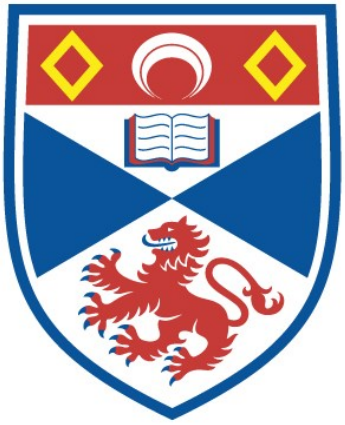
USING SUBGOAL CHAINING TO ADDRESS THE LOCAL MINIMUM PROBLEM

Jonathan Peter Lewis

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



2002

Using Subgoal Chaining to Address The Local Minimum Problem

A thesis submitted to the

UNIVERSITY OF ST ANDREWS

for the degree of

DOCTOR OF PHILOSOPHY

by

Jonathan Peter Lewis

School of Computer Sciences

University of St Andrews

September, 2001

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Declarations

I, Jonathan Peter Lewis, hereby certify that this thesis, which is approximately 44,000 words in length, has been written by me, that it is the record of work carried out by me, and that it has not been submitted in any previous application for a higher degree.

date 4.6.02 *signature of candidate* _____

I was admitted as a research student in November 1997 and as a candidate for the degree of Doctor of Philosophy in November 1998; the higher study for which this is a record was carried out in the University of St Andrews between 1997 and 2001.

date 4.6.02 *signature of candidate* _____

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date _____ *signature of supervisor* _____

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any *bona fide* library or research worker.

date 4.6.02 *signature of candidate* _____

Abstract

A common problem in the area of non-linear function optimisation is that of not being able to guarantee finding the global optimum of the function in a feasible time especially when local optima exist. This problem applies to various areas of heuristic search. One of these areas concerns standard training techniques for feedforward neural networks. The element of heuristic search consists of attempting to find a neural weight state corresponding to the lowest training error. This problem may be termed the local minimum problem.

The local minimum problem is addressed for feedforward neural networks. This is done by first establishing the conditions under which local minimum interference for the training process is to be expected. A target based approach to subgoal chaining in supervised learning is then investigated. This is a method to improve travel for neural networks by directing it more precisely through local subgoals than may be achieved through a more distant goal. It is shown however that linear subgoal chains are not sufficient to overcome the local minimum problem. Two novel training techniques are presented which use non-linear subgoal chains and are examined for their capability to address the local minimum problem.

It is found that attempting to target a neural network to do something it cannot may lead to suboptimal training. It is also found that targeting a network to do something it is capable of generally leads to successful training. A novel system is presented which is designed to create optimal realisable targets for unrealisable goals. This allows neural networks to subsequently achieve the optimal weight state through a sufficiently powerful training method such as subgoal chaining. The results are shown to be consistent with the theoretical expectations.

Acknowledgements

The most sincere thanks to Mike Weir, without whose constant encouragement and tireless supervisory commitment this thesis and the work it entailed would not have been possible.

My deepest gratitude goes to my parents, whose moral and financial support over the years has enabled me to complete my work.

I would also like to thank Brian, Helen and Joy, from the School of Computer Science.

Last, but certainly not least, I would like to thank all my friends for their unwaivering support.

This thesis was supported by the EPSRC.

Contents

Chapter 1

Goal Directed Learning for Neural Networks

1.1 Neural Networks

1.1.1 The Brain as a Neural Computer

When looking at the brain as a computing unit we can see that in many areas it is more powerful than some of today's most sophisticated computer systems, especially in areas concerning vision, speech, motor control and facial recognition, i.e. tasks which would be considered to be easy for humans to perform. The brain has the ability to form generalisations and characterise objects or circumstances that it has not been subjected to before, to a greater extent than traditional computer systems. It is also far more robust than a computer

in the sense that if parts of it are slightly damaged it can still function and in some cases will adapt. Flexibility, fault tolerance and generalisation ability coupled with a highly parallel design may be seen as features which are desirable when a system is required to function in the real world. This realisation can be seen as the main motivation to investigate ways of making the architecture of computers more similar to that of our brain.

A neural network can be seen as a group of interconnected simple units in which the structure and type of connections between the units define the computational capability of the network.

One of the early models which is a basis for most neural networks was suggested by neurophysiologist Warren McCulloch and logician Walter Pitts in 1943 and described the brain as consisting of many interconnected neurons. Their model has these neurons sending simple messages to each other which are excitatory or inhibitory pulses and has the neurons updating their excitations on the basis of these messages. They subsequently proposed a model for a neural network consisting of an array of variable resistors on connection lines between summing amplifiers to mimic the way in which neurons operate in the brain.

In 1949 Donald Hebb suggested that a group of neurons could reverberate in different patterns which could correspond to the brain remembering different experiences. Hebb also speculated about synaptic modification by which learning could possibly be achieved. In 1962 Frank Rosenblatt coined the expression *perceptron* with which he referred to a learning image recognising unit based on McCulloch and Pitts' model combined with some non-learning feature detectors. Today one would use the term perceptron in a slightly dif-

ferent context, i.e. omitting the feature detectors. With this adjustment the term perceptron has survived and denotes a simple unit which is capable of learning and may be a part of an arrangement of many such units, i.e. a neural network. He also produced a convergence theorem for perceptrons that guarantees them converging to a solution in finite time if one exists, by using a learning rule which can either increase or reduce the connective strengths.

In their book *Perceptrons* Minsky and Papert describe the class of problems which may be solved by perceptrons as very restricted, i.e. to those only involving linear mappings. This limitation on the use of perceptrons may be seen as a reason for a decline in research taking place in that area.

A way to overcome the limitation of the perceptron was found by extending its architecture to allow multiple layers of neurons and finding a learning rule to train them. Today this type of network is generally called a multi-layer feedforward network and the learning rule used to train it is now called error propagation, back-propagation or generalised delta rule and was popularised in 1986 by Rummelhart, Hinton and Williams in [RM86].

The mapping ability of multi-layer feedforward networks was shown to be superior to that of the perceptron. In particular, the class of solvable problems broadens to include non-linear as well as linear mappings and the availability of back-propagation as a learning rule may be seen as a reason for a second wave of interest in neural computation.

1.1.2 Feedforward Neural Networks

There are many different types of artificial neural network (ANN), but I will be merely concentrating my studies on so-called *feedforward* neural networks. This type of network can be seen to describe an arrangement of units in different layers such as *input* layer, *output* layer and intermediate *hidden* layers. Figure 1.1 is a stylised view of a feedforward neural network. In a fully connected network each unit in one layer is connected to all units in adjacent layers via *weighted* connections. If the network is *strictly layered* then connections are only between units in adjacent layers. The weight on a connection modifies the strength of a signal value being passed along the connection by multiplying the signal value by the numerical value of the weight.

In general an input signal consisting of a numerical value for each unit in the input layer is applied. Each unit in the adjacent layer of units calculates the weighted sum of the incoming signal values which is termed its *excitation*. The units' excitation values, which may range between negative and positive infinity, are squashed down to output values which range between 0 and 1 using a squashing function which is called the *activation function* for neural networks. A commonly used activation function is the sigmoid for which the output of a unit can be calculated as a function of its excitation as

$$\text{out} = \frac{1}{1 + e^{-\text{ex}}} \tag{1.1}$$

where out denotes the output value and ex denotes the excitation value.

Each output from these units is in turn fed forward along weighted connections to units in subsequent layers and each unit along the way produces an output as a function of its

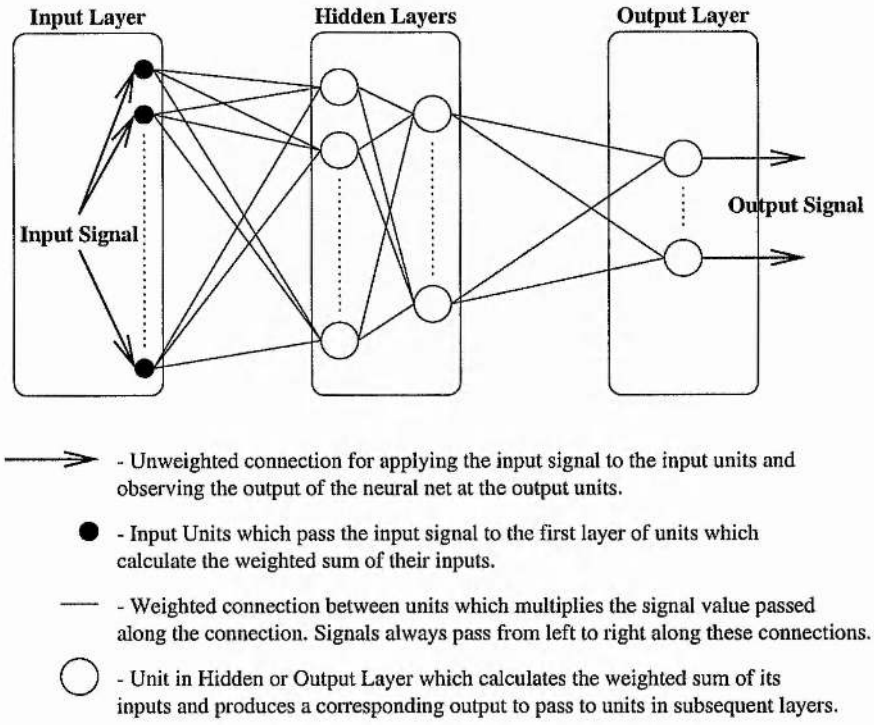


Figure 1.1: Stylised view of a fully connected and strictly layered feedforward neural network. The input signal is applied to the input units in the input layer and passed along weighted connections to units in subsequent layers until the output signal is emitted by the output units. Although 2 layers of units have been depicted as the hidden layers, any number of hidden layers may constitute such a neural network.

weighted sum of inputs. In this way the input signal is fed forward through the entire network such that a signal is eventually emitted from the output units in the output layer.

The only units which do not calculate the weighted sum of inputs are the input units in the input layer which have an input signal applied to them directly and pass this signal to the adjacent layer. The input layer is consequently treated as a special layer in this work and is disregarded when referring to the number of layers in the network. In the following a single layer network for example refers to a network with an output layer and its inputs in the input layer. A multi-layer network refers to a network with an output layer, at least one

hidden layer and the network inputs in the input layer.

The output emitted from the neural network for an input presented to the network is a function of the neural weight settings. These settings comprise the internal state of the network and as a collection are commonly referred to as the *weight state*. The weight state can be seen as a vector containing the value of the weight setting for every weighted connection in the network.

1.1.3 Learning for Neural Networks

Learning for neural networks can be divided into two parts, the first being *training*. For feedforward networks the training phase typically consists of starting the network at some randomly generated initial weight state and attempting to adjust this weight state via a learning rule such that for an array of input signals, the corresponding output responses match the desired responses, otherwise known as targets, as closely as possible. Because each training input has an associated target response this form of neural training falls into the category of *supervised learning*. Each training input is an N-dimensional vector, where N is the number of input units in the network. Similarly the target output is an M-dimensional vector where M is the number of output units in the net. A training pattern consists of a training input and its desired or target response. The set of all training patterns the neural network is required to learn is called the *training set*.

The second part of the learning process is *generalisation*. The aim of generalisation is that the trained neural network should produce fairly accurate outputs, i.e. outputs that match

their targets reasonably well, for inputs it was not trained on as well as for the trained inputs. One example is to train a neural network to recognise a specific hand written character "A" and in so doing to get the network to generalise in terms of recognising a number of "A"s in different shapes and sizes. The capability of neural networks to generalise in this way may be seen as a very important feature but within the scope of this thesis I will only be dealing with the training aspect to learning.

To recap, the objective of training a feedforward network in supervised learning is to find a weight state for which the neural network produces an input to output (I/O) relation which is as close as possible to the desired I/O relation. A space called *input space* can be defined such that each training pattern's input vector corresponds to a point in that space. The neural network will produce an output for each of the points in input space depending on its weight state. The output produced for each training point can be used to effectively colour the point in input space. This extension of input space is commonly called I/O space because the neural network's I/O relation can be represented in it.

When training within the context of supervised learning an error measure can be defined which reflects how well the neural outputs match their target values over all patterns in the training set. This error measure is either greater than zero or a zero valued quantity if all outputs match their targets exactly. The aim of training is then to find a weight state that minimises this error measure for the training set. A space called *weight space* can be defined in which a weight state corresponds to a point. The dimensionality of weight space is the number of weights in the neural network. The error is a function of the neural outputs for the training set which in turn are a function of the neural weight state. This weight space

can be extended by 1 dimension to include the error measure, a function of the weight state, as an additional height in which case the space is called *error-weight space* and contains error-weight states. The error as a function of the weight state defines an error-weight surface in error-weight space. Training the neural network is then equivalent to finding the lowest point on the error-weight surface.

Some people working with neural networks may argue that weight states producing good generalisation are seldom very close to those which minimise training error. Hence they may argue that there is no reason to find the lowest point on the error-weight surface for the training set when the ultimate aim is to get the neural network to generalise well. By the end of the thesis this will have been addressed such that various responses to this point may be made.

1.2 The Local Minimum Problem

Many techniques for optimising functions in multiple dimensions, that is finding the lowest or highest extreme point on them, have been developed within the framework of standard numerical analysis in mathematics. It is maybe not surprising then that in general the development of training techniques for feedforward neural networks can be seen to have followed the development of optimisation techniques in numerical analysis. The dominant training techniques in particular follow those in a major book on numerical methods [Pre94]. While the individual training techniques undoubtedly differ in various ways it is a similarity shared by most of them that is of some concern.

This similarity is that most of the techniques are based on some form of error reduction by a technique which is commonly referred to as *gradient descent*. Gradient descent or *hill-descent* is an iterative approach to minimising a function. The process is initialised at a randomly generated starting state and effectively takes a sequence of steps down the hill until it reaches the bottom of a valley. The concern being raised with this is that such techniques have a well known limitation.

Minsky makes a strong point to this effect with regard to a neural training technique called *Error Back-propagation* in the expanded edition of Perceptrons [MP88]. He also makes a stab at *connectionists*, that is people involved with neural networks, for possibly not having recognised the limitation of the technique.

> "We have the impression that many people in the connectionist community do not understand that this [Back-propagation] is merely a particular way to compute a gradient and have assumed instead that Back-propagation is a new learning scheme that somehow gets around the basic limitation of hill-climbing."
> *Minsky*

The basic intuition of performing hill-climbing is the same as for hill-descent, only the objective is to find the maximum of the function rather than the minimum. Hill-climbing will ensure finding the highest peak on a hill in the local vicinity of where it was started. The basic limitation is that this peak may not be the highest peak on the highest hill. This same limitation also applies to the opposite of hill-climbing, namely gradient descent. Gradient descent is able to find a minimum in the local vicinity of its starting point, i.e. a *local*

minimum, but is unable to find the global minimum in a single attempt.

Undoubtedly, many connectionists do understand the limitation, the issue is what should be done about it. Minsky's point is echoed and extended by Bianchini et al in [BFGM98].

> "Because simple gradient descent algorithms get stuck in local minima, in principle, one has no guarantee of learning the assigned task ... Hence it turns out to be very interesting to investigate the presence of local minima and particularly to look for conditions that guarantee their absence." *Bianchini*

The lack of being able to find the global minimum in a single attempt is mostly approached by allowing multiple random initialisations. The problem with this is that the number of initialisations needed for success is a quantity which is unknown a priori as it depends on the relative sizes of local and global minimum *attractor basins*, which cannot be established without knowing where the global minimum is. A basin of attraction for a minimum in weight space defines the set of weight states that will produce convergence to the minimum when used as initial weight states for gradient descent.

In addition the error value at the global minimum may well be a non-zero quantity which is also unknown a priori. Without knowing the error at the global minimum the only stopping criterion that guarantees having found the global minimum is finding a state corresponding to zero error, because there can be no minima with a lower error than zero. If the error at the global minimum is indeed a non-zero quantity then the training technique will in theory continue searching ad infinitum. Trivially this is infeasible so that there is no guarantee for success when using multiple random initialisations.

In practice the number of runs is specified a priori and the run with the lowest final error is taken in the knowledge that it may not be the global minimum state if its associated error is non-zero. The problem with attempting to establish the global minimum state in the presence of local minima will in future be referred to as the *local minimum problem.*

The main area of concern for the thesis is to find a way to address the local minimum problem when training feedforward neural networks. In order to address the local minimum problem a fundamentally novel way of directing neural training is desired.

1.3 Goal Directed Training

Behaviour which is directed by trying to achieve *goals* is a very familiar concept for us because the animate world, including ourselves, exhibits such behaviour in everyday life. Quite often the goal may be broken down into a set of *subgoals*, a subset of which may need to be achieved in series and/or possibly in parallel in order to reach the goal. Examples of human activities in which the use of subgoals is readily apparent are route planning and problem solving.

The use of subgoals may have come about in animate behaviour because subdividing the goal into a sequence of subgoals allows the behaviour to be directed towards the goal more successfully in stages than by going straight for the goal. This is in essence because of the unpredictable nature of the constantly changing environment in the real world in which, as an added difficulty, the quantitative relation between cause and effect is mostly excessively

complex.

The benefit of using subgoals is that the behaviour may constantly adapt to the subgoals as they are placed on the agent's agenda. In addition the agent's behaviour may be directed more precisely at each stage by the nearby subgoal in comparison to being directed by the more distant long term or overall goal. The improved precision is because simplified approximations to the complex relation between cause and effect used to guide the agent's behaviour are more precise for small changes in desired effect, i.e. for nearby subgoals.

Although the use of subgoals can be seen as beneficial in terms of aiding human activities the use of subgoals is not commonly seen to be represented within the realm of neural training. The overall goal can be thought of as a desired I/O relation. However, the term "goal" as used in the thesis will normally be target based. The way in which goal directedness will be incorporated into the context of neural networks in this thesis will involve investigating how output targets including subgoal targets may be set in order to aid the neural training process.

In order to do this it is helpful to be able to form a graphical representation of goal and subgoal targets that will be used in the training process. For this purpose it is necessary to introduce two more spaces in addition to the commonly used input space, I/O space, weight space and error-weight space. Both spaces are only defined for output units here. For simplicity the spaces will be defined for a network containing a single output unit although the concept may be generalised to larger numbers of output units.

Each training pattern's input vector generates a corresponding network output as a function

of the weight state. A vector containing the network outputs for every input vector in the training set is the output state of the network. For a single output unit and P training patterns, output states are P-dimensional. The first space to be defined is *output space* in which any output state can be represented as a point. Consequently output space is P-dimensional.

The target output state, i.e. the target outputs for all the training patterns as defined in the training set, may be seen as the goal state for the training process. This goal can, like any other output state, be represented as a point in output space. Consequently setting a subgoal for the neural training process corresponds to temporarily substituting the goal targets for a subgoal output state which again can be represented by a point in output space.

The second space to introduce at this point is one closely related to output space and is called *excitation space*. Like output space excitation space is only defined for output units and has the same dimensionality as output space. The relation between excitation space and output space is that every state in excitation space, i.e. *excitation state*, corresponds to a unique output state which can be calculated by applying the neural activation function. An excitation state is merely a vector containing the excitation values of the output unit for every training pattern's input vector. Neural output states, the goal and subgoals may now be represented in output space or by their equivalent excitation states in excitation space.

1.4 Thesis Structure

In chapter 2 I will present standard neural training techniques, most of which are based on some form of gradient descent, do not make use of subgoals and suffer from the local minimum problem.

In chapter 3 it is explained how symbolic AI attempts to overcome the local minimum problem using various search techniques. The use of subgoals in symbolic AI to break down large search spaces in order to facilitate complex problem solving will also be described.

Chapter 4 presents some neural training techniques which set subgoals in a straight line along a so called *subgoal chain* in output space. Two of the new linear subgoal techniques are further developments of the Tangent Hyperplanes (TH) technique originally developed by Antonio Fernandes during his PhD [Fer97]. The further developments to this technique were conceptualised and implemented by the St Andrews neural group from 1998 until 2001. Some issues with using linear subgoal chaining are presented. Another technique called ERA is examined with respect to the claims that were made for it by its creators, namely that it overcomes the local minimum problem. The reasons why ERA must in theory fail to avoid convergence to local minima are developed.

Techniques for creating local minima on a neural network's error-weight surface by manipulating the neural training set are developed in the first part of chapter 5. Specific training set examples are given which were created using these techniques. A conceptual reason for the existence of local minima and when local minima are to be expected, and when not, is explained in the second part of chapter 5. This explains the notion of *unrealisable*

regions in output space as regions of output space containing output states which are not producible, i.e. *realisable*, by the neural network for any weight state. Third, the training examples created in the first part of the chapter are used to test the claims made for the ERA technique empirically and the results are presented which show ERA failing to overcome the local minimum problem. Making use of the concepts regarding local minima and unrealisable regions from earlier parts of this chapter allows the reasons and conditions for ERA's failure to be re-examined.

Chapter 6 presents the key components of a new model and method to allow a subgoal chain to adaptively shape itself during training into a non-linear shape in order to allow training to overcome local minima. The training technique is tested on various test problems including those on which ERA was shown to fail to reach the global minimum. The results are very promising in as much that the new technique outperforms both standard techniques and ERA on the test problems. Nonetheless it is concluded that the technique has scope for improvement in terms of the precision with which it aims to achieve subgoals.

In chapter 7 a method is described which is aimed to provide the desired improvement of the method described in chapter 6. The initial implementation of the new method is successful in terms of improving the accuracy of aiming for subgoals. Early on in testing however it became clear that oscillations are being introduced into the travel when aiming for unrealisable subgoals by precisely the mechanism which prevents the technique from getting stuck at a local minimum. It is also observed that setting a realisable goal induces realisable subgoals which leads to successful training. This in turn leads to the final design of a system to overcome the local minimum problem.

Chapter 8 presents the key components of a method to establish an estimate of the global minimum output state prior to training which is called the global positioning system (GPS). The idea is that training a neural network to achieve the global minimum output state, which is realisable, is likely to induce a realisable subgoal chain that may enable successful training to the global minimum weight state. Training to the global minimum weight state occurs on a first time basis and training is stopped when the weight state produces an output state close enough to the global minimum output state. GPS is practically a stand alone system which can be run once before training to establish an estimate of the global minimum output state in polynomial time in terms of the number of training patterns. TH may then be used to establish the global minimum weight state. GPS is tested on one of the training examples used for testing ERA, and on some other test problems which allow a graphical insight into the GPS procedure.

Chapter 9 presents the conclusions and includes a further work section.

1.5 Notation

1.5.1 Mathematical Notation

Scalar variables consisting of 1 character are represented in either lower or upper case non-bold italics such as x or E for example. If multiple characters are used such as for excitation being denoted by ex, then non-italic characters are used.

The multiplication of two scalars a and b is denoted as $a\,b$. The multiplication of numerical

values on the other hand is denoted as $9 \times 8 \times 7$ for example. The multiplication of a numerical value 3 with a scalar a for example is denoted as $3a$.

The absolute value of a scalar a is written as $|a|$.

Vectors are represented in upper or lower case bold type non-italic such as the vector $\mathbf{v}$. This notation generally refers to vectors whose elements are arranged in different columns rather than rows unless the elements for a specific vector are written out in which case the row form of the vector is adopted for the reason that it takes up less space on a page.

The length of a vector $\mathbf{v}$ is denoted by $||\mathbf{v}||$.

The elements of vector $\mathbf{v}$ are italicised such as v_i where the subscript indicates the element's position in the vector and can range from 1 to $\dim(\mathbf{v})$ where $\dim(\mathbf{v})$ is the dimensionality of the vector $\mathbf{v}$.

The scalar product or dot product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{a} \cdot \mathbf{b}$. Vectors may also have subscripts to denote there is an array of say P individual vectors $\mathbf{v}_p$ where p ranges between 1 and P.

The multiplication of a scalar s with a vector $\mathbf{v}$ is represented as $s\,\mathbf{v}$.

Scalar functions, i.e. functions which return a scalar value as a function of a scalar or a vector, are denoted in the same way as scalars are. For example $g(x)$ and $E(\mathbf{w})$ and $\text{out}(\mathbf{w})$ denote scalar functions. Vector functions, i.e. functions which return a vector as a function of scalars or vectors are denoted in the same way as vectors are, such as $\mathbf{v}(t)$ and $\mathbf{O}(\mathbf{w})$.

The exception to displaying vectors in bold type non-italic is for a vector resulting from

an operator such as the gradient operator ∇ operating on a scalar function E which is represented as ∇E.

For a function of many variables $f(x_1, \ldots, x_i, \ldots)$ the partial derivative of the function with respect to one of the variables x_i is written as $\frac{\partial f}{\partial x_i}$.

Matrices are represented in upper case bold type non-italic such as a matrix $\mathbf{A}$ and its elements are represented in lower case italic as a_{ij} for example, where i denotes the i^{th} row and j denotes the j^{th} column.

The transpose of a matrix $\mathbf{A}$ is displayed as $\mathbf{A}^{\mathrm{T}}$. Similarly the transpose of a vector $\mathbf{v}$, which transforms a column vector into a row vector and vice versa, is written as $\mathbf{v}^{\mathrm{T}}$. The inverse matrix of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^{-1}$

The multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$ is written as $\mathbf{A}\,\mathbf{B}$ and the multiplication of a matrix $\mathbf{A}$ and a vector $\mathbf{v}$ is written as $\mathbf{A}\,\mathbf{v}$.

1.5.2 Figures, Tables and Equations

All figures, tables and equations are numbered to include the chapter number they occur in and the number denoting their relative position in the chapter.

When referring to a figure or table from within the text the word Figure or Table will be put before the reference number. For example Figure 3.1 refers to the 1st figure in chapter 3.

When referring to an equation from within the text the reference number is placed within

curved brackets. For instance (4.2) refers to the 2nd equation in chapter 4.

Chapter 2

Common Neural Training Techniques

In this chapter I will be taking a look at a few standard training techniques for feedforward neural networks. Feedforward neural networks use supervised learning which means that the desired or goal neural response to a set of training inputs is known. The neural response can be graded according to the desired response and be used to direct training. In this sense all supervised training regimes are goal seekers because they aim to adjust the neural weights in order to obtain the goal response, but are not generally goal directed in the sense that subgoals may be set in order to facilitate learning. In fact neural training techniques seem to have followed in the footsteps of classical numerical analysis.

2.1 Back-propagation

The back-propagation algorithm, commonly referred to as backprop (BP), was first popularised in [RM86]. It was the first algorithm which allowed feedforward networks with hidden layers to be trained. Feedforward networks are networks in which the input signal is fed forward from input units in the input layer along weighted connections to units in successive layers until the output layer is reached. Multi-layer networks are networks which have so called hidden layers in between input and output layers. After feeding an input signal forward through the layers the neural network produces an output.

Before the introduction of BP, single layer networks could be trained using an algorithm called the delta-rule [BE60]. Single layer networks can only learn to classify linearly separable data which imposes a big limitation on their use because many classification problems in the real world are not linearly separable. When BP was introduced it offered a way to train multi-layer networks which can perform non-linear mappings. This caused a resurgence of interest in neural networks.

When training a neural network using supervised learning, the desired target output of the neural network corresponding to the presented input is a known quantity. The pairing of the neural network's input signal and the desired target output constitutes a training pattern for the neural network. It is possible to assign a measure to the neural network's output which reflects how well the neural output matches the target output. The objective of the training process is to set the neural weights such that the neural output matches the desired target output as closely as possible. If we define an error function e_p for an output unit for

training pattern p as

$$e_p = \frac{1}{2}(\text{out}_p - T_p)^2 \tag{2.1}$$

where out_p is the neural output for training pattern p and T_p is the target output for pattern p then there is an error for each pattern associated with the neural output for that pattern. The error measure will either be a positive quantity or zero when the neural output matches its target. The training process now becomes a matter of reducing the error measure. Because the neural output is a function of the neural weights, the error function is implicitly also a function of the neural weights. Due to this correspondence it is possible to talk of an error-weight surface for each pattern, where the error forms the height of the surface for every weight setting. [1] When a function of many variables is differentiated with respect to one of the variables this forms a so called *partial* derivative which I will also refer to as partial gradient. For information on derivatives, partial derivatives, gradients and ways to calculate them using a rule called the chain rule I refer the reader to [Jef85] for example. Essentially BP is a way to apply the chain rule to neural networks in order to compute partial gradients on the error-weight surface with respect to the neural weights, in order to obtain a direction of weight adjustment which reduces the error.

Specifically BP provides a way to calculate the partial derivative $\frac{\partial e_p}{\partial w_{ij}}$ for any weight w_{ij} leading from unit i to unit j in the network by propagating the error signal at the output layer back to weights connecting to previous layers. BP itself is not an optimisation method

[1] The shape of the error-weight surface for a given training set may be seen to depend on the choice of error function, the number of hidden units and the activation function for the output unit. In certain circumstances this choice may guarantee the absence of local minima on the surface. Investigations along these lines may be found in [BFG95], [BH89], [FGT92], [BF91] and [SS91].

but merely provides these partial gradients which can be used to update the neural weights in a multi-layer network in order to reduce a chosen error measure.

When training a neural network it makes sense to reduce the error not only for one pattern but for all patterns and thereby get a good performance over all patterns. The error measure to be minimised is commonly termed Least Mean Square (LMS) error, where "Least" refers to the minimisation and "Mean" refers to taking the mean over all patterns. For P training patterns LMS error can be written as

$$E_{\text{LMS}} = \frac{1}{P} \sum_{p=1}^{p=P} (\text{out}_p - T_p)^2 \tag{2.2}$$

Quite often the mean $\frac{1}{P}$ is replaced with $\frac{1}{2}$ to make the differentiation of the error function nicer and so it becomes

$$E = \frac{1}{2} \sum_{p=1}^{p=P} (\text{out}_p - T_p)^2 = \sum_{p=1}^{p=P} e_p \tag{2.3}$$

This error function is either always positive, as before, or zero now if all patterns' outputs match their respective targets. The objective of the training process is now to minimise this overall error function E. BP is used to calculate the partial gradients on the error-weight surface which is now the superposition of all individual patterns' error weight surfaces and the gradients are used to obtain a direction of decreasing error. The partial gradients for the overall error function are simply the summation of the individual patterns' error-weight gradients over all patterns.

$$\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{p=P} \frac{\partial e_p}{\partial w_{ij}} \tag{2.4}$$

Some form of gradient descent can then be used to iteratively step down the combined error-weight surface in order to find a minimum on the surface. Steepest gradient descent,

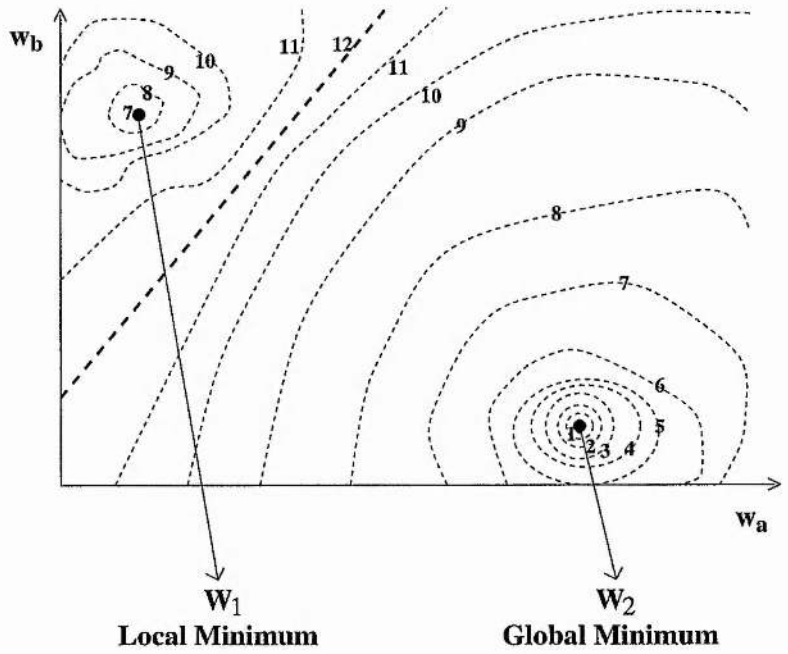


Figure 2.1: Stylised view of error-weight space with weight axes w_a and w_b and error is indicated by the number on the contour lines. W_1 is a local minimum and W_2 is the desired global minimum.

as the theoretically most pure method, takes a step down the surface in the direction in which the error decreases most steeply.

Error weight surfaces for each pattern individually have empirically been found to be relatively simple for gradient descent techniques to traverse. That is local minima, which are defined as being states on the surface surrounded by higher error in all directions locally, while not being the lowest error on the whole surface, have not been reported for these surfaces. When performing steepest gradient descent on such simple surfaces the global minimum is usually found in a few iterations of BP. This is also reported by Weir and Chen in [WC90].

In contrast, the superposition of the individual pattern error-weight surfaces often creates

complex surfaces which are difficult to travel on using gradient descent techniques. One such complexity is the occurrence of local minima in the error-weight surface. Local minima have often been reported such as by Brady in [BR88] and examined by Sontag and Sussmann in [SS89]. Figure 2.1 shows what a global and a local minimum may look like for a neural error weight surface. The problem of performing gradient descent on such an error weight surface is that once the state is within the local minimum's basin of attraction, training will eventually get stuck at the local minimum. When initialising the training process at random states the probability of reaching the desired global minimum is proportional to the ratio of the global minimum basin size to the local minimum basin size. For single layer networks consisting of one neuron only it has been shown that the occurrence of multiple local minima is common place. Auer et al reported in [AHW97] that the number of local minima rises exponentially with respect to the dimensionality and number of input patterns for single neurons. Various examples of local minima for single layer networks are also described by Lewis and Weir in [LW99] which present great difficulties for BP.

The superposition of the individual pattern error weight surfaces also creates other difficulties for gradient descent techniques such as ravines as reported by Weir in [Wei91]. Figure 2.2 is a stylised view of a ravine shaped error weight surface. Following the steepest descent direction on such ravines will cause a *zig-zagging* travel path if the step-size is not exactly such to direct the transition to the centre of the ravine. Each blob on the *zig-zag* path in Figure 2.2 denotes a state achieved through one single step from the previous state. In order to avoid transitions bouncing out of the ravine to higher error states the step size

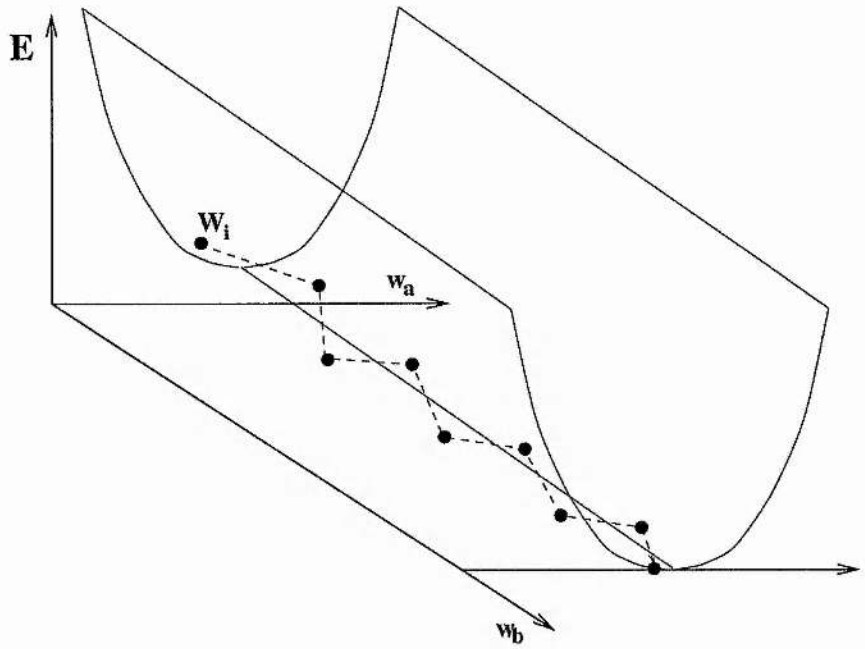


Figure 2.2: Stylised view of error-weight space with weight axes w_a and w_b and an axis for the error **E**. W_i denotes an initial starting state and the dashed lines indicate the *zig-zag* line of travel obtained when following the steepest gradient descent.

must be set low which slows the training process down. The problem is then that training may often be so slow that a minimum is not found in the time allowed.

Another problem for BP is the occurrence of shallow regions on the error-weight surface with almost constant high error. Such regions are commonly termed plateaus. When the error-weight state is situated on such a plateau, training will proceed so slowly, due to the shallow gradients, that it may take a very long time to reach regions of lower error, and again a minimum may not be reached in the time allowed.

In [RM86] two methods are presented for performing gradient descent using the partial gradients obtained via BP. One of the methods is off-line BP, also commonly referred to as batched BP, and the other is on-line BP. Off-line BP is the more theoretically pure method

in which the weights are updated after all the patterns have been presented and the gradients have been calculated for each weight over all the patterns p according to (2.4). Specifically each weight w_{ij} is then updated in proportion to the calculated gradients, i.e.

$$w'_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} = w_{ij} - \eta \sum_{p=1}^{p=P} \frac{\partial e_p}{\partial w_{ij}} \tag{2.5}$$

where w'_{ij} is the weight value after the weight transition is made, and η is the step size of the weight update, commonly known as the learning rate. This kind of weight update is equivalent to performing steepest gradient descent and so off-line BP finds surfaces with plateaus, local minima and ravines difficult to traverse. It is possible to add a momentum term to the weight update which has been shown to smooth out the oscillations encountered when travelling on ravines to a limited extent. The smoothing occurs by adding a fraction of the previous transition direction to the current gradient in the hope that they will cancel out to provide a transition which does not shoot up the other side of the ravine. A low learning rate will also reduce the oscillations when travelling on ravines, but will increase the overall training time and so has to be set with care. In practice a useful learning rate and momentum setting depends on the shape of the surface. The wrong learning rate and momentum setting may cause learning to follow oscillatory trajectories of increasing amplitude and bounce out of the ravine. Ironically this undesired effect has been quoted as being beneficial for dealing with local minima. High momentum and learning rate may bounce gradient descent training out of a local minimum's basin of attraction, but the cause of this success is by creating training which does not always converge to a minimum in the current basin of attraction. Hence high momentum and learning rates may likewise bounce the trained state into a local minimum basin.

For on-line BP the procedure is very similar, the only difference being that the weight update occurs after each pattern is presented. All the weights are updated as follows

$$w'_{ij} = w_{ij} - \eta \frac{\partial e_p}{\partial w_{ij}} \tag{2.6}$$

Compared to batched BP, on-line BP offers the advantage of multiple directions for the price of one. That is because a different transition direction is explored for every pattern as it is presented, but the overall cost of computing the gradients is the same as for batched BP. The drawback of on-line BP is that a good direction for one pattern is not necessarily a good direction for another pattern and hence multiple oscillations may occur within the cycle of presenting all the patterns once. The overall merit of one version of BP against the other appears unclear conceptually. While on-line BP may possibly be more suited for certain types of training sets batched BP may be better suited for others.

In summary, BP offers a general method for optimisation through error reduction, but suffers from finding complex surface traversal difficult. Its versions of gradient descent fail to overcome the classical difficulties for gradient descent, namely ravines and local minima.

2.2 Conjugate Gradient Descent

Conjugate Gradient Descent (CGD) is a method to improve on the directions obtained for training a neural network compared to those obtained using steepest gradient descent. When performing some version of steepest gradient descent on an error surface, such as using BP as described in section 2.1, the direction does not generally point towards a minimum error state on the error-weight surface. Depending on the learning rate and momentum setting this often leads to arbitrary oscillations in surface regions with ravines. The oscillations substantially increase the training time required to reach the global minimum, provided a local minimum is not reached instead, in which case failure occurs. The basic idea of CGD is to, at each stage, preserve the goodness of previous transitions as much as possible by choosing successive travel directions which avoid oscillations.

Part of the CGD technique uses line minimisation, also referred to as line search, which in a neural sense is a search for minimum error along a fixed line in weight space. Line search alone does not improve matters compared to BP, although it achieves maximum goodness along its direction. If no momentum term is used and the direction for the individual line search is along the steepest descent direction then each search direction will be orthogonal to the previous direction as shown in [Pre94]. Even for a simple quadratic 3-D error-weight surface, with 2 weight axes and one error axis, the orthogonal directions do not generally point directly towards the overall global minimum on the surface. Performing iterative line search on such a surface will then lead to many steps being taken in orthogonal directions with the step size decreasing as the search nears the minimum.

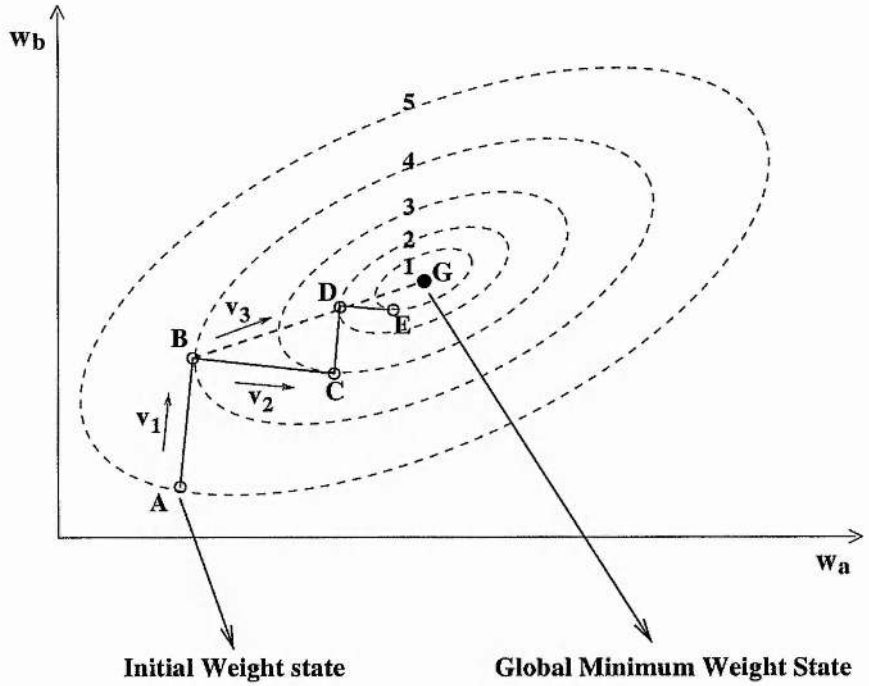


Figure 2.3: Stylised view of weight space with 2 weight axes $\mathbf{w_a}$ and $\mathbf{w_b}$. Numbers on the dashed error contour lines are not to scale but denote the increasing height of this simple quadratic error function around its one minimum which is therefore a global minimum.

Figure 2.3 is a schematic view of 2-D weight space with a quadratic error function of the weights as the 3rd dimension of the error-weight surface. The direction of increasing error is denoted by numbers on the dashed error contour lines. When starting at the initial weight state **A** and performing line minimisation along the steepest descent direction $\mathbf{v}_1$, **B** is obtained as the minimum error state. It should be noted that the steepest descent direction, at any state, is orthogonal to the tangent on the error contour line containing the current state. It should also be noted that the minimisation direction forms the tangent to the error contour line containing the minimum along the minimisation direction. If the minimisation direction is always chosen to be the steepest descent direction, then successive search directions are necessarily orthogonal to each other. Thus line searches from **B** along the

steepest descent direction at each stage will take the search to the states **C**, **D**, **E** and will continue in a zig-zag path along the directions $\mathbf{v}_1$ and $\mathbf{v}_2$, with an ever decreasing step size, towards the minimum state at **G**. Following such a path means that a lot more steps are taken than necessary for such a simple surface.

The other part of the CGD technique offers a way to preserve the goodness achieved through previous line searches so as to avoid travelling on the zig-zag path just described. Related to the simple example in Figure 2.3, CGD offers a way to establish a search direction $\mathbf{v}_3$ for the 2nd line search, when situated at **B**, which passes through the minimum at **G**. When starting at **A** in Figure 2.3 the initial direction for the line search is chosen to be along the steepest descent direction $\mathbf{v}_1$ and so **B** is the obtained minimum for that line search. For the second line search CGD allows the calculation of a conjugate direction to $\mathbf{v}_1$. The conjugate direction to $\mathbf{v}_1$ is a direction along which the value of the gradient on the error-weight surface along the $\mathbf{v}_1$ direction is the same as it was at **B**. The conjugate direction at **B** for $\mathbf{v}_1$ and an error-weight gradient of zero along the $\mathbf{v}_1$ direction is $\mathbf{v}_3$, which passes through the minimum at **G** and any other line minimisation along the $\mathbf{v}_1$ direction, such as **D** for example. The 2nd line search can then be performed along the $\mathbf{v}_3$ direction to obtain the desired minimum weight state at **G**.

In order to calculate the conjugate directions, CGD makes use of 2nd order derivatives of the error function with respect to the weights. These 2nd order derivatives are grouped into a matrix called the Hessian after the mathematician Otto Hesse. For an error function E of

N weight parameters w_i where i can range from $1 \ldots N$, the Hessian can be written as:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 E}{\partial {w_1}^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E}{\partial {w_N}^2} \end{pmatrix} \tag{2.7}$$

Finding the conjugate direction to $\mathbf{v}_1$ involves evaluating the Hessian and solving the following equation expressed here in matrix notation

$$\mathbf{v}_1{}^{\mathrm{T}} \mathbf{H} \mathbf{v}_3 = 0 \tag{2.8}$$

in terms of the components of the vector $\mathbf{v}_3$

$$\mathbf{v}_3 = \begin{pmatrix} v_a \\ \\ v_b \end{pmatrix} \tag{2.9}$$

where v_a and v_b are values along the $\mathbf{w_a}$ and $\mathbf{w_b}$ axes in Figure 2.3.

The 2nd order derivatives of the error function with respect to the weight parameters in the Hessian indicate how the gradient on the error-weight surface will vary for changes in weight. For error-weight surfaces which are quadratic functions of the weight parameters all entries in the Hessian are constant over all weight settings. The Hessian is an exact model of the surface shape in this case and CGD is able to find the minimum in at most N applications of line search for a quadratic error function of N weight parameters.

In order to find a set of conjugate directions it is not necessary to compute the Hessian explicitly. Instead conjugate directions may be generated as follows. At time $t = 1$ set the first search direction, $\mathbf{v}(1)$, to the negative of the gradient $(-\nabla E)|_{t=1}$, which is a column

vector of the error derivatives with respect to each weight evaluated at time $t = 1$. For subsequent time steps with $t > 1$, search directions may then be set using

$$\mathbf{v}(t) = (-\nabla E)|_t + \beta(t)\mathbf{v}(t-1) \tag{2.10}$$

where $\beta(t)$ is a scalar value much like an adaptive momentum term which adds a fraction of the previous direction to the current steepest descent direction. Many rules have been developed for computing a suitable $\beta(t)$ such that the directions obtained between time $(t - N + 1)$ and t are all mutually conjugate. One of these rules is the commonly used Polak-Ribiere formula from [Pol71].

$$\beta(t) = \frac{\left[(\nabla E)|_t - (\nabla E)|_{t-1}\right] \cdot (\nabla E)|_t}{(\nabla E)|_t \cdot (\nabla E)|_t} \tag{2.11}$$

Using the search directions obtained with (2.10) and (2.11) will lead, just as when calculating the Hessian explicitly, to finding the minimum in as many steps as there are dimensions in the weight space for quadratic error functions. For non-quadratic error-weight surfaces, the minimum may not have been reached after this number of steps. In this case the procedure is re-initialised to search along the steepest descent direction from the current state by setting $\beta(t)$ to 0 for one step. In successive steps, search is performed along conjugate directions again by adjusting $\beta(t)$ according to (2.11).

CGD is certainly a big improvement for quadratic surfaces compared to using steepest gradient descent in as much as it is guaranteed to reach the minimum in a low number of steps. Even for non-quadratic surfaces CGD, would appear to have the advantage of making use of more information in its search than BP for example.

The strength of using conjugate directions in search is only apparent though when com-

bined with line search along these directions. Ironically, line search can be a cause of problems for CGD. Searching for the lowest error along a specific direction can, depending on the surface, lead to travelling large distances in weight space from which training has to return and may only do so with great difficulty. This can slow down training and in some cases may prevent CGD from converging to the desired minimum in the allowed time. From work done in the group here at St. Andrews CGD has been found to be less robust than BP in these circumstances [Wei00].

Other difficulties may arise for CGD which originate from the assumption made that the travel surface is roughly quadratic in shape. If the error surface is more complex and far from quadratic in shape, the conjugate directions will not point towards the global minimum after N steps and may not lead to a convergence within the time allowed. Baum and Lang for example reported in [BL90] that CGD never managed to converge to the global minimum which is known to exist for the 2-spirals problem for a fixed 2-50-1 network architecture (2 input units, 50 hidden units and 1 output unit). Such failure is not perhaps surprising since although CGD is a big improvement for travelling on quadratic surfaces, it does not overcome the basic limitation of gradient descent techniques and so will still find complex surfaces difficult to traverse and get stuck in local minima.

2.3 Levenberg-Marquardt

Levenberg-Marquardt is a training method which aims to provide an error-weight surface search, more informed than the techniques described so far, by combining the strengths of both first order methods, such as gradient descent, and methods which make use of second order derivative information, when it appears to be useful. As such, Levenberg-Marquardt is a hybrid method between making use of 2nd derivative information in terms of the inverse of the Hessian and simple steepest gradient descent. For a detailed presentation of the Levenberg-Marquardt technique see [Pre94].

As shown in section 2.2, the actual implementation of CGD only makes indirect use of second order derivatives. This is because the Hessian, as shown in (2.7) on page 32, does not need to be evaluated explicitly to obtain conjugate directions. A method called Newton's method on the other hand, which for this discussion can in theory be treated as part of the hybrid Levenberg-Marquardt technique, does make explicit use of the Hessian matrix. The idea is that the direction and required step size leading to the global minimum can be calculated exactly for a quadratic error surface. For a change in the weight state of $\Delta\mathbf{w}$, the change in the error gradient $\Delta\mathbf{G}$ is

$$\Delta\mathbf{G} = \mathbf{H}\Delta\mathbf{w} \tag{2.12}$$

where $\Delta\mathbf{G}$ is a vector of gradient changes for all weights, $\mathbf{H}$ is the Hessian and $\Delta\mathbf{w}$ is a vector of weight changes for all weights. At a minimum all the error-weight gradients are zero. So in order to make a transition to a minimum, the change in error gradient is required

to be equal to the negative of the current error-weight gradient, that is

$$\Delta \mathbf{G} = \mathbf{H} \Delta \mathbf{w} = -\nabla E \tag{2.13}$$

where ∇E is a vector containing the first derivatives of the error function with respect to all the weights. Re-arranging (2.13) in terms of the change in weight state $\Delta \mathbf{w}$ gives us the desired weight transition, also known as a Newton step, which has the appropriate step size and direction to the global minimum for a quadratic surface

$$\Delta \mathbf{w} = -\mathbf{H}^{-1} \nabla E \tag{2.14}$$

where $\mathbf{H}^{-1}$ is the matrix inverse of the Hessian.

The basic intuition behind Levenberg-Marquardt is to use information from the inverse Hessian to perform a Newton step when it is useful to do so, and otherwise to perform steepest gradient descent. In practice implementations of Levenberg-Marquardt mostly use an approximation of the inverse Hessian to speed up computation, but the effect on convergence is the same such that the methods for approximating the inverse will not be discussed here. If the error E is repeatedly very low, the assumption is made that the currently trained state must be close to the global minimum, in which case the error-weight surface is expected to be roughly quadratic in shape. In this case Levenberg-Marquardt performs a weight adjustment which by approximating the inverse Hessian is very similar to the above Newton step, which should lead to the global minimum of a quadratic surface. If on the other hand the error is repeatedly high, then no assumptions are made about the shape of the error-weight surface and a step is taken which is close to the steepest descent

direction given by

$$\Delta \mathbf{w} = -\eta \nabla E \tag{2.15}$$

where η is the step size, otherwise known as the learning rate.

Being a hybrid method, Levenberg-Marquardt will suffer from problems which affect each part of the hybrid individually. One possible problem for the Levenberg-Marquardt method is that on various surfaces the error may be low, while the error surface between the current state and the goal weight state is not at all quadratic in shape. This will certainly occur when situated near a local minimum with a low error. Here a Newton step does not point towards the global minimum and will cause convergence to the local minimum.

Alternatively, the shape of the surface may be such that it takes a long time for the error to decrease, such as when situated on a high plateau of almost constant error on the error-weight surface. In this case the transitions performed by Levenberg-Marquardt are mainly dominated by steepest gradient descent. It is known that shallow gradients, as found on plateaus, present problems for simple gradient descent techniques, as described in section 2.1. If the gradient is shallow then it can take many iterations to decrease the error towards the minimum, in some cases more iterations than allowed. Levenberg-Marquardt will find surfaces with plateaus difficult to traverse in the initial stages, just as simple gradient descent does.

In summary Levenberg-Marquardt offers a way to smoothly vary between an inverse Hessian approach performing quadratic descent and steepest gradient descent, but can still find it difficult to travel on complex surfaces such as surfaces with plateaus. Although

Levenberg-Marquardt is a more sophisticated approach than BP to minimising the error function, it does not overcome one of the most basic limitations of gradient descent, namely that of getting trapped in local minima. The sting of Bianchini's point in [BFGM98], made with regard to BP, is still not removed when looking at such more sophisticated training methods which attempt quadratic descent by approximating the inverse of the Hessian or alternatively some sort of gradient descent on a fixed error-weight surface. Both parts of Levenberg-Marquardt, that is quadratic descent and steepest gradient descent, are only able to offer local convergence. Like BP and CGD, Levenberg-Marquardt will still converge to the attractor of the basin its initial state is contained within and is therefore not able to overcome the local minimum problem.

2.4 Simulated Annealing

A technique possibly not used for training feedforward neural networks as commonly as it is used in other areas of optimisation and for training a particular type of neural network called a Boltzmann Machine is simulated annealing which originates from work in [KGV83]. Although it may seem strange to make mention of simulated annealing without explaining the Boltzmann Machine I wish to place simulated annealing in the framework of continuous function optimisation which is in theory applicable to training feedforward neural networks.

The idea for simulated annealing takes its inspiration from engineering and chemical physics, where the annealing (slow cooling) process in solids was observed to produce perfect crys-

tals if cooling occurred sufficiently slowly. The mathematical reasoning for this behaviour was explained by statistical mechanics in physics.

The intuition is that at high temperatures the atomic configurations are able to reach states corresponding to a large enough increase in energy to allow energy barriers between a shallow local minimum basin and a relatively deep global minimum basin to be traversed both ways. As the temperature decreases slowly the state transitions to higher energy levels become increasingly unlikely. This means that as the temperature is lowered the chances are that the system will still be able to escape the shallow local minima but be unable to escape the relatively deep global minimum basin. At low temperatures the system is finally expected to stabilise at the global minimum energy state. The amazing fact is that nature has its own way of finding the global minimum energy state by aligning the atoms in the crystal with such precision as long as enough time is given for the cooling process.

The probability p_E of a thermodynamic system in thermal equilibrium, at a temperature T being in an energy state E is given by the Boltzmann probability distribution. This is

$$p_E(E) = \frac{1}{Z} \exp(-E/k_B T) \tag{2.16}$$

where k_B is the Boltzmann constant which relates temperature to energy and Z is a normalisation factor to ensure that the probabilities of the system being in an energy state add up to 1 over all possible energy states.

In [KGV83] an algorithm was presented to implement simulated annealing in order to optimise functions. Translated into the context of minimising a neural error function E in place of the energy in the annealing process, simulated annealing makes use of a virtual

temperature T to control its processes which can be divided into two stages. The first stage is to propose a change in weight state. One way to do this is to randomly select a weight state surrounding the current weight state using a certain probability distribution. The probability distribution defines the probability of exploring a weight state. This probability may depend on the distance of the weight state from the current weight state and the temperature for instance. The second stage is to examine the error increase or decrease effected by making the transition to the suggested weight state. Depending on the change in error and the temperature the transition will either be accepted or not.

There are various annealing schedules which define when to accept a certain change in error and when not and with what rate to cool the system. In the Metropolis algorithm from 1953 as mentioned in [Pre94] the transition from one energy state to another in a simulated thermodynamic system was assumed to occur with the probability

$$p_E(\Delta E) = \begin{cases} 1 & \text{, if } \Delta E \leq 0 \\ \exp(-\Delta E / k_B T) & \text{, if } \Delta E > 0 \end{cases} \tag{2.17}$$

where ΔE represents the change in E effected by the transition. The metropolis algorithm is one example of a procedure used for performing simulated annealing.

Translated into neural terms by removing Boltzmann's constant k_B and taking E to represent the neural error one may observe the following: If the proposed weight change corresponds to no change or a decrease in error the transition to that state is always accepted with a probability of 1. For an infinite temperature the probability of accepting a state representing an increase in error is also 1. So at an infinite temperature an increase in error is just as likely to be accepted as a decrease in error. For any finite temperature the

probability of accepting increases in E is less than 1, and the greater the increase in E, the less likely it is to be accepted. As the temperature is reduced, the probability of accepting increases in E decreases until at $T = 0$ only decreases in E will be accepted. There are other procedures than the Metropolis algorithm to implement simulated annealing and although they will undoubtedly differ in detail for the various versions they all obey the same general concept.

The idea behind the claim made for simulated annealing to converge to the global minimum relies heavily on the rate at which the system is cooled. There are various ways in which to perform this virtual cooling of the system. In essence it can be said though that the probability of convergence to the global minimum tends towards 1 for cooling schedules which allow an amount of time tending towards infinity. This means that the local minimum problem still applies to simulated annealing.

2.5 Conclusion

All training techniques described in this chapter, are based on optimisation techniques from classical numerical analysis. Apart from simulated annealing all techniques depend on 1st or 2nd order derivative information or a combination of both to provide a direction which should minimise the error function. Such techniques can at best converge to the minimum in the same attractor basin as the starting state, which may not be the global minimum. Multiple random initialisation is one way to attempt finding the global minimum using such techniques. The number and range of such multiple initialisation states needed for success

is unclear a priori. If the global minimum error is a non-zero quantity then recognition of the global minimum state is an additional problem. In this case only an infinite number of random initialisations can guarantee success because this ensures that all minima will have been found.

Simulated annealing on the other hand makes use of a statistic update rule which allows large increases in error to take place at high temperatures and error decreasing transitions to take place increasingly as the temperature is decreased. Theoretically it can be shown that if an infinite time is allowed for cooling then the global minimum will be reached, for finite cooling schedules there is no guarantee.

In summary, all techniques currently employed for training neural networks suffer from the local minimum problem. That is to say that they are not guaranteed to converge to the global minimum in a feasible time, if at all when disregarding multiple random initialisation.

Chapter 3

Goal Directed Behaviour and Symbolic AI

3.1 Common Search Techniques in Symbolic AI

There are a variety of search techniques available for problem solving in the domain of symbolic Artificial Intelligence (AI). Symbolic AI refers to the part of AI which adheres to the physical symbol systems hypothesis developed by Newell and Simon in 1976.

At this point it is useful to define some terminology that will be used later. A search procedure may be said to be *fully informed* if it uses sufficient information known about the problem which is to be solved in order to ensure that a desired outcome is realised without search having to branch, i.e. explore other alternatives. Conversely a search procedure may be termed *uninformed* if it does not use any problem-specific information in its operation.

In between these two extremes of search there are various degrees of informedness, such that a procedure may be termed *semi-informed* if problem-specific information is used to attempt a desired result, but which is insufficient to prevent the outcome being uncertain. In other words semi-informed search cannot guarantee that a desired outcome is going to be reached in a feasible time.

Trying to obtain a solution to a problem in symbolic AI is commonly viewed as searching for a *goal* state on a *search tree*. Figure 3.1 shows what part of such a search tree looks like for the 8-puzzle in which the objective is to move the tiles around on the square tray from the initial state so as to obtain the depicted goal state.

Each node or state of the search tree constitutes a potential solution to the problem and the search tree contains sequences of states from the state space for that problem. Part of the procedure when searching for the goal solution is to expand child nodes at level $n+1$ of the parent node or nodes at level n currently under consideration. In Figure 3.1 the only child nodes that have been expanded in successive layers for simplicity are child nodes which lie on the optimal path from the initial state to the goal, i.e. along the path connecting states with solid lines.

If the search procedure knows exactly which child node to go to on the optimal path to the solution from any initial state and forms a single direct path to the goal then the search procedure is fully informed. The problem is that fully informed search procedures are not generally available for conducting search in symbolic AI and so other forms of search are generally used instead.

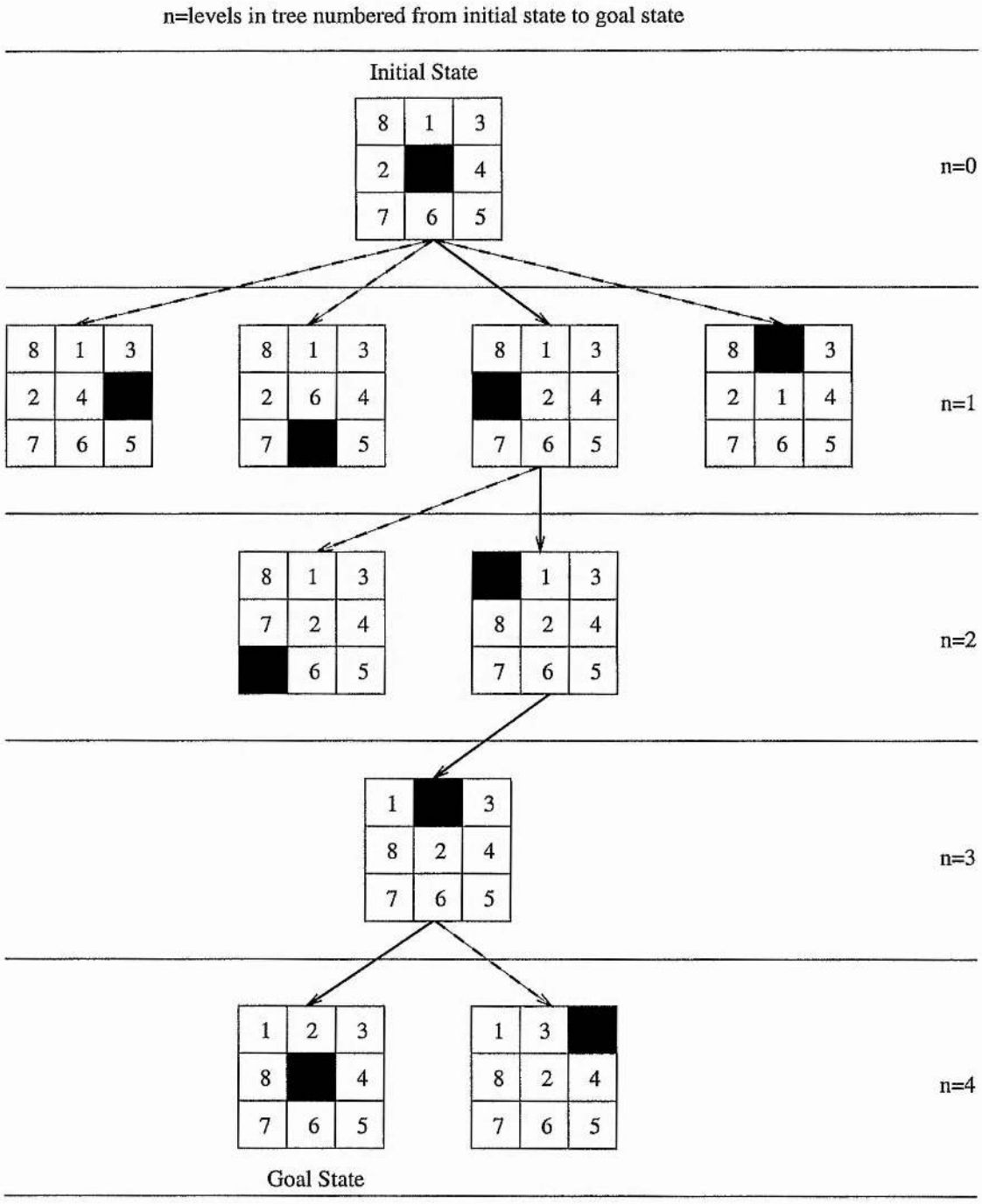


Figure 3.1: Example of a search tree for the 8 puzzle, with an initial and a goal state. The solid lines represent the shortest path to the goal, and the dashed lines lead to states which are not expanded here. Repeated states caused by taking the opposite of the last move are not depicted.

One feasible alternative to fully informed search is uninformed search. As mentioned above this is search which does not use any problem-specific information in its operation. Uninformed search has the severe drawback though of possibly searching without convergence until resources are exhausted. That is the search may run out of time by getting stuck in state transition loops if a test is not included to avoid travelling around a loop. A loop is formed when the transition from the final state in a set of states interconnected by transitions leads back to the first state in that set. It is these loops in transitions between states that effectively create paths of infinite length in the search tree which may not contain the goal solution. If search proceeds down such paths it will search without ever reaching the goal. The other resource to exhaust is space, or in other words memory. To be able to search effectively the search procedure has to keep track of at least a subset of states it has and has not previously visited in order to avoid re-visiting already visited states and to be able to explore previously unexplored branches in the search tree. For large search problems some uninformed search techniques may need more memory when run on a computer than is actually available.

One example of an uninformed search technique which can suffer from travelling along lengthy search paths which do not contain the solution is *depth-first* search. Depth-first search always tries to descend as far as possible down one branch of the search tree until either no successor states exist or the so called *depth-limit* is met, before ascending back up the tree to explore previously unexplored branches. When using depth-first search the simplest way to avoid the search proceeding down paths of infinite length is to set a limit to which depth-first search may descend, this limit is the so called depth-limit. The proce-

dure of allowing the search to ascend back up the tree to examine previously unexplored branches is called *backtracking*.

Applied to the example shown in Figure 3.1 depth-first search may expand unexplored nodes at each depth level in the order from left to right. This means that depth-first search will definitely not initially head along the optimal path from the initial state to the goal i.e. along the solid lines, although it may after having reached its depth-limit and having ascended up the tree again. One problem is how to set the depth-limit a priori. If the depth-limit is set to be too low search will never find any solution. On the other hand it may be set too high in which case search may well spend a lot of time on deep paths that may not contain a solution state. If of course no limit is set at all travelling down infinite paths created by loops in state space transitions becomes a problem.

In order to be able to backtrack depth-first search needs to keep track of child states it has visited and possibly also states it has not visited, at each level as it proceeds down the tree. For a depth limit of L the memory usage has an upper bound of the order $B\,L$ where B is the *branching factor* of the tree. The branching factor is the factor which determines how many child nodes at level $n+1$ branch out from each node at depth n.

For our simple 8 puzzle the branching factor is not constant over all states, but rather depends on the current state. Depending on where the empty square is at the current state a branching factor can be calculated. If the search algorithm avoids taking the opposite of the last move, which would effectively create an immediate loop in the search tree, then branching factors for states other than the initial state are as follows. When the empty

square is at a corner $B = 1$, when on the edge $B = 2$ and when at the centre $B = 3$. There are 4 edge states, 4 corner states and 1 centre state. A branching factor averaged over the 9 empty square states can be calculated as $(4 \times 1 + 4 \times 2 + 1 \times 3)/9 = 1\frac{2}{3}$. So if the goal were reachable in about 20 steps and the depth-limit for the search were set to 20, then in the worst case depth-first search would have to examine about $1.\overline{666}^{20} \approx 2.7 \times 10^4$ states.

Depth-first search neither guarantees that the optimum path to the goal will be found nor that the goal will be found at all. The goal may be obtainable along the optimal path at a depth of 4 from the initial state such as shown in the example in Figure 3.1. Depth-first search may have pursued a different path though, by initially expanding all the states in the order from left to right in the search space diagram at each level n, and thereby would have explored many states which are not shown in Figure 3.1. In this way the search would have initially missed the optimal path denoted by the states connected with the solid line and may have found a suboptimal path to the goal state and at a depth greater than 4.

On the other hand the optimal route to the goal for some other initial state may be of length 21 or some other depth from the initial state, in which case it would not have been found with a fixed depth limit of 20. If the depth limit for the search is removed then depth-first search has the drawback of possibly continuing along an infinite path created by loops in state space transitions as mentioned above, and which may not contain the goal. For a more detailed description of depth-first search than in the scope of this section see [LS93] or [RN95].

Another uninformed search technique is called *breadth-first* search. This technique will

examine all nodes at level n in the search tree before continuing to level $n + 1$. It is therefore guaranteed to find the shortest path to the solution if one exists, but its memory usage in order to ensure convergence grows exponentially with the length of the path to the solution. For a branching factor B the memory usage at level n of the search tree is of order B^n. Breadth-first search also has an exponentially growing time usage of order B^n because all the B^n states have been examined when the procedure reaches a depth of n .

For the 8 puzzle and for the case where the goal is at a depth of 20 from the initial state breadth-first search will have to remember approximately 2.7×10^4 states when it is at level 20. For a more detailed description of breadth-first search see [LS93] or [RN95].

A further uninformed search technique exists which requires mentioning, namely *depth-first search with iterative deepening*. The idea is to start with a depth limit L of 0 and to perform a depth-first search with this limit. As long as the Goal is not found in the search space within a depth of L, L is increased by 1 and a new depth-first search is started. This technique carries the same guarantees as breadth-first search in terms of finding the shortest path to the goal if the goal exists, because all states at a depth level of L are examined before looking at states at $L + 1$. Because depth first search is executed for each limit L this sort of search is able to reduce the memory usage to a maximum of $B\,L_g$ where L_g is the depth at which the goal is eventually found. Although states will generally be examined many times during this type of search it turns out that the cost in terms of running time is still of the same order as breadth-first search, i.e. B^{L_g}. For a more detailed description of iterative deepening search see [LS93] or [RN95].

It is trivial to see how the combinatorial explosion resulting from having to examine exponentially many states is drastic for search problems with high branching factors, such as for Chess or Go for example. Chess has an average branching factor of 35. If a normal game lasts about 50 moves per player then the number of nodes on the search tree for Chess would be $35^{100} \approx 2.5 \times 10^{154}$, which is arguably greater than the number of Hydrogen atoms in the universe, although there are only about 10^{40} different legal positions in Chess as described in [RN95]. There are more states in the tree than there are legal positions because the tree encompasses all possible games, i.e. sequences of legal board states.

This can be seen more easily for a simpler example than Chess. Take Tic-Tac-Toe, otherwise known as noughts and crosses, for example. For demonstration purposes the version of Tic-Tac-Toe has been altered slightly here. In this altered version there is a 3 by 3 board onto which players alternately place their playing pieces one at a time until the board is full. The aim of the game is to get as many of your pieces in straight lines of 3 adjacent pieces as possible and the winner is decided after the board is full. When placing k playing pieces of one type onto a board with n positions there are

$$^{n}C_{k} = \frac{n!}{k!\,(n-k)!} \tag{3.1}$$

ways to place the pieces when the ordering is not important. With 9 positions on the board a function

$$F(i,j) = {}^{9}C_{i}\ {}^{9-i}C_{j} \tag{3.2}$$

can be defined such that $F(i,j)$ denotes the number of board states that exist with i pieces belonging to player 1 and j pieces belonging to player 2 on the board, irrespective of the order in which they were placed. The intuitive reasoning for this is that for each of the

${}^{9}C_i$ board states which have i pieces on them belonging to player 1, there are $9 - i$ board positions left over which to distribute j of the 2nd player's pieces, which works out to a further ${}^{9-i}C_j$ combinations.

A final board state is one in which the 9 positions have been taken up by 5 pieces belonging to the 1st player and the remaining 4 are the 2nd player's pieces. There are $F(5,4) = 126$ final board states for this game. The total number of board states can be calculated using the following reasoning. Player 1 goes first and the players continue to place pieces alternately until the board is full. This means that player 1 can either have the same number of pieces as player 2 or one more piece than player 2 on the board. The number of total board states can be calculated as the sum

$$\sum_{c=0}^{4} F(c,c) + F(c+1,c) \tag{3.3}$$

which works out to 6046.

In comparison the number of sequences of states which can lead to the eventual 126 full board states does take the order in which the pieces were placed into account and is $(9 \times 8 \times 7 \times \ldots \times 2 \times 1) = 9! = 362880$ which is 2 orders of magnitude greater than the total number of board states and 3 orders of magnitude greater than the number of final board states.

When playing Chess it is hard to conceive of an opening move which ensures victory. It is not clear whether such a move exists or not since the state space has never been examined in full. Maybe an opening move exists which can ensure at least a draw though. In order to talk of an optimal move in Chess I will re-define optimality here for purposes of discussion.

An optimal move in Chess is one which will never cause a loss if a draw is possible and will never cause a draw if a win is possible.

In order to search the space using uninformed search to find such an optimal move in Chess, you would most probably end up examining a number of states which is larger than the number of atoms in the universe and also larger than the number of nanoseconds since the Big-Bang. Trivially this is infeasible in terms of the time it would take. Go has a branching factor approaching 360, i.e. one order of magnitude greater than the branching factor in Chess. Like Chess, Go is a game impossible to tackle with uninformed search techniques. High branching factors and long successive moves to solution make finding solutions to such games completely intractable for uninformed search and as such this emphasises the need for more informed search.

Generally the search is made more informed by supplying the search procedure with some knowledge about the problem it is trying to solve. The search is now provided with a guiding measure of which states and search directions are more promising than others in order to hopefully cut down the size of the search space it will have to examine. This guiding measure is commonly called a *heuristic*, which comes from an ancient Greek verb which means *to find.*

There are various technical meanings attached to the word heuristic and many changes have been gone through in the history of AI. In the specific area of search algorithms though, heuristic refers to a function which provides an estimate of the solution cost and is generally gained by using problem specific knowledge. The heuristic measure comprising the

knowledge can be discussed independently of the problem it is to tackle, much like a general search technique can be, but in order to function the heuristic has to be combined with a general search technique which uses the heuristic measure as a guideline for the search. The place to inject this heuristic knowledge into a search procedure is at the point where it is decided which node to expand next. This is done by designing an evaluation function to indicate how promising it is to expand a certain node in the search tree. Generally such a heuristic function h is a measure which estimates the distance from the current node to the goal node, i.e. it indicates the estimated cost of reaching the goal by its function value.

One search procedure which makes use of this heuristic measure h alone is called *gradient descent* (or alternatively *hill-climbing* if the evaluation function is viewed as a quality rather than a cost). The idea of gradient descent is to minimise the estimated distance to the goal by selecting the least costly child node of the current node for further expansion, i.e. the node associated with the smallest heuristic cost. Gradient descent has a very low memory usage, because it only ever needs to keep track of the current node and its heuristic value, but a severe problem for gradient descent is that it can easily get stuck.

Once a node has been found which is estimated to be closer than or as close to the goal as any of its children progress will stop, even if the found state is not a solution state. In other words gradient descent suffers from getting trapped in local minima and plateaus of constant estimated distance to the goal on the heuristic measure surface. If on the other hand the chosen heuristic does not have any local minima or plateaus for the problem's state space it was designed for, gradient descent will perform well and find the goal.

Figure 3.2 shows the same state space for the 8 puzzle as before in Figure 3.1, but now heuristic measures have been included. Two heuristic measures are shown in the state space diagram, namely $h1$ and $h2$. $h1$ is obtained by summing the *city block* distances of all the tiles from their goal state positions for the puzzle state being examined. The city block distance is obtained by adding the number of horizontal and vertical moves which would get the tile to its goal position, diagonal moves are not allowed. $h2$ is merely the number of tiles out of place with respect to their goal positions for the puzzle state under consideration.

Each puzzle state in Figure 3.2 has an $h1$ and $h2$ value associated with it which is depicted in the figure. Performing gradient descent for these heuristic measures would mean choosing the best state at each stage for expansion, i.e. the state with the lowest heuristic value. Using the heuristic $h1$ would lead the descent process directly from the initial state along the path indicated by the solid line to the goal, because $h1$ decreases *monotonically*, that is to say without ever increasing, along this path. On the other hand if we had chosen to use $h2$, gradient descent would get stuck at the initial state, because its heuristic value $h2 = 3$ is lower than or equal to any values of $h2$ at level $n = 1$, and so the initial state lies on a plateau of constant heuristic value. Although not depicted here, it is also easy to imagine a local minimum occurring elsewhere for both heuristics $h1$ and $h2$. Such a local minimum would occur for example when a tile which is either close to or at its desired state has to be moved away, thereby generating an increase in the heuristic measure, before another tile can be moved towards its desired state.

One way of getting round such local minima is to use multiple random sequence initialisa-

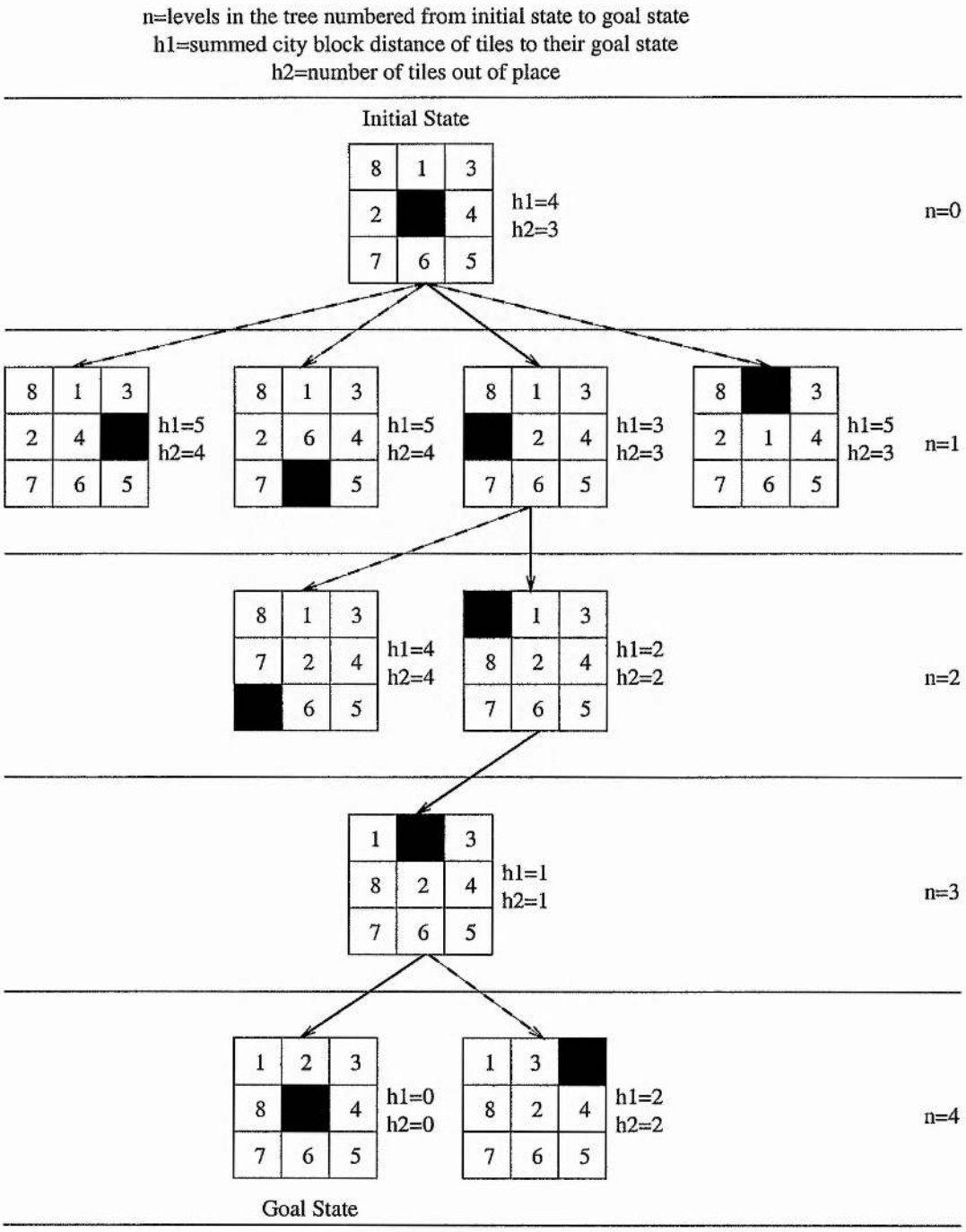


Figure 3.2: Example of a search tree for the 8 puzzle, with heuristic values indicated for some of the states between initial and goal state for two heuristics.

tions, but there is a trade-off with the extra resource cost which is not clear a priori. This is because it is the positions and relative size of local and global attractor basins which determine the probability of a random initialisation converging to the global minimum using gradient descent. The positions of the basins and the ratio of their sizes is not known a priori because this knowledge constitutes having already solved the problem. And so it is not possible to know how many random initialisations are necessary to be able to expect convergence to the desired global minimum a priori. See [RN95] and [LS93] for a more detailed description of gradient descent or hill-climbing.

In symbolic AI, alternative techniques to gradient descent, with or without multiple initialisation, keep on the move by ignoring the merit of the current state and considering the merit of the other known states already developed on other branches. If multiple branches are explored this is a non-random form of using multiple initialisations and so again extra resource usage in terms of time and memory is incurred.

One form of allowing non-random multiple initialisations is to allow searching to continue along more promising directions if the current direction turns out to be poor or a dead end. One such systematic control strategy which makes use of heuristic evaluation functions by expanding minimum cost nodes first is called *best-first search.* As a control strategy best-first search is quite similar to gradient descent, the only difference being that best-first search always expands the most promising node at each stage. It is able to do this at the expense of keeping track of a subset of states which have not been expanded, and which may contain promising child nodes in the future. In other words it has a list of states ordered according to a heuristic value function from which to choose the best node to expand next

rather than just looking straight ahead like gradient descent. Best-first search will always pick the most promising state even if it has a higher estimated cost associated with it than the currently explored node and will thereby keep moving.

Best-first search can be used to minimise the estimated cost to reach the goal by minimising the heuristic measure h similarly to how gradient descent was used. This is called *greedy search* because the strategy aims to bite off the biggest bite possible out of the remaining estimated distance to the goal, without taking into consideration whether this step will be optimal in the long run. This form of greedy search can suffer from similar problems as depth-first search does, namely a bad start can lead search to travel along non-optimal paths and possibly go round in loops (infinite paths) because greedy search doesn't take into account how far it has travelled. This means that greedy search cannot guarantee finding the shortest path to solution.

Best-first search can also be used to minimise the estimated total length of the path from the initial state to the goal. In order to do this two evaluation functions are combined by adding them to form one evaluation function $f = g + h$. The first part of the combined evaluation function f is g which is an exact measure of the distance of the node currently under consideration to the starting node in the search. The second part h is the same as before, namely the estimated distance from the current node to the goal.

When used to minimise f best-first search can be shown to be guaranteed to find the goal if one exists and furthermore the shortest connection to the goal, for the case where the employed heuristic measure fulfils certain criteria. If the heuristic measure h never overes-

timates the true distance to the goal, h is termed an *admissible* heuristic. Best-first search using an evaluation function f which has an admissible heuristic h is equivalent to a search algorithm called A*. A* is guaranteed to find the shortest path to the goal if the goal exists.

A* will perform quite differently though when the goal is not exactly realisable, i.e. there is no exact solution to the problem. In this case A* will probably continue searching for ever as it never reaches a state for which $h = 0$. A* only deals with problems which have an exact solution, i.e. $h = 0$ at the goal, it cannot deal with finding the globally optimal approximation to the goal state where $h > 0$. There are techniques in symbolic AI to deal with finding optimal approximations to the solution. These techniques either attempt to exhaustively examine the entire space which is not feasible for large state spaces, or rely on gradient descent with simulated annealing as discussed in chapter 2 and so can either take an infeasibly long time or get stuck in local minima.

An intuitive explanation of why A* is complete in terms of finding the goal if it exists and optimal in terms of finding the shortest path to the goal goes as follows. A* always examines nodes with lowest f first, i.e. lowest estimated total path length. If h never overestimates the actual distance to the goal then the search can never come up with a path which is longer than the optimal path because it would have already examined all paths associated with a total cost lower or equal to the optimal goal path. See [RN95] and [LS93] for a more detailed description of best-first search and A* search.

Figure 3.3 shows the same state space for the 8 puzzle again, but now includes the function value of 2 heuristic evaluation functions $f1$ and $f2$ for every state displayed. Both $h1$ and

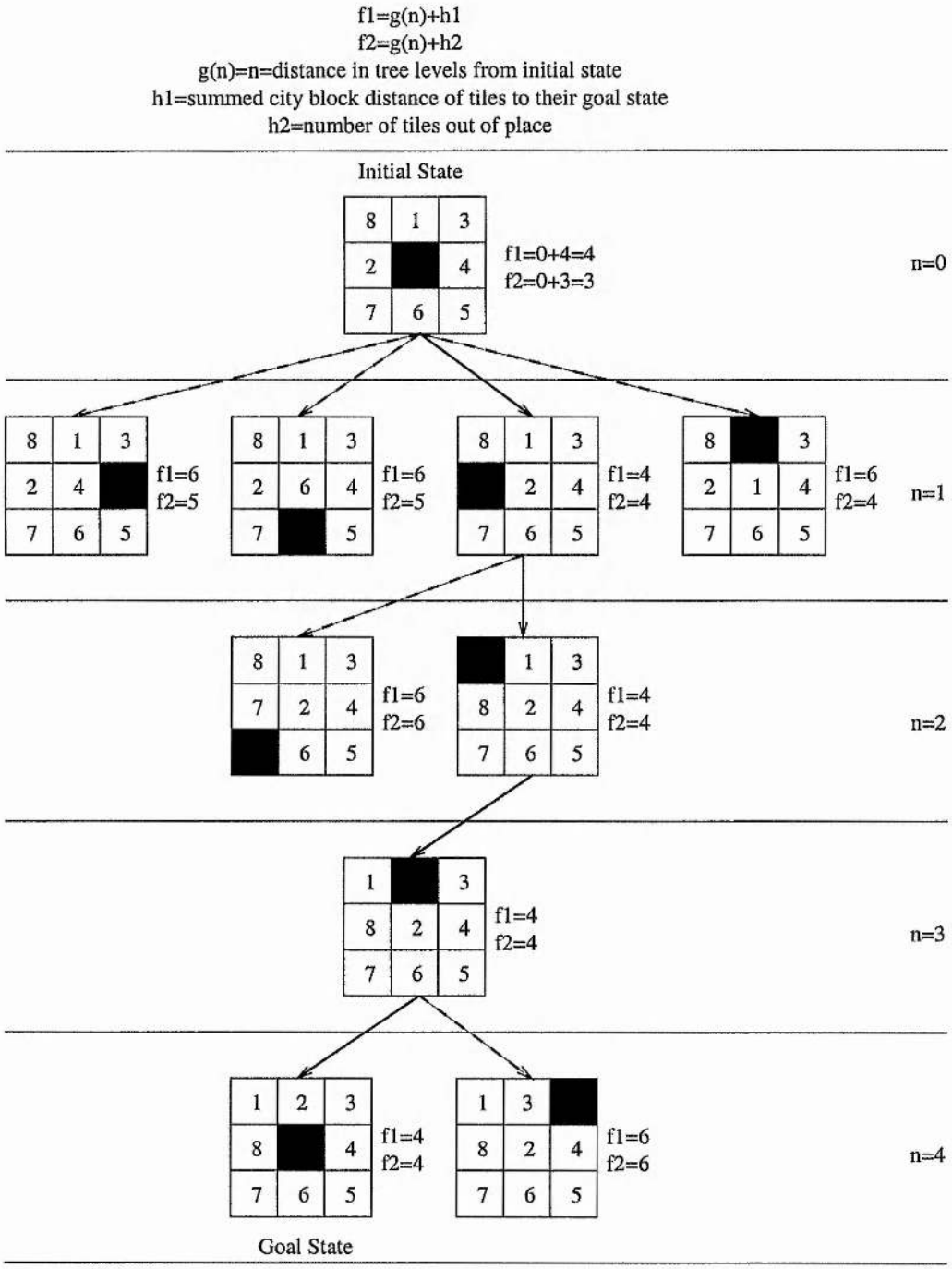


Figure 3.3: Example of a search tree for the 8 puzzle, with the value of two A* heuristic evaluation functions $f1$ and $f2$ indicated for some of the states between initial and goal state.

$h2$ which measure city block distance to goal and tiles out of place respectively can be shown to be admissible, because they can never over-estimate the actual distance to the goal. This is because if x tiles are out of place by a summed city block distance of y you will need at least y moves to solve the 8 puzzle where $y \geq x$. With both $h1$ and $h2$ being admissible, both $f1 = g(n) + h1$ and $f2 = g(n) + h2$ implemented with best-first search are A* algorithms which are guaranteed to find the shortest path to the goal for the 8 puzzle.

It can be noted that although both heuristics are admissible there are differences between the two. For any state the cost estimated by $h1$ will be greater or equal to the cost estimated by $h2$ for the same state. This means $h1$ is a more precise predictor of the actual distance to the goal and so it can be said to be more informed than $h2$. In general one might expect the more informed heuristic to examine less states than the less informed heuristic, but this may not always be true, even though it happens to seem plausible when looking at the example shown in Figure 3.3.

Using $f1$ with best-first search will result in travelling straight down the optimal path connected by the solid lines from the initial state to the goal, along which every state has a constant function value $f1 = 4$ which is lower than the function value for any other states. In contrast using $f2$ will result in possibly exploring at least one more state namely the far right state in level $n = 1$ which has an identical $f2 = 4$ value to the state in that level along the optimal path.

Although semi-informed search is mostly less costly than uninformed search, its alternative techniques to gradient descent still suffer from the same problems as uninformed search,

namely of possibly searching without convergence until resources are exhausted. Depending on the size of the search space semi-informed search, including A* search, may still turn out to be too costly. A trivial example of a costly A* search is best-first search implemented with the admissible heuristic $h = 0$ for all states. This is equivalent to the uninformed breadth-first search which as mentioned above has a memory and time usage which rises exponentially with the search depth, i.e. B^n.

Standard A* is not guaranteed to converge in finite time without running out of memory. A technique called *iterative deepening A** (IDA* see [RN95]) much like depth-first search with iterative deepening, manages to remove the memory issue, but may still fail to converge to the solution in feasible time.

Although the number of states examined may decrease as the heuristic becomes more informed the cost of evaluating the heuristic may likewise grow and cannot be disregarded. As mentioned in [RN95], unless $|h - h^*| \leq O(\log h^*)$ where h^* is the actual distance to the goal then the number of states examined for most problems will still be exponential in terms of the path length and so A* can still fail to find the goal in a feasible time.

Summary

Symbolic AI generally uses semi-informed techniques to try to make search feasible. Fully informed search is generally not available. Some of the search techniques like those based on gradient descent suffer from the Local Minimum Problem in the strict sense of literally getting stuck at suboptimal states, some do not. Multiple initialisation increases the chances

of success, but there is an unclear trade-off with the extra resource cost a priori. Semi-informed search such as A* search uses a non-random form of multiple initialisation to ensure success theoretically and aims to converge on the shortest path to the goal by the time it reaches the goal.

A* does not suffer from the local minimum problem in the sense of getting stuck at local minima, but nevertheless may not feasibly converge to the goal, i.e. the global minimum of the heuristic h, which is part of the problem as defined in section 1.2. A* is committed to following the best available transition at each stage, but since this transition may not be towards the global minimum, it may not be successful within the feasible resource allowed. This is especially true since a large number of paths may be started in pursuit of the best available transition. Furthermore, without having local attractors, A* may continue searching without converging even to a local optimum in the allowed resource. Alternative techniques such as uninformed search suffer from the local minimum problem in the same way as well.

For A* this behaviour will occur when the heuristic is not informed enough and for large search spaces, where the time needed for convergence is infeasible. It will also occur when dealing with an unrealisable goal for which the global minimum of the heuristic measure h is greater than 0.

One of the ways to deal with an unrealisable goal is to minimise the heuristic measure rather than insist it be zero. The only way to guarantee that the global minimum of h has been found is to perform an exhaustive search of the whole state space which may often

be infeasible. In practice simulated annealing can be used in combination with gradient descent for such problems. This will only guarantee success in terms of finding the global minimum if infinite time is allowed (see chapter 2). As this is infeasible as well finite times have to be used in the search which do not guarantee finding the global minimum.

In summary, search techniques in symbolic AI suffer from the local minimum problem just as training techniques for neural networks do, as both come to rely on methods taken from numerical analysis when attempting to minimise a heuristic measure.

3.2 The Use and Selection of Subgoals in Symbolic AI

In a sense, being directed by a heuristic measure based on the criterion for goal success, all semi-informed search presented in section 3.1 is goal directed. The notion of goal direction in symbolic AI can be extended though to incorporate the use of *subgoals*, which are often used in procedures which perform automated *planning*. This consists of using an automated planner to generate a sequence of moves or steps which may lead from the current state to a goal state when executed. Depending on the problem, the planner may not be able to reach the goal state with certainty, but it may be able to gauge various alternate plans with respect to the probability of them actually reaching the goal. Planning intuitively incorporates the use of subgoals because subgoals are incorporated into human planning.

When planning to make bread and butter for example, subgoals will naturally arise in the form of buying both bread and butter if necessary. Buying bread may constitute one subgoal

and buying butter another. These are two conjunctive subgoals because both subgoals need to be achieved in order to achieve overall success.

One symbolic AI technique which uses subgoals is *Divide and Conquer*. It aims to tackle a problem by decomposing it into subproblems and then re-combining the individual solutions for each subgoal to form the overall solution. The main problem for Divide and Conquer techniques is when the cost of combining the solutions for the individual subgoals is higher than solving the entire problem in one go. This occurs when actions performed to solve one subgoal affect the outcome of actions performed with another subgoal in mind. In many cases in the real world subproblems are carved out to be independent of each other. For example, buying bread and subsequently buying the butter does not mean that you lose your bread. This is not always true though and a naive planner attempting to solve a problem may break the whole problem down such that it includes an inter-dependence of actions between subproblems which is not anticipated. One way of attempting to solve the 8 puzzle is a perfect example of this.

When trying to solve the 8 puzzle as a planning problem a naive planner may attempt to solve 8 subproblems, each subproblem consisting of getting one tile in its correct position. Trying to get tile 1 in its correct position may be seen as being trivial, but trying to get tile 2 in its correct position may in all likelihood move tile 1 from its goal position again. It is therefore not very easy to combine the individual problem solutions to one solution for the whole 8 puzzle. It is this inter-dependence of actions performed to solve a conjunctive set of subgoals which causes problems for divide and conquer. Trivially, any inter-dependence of actions when trying to solve a disjunctive set of subgoals is no problem, because once

one subgoal has been achieved the whole goal has been achieved.

Divide and Conquer can be used to focus on the spatial nature of subgoals by forming small subgoal state spaces. Spatial decomposition consists of breaking the whole search space into subspaces representing subproblems which can be done in any order, possibly even in parallel. Generally this can be done if the subproblems do not have to be sequenced. That is there is no temporal inter-dependence between the subgoals. A trivial example is that buying butter and bread can be done in any order when wanting to eat bread and butter.

Temporal decomposition in planning consists of recognising subtasks which must necessarily be finished before being able to attempt other subtasks, such as going to the bank to get money before buying the bread. *Agenda* driven search carries out temporal as well as spatial decomposition, i.e. divide and conquer in space and time. That is it allows difficult tasks to be broken down into a temporal sequence of subtasks which in turn may be broken down spatially, see [RK91] for a detailed description of Agendas. Agenda driven search may be seen as a very flexible goal directed search which is quite close to goal directed behaviour as observed in the animate world including ourselves.

Similarly, as with any automated search or planning technique, agendas have costs that need to be controlled. A significant overhead may be created due to maintaining the list of the tasks to be attempted in order of preference in which they should be attempted. When a new task is added the order of the list will have to be re-computed at some point so that the list of tasks reflects the preferred order in which tasks should be attempted. The overhead generated by maintaining the list depends on how and how often the list is updated and

how many problems are on the list. The number of problems on the list depends mainly on how many subproblems the whole problem is allowed to be split into. The variable which determines this is generally set before the automated planner can proceed and is called the *grain size.*

The question is how to set the grain size to determine the number of subproblems which are allowable, a priori. If the grain-size is too large hardly any decomposition will take place and the planner may be trying to tackle a problem which is too complex for quick solution. In this case a lot of time will be spent on trying to solve big and complex subgoal chunks of the whole problem. On the other hand, if the grain size is set to be too small a lot of temporal and spatial decomposition will take place and so time spent on solving each simple and small subproblem will be small, but the sum of the small times may be larger than the sum of a smaller number of larger times. Furthermore when a large number of tasks are on the agenda a lot of time will be used in deciding which subtask to attempt next and so slower progress may be made towards solving the overall problem.

There is therefore a trade-off between *progress*, which is maximised using larger grain sizes, and *realisability*, which is maximised using smaller grain sizes. The optimal grain size is of course problem dependent.

The complexity of the subgoals in relation to the entire problem complexity should be such that maximum progress is being made towards solving the entire problem with each subgoal while each subgoal remains realisable. Imagine for example playing a game of Chess. Setting a subgoal such as setting the king to check may be highly progressive

towards winning the game but not very realisable, because it is close to winning the entire game and is therefore a difficult position to achieve. In contrast aiming to move a pawn to a position from which it cannot be taken may be a very realisable subgoal, but not very progressive towards winning the entire game. It is the combination of both progress and realisability that needs to be maximised in order to achieve optimal search.

Summary

Decomposition extends the range of problems that can be solved feasibly. This is intuitive because it is how difficult problems tend to be tackled in the real world by ourselves. If the decomposition is done in the right way for the problem at hand then finding the solutions to small subproblems and their recombination to the overall solution should be simpler than solving the entire problem in one shot.

Subgoal sequencing through agendas is one way to enable this decomposition, but there is a progress versus realisability trade-off which has to be got right for successful solution. An a priori setting of the grain size may be unsuccessful either by enforcing attempts on subgoals which are highly progressive yet unrealisable or subgoals which are highly realisable but not progressive. The local minimum problem is applicable within each subgoal as well.

3.3 Conclusions

Although some techniques in symbolic AI are sophisticated in terms of extending the range of problems to be attempted the local minimum/attractor problem still applies. Namely if a technique converges to local attractors it may get stuck at suboptimal attractors and in contrast if the technique does not converge to local attractors then it may search without even converging to locally optimal states in the allowed time. That is to say that even with A* and subgoal sequencing, problem solving search techniques in symbolic AI do not lack scope for improvement and thus are not as optimal as one naively might be lead to believe.

Symbolic AI has managed to use techniques in problem solving which do not directly originate from numerical analysis. The use of subgoals to facilitate problem solving is one example. The use of subgoals is relatively common within the realm of symbolic AI compared to within neural training. The question might be asked as to whether neural training can benefit from using subgoals.

Chapter 4

Linear Subgoal Chaining for Neural Networks

4.1 Tangent Hyperplanes

Tangent Hyperplanes is a technique which was developed here at St. Andrews as a part of Ph.D. research carried out by Antonio Fernandes [Fer97]. Its aim is to provide a locally good *direction* for the supervised neural training process which is not dependent on the shape of the error-weight surface used in common neural training techniques.

The reasons for bad directions obtained when using standard training techniques are discussed in more detail in chapter 2, but can be summarised in short as follows. The summation of error-weight surfaces over all the input/output training pairs often creates a complex error-weight surface on which steepest gradient descent and other error-weight gradient

based techniques do not point towards the minimum on the error-weight surface. For example, error-weight surfaces may arise which look like ravines. Travelling in finite step sizes on such surfaces during training can cause oscillations from side to side across the ravine, and can even cause travel to bounce out of the ravine, rather than smoothly progress towards lower error along the bottom of the valley.

One component of a technique for obtaining a better direction in weight space is Singular Value Decomposition (SVD). See [Pre94] to find a detailed description of SVD. To illustrate the technique I will start by considering a single layer neural network with N inputs and 1 output unit with a sigmoidal activation function. The neural excitation of the output unit for input pattern p can be defined as

$$\mathrm{ex}_p = \sum_{i=1}^{N} w_i \, \mathrm{in}_{ip} \tag{4.1}$$

where w_i is the weighted connection from input i to the output unit and in_{ip} is the i^{th} component of the input for pattern p. One of the weights w_i is the bias weight and one of the inputs in_{ip} is the bias unit's output value of 1. With the sigmoidal activation function g given by

$$g(\mathrm{ex}_p) = \frac{1}{1 + e^{-\mathrm{ex}_p}} \tag{4.2}$$

the neural network's output out_p for pattern p can then be obtained by applying the activation function g to the excitation such that we get

$$\mathrm{out}_p = g\left(\mathrm{ex}_p\right) \tag{4.3}$$

When training a neural network one tries to find a weight state for which the network output matches the desired output for each pattern and failing this, one attempts to minimise the

difference between desired and actual output. In other words one attempts to solve the following equation for the neural network's output out_p in terms of the weight state $\mathbf{w}$ for all patterns p simultaneously.

$$\text{out}_p(\mathbf{w}, \mathbf{in}_p) = T_p, \ \forall p \tag{4.4}$$

where $\mathbf{w}$ is the weight state vector $(w_1, w_2, \ldots, w_N)$ containing the network's N weight values including the bias weight, $\mathbf{in}_p$ is the vector $(\text{in}_{1p}, \text{in}_{2p}, \ldots, \text{in}_{Np})$ containing all N input values including the bias unit's output of 1 for pattern p and T_p is the target output value for input pattern p. The convention of including bias terms in the weight state vector and pattern input vector will be maintained throughout the thesis. With the inverse of the activation function defined as

$$g^{-1}(\text{out}_p) = -\ln\left(\frac{1}{\text{out}_p} - 1\right) = \text{ex}_p \tag{4.5}$$

which maps outputs to excitation values then (4.4) can be re-written in terms of neural excitations as

$$\text{ex}_p = \mathbf{w} \cdot \mathbf{in}_p = \sum_{i=1}^{N} w_i \, \text{in}_{ip} = T_{\text{ex}p} = g^{-1}(T_p), \ \forall p \tag{4.6}$$

where $T_{\text{ex}p}$ is now the target excitation value for pattern p. For each pattern p individually there will be a manifold of weight states producing the desired output. These weight states form what will in future be referred to as the *solution manifold* for pattern p. More specifically for this single layer network one can see that the solution manifolds for each pattern are hyperplanes of dimension $N - 1$ in N dimensional weight space. Attempting to solve the P simultaneous equations given by (4.6) is equivalent to trying to find the intersection of the P solution manifolds.

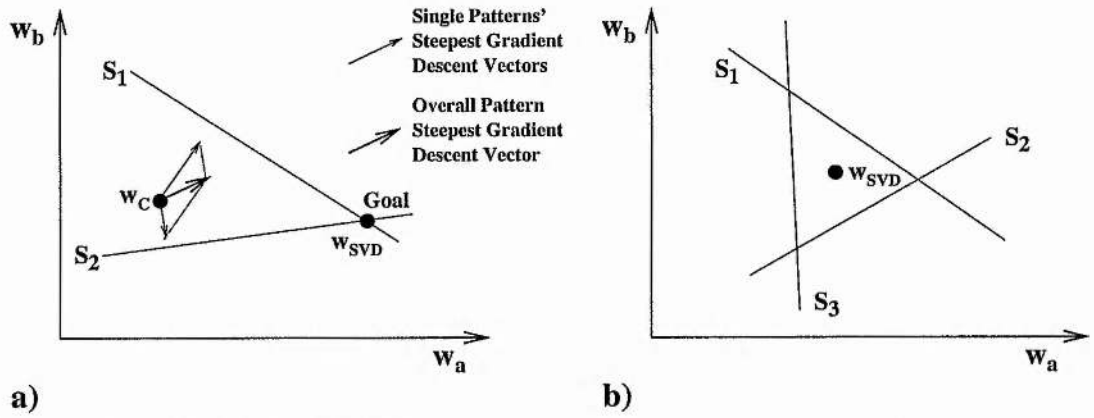


Figure 4.1: Stylised view of weight space with weight axes $\mathbf{w}_a$ and $\mathbf{w}_b$, $\mathbf{S}_p$ as the solution manifold for pattern p and $\mathbf{w}_{SVD}$ denoting the weight state found through applying SVD. **(a)** shows steepest gradient descent summed over all the patterns not pointing at the goal when starting at some current weight state $\mathbf{w}_c$. In contrast the intersection of the solution manifolds as found by SVD is at the goal. **(b)** shows the case where SVD finds the optimum weight state with respect to solution manifolds which do not intersect.

Figure 4.1**(a)** shows the case where the intersection of the two patterns' solution manifolds exists. The steepest gradient descent vectors for each individual pattern are depicted as the vectors from the current weight state $\mathbf{w}_c$ orthogonal to the patterns' solution manifolds. The steepest descent vector for a transition obtained through vector summation of the individual patterns' gradient vectors does not point towards the goal. The better direction offered by SVD is readily justifiable here because the intersection of the solution manifolds, obtained by SVD, actually is the goal.

Figure 4.1**(b)** shows the case where there is no intersection of the solution manifolds. In this case SVD finds the weight state which is *closest* to all the solution manifolds. Close is a relative term here because SVD measures closeness in weight space dependent on the magnitude of the vector defining the solution hyperplane. This defining vector is one

orthogonal to the hyperplane. Normalisation of this defining vector is an issue which has to be addressed when employing SVD, because different lengths of defining vectors will result in a different optimal closeness to all the solution manifold hyperplanes. The various types of normalisation and their effects on the resulting optimal closeness are described in detail in [Fer97] and [WF94].

The strength of this technique is quite apparent for single layer networks because it offers a way to train to the optimum solution in one single iteration, i.e. one single application of SVD. The capability of obtaining a *one shot* solution applies to all N-M single layer networks regardless of the type of activation function and to multi-layer networks where the hidden units have linear activation functions. It is well known though that single layer networks and networks with linear activation functions are of limited use. The above described basic technique is extendable however for use with multi-layer networks with non-linear activation functions.

In the latter case the solution manifolds are no longer hyperplanes, they are non-linear hyper-surfaces. The training process now becomes an iterative one by trying to approximate the non-linear manifolds by hyperplanes tangent to them and applying SVD to the *tangent hyperplanes*, hence the name of the technique. In order to make the approximation to the non-linear manifolds as good as possible one can exploit a characteristic of the manifolds. The manifolds for the output state corresponding to the current weight state all pass through the current weight state and are locally linear, as indicated in Figure 4.2(**a**). Consequently, if the manifolds are at some stage close enough to the current weight state, they may be sufficiently linear locally for the hyperplanes tangent to them to give a good approximation.

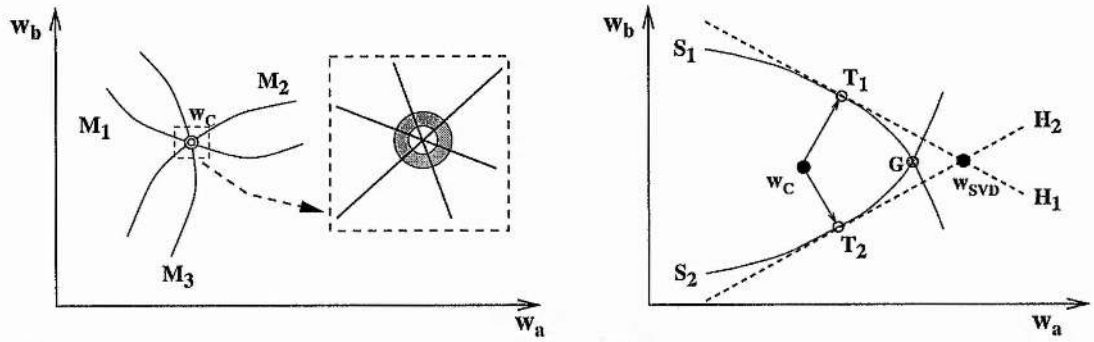


Figure 4.2: Stylised view of weight space with weight axes $\mathbf{w}_a$ and $\mathbf{w}_b$, and current weight state $\mathbf{w}_c$. **(a)** shows how the curved manifolds may be treated as being locally linear in the vicinity of the current weight state. The output manifolds $\mathbf{M}_p$ for all patterns p corresponding to the output state at the current weight state $\mathbf{w}_c$ all pass through $\mathbf{w}_c$ and are locally linear. In **(b)** $\mathbf{S}_p$ denote the solution manifolds for the goal and pattern p, $\mathbf{H}_p$ are the hyperplanes tangent to the solution manifolds at the tangency points $\mathbf{T}_p$ for pattern p. $\mathbf{w}_{\mathrm{SVD}}$ denotes the weight state found through applying SVD.

When looking for a point on the solution manifold at which to place the tangent hyperplane it makes sense to find a point which is as close as possible to the current weight state. This is because a hyperplane close to the current weight state is most likely to give a good approximation to the manifold. Figure 4.2**(b)** shows such *tangency points* $\mathbf{T}_1$ and $\mathbf{T}_2$ on the solution manifolds $\mathbf{S}_1$ and $\mathbf{S}_2$ for patterns 1 and 2 respectively. These tangency points can easily be found through line minimisation along the steepest descent direction for each pattern p individually, from the current weight state $\mathbf{w}_c$ to the patterns' solution manifolds $\mathbf{S}_p$.

Once the tangent hyperplanes have been established SVD can be applied to them in order to find the weight state which is closest to all the manifolds as in the single layer network case. But in the multi-layer network case the found weight state, denoted by $\mathbf{w}_{\mathrm{SVD}}$

in Figure 4.2(**b**), constitutes a test weight state that may need to be revised. Only when the approximation provided by the tangent hyperplanes is good enough will $\mathbf{w}_{SVD}$ be significantly near the weight state which is closest to all the solution manifolds, denoted by **G**.

The Tangent Hyperplanes technique starts out with a random weight initialisation. It is therefore quite common that the initial tangency points are far from the initial weight state and that their tangent hyperplanes do not provide a near optimal approximation to the solution manifolds. In this case it is possible to set intermediate *subgoal* target states according to a linear bisection regime such that the subgoals become increasingly close to the network output for the current weight state (current output state). Each such subgoal will have an associated set of solution manifolds which in future I will refer to as *subgoal manifolds*. These subgoal manifolds are manifolds of weight states which produce the desired subgoal outputs for each pattern individually. As the subgoals near the current output state, the subgoal solution manifolds become increasingly close to the current weight state. Because of local linearity the tangent hyperplanes are more likely to produce a good approximation to such subgoal manifolds.

Figure 4.3 is intended to demonstrate how setting subgoals can improve the approximation obtained from the tangent hyperplanes. The test weight state **A** is the approximation to **G** as obtained from applying SVD to the tangent hyperplanes $\mathbf{H}_1$ and $\mathbf{H}_2$. Subgoal target states may be set which correspond to the output at bisected tangency points $\mathbf{T}_p'$. In other words, each pattern's subgoal target value is set to the network output at the bisected tangency point for the respective pattern. The test weight state $\mathbf{A}_j$ is the approximation to the j^{th}

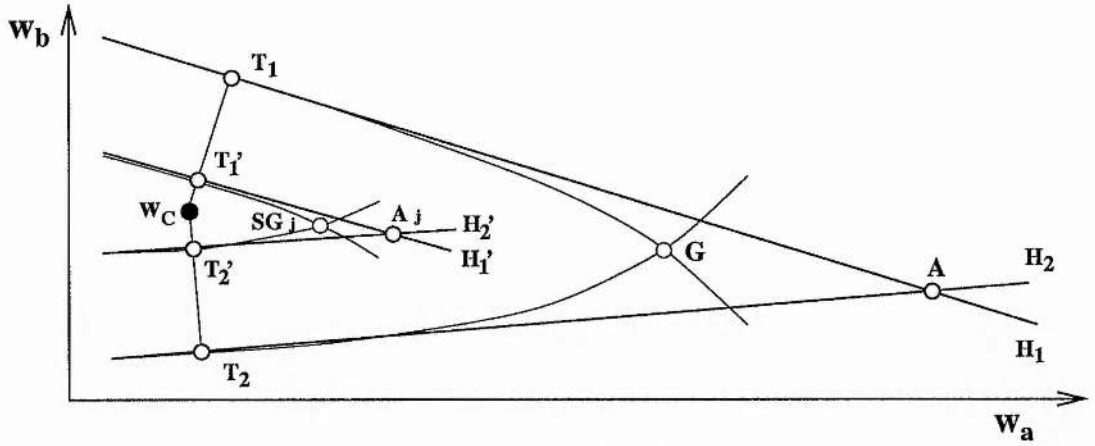


Figure 4.3: Stylised view of weight space with weight axes $\mathbf{w}_a$ and $\mathbf{w}_b$, and current weight state $\mathbf{w}_c$. $\mathbf{T}_p$ are the tangency points on the solution manifolds for pattern p, $\mathbf{H}_p$ are the tangent hyperplanes to the solution manifolds for pattern p, $\mathbf{G}$ denotes the goal weight state and $\mathbf{A}$ the approximation to the goal through applying SVD on the tangent hyperplanes. $\mathbf{T}_p{}'$ denote the bisected tangency points for patterns p, $\mathbf{H}_p{}'$ are the tangent hyperplanes through the bisected tangency points and parallel to $\mathbf{H}_p$. $\mathbf{SG}_j$ denotes a subgoal solution weight state for subgoal targets derived from the output state at the bisected tangency points $\mathbf{T}_p{}'$. $\mathbf{A}_j$ denotes the approximation to $\mathbf{SG}_j$ found through the intersection of the tangent hyperplanes $\mathbf{H}_p{}'$.

subgoal $\mathbf{SG}_j$ as obtained from applying SVD to the hyperplanes $\mathbf{H}_1$ and $\mathbf{H}_2$.

In order to avoid applying SVD for each subgoal it is possible to exploit another characteristic of the manifolds. It turns out that manifolds belonging to successive subgoals are highly parallel such that $\mathbf{H}_p{}'$ in Figure 4.3 are parallel to $\mathbf{H}_p$ for all patterns p. This means that $\mathbf{A}_j$ and $\mathbf{A}$, as the intersection of all $\mathbf{H}_p{}'$ and $\mathbf{H}_p$ respectively, are co-linear with the current weight state $\mathbf{w}_c$. Because of this co-linearity multiple applications of SVD to find the intersections of $\mathbf{H}_p{}'$ after having found the intersection of $\mathbf{H}_p$ are not necessary. After applying SVD to the hyperplanes tangent to the goal's solution manifolds one obtains the test weight state $\mathbf{A}$. In order to obtain the test weight states $\mathbf{A}_j$ for subsequent subgoals

it is merely necessary to bisect the vector from $\mathbf{w}_c$ to $\mathbf{A}$ by the same amount as the tangency point distances from $\mathbf{w}_c$ were bisected in order to get the $\mathbf{T}_p{}'$. The above mentioned co-linearity of successive test weight states $\mathbf{A}_j$ also holds for the inexact solution case, where there is no intersection of the solution manifolds and so there is no need for multiple applications of SVD for such cases either.

As the subgoals near the current output state and the subgoal solution weight state nears the current weight state, a subgoal will be found for which the approximation obtained from the tangent hyperplanes, in form of a test weight state and corresponding test output state, is good enough with respect to the subgoal output state. The test for the goodness of the approximation delivered by the tangent hyperplanes is based on how closely the test output state matches the subgoal output state. Eventually a subgoal and its approximation in form of the test output state will be accepted during the bisection process and a transition will be made to the approximation of the current subgoal.

Figure 4.4 shows how, while bisection is taking place, a subgoal is either accepted as being a *good approximation* to the current subgoal or accepted as being *too close* to the current output state. If the Euclidean distance of the test output state $\mathbf{O}(\mathbf{A}_j)$ to the subgoal state $\mathbf{SG}_j$ is within a small fraction k_1 of the distance d_1 from the current state $\mathbf{O}_c$ to the subgoal state $\mathbf{SG}_j$, then the test output state is seen as being a good approximation to the subgoal and the transition is made to the test output state. If on the other hand the Euclidean distance of test output state $\mathbf{O}(\mathbf{A}_j)$ to the current state $\mathbf{O}_c$ is within a small fraction k_2 of the distance d_2 from the current state $\mathbf{O}_c$ to the goal output state $\mathbf{G}$ then it is supposed that no significant further progress will be made towards the subgoal. In the latter case the transition is made

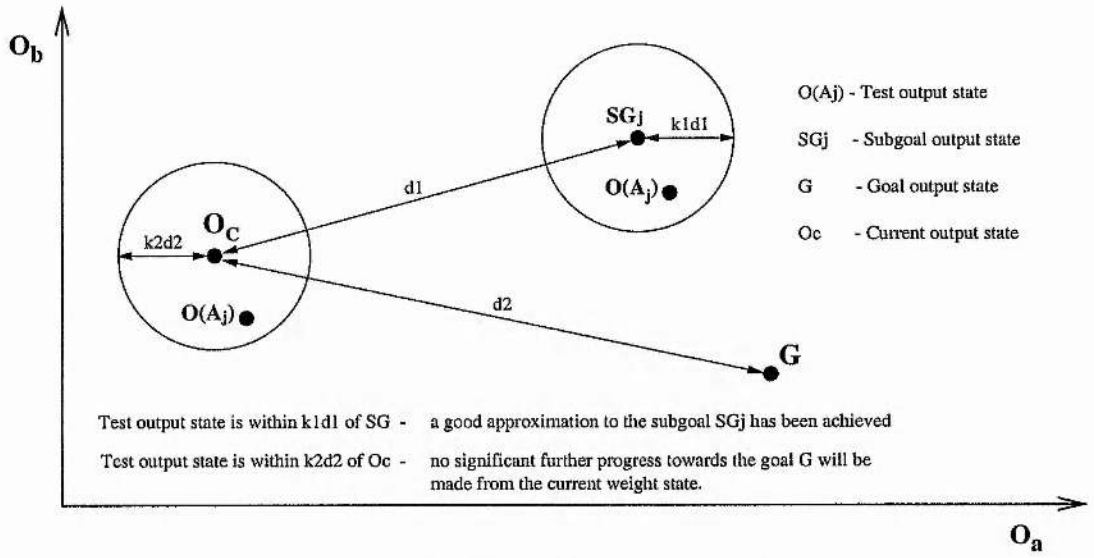


Figure 4.4: Description of the subgoal acceptance test. Stylised view of Output space with output axes $\mathbf{O}_a$ and $\mathbf{O}_b$, the current output state $\mathbf{O}_c$ and the goal output state $\mathbf{G}$. $\mathbf{O}(\mathbf{A}_j)$ denotes the test output state approximation to the subgoal output state $\mathbf{SG}_j$, corresponding to the test weight state $\mathbf{A}_j$ in Figure 4.3.

to the close-by state in order to re-initialise the process close to but not at the current state and thereby avoid getting stuck at the current state.

The algorithm for making one iteration of tangent hyperplanes may be written as follows:

```
(1)  For each goal output training pattern
       find the tangency point using line search along the
       steepest descent direction.
(2)  Compute the SVD weight state (A) for the goal's tangent
     hyperplanes.
(3)  Set (A) to be the 1st test weight state
(4)  repeat {
       if (output from test weight state is a good approximation
           to the subgoal state or too close to the current
           output state)
       then {
         flag ''DONE''
       }
       else {
         set a closer subgoal using linear bisection of the
         vectors to the tangency points and the test weight
         state from the current weight state.
       }
     } until ''DONE''
```

Although tangent hyperplanes is now an iterative procedure for multi-layer networks and

SVD has to be applied more than once in general, there is a theoretical argument and empirical evidence that the search remains strongly focussed over the sequence of iterations. In theory the focus is because the sequence of transitions will most probably contain output states close to each other for which the solution manifolds are locally parallel and locally linear with respect to the previously realised state in the sequence. This means that the tangent hyperplanes will provide a good approximation to the subgoal manifolds at each stage and hence direct training smoothly towards the goal state. Empirically this is backed up by measuring angle differences for the travel direction between successive iterations and finding these to be very small [Fer97].

The subgoal acceptance criteria described above were developed in the neural group between 1998 and 2000 in order to clarify the reasons for TH's success and possibly improve TH. The modified version of TH has been tested on a variety of benchmark problems. The results show that the technique offers high success rates with a low number of iterations and shows robustness to learning parameter variation [WLM00]. The fact that successive iterations have small angle differences and few iterations are needed to solve many problems shows that indeed the direction provided here is better than for gradient descent based training regimes in which oscillatory behaviour and a large number of iterations are both not uncommon. Its ability to solve the 2-spirals problem from [LW88], with a 2-50-1 fixed topology, with 100% success first reported in [WF94], something unmatched by gradient descent based algorithms to date, shows the strength of the direction provided by the technique for organising hyperplanes to fit the data. By using subgoals, the technique is able to cater for curved solution manifolds inherent to multi-layer networks. Subgoals allow

the tangent hyperplanes to be used as good linear approximations to the curved manifolds locally, and to achieve smooth iteration sequences to success.

Although tangent hyperplanes is a big improvement on standard gradient descent based techniques in terms of providing better direction towards the goal, it has been shown to fail to reach the best approximation to a problem solution, i.e. the global minimum in LMS error, in a feasible time for some problems. These problems are those which have a goal target state which is not producible by any weight state, i.e. the global minimum has non-zero LMS error. The reason for TH failing would appear to lie with the subgoal acceptance criteria described above.

Nonetheless tangent hyperplanes as it stands remains a powerful technique for training neural networks by using subgoals but needs further work to allow it to find solutions for more general types of training problem.

4.2 Extending TH's Subgoal Acceptance Criteria

As mentioned in section 4.1 tangent hyperplanes (TH) encounters difficulties in training when dealing with problems which have a non-zero global minimum. The reason for this may be shown to lie with the employed subgoal acceptance criteria.

TH uses SVD to suggest a weight state $\mathbf{A}_j$ with a corresponding output state $\mathbf{O}(\mathbf{A}_j)$ for the subgoal output state $\mathbf{SG}_j$ currently under consideration. k_1 is the parameter in TH's subgoal acceptance test which designates the fractional radius from $\mathbf{SG}_j$ within which $\mathbf{O}(\mathbf{A}_j)$ must

lie to be seen as making acceptable progress as shown in Figure 4.4. The radius belongs to a hypersphere in output space and surrounding the subgoal state. Generally k_1 is set to a value between 0 and 1 and for the experimental results described in [WLM00] for example a value of 0.2 was used for all training problems. A value of 0.2 for k_1 for example ensures that only SVD transitions are accepted which make significant progress towards the subgoal, i.e. lie within a radius surrounding the subgoal equal to 20% of the distance from the current output state to the subgoal. For all the training examples investigated in [WLM00] the global minimum found had near zero error. This means that the setting of $k_1 = 0.2$ made for successful penetration to solution for all problems by keeping the search closely focussed on each subgoal.

In contrast, for training problems for which the goal output state cannot be realised by any weight state, i.e. the global minimum has a non-zero valued error associated with it, achievable progress towards the goal is limited. Without knowing how close the global minimum output state is to the goal output state one cannot set a priori a value for k_1 which is either universal or suited to the problem at hand. If k_1 is set at a value too small for the problem at hand one is requiring that more progress be made towards the goal and subgoals than is actually possible and so no SVD suggestion will ever be deemed acceptable. In this case TH will only perform small jumps when *too close* has been triggered. These jumps may be small enough that significant progress towards the goal is not made in the allowed time.

If on the other hand the amount of progress to be made is relaxed, i.e. if k_1 is set to a higher value nearing 1, transitions will be accepted where the new output state may lie

near the surface of a relatively large hypersphere surrounding $\mathbf{SG}_j$. The transitions may consequently bounce around the hypersphere in an unfocussed manner with only slow convergence towards $\mathbf{SG}_j$.

In addition to being able to produce poor travel when accepting or rejecting the SVD suggestion for subgoals, the acceptance criterion just described can also fail to accept even the global optimum output state for the goal if provided it by SVD. Specifically, at a global minimum associated with a non-zero valued error E_{GM} there is a corresponding least distance to the goal in output space d_{GM} achievable by the network. If $k_1\, d_1 < d_{\text{GM}}$ then the global minimum output state will be rejected by TH as not being close enough to the goal in output space. The ideal value at which to set k_1 is also problem dependent for this case and as such cannot be known without knowing the non-zero valued error at the global minimum.

The subgoal acceptance criteria used for TH and described in section 4.1 need revising in order to allow TH to train successfully on problems which have a non-zero valued error at the global minimum. This revision of the subgoal acceptance criteria is work conducted in the neural group at St Andrews between 2000-2001.

One strategy is to adopt a subgoal acceptance criterion which can recognise an optimum for both the goal and subgoal targets regardless of whether the minimum has zero or non-zero error. An error-weight gradient based approach is one way to do this.

The travel direction obtained via error-weight gradient descent as used in BP has been shown to be poor for complex error-weight surfaces as reported for one in [WLM00], but

the gradients can still be exploited for recognising local extrema on the error-weight surface. One potential drawback is that in theory minima and maxima may be confused by using error-weight gradients as an indication of optimality. In order to avoid this confusion the change in error relative to the current weight state may be used in combination with the error-weight gradients as an indication of optimality. Relative to the current weight state, a next weight state should decrease in error and be local enough to be part of the same error-trend.

In essence a small alteration to the TH procedure as described in section 4.1 can be made to use error-weight gradients and error-values where they are useful, i.e. for recognising travel towards locally optimal states for solution manifolds to near by subgoals.

The error weight gradients can be calculated with respect to the subgoal targets at the weight state suggested by SVD for each weight. The error-weight gradient values can be written as

$$g_i = \frac{\partial E}{\partial w_i} = \sum_{p=1}^{P} \frac{\partial e_p}{\partial w_i} \tag{4.7}$$

where E denotes standard LMS error summed over all training patterns as described in (2.3) in section 2.1, e_p is the pattern error for training pattern p as described in (2.1), P is the number of training patterns and i ranges from 1 to the number of weights N in the network. The gradient values g_i for each weight i may be grouped into a vector $\mathbf{G}$ which is equivalent to ∇E and which points in the direction in weight space in which E increases most steeply.

The weight state $\mathbf{A}_j$ suggested by SVD for a subgoal $\mathbf{SG}_j$ during the bisection process can

now be accepted on the following basis. If the magnitude of the error-weight gradients is small, i.e. the error weight surface for the subgoal is nearly flat, and if a decrease in subgoal error is registered at $\mathbf{A}_j$ with respect to the error at the current state then the proposed state may be close to an optimum.

$$\|\mathbf{G}\| < t \tag{4.8}$$

describes one part of the subgoal acceptance criterion where t is the threshold of gradient magnitude below which the subgoal's error weight surface is assumed to be flat enough for optimality to occur.

Another part of the subgoal acceptance criterion to check for a decrease in subgoal error at $\mathbf{A}_j$ with respect to the subgoal error at the current weight state. In order to do this a fractional error measure F is defined as

$$F = \frac{\|\mathbf{O}(\mathbf{A}_j) - \mathbf{SG}_j\|}{\|\mathbf{O}_c - \mathbf{SG}_j\|} \tag{4.9}$$

where $\mathbf{O}(\mathbf{A}_j)$ is the output state at the weight state suggested by SVD, $\mathbf{O}_c$ is the output state at the current weight state and $\mathbf{SG}_j$ is the subgoal output state. Checking for an error decrease is done by checking whether the fractional error value F is smaller than 1.

A further alteration was implemented with this new version of TH which aimed to increase the accuracy of the transitions obtained via the tangent hyperplanes. This was to exploit local linearity of the solution manifolds additionally to as described in section 4.1. By allowing bisection to take place before applying SVD, i.e. making the manifolds on which TH forms the tangent hyperplanes and applies SVD closer to the current state, one can expect a better approximation via SVD due to local linearity. If a low gradient magnitude

is registered after applying SVD, indicating the test weight state is near a minimum, it is possible to double the suggested weight transition and tangency vectors, i.e. try for a subgoal closer to the goal. Doubling may continue while near optimality continues to be flagged up and as long as the goal manifolds have not been reached. The doubling procedure is in order to allow as much progress towards the goal as is possible.

If on the other hand the approximation is not good then bisection of the suggested weight transition and tangency vectors can take place until a good approximation is found or until the suggested output state corresponding to the suggested weight state is deemed *too close* to the current output state. The test for the suggested output state being too close to the current output state is identical to before in section 4.1. Its function, as before, is to re-initialise the system at a state close to the current state if no progress is going to be made towards the current subgoal.

The algorithm for this altered version of TH can be described as follows.

REPEAT:

0) provide tangency vectors for goal.

1) Bisect back until we want to apply SVD.

2) Apply SVD to the solution manifolds found at that stage.

3) Calculate the magnitude of the current error-weight gradient
over all patterns and weights, i.e. $\|\mathbf{G}\|$ and
the fractional error value F.

4) If ($\|\mathbf{G}\| < t$ AND $F < 1$) {
double out while ($\|\mathbf{G}\| < t$ AND $F < 1$)
}
else if (NOT *too close*) {
bisect back until (($\|\mathbf{G}\| < t$ AND $F < 1$)
OR *too close*)
}

UNTIL (*time out*
OR ''little change'' is registered over a number of iterations)

The termination criterion for a problem which has non-zero error at the global minimum, the value of which is not known a priori, has to be different than the termination used for the version of TH described in section 4.1. Previously a sequence of TH cycles could be terminated when the outputs were within a certain analogue tolerance of the goal targets.

Now, the correct output tolerance to use in order to recognise successful training is not known because the optimal output state with respect to the goal target is not known. Instead, averaging the weight transition magnitudes over a number of cycles has been adopted as a measure of how much progress is being made. If this measure drops below a certain value then it is assumed that no further significant progress will be made towards the goal and the run can be terminated.

The implementation of this extended version of TH was shown to be successful. In initial tests the above method was found to work well for XOR, for which a zero-valued error is achievable, and was also found to work well for problems that are known to have global minima with non-zero error. In essence the technique was able to match the old version of TH (as reported in [WLM00]) in terms of performance for problems with a zero-valued error at the global minimum state and in addition was able to tackle problems which do not have a global minimum with zero error.

4.3 Linear Subgoal Chaining Issues

Among research carried out in the group here at St Andrews is a technique which was investigated by Antonio Fernandes and Mike Weir [Fer97]. The technique makes use of subgoals within supervised training which means that the targets used for directing training may at intermediate training stages be set to other values than the original goal targets. Here, goal targets, or the goal for short, refers to the neural target output values which define the original training problem. In future I will refer to this technique as a linear

subgoal chaining.

In essence, linear subgoal chaining allows subgoals to be set along a linear chain in output space. The initial subgoal is generally chosen to correspond to the neural output state at the start of training, and the subgoals are then set in regular steps towards the goal of the original training set. The neural network can be trained on each subgoal in succession, using any of the aforementioned training techniques. In the simplest implementation developed here in the group standard batched BP was used, see section 2.1 for a description of BP.

As the reader may recall from (2.2) in section 2.1, standard LMS error for P training patterns was defined as

$$E_{\text{LMS}} = \frac{1}{P} \sum_{p=1}^{p=P} (\text{out}_p - T_p)^2 \tag{4.10}$$

In other words the error is a function of the neural output out_p and the desired target output value T_p for each pattern p. When initialising the neural network with random weight values and setting the initial subgoal to correspond to the current neural output state, the LMS error measure is trivially zero, and so the initial weight state is immediately placed in a global minimum for this initial subgoal.

It is known that the directions obtained via gradient descent do not generally point at the minimum, especially if the minimum is far away. It was hoped that linear subgoal chaining would improve the ability of gradient descent techniques to find a solution to a training problem and the time taken to find the solution. The reason for the anticipated improvement naively was that it would appear easier to get a neural network to realise an intermediate subgoal which is closer to the current neural output state, than the goal consisting of the

original training targets, which is further away from the initial neural output state.

It was found though, that the directional ability of BP was so weak such that training times to reach the 1st subgoal in the chain were generally longer than the times taken to reach the goal when training on the goal target from the initial state, without the use of intermediate subgoals. The reasoning for this is that the local direction obtained from BP is as weak as its global direction.

In order to address the weak direction obtained using techniques such as BP, the tangent hyperplanes (TH) technique was developed, as described in section 4.1, for which the local direction is indeed better than the global direction. TH thereby successfully made use of linear subgoal chaining to direct the local training process at each stage more precisely.

Linear subgoal chaining was never intended to address the local minimum problem. The local minimum problem is a problem which affects all gradient descent based supervised learning techniques described so far. As described before in section 2.1, a local minimum is a region or point on the error-weight surface surrounded by higher error in all directions locally, yet which is not the lowest error on the whole surface. Once the weight state, at any time during training, is within the local minimum's basin of attraction, pure gradient descent without the use of momentum will direct training to the local minimum. At this state $\nabla E = \mathbf{0}$, where ∇E is a vector containing first order derivatives of the error function with respect to all weights and $\mathbf{0}$ is a vector whose elements are all zero. The problem when using some form of gradient descent, is how to avoid converging to a state satisfying the minimisation condition of $\nabla E = \mathbf{0}$ which is not the global minimum.

When situated at a local minimum for the goal, it is by definition impossible to find states with lower errors in the local vicinity. The only way out of the local minimum is by means of temporarily increasing the error, that is, if tunnelling through the error-weight surface is not allowed. In other words, when situated at a local minimum for the goal it is by definition impossible to immediately improve the output towards the goal. This awareness makes any attempt to overcome local minima with the use of linear subgoal chaining futile. If at any time during the training process the trained state is at a local minimum for the goal, then progress towards any subgoals further along the chain including the goal is not possible. That is, subgoals in the vicinity of a local minimum for the goal which are closer towards the goal than the local minimum are not *realisable*. Any technique which points travel towards such unrealisable states will get training stuck at the local minimum.

Expanded range approximation (ERA) is a technique presented in 1997, which purported to deal with the local minimum problem for supervised feedforward neural networks. The authors of the ERA paper [GST97] claim to overcome local minima by varying the error-weight travel surface during training by setting subgoals. When reading the ERA paper it became apparent that the authors were not aware of the restrictions imposed on travel and the surface variation by using linear subgoal chains. It was believed in the group that the authors of the ERA paper were therefore falsely claiming to have solved the local minimum problem, albeit unbeknown to them. This made it important to rebut their work so that the need for non-linear chains could be seen.

4.4 Expanded Range Approximation

Expanded Range Approximation is a technique which attempts to deal with the local minimum problem for supervised feedforward neural networks by using subgoals to vary the error-weight travel surface. The ERA method consists of defining a modified training set by compressing the P goal target values T_p down to their mean-value $\langle T \rangle$ for each output unit

$$\langle T \rangle = \frac{1}{P} \sum_{p=1}^{P} T_p \tag{4.11}$$

and then progressively expanding these compressed targets linearly back toward their original values. That is, a modified training set for the inputs $\mathbf{x}_p$ is defined as

$$S(\lambda) = \{\mathbf{x}_p, T_p(\lambda)\} = \{\mathbf{x}_p, \langle T \rangle + \lambda(T_p - \langle T \rangle)\} \tag{4.12}$$

where λ is increased in regular discrete steps from 0 to 1. In terminology commonly used in the group at St Andrews, the value of λ defines a particular subgoal setting. The increases in λ generate a linear subgoal chain from the mean-valued targets to the final goal targets for the original training set. Similarly to the linear subgoal chaining technique described in section 4.3 ERA uses standard gradient descent based training to get the network to attempt realisation of the subgoal outputs.

The creator's of the ERA technique argue for 100% success in finding the global minimum for a wide variety of problems. Their claim would appear to be based on the following 3 step argument and obtaining 100% success when training a 2-2-1 network on the XOR problem.

Firstly it is believed that the mean-valued subgoal state, defined by $\lambda = 0$, only has global minima which can always be found using gradient descent.

The second stage is to make a small enough step along the linear chain. A small enough perturbation of the subgoal target state produces an error-weight surface with a slightly shifted global minimum. It is argued that this slightly perturbed error-weight surface contains no minima other than the slightly shifted global minimum from the stage $\lambda = 0$. Consequently training will converge to the one and only minimum which is the shifted global minimum.

The claim is then made that the system can be re-expanded to the original training problem, by successively setting subgoals closer to the goal target state, defined by $\lambda = 1$, without displacing the trained state from the global minimum for the subgoal at any stage. The reason given is that each successive increase in λ will create an error weight surface which has a minimum at a slightly shifted position compared to a minimum position for the previous λ setting. For small increases in λ it is furthermore argued that the basin of attraction influencing the weight state transition contains the previous trained weight state and so training will converge to the shifted minimum. This gives rise to the notion that local minima are avoided, so that travel to the goal's global minimum is always successful.

It is this third stage in their argument which provided the most concern when reading the paper. The main design feature in ERA is that the travel surface is changed with every subgoal. This surface change however may not always fit with the designers' intentions.

From previous experimentation with linear subgoal chaining, as mentioned in section 4.3, it had become apparent that linear chains were not able to overcome local minima due to

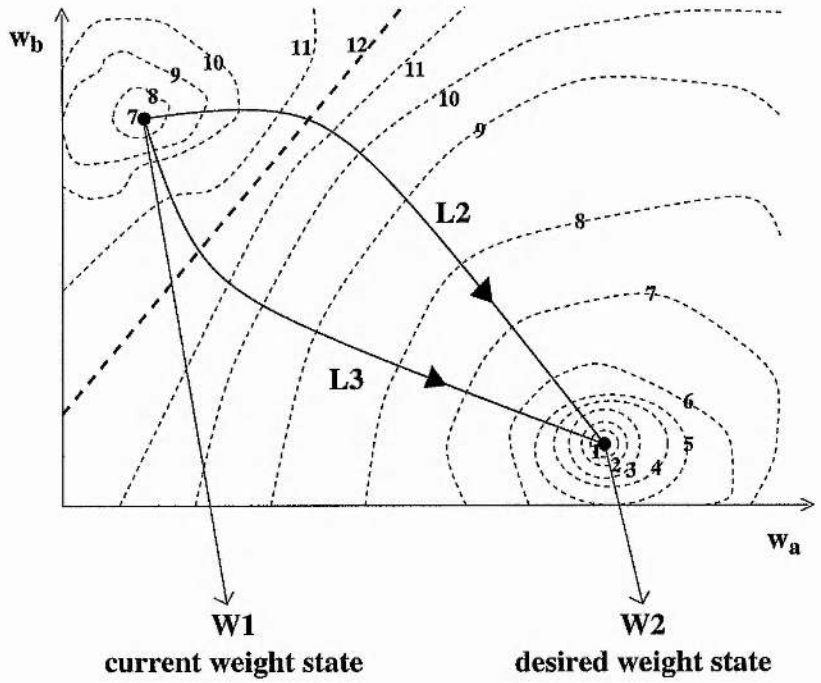


Figure 4.5: Local and global minimum in error-weight space. Weight axes are w_a and w_b. LMS error is indicated by the numbers on the dashed contour lines. **L2** and **L3** denote paths leading out of the local minimum to the global minimum.

the restriction they impose on the travel. The restriction is that setting a linear subgoal chain towards the goal forces travel to always proceed towards lower error with respect to the goal, just as standard unchained training does, such as BP without the use of subgoals. When situated at a local minimum for the goal, it is by definition impossible to immediately improve the neural outputs towards the desired goal output targets and so any attempt to escape a local minimum with the use of linear chaining must fail.

This point is illustrated in Figures 4.5 and 4.6. Figure 4.5 is a stylised view of weight space with a local minimum **W1** as the current weight state and the desired global minimum weight state **W2**. Figure 4.6 is a stylised view of the output space corresponding to the weight space in Figure 4.5. Output space is a way of viewing the current state and

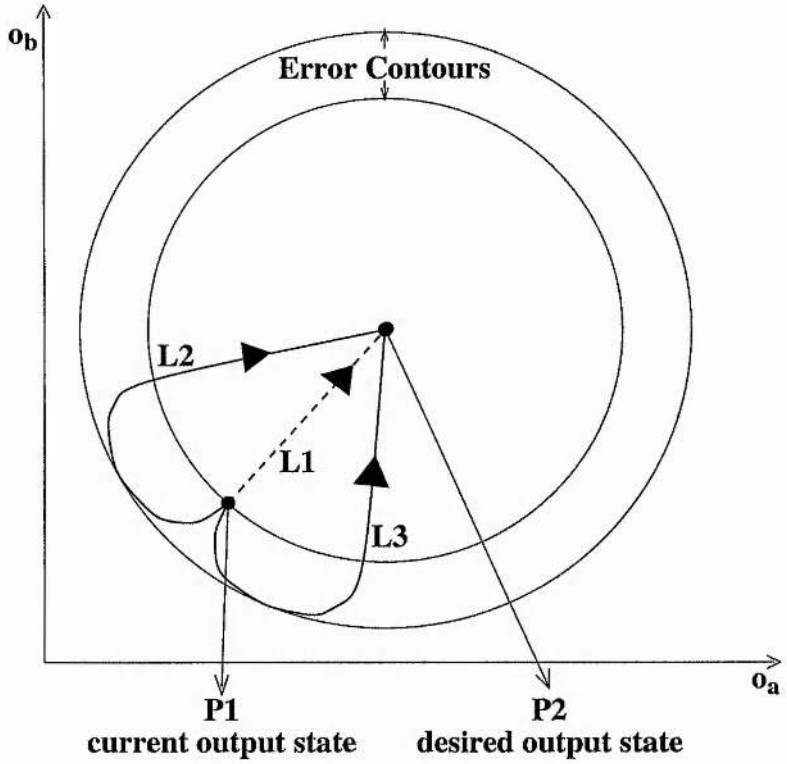


Figure 4.6: Local and global minimum in output space. Output axes are $\mathbf{o_a}$ and $\mathbf{o_b}$. The two concentric circles denote error contours with respect to the goal at **P2**.

the desired state of a neural network, namely in terms of the outputs the neural network produces for each pattern. The output axes $\mathbf{o_a}$ and $\mathbf{o_b}$ in Figure 4.6 denote the output the neural network is producing for pattern **a** and **b** respectively. Output space provides us with a visible direction to the goal and subgoals, because target output states can be viewed in output space. This is additional information which is not visible on a neural network's error-weight surface, on which the current error-weight state and the current gradient are commonly the only two measurable quantities.

The current output state **P1** in Figure 4.6, corresponding to the neural outputs obtained at the current weight state **W1** in Figure 4.5, is a local minimum for the goal at **P2**. For

this case, various paths such as the line **L1** are not realisable, because they constitute an immediate decrease in error with respect to the goal at **P2**. In particular, paths from local minima which monotonically decrease error with respect to the goal do not exist, no matter what surface variation occurs. Any paths which do exist, such as **L2** or **L3**, initially lead away from the goal and so constitute an initial increase in error. The increase is not only with respect to the goal but any subgoal along **L1**. Consequently, the weight state will converge to the local minimum rather than the global minimum when linear chaining is used.

ERA uses linear subgoal chaining and specifically places subgoals along the line **L1**, if **P1** is ERA's mean-valued starting state. As just described, **L1** is not realisable if **P1** is a local minimum for the goal and so ERA's weight transitions will therefore converge to the local minimum **W1** rather than the global minimum **W2**. In particular the attractor influencing the weight state transitions will change from being a global minimum, for all subgoals up until some specific subgoal, to being a local minimum for the next and every remaining subgoal on the linear chain **L1**. Point 3 in their argument therefore does not hold and the ERA method can no longer rely on passing from one global minimum to the next. Quite to the contrary, local minima are a major problem for ERA, and once they are encountered, ERA's linear subgoal chain will converge to them just as standard unchained chaining can.

Training examples have been constructed to show this empirically. Figure 4.7 shows a simple training example, one of many training examples which have been designed to place a local minimum on ERA's travel path. The principles behind constructing training examples which create local minima on the error-weight surface are described later in section 5.1.

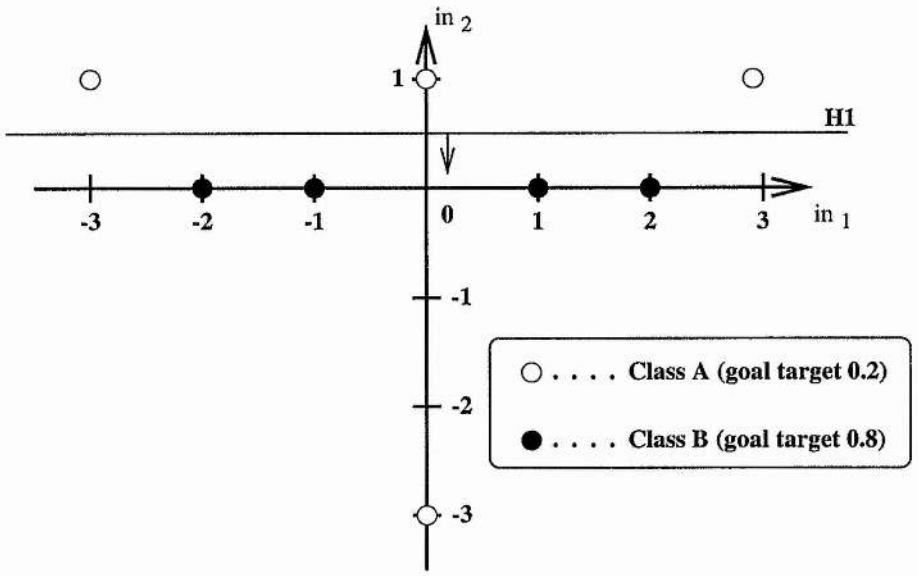


Figure 4.7: An inseparable training set for 2-1 network depicted in input space. It consists of 4 training patterns per target class. This training set creates a local minimum for the goal at ERA's mean-valued output state.

This example in particular, for a 2-1 network, has a local minimum at ERA's mean-valued starting state, which corresponds to an output of 0.5 for all training patterns. This mean-valued output state corresponds to an identical colouring for all training patterns as opposed to the desired colouring indicated by the training points' target classes in Figure 4.7. The local minimum occurs for zero valued weights, the global minimum separation is for a hyperplane close to the line indicated by **H1**.

Both standard BP and ERA were tested on this and many other examples, the results of which have been published in [LW99]. For this training example in particular, ERA fails completely to reach the global minimum and always gets stuck at the local minimum, while standard BP manages to converge to the global minimum in 8% of trials.

As mentioned before, the authors of the ERA paper partly base their claim for successfully

avoiding local minima on reaching the global minimum for the XOR problem in 100% of trials. The successful construction and testing of counter examples for linear subgoal chaining leads to the question of why the success and failure rates for ERA's linear subgoal chaining can be so extreme, i.e. 100% success for XOR and 100% failure for some counter examples.

The answer would appear to lie in ERA's starting conditions in particular and in the mechanism of linear subgoal chaining in general. ERA's starting conditions are such that it is necessary to pass through or near the mean-value state to begin with for every problem. This means that the outcome is dependent on the mean-value state rather than the initial weight-state. Empirical investigations have established that for XOR the first subgoal after the mean-value state is exactly realisable. The same is true for all subsequent subgoals including the goal, with a successful path to the goal being the result. It is not surprising therefore that all weight initialisations yield success.

It has also been reported that there are no local minima for a 2-2-1 network training on the XOR problem in [SKB96]. It is reported that although error-weight states with non-zero error but zero valued error gradients theoretically exist for infinite weight settings, a weight state path effecting strictly decreasing error exists from all finite weight settings to a zero valued global minimum. Consequently ERA's 100% success for XOR does not indicate that it is avoiding local minima.

The counter examples for ERA in [LW99] causing failure placed a local minimum for the goal on ERAs travel path in order to test whether the state transitions converge towards it

and become stuck. This resulted in ERA's failure every time, because, once convergent towards a local minimum for the goal, no further progress can be made immediately towards the remaining subgoals and so ERA did get stuck.

In summary, ERA is an example of using subgoals to vary the surface, but does not in any way deal with the local minimum problem it is purported to deal with. Quite contrary to the author's claims, local minima are a major difficulty for ERA. Linear subgoal chaining is in fact very similar to the standard unchained approach in the reasons for its success and failure. That is, while it undoubtedly generates different travel paths to those of the unchained approach due to using varying error-weight surfaces, it has similar attractor basins.

In essence ERA can be seen as having an underlying fixed travel surface which has the same attractors as the standard unchained approach. Success or failure for ERA for example depends on whether the mean-value state is in the goal's attractor basin on the underlying travel surface or not, and step-size issues apart, nothing else. That is why it is all-or-none for some problems. Linear subgoal chaining without the mean-value starting condition and initialised randomly may be expected to have a more variable success rate depending on the basin distribution for a problem.

Chapter 5

Local Minima and Unrealisable Regions

5.1 Local Minima

In this section, I will be describing some techniques to set the error-weight gradients to zero, in order to create a local minimum on the error-weight travel surface for gradient descent based techniques, such as BP for example. One of the reasons behind wanting to create local minima was in order to rebut the claims made for a technique called ERA [GST97], as discussed in sections 4.3 and 4.4, namely that ERA avoids local minima for a wide variety of problems. Another reason for creating local minima was to test a new subgoal chaining technique, which uses non-linear subgoal chains.

The method for creating zero-valued error-weight gradients will be developed here in terms of standard Least Mean Squared error, which is defined in (2.2) in section 2.1. The mean $\frac{1}{P}$ is replaced with $\frac{1}{2}$ to make the differentiated expression of the error function simpler. Just

to re-iterate the error function, as written in (2.3) in section 2.1, the total error for an output unit over all P training patterns is defined as

$$E = \frac{1}{2}\sum_{p=1}^{p=P}(\text{out}_p - T_p)^2 = \sum_{p=1}^{p=P} e_p \tag{5.1}$$

where out_p is the neural output, T_p is the desired output for pattern p, and e_p is the squared error for each training pattern p, as defined by (2.1) in section 2.1.

The main intuition behind the technique described here is to arrange the input coordinates and target values of the patterns in the training set such that the error-weight gradients are zero for certain weight states, i.e.

$$\frac{\partial E}{\partial w_{ij}} = 0, \forall i, j \tag{5.2}$$

where w_{ij} is a neural weight connecting from unit i to unit j in the network.

Testing for a Minimum

To check that the weight states producing zero-valued gradients correspond to minima, 2nd order derivatives or convergence tests can be used.

Using convergence in order to test for a minimum involves initialising the network at small random distances from the weight states to be tested, and then observing whether all trial runs converge towards the weight states under consideration.

Using 2nd order derivatives involves calculating the Hessian matrix of the error function with respect to the neural weights, and testing whether the Hessian fulfils certain properties. The Hessian is a matrix of second order partial derivatives of the error with respect to the neural

weights. In order to simplify the indices of the weights in the Hessian, the neural weights displayed in (5.3) are numbered from 1 to N for a neural network with N weights in total, but each neural weight w_n for $n = 1 \ldots N$ represents a unique weight w_{ij} from some unit i to unit$j$.

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 E}{\partial {w_1}^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E}{\partial {w_N}^2} \end{pmatrix} \tag{5.3}$$

If one can show the Hessian to be positive definite or positive semi-definite at the weight states under consideration then the zero-valued gradients correspond to a bowl shaped point minimum or a trough shaped minimum. This is because of the meaning of positive definite and positive semi-definite. The Hessian $\mathbf{H}$ is said to be positive definite if pre-multiplying and post-multiplying the Hessian by any non-zero length vector $\Delta\mathbf{w}$ produces a positive number. Put into a neural context, $\Delta\mathbf{w}$ is a vector containing weight changes with respect to the current weight state for all neural weights, and the positive number corresponds to the change in LMS error ΔE effected by the weight change. In mathematical notation this can be written as

$$\Delta E = \Delta\mathbf{w}^{\mathrm{T}}\,\mathbf{H}\,\Delta\mathbf{w} > 0, \forall\, \Delta\mathbf{w} \neq \mathbf{0} \tag{5.4}$$

In other words, if the Hessian is positive definite then any change in weight state around the current weight state at which the Hessian has been evaluated produces an increase in error which means that the current weight state must be a bowl shaped point minimum. The Hessian is not generally a constant over weight space, in fact it is only constant when

the error-weight relation is a quadratic equation. But it may be supposed that the error-weight surface can be approximated closely by a quadratic equation in the local vicinity of the current weight state. In other words the Hessian can be treated as being locally constant such that if the Hessian is positive definite at the current weight state then there is a minimum at the current state.

If the Hessian is positive semi-definite then any non-zero change in weight state $\Delta\mathbf{w}$ will produce a change in error $\Delta E \geq 0$. In a neural context positive semi-definiteness can be written in mathematical notation as

$$\Delta E = \Delta\mathbf{w}^{\mathrm{T}}\,\mathbf{H}\,\Delta\mathbf{w} \geq 0,\ \forall\,\Delta\mathbf{w} \neq \mathbf{0} \tag{5.5}$$

This means that the current weight state must be at the base of a trough shaped minimum. The directions in weight space along which $\Delta E = 0$ correspond to the directions along the base of the trough and the directions along which $\Delta E > 0$ correspond to walking up the sides of the trough. Again the prediction by the Hessian only holds in the local vicinity of the current weight state and so the trough may only look like a trough in a very small neighbourhood of the current state. For a more detailed discussion of positive definite and positive semi-definite and how to establish whether a matrix is positive definite or semi-definite see [Nob69].

Although Hessian tests may be more costly to implement than convergence tests one should bear in mind that convergence tests may lead to falsely recognising extended saddle points as minima if they look like plateaus of almost constant error. Convergence tests would merely show extremely slow travel in such regions and therefore one might falsely assume

a trough shaped minimum to be present. The advantage of Hessian tests is that once the Hessian has been evaluated one can see at a glance whether we are dealing with a minimum or not.

Zeroing the Error-weight Gradients

The main reason for developing techniques to zero the error-weight gradients was in order to create training examples for neural networks which place a local minimum on the travel path attempted by the ERA technique, as reviewed in section 4.4, which show that ERA fails to overcome local minima. The general principle of construction behind all the examples is therefore much the same, namely to place a local minimum for the goal at or near a weight state satisfying the initial ERA subgoal. This makes it either certain or at least likely that the local minimum basin on the goal's error-weight surface contains weight states producing ERA's mean-value output state. The goal's error-weight surface refers to the error-weight surface where the targets are those defined in the original training set. The mean-value output state is the output state satisfying ERA's initial subgoal. As described in section 4.4 the initial subgoal for the ERA process consists of target values T_p which have been compressed down to their mean-value $\langle T \rangle$ for each output unit

$$\langle T \rangle = \frac{1}{P} \sum_{p=1}^{P} T_p \tag{5.6}$$

The mean-value output state corresponds to the neural network producing an output of the mean-value $\langle T \rangle$ for each pattern in the training set.

By using a technique similar to one used by Brady [Bra89] and examined in [SS89] it is

possible to establish local minima in LMS error. Many of our problems differ from Brady's though in using inseparable data so that the best fit to the data involves misclassification.

The training set is divided between *non-spoiler* points and a relatively small set of *spoiler* points. Without the spoiler points, non-spoiler points, by definition, meet their targets exactly at the global minimum. When the spoilers are added there is no weight state where the new data set meets all of its targets exactly. Furthermore, the spoilers' positions are set so that there is, apart from the global minimum, at least one local minimum for the goal at or near ERA's mean-value output state.

The local minimum occurs where the points all produce exactly or nearly the mean output value. The global minimum has the non-spoilers nearly meeting their targets exactly, leaving the fewer spoilers with relatively high error. Figure 5.1, explained in detail below, illustrates a design for a 2-1 network where the points have exact mean-value outputs at the local minimum. The local minimum has zero weight values in this case while the global minimum is associated with a hyperplane near the separation line **H1** in Figure 5.1.

Single Layer Networks

The following training examples are for a 2-1 network and the local minimum for the goal is at or near ERA's mean-value state. The weight state which produces the mean-value output for a 2-1 network with a sigmoidal activation function, as shown in [GST97], is defined for the 3 weights w_{03}, w_{13} and w_{23} in Figure 5.2 as

$$w_{03} = \ln\left(\frac{\langle T\rangle}{1-\langle T\rangle}\right),\ w_{13} = 0,\ w_{23} = 0 \tag{5.7}$$

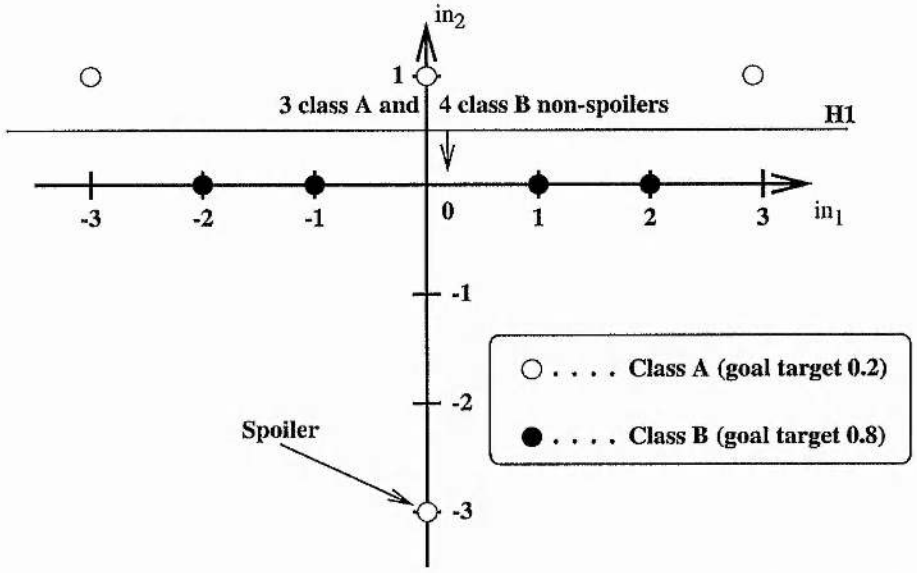


Figure 5.1: An inseparable input point set for a 2-1 network, consisting of 4 training patterns per target class, 7 non-spoilers and 1 spoiler, which creates a local minimum for the goal at ERA's mean-valued output state. The local minimum corresponds to zero-valued weights and the global minimum is associated with a hyperplane near the separation line **H1**.

A sigmoidal activation function is defined by (4.2) in section 4.1.

The weight state corresponding to the mean-value output may be set to be a stationary point by arranging the spoiler and non-spoiler points relative to one another so that the error-weight gradients are all zero. In order to ensure that (5.2) holds, i.e. that all error-weight gradients are zero-valued, we make the following observations. Firstly, the error-weight derivatives for an output unit j can be written as

$$\frac{\partial E_j}{\partial w_{ij}} = \sum_{p=1}^{P} \left(\frac{\partial e_{jp}}{\partial \mathrm{ex}_{jp}} \frac{\partial \mathrm{ex}_{jp}}{\partial w_{ij}} \right) = \sum_{p=1}^{P} \Delta_{jp} \, \mathrm{in}_{ip} = \sum_{p \in \text{class A}} \Delta_{jp} \, \mathrm{in}_{ip} + \sum_{p \in \text{class B}} \Delta_{jp} \, \mathrm{in}_{ip} \tag{5.8}$$

where E_j is the squared error for output unit j over all patterns, e_{jp} is the squared error for output unit j and pattern p and in_{ip} is the i^{th} component of the input to the neural network for

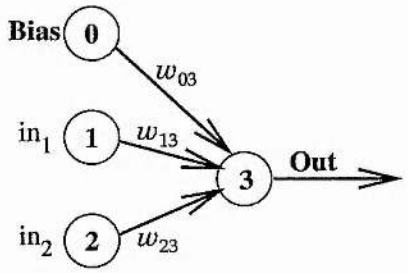


Figure 5.2: A 2-1 single layer network with 4 units numbered 0 to 3. It has 2 input units, 1 output unit and a bias unit which always has an output value of 1. The input units take on the values of training pattern coordinates on the in_1 and in_2 axes.

pattern p. The input to the network consists of the bias unit's output of 1 and the position of pattern p in input space. Δ_{jp} is defined for each pattern p and output unit j with a sigmoidal activation function as

$$\Delta_{jp} = \frac{\partial e_{jp}}{\partial \mathrm{ex}_{jp}} = -(T_{jp} - \mathrm{out}_{jp})\,\mathrm{out}_{jp}(1 - \mathrm{out}_{jp}) \tag{5.9}$$

where T_{jp} is the target output for pattern p and output unit j, out_{jp} is the output of unit j for pattern p. If the number of patterns in each target class is set to be equal we establish the mean-value output state as

$$\mathrm{out}_{\mathrm{MV}} = \frac{T_{jA} + T_{jB}}{2} \tag{5.10}$$

where T_{jA} and T_{jB} are the goal targets for patterns of class **A** and **B** respectively. Using (5.9) and (5.10) then yields

$$\begin{aligned} \Delta_{jA} &= -\,(T_{jA} - \mathrm{out}_{\mathrm{MV}})\,\mathrm{out}_{\mathrm{MV}}(1 - \mathrm{out}_{\mathrm{MV}}) \\ &= \quad (T_{jB} - \mathrm{out}_{\mathrm{MV}})\,\mathrm{out}_{\mathrm{MV}}(1 - \mathrm{out}_{\mathrm{MV}}) = -\Delta_{jB} \end{aligned} \tag{5.11}$$

Because there is an equal number of patterns in each class and the Delta for patterns of one class is the negative of the Delta for patterns in the other class one can make an observation

for weights connecting from the bias unit to the output unit. For the bias unit with $i = 0$ the sum of pattern Deltas times the bias unit's output $\text{in}_{0p} = 1, \forall p$ is zero, i.e. (5.8) is zero which means that zero-valued gradients have been created.

In order to create zero-valued error-weight gradients for weights connected from all other input lines with $i \neq 0$ it is required, in addition to having the same number of points in each class, that

$$\sum_{p \in \text{class A}} \text{in}_{ip} = \sum_{p \in \text{class B}} \text{in}_{ip} \tag{5.12}$$

The training set in Figure 5.1 is designed to obey (5.10) and (5.12) and so has zero-valued error-weight gradients and a local minimum at the mean-value state. The mean-value state corresponds to having zero-valued weights which produce zero excitation for every training pattern which corresponds to an output of exactly 0.5 when using a sigmoid activation function. At this local minimum mean-value state no separation of the training patterns is obtained. So any classification imposed on the obtained borderline output value of 0.5 will miss-classify 4 patterns with respect to their target class. In contrast all training patterns apart from the spoiler can be classified correctly at the global minimum indicated by the separation line **H1**.

Once a local minimum has been established, it is possible to move the position of the local and global minimum for single layer networks, by perturbing the position of the input points. If the spoiler point in Figure 5.1 is moved from (0, -3) to (0, -3.1) for instance, the gradients no longer cancel at the zero-valued weight state but do cancel at a weight state not far away. Effectively the local minimum is shifted from the zero-valued weight state producing the mean-valued output, to a state which in this case produces an output below

0.5 for the class **A** spoiler point, outputs slightly above 0.5 for all class **B** non-spoilers and a higher output for all class **A** non-spoilers. For this separation, 3 class **A** non-spoilers are incorrectly classified at the local minimum. The global minimum position is also perturbed by changing the position of the spoiler, but the separation at the global minimum still only miss-classifies the one spoiler point.

The input coordinates can be perturbed in a more random way than just described, but which also effects a shift in local and global minimum state. The points in Figure 5.3 have been generated in a more random way to make the example less artificial and to show to what extent misclassification can occur at the local minimum. The latter input point set no longer has a local minimum at the mean-value state exactly, but merely close to the mean-value state. The input points have been moved relative to the positions determined by (5.12) in a random fashion. At the global minimum, denoted by the separation line **H1**, all non-spoilers are separated according to their respective target class, while the outputs for the 4 spoilers are misclassified. In contrast, 27 points are misclassified with respect to their target class at the local minimum, denoted by the separation line **H2**. This is more than half the training set.

Figure 5.4 shows an association problem. This counter example is an exception to the previous design placing a local minimum at or near the mean-value state. This design is similar to one described in [Bra89], which was modified so that the previous global minimum state was shifted in weight space and status to become a local minimum at the end of ERA's subgoal path. The goal targets of 0.29 and 0.77 created the desired local minimum.

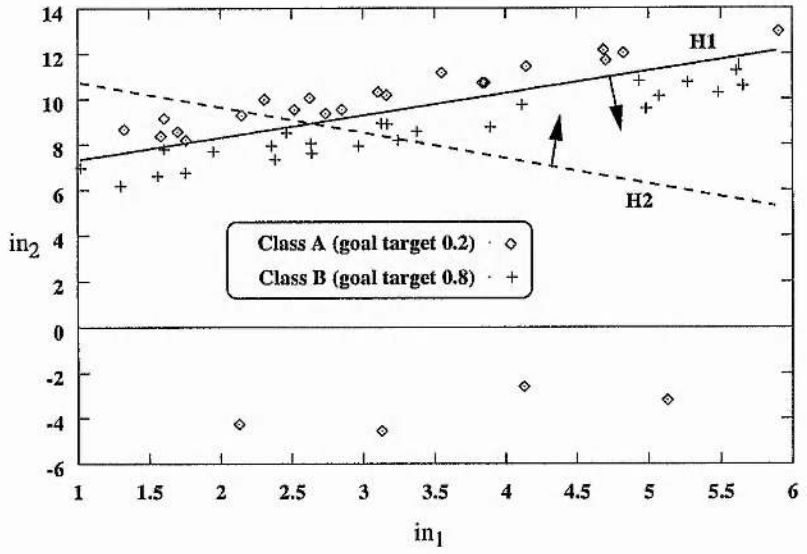


Figure 5.3: An inseparable input point set for a 2-1 network which creates a local minimum for the goal near the mean-value output state. There are 25 points in each class including the 4 class **A** spoilers. **H1** and **H2** denote global and local minimum separation lines respectively.

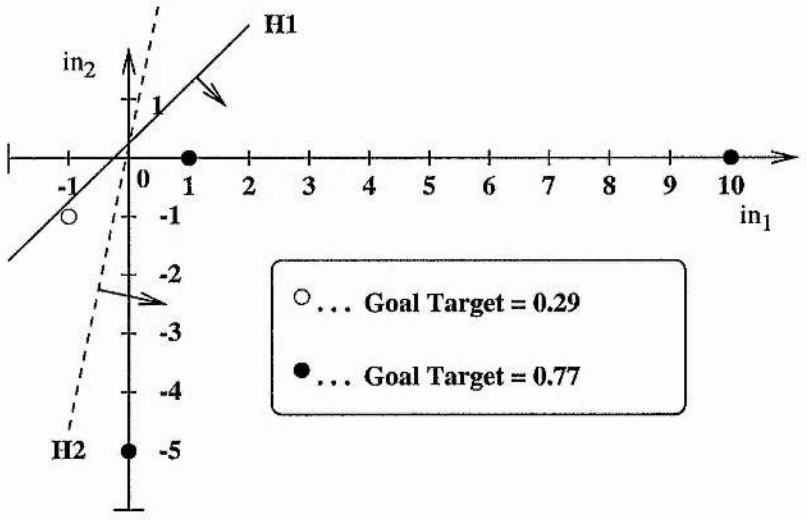


Figure 5.4: An association problem for a 2-1 network which creates a local minimum associated with a hyperplane near the line **H1** at the end of the path travelled by ERA. The global minimum is associated with a hyperplane near the separation line **H2**.

The construction of local minima for single layer networks can be extended to be more general. Specifically, one can calculate a position and target value of a point to be added to the training set which produces zero-valued error-weight gradients over all patterns in the expanded training set. If the error-weight gradients are already zero then a point can be added to the training set with any input coordinate and with a target setting equal to the current network output for that point, without altering the overall error-weight gradients. This is because the addition to the overall error-weight gradient by adding an extra point will be zero if the output for that point is identical to its target. The error-weight gradient for a single point p and output unit j can be written as

$$\frac{\partial e_{jp}}{\partial w_{ij}} = \left(\frac{\partial e_{jp}}{\partial \mathrm{ex}_{jp}} \frac{\partial \mathrm{ex}_{jp}}{\partial w_{ij}} \right) = \Delta_{jp} \, \mathrm{in}_{ip} = -(T_{jp} - \mathrm{out}_{jp}) \, \mathrm{out}_{jp}(1 - \mathrm{out}_{jp}) \, \mathrm{in}_{ip} \qquad (5.13)$$

which is zero if the target T_{jp} is set to be equal to the output out_{jp}.

If on the other hand the error-weight gradients are non-zero for the current weight state and training set, a single input point coordinate and target pairing exists for an extra point to be added to the training set which produces zero valued gradients over all patterns at the current weight state. For simplicity, the formulae in the following description for zeroing the error-weight gradients will be for a 2-1 network, but the procedure can easily be extended to any single layer network with N input units and M output units.

Imagine a 2-1 network which has non-zero error-weight gradients for the current weight state. In order to zero the error-weight gradients, we require the insertion of a point to the training set which balances out the current error-weight gradients. For error-weight gradients of $\frac{\partial E_j}{\partial w_{ij}}$ at the current weight state for the current training set we require an additional

gradient

$$G_{ij} = -\frac{\partial E_j}{\partial w_{ij}} \tag{5.14}$$

such that the gradients add to zero after insertion of the new training point. Specifically for the additional point p and this 2-1 network as shown in Figure 5.2 we require that the error weight gradient for the weight w_{03} from the bias unit 0 to output unit 3

$$\frac{\partial e_{3p}}{\partial w_{03}} = \Delta_{3p}\, 1 = G_{03} \tag{5.15}$$

For the two weights w_{13} and w_{23} along the in_1 and in_2 input lines we require that

$$\frac{\partial e_{3p}}{\partial w_{13}} = \Delta_{3p}\, \text{in}_{1p} = G_{13} \tag{5.16}$$

and

$$\frac{\partial e_{3p}}{\partial w_{23}} = \Delta_{3p}\, \text{in}_{2p} = G_{23} \tag{5.17}$$

where in_{1p} and in_{2p} are the input coordinates of the training point being added to the training set. The output unit's delta Δ_{jp} is as previously defined in (5.9) and so from (5.15) we get

$$-(T_{3p} - \text{out}_{3p})\, \text{out}_{3p}(1 - \text{out}_{3p}) = G_{03} \tag{5.18}$$

which gives the required target value

$$T_{3p} = \text{out}_{3p} - \frac{G_{03}}{\text{out}_{3p}\,(1 - \text{out}_{3p})} \tag{5.19}$$

which means we can satisfy (5.15) at the current weight state for any value of G_{03} within reason, by setting the appropriate target value for that point depending on its output.

Now if (5.15) is satisfied then $\Delta_{3p} = G_{03}$ and so (5.16) is clearly satisfied if and only if

$$\text{in}_{1p} = \frac{G_{13}}{G_{03}} \tag{5.20}$$

and (5.17) is satisfied if and only if

$$\text{in}_{2p} = \frac{G_{23}}{G_{03}} \tag{5.21}$$

A training point p with input coordinates $(\text{in}_{1p}, \text{in}_{2p})$ creates an output out_{3p} which inserted into (5.19) allows the calculation of the required target for which now (5.15) and hence (5.16) and (5.17) are satisfied.

As previously described all the error-weight gradients can be zeroed in this way. Thereafter the Hessian can be examined for being positive definite or positive semi-definite which ensures that (5.4) or (5.5) holds, i.e. that the weight state is either a bowl-shaped point minimum or trough shaped minimum. If the Hessian is neither then the zero-valued gradients may correspond to a Saddle point or a local maximum. Instead or in addition to examining the Hessian, convergence tests can be used to determine whether the weight state is a minimum or not.

5.2 Unrealisable Regions

When using a neural network to perform pattern classification for example, the objective is that the neural network learns to assign categories or target classes to the inputs of the training patterns. The neural network is supposed to do this by producing an appropriate output for each pattern which represents the class into which the neural network places the input pattern.

As mentioned before in section 2.1, a single layer network can only perform linear separa-

tion of the training data. Even if the underlying mapping of the true data values is linear but the actual data are sufficiently noisy then the relation between training input and target class can be non-linear and the single layer network will not be able to assign all patterns to their respective target classes. This means that the vector state comprising the goal target output values for all patterns is not realisable in this case and so there is a clear distinction here between the goal and the output state at the global minimum with non-zero error, because all patterns' outputs will not match their target at the global minimum in error-weight space.

It is commonly understood that unrealisable states must exist in various cases depending on the topology of the network and the training data. In fact this unrealisability of states can even be desired because it encourages the network to *generalise* and not *memorise*. If the network can always produce the desired target state it is in other words able to memorise the entire data set and so is able to over-fit the underlying noiseless mapping of the noisy data by precisely fitting the noisy data. Mostly it is desired that the network do the opposite, i.e. the correct fit often has the appearance of under-fitting the actual data which is likely to be noisy. This appearance of under-fitting occurs by producing the best possible linear fit to the seemingly non-linearly separable data set for example. In this state the network is more likely to be approximating the underlying clean mapping and not the noisy fluctuations in the data.

Especially when a local minimum exists on the goal's error-weight surface, then an unrealisable region must exist in output space. The reason for this was explained in section 4.4. When the current weight state for a neural network is a local minimum for the goal it

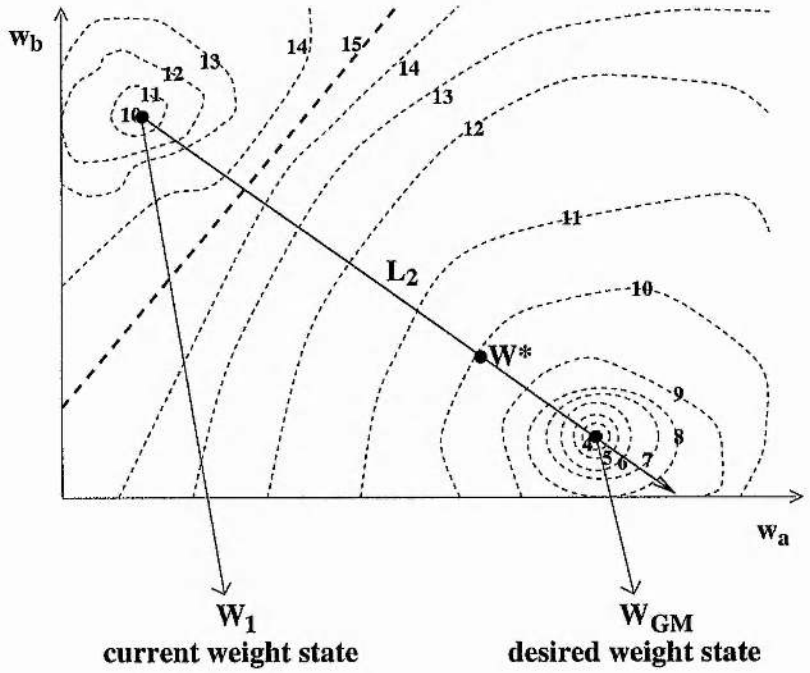


Figure 5.5: Local and global minimum in error-weight space. Weight axes are w_a and w_b. LMS error with respect to the goal is indicated by the numbers on the dashed contour lines.

follows that any motion away from the local minimum results in an initial increase in error with respect to the goal. This situation is displayed in Figure 5.5 where the current weight state $\mathbf{W_1}$ is a local minimum and $\mathbf{L_2}$ denotes a straight line path leading out of the local minimum to the global minimum $\mathbf{W_{GM}}$ by initially increasing the error before decreasing the error.

The fact that all paths leading out of the local minimum to the global minimum must initially increase in error with respect to the goal means that certain paths in output space are not realisable. Paths in output space which do not initially increase error with respect to the goal, and especially those used by ERA as explained in section 4.4, which lead directly towards the goal from the local minimum are not realisable and will cause training

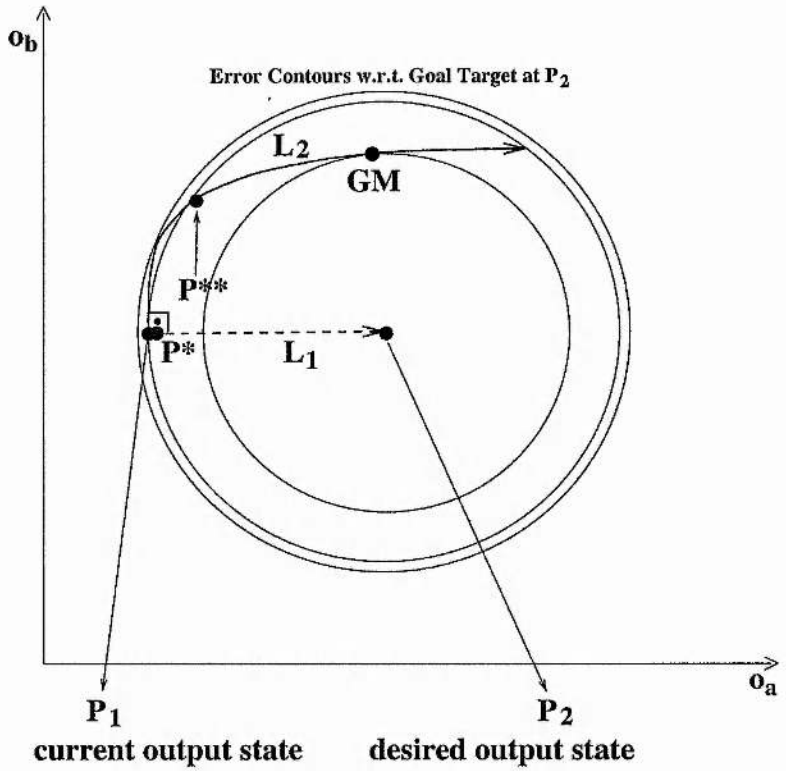


Figure 5.6: Local and global minimum in output space. Output axes are o_a and o_b. The three concentric circles denote error contours with respect to the goal at P_2. L_1 denotes an unrealisable path, P* is the first state from P_1 along L_1 closer to the goal than P_1. L_2 denotes a realisable path to the global minimum GM, P** is the first state from P_1 along L_2 which is closer to the goal than P_1.

to converge to the local minimum rather than the global minimum.

This situation is shown in Figure 5.6 where the straight line path L_1 cannot be realisable, because P_1 is a local minimum for the goal at P_2. The path L_2 on the other hand denotes a realisable path which allows travel from the local minimum P_1 to the global minimum GM, by initially increasing in error before decreasing error with respect to the goal at P_2. It should be noted here that any realisable trajectory away from a minimum in general and at a local minimum P_1 in particular must initially be tangential to the hypersphere through

the current output state $\mathbf{P_1}$ and centred on the goal at $\mathbf{P_2}$. This is because only the tangent hyperplane to the hypersphere contains directions which satisfy the condition of being at a minimum, namely that $\frac{\partial E}{\partial w_i} = 0, \forall i$, i.e. all error-weight gradients are zero. E denotes standard LMS error and w_i denotes the i^{th} weight in the neural network. A proof basically consists of looking at a re-write of the error-weight derivatives in terms of the vector from the current state $\mathbf{P_1}$ to the goal $\mathbf{P_2}$ and the vector denoting how the output will vary for a given weight change. What follows is the one line equivalence which shows this.

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\left(\frac{1}{2}(\mathbf{P_2} - \mathbf{o})^2\right) = -\left(\frac{\partial \mathbf{o}}{\partial w_i}\right)_{\mathbf{o}=\mathbf{P_1}} \cdot (\mathbf{P_2} - \mathbf{P_1}) = -\left|\frac{\partial \mathbf{o}}{\partial w_i}\right|_{\mathbf{o}=\mathbf{P_1}} |(\mathbf{P_2} - \mathbf{P_1})| \cos(\phi) \quad (5.22)$$

where $\mathbf{o}$ is a vector function which returns the output state as a function of the weight state, comprised of all w_i, and the training set inputs. The squared term $(\mathbf{P_2} - \mathbf{o})^2$ is being used as a short form of the scalar product of the vector $(\mathbf{P_2} - \mathbf{o})$ with itself. $\mathbf{P_2}$ is the current goal target in output space. The angle ϕ is between the direction to the goal $(\mathbf{P_2} - \mathbf{P_1})$ and the linear approximations of how the output state will change for a given weight change, which are denoted by $\frac{\partial \mathbf{o}}{\partial w_i}$. As a reminder, the derivative of a vector function is merely a vector containing the derivatives of the original vector's function elements.

One can see that if the current state is a minimum, which does not produce the goal target state then $|(\mathbf{P_2} - \mathbf{P_1})| \neq 0$. Consequently $\frac{\partial E}{\partial w_i} = 0, \forall i$ only if either $\left|\frac{\partial \mathbf{o}}{\partial w_i}\right| = 0$ or the realised direction is orthogonal to the current goal direction. It may be shown that the expression $\left|\frac{\partial \mathbf{o}}{\partial w_i}\right|$ is only zero if all the elements o_p of the output state $\mathbf{o}$ are either 0 or 1. With a sigmoid activation function this will never occur. It follows that in order for $\frac{\partial E}{\partial w_i}$ to be zero for all weights w_i the only realisable directions when escaping a minimum are tangential to the hypersphere surrounding the goal state.

A fact has been established here and in section 4.4 that there cannot be an instantaneous connection in weight space from $\mathbf{W_1}$ to a weight state **W*** say, which produces an output state with infinitesimally lower error than at the local minimum $\mathbf{P_1}$ because such a connection would allow travel immediately towards the goal which is impossible when situated at a local minimum.

The question arises in this section as to how this observation provides us with an indication of the extent of unrealisable regions. Does the lack of an instantaneous connection in weight space, for instance, show a lack of existence of some or possibly even all states along an unrealisable path such as $\mathbf{L_1}$ for example? In other words does an unrealisable path mean that some or maybe all of its states are not realisable, i.e. not producible with any weight state?

To argue this point, one might ask whether it possible that a weight state **W*** produces the first output state **P*** along the path $\mathbf{L_1}$ which decreases error with respect to the goal $\mathbf{P_2}$. If an output state **P*** infinitesimally close to $\mathbf{P_1}$ but along $\mathbf{L_1}$ and so closer to $\mathbf{P_2}$ were realisable, it would have to be produced by a weight state similar to **W*** that it is relatively far from the local minimum weight state $\mathbf{W_1}$, as indicated in Figure 5.5. If one were to draw a straight line from $\mathbf{W_1}$ to **W*** then the change in weight state along that line would for a single layer N-1 network mean that the excitation for each pattern p changes in a straight line as well. This is because the change in excitation $\Delta \mathrm{ex}_p$ for each pattern p is a linear function of the change in weight state as can be seen in (5.23)

$$\Delta \mathrm{ex}_p = \sum_{i=1}^{i=N} \Delta w_i \, \mathrm{in}_{ip} + \Delta w_\beta \tag{5.23}$$

where Δw_i represents the change in the weight w_i connecting from input i to the output unit, Δw_β is the change in the bias weight connecting to the output unit and in_{ip} represents the i^{th} input of pattern p.

If the change in weight state $\Delta\mathbf{W}$ containing all weight changes Δw_i and Δw_β is parameterised by a parameter λ such that

$$\Delta\mathbf{W} = \lambda\left(\mathbf{W}^* - \mathbf{W}_1\right) \tag{5.24}$$

where λ can take on any value between 0 and 1 then the change in excitation Δex_p for each pattern can be expressed as a function of λ. The function $\Delta\text{ex}_p(\lambda)$ will be a straight line function of the parameter λ. The change in the outputs Δout_p can similarly be written as a function of λ by applying the activation function to the change in excitation. With a sigmoidal activation function $\Delta\text{out}_p(\lambda)$ will be a strictly increasing or strictly decreasing monotonic function of λ due to the monotonicity of the sigmoid.

Consequently a change in weight state from $\mathbf{W_1}$ towards $\mathbf{W}^*$ will effect a monotonic change in output for all patterns as the error-weight state initially climbs over the error height separating local and global minimum and descends into the global minimum basin surrounding $\mathbf{W_{GM}}$. With the outputs for each pattern changing monotonically to effect significant changes in error it is impossible that the output state produced at $\mathbf{W}^*$ is infinitesimally close to $\mathbf{P_1}$ and on $\mathbf{L_1}$. So we have disproved our original assumption that $\mathbf{P}^*$ is produced by $\mathbf{W}^*$ and is infinitesimally close to $\mathbf{P_1}$ and on $\mathbf{L_1}$. In other words $\mathbf{P}^*$ as the first output state along $\mathbf{L_1}$ which decreases the error with respect to the goal at $\mathbf{P_2}$ cannot be realisable, i.e. producible by any weight state. Instead the weight state $\mathbf{W}^*$ will produce an output

state more like **P**** in Figure 5.6.

It is possible to establish a conjecture at this point. It concerns the realisability of the goal target state for single layer networks. When starting at a local minimum as established before the only realisable directions are initially tangential to the goal direction. For a single layer network the straight line connection between a local and global minimum weight state produces monotonically changing outputs for all patterns as described above. Let us now imagine that the global minimum weight state produces the desired goal output state. The straight line path in weight space between local and global minimum weight state must then produce a monotonically changing output state path starting at the local minimum and ending at the goal with zero error.

One may argue that if the initial direction of this monotonically changing output path is tangential to the hypersphere centred on the goal and through the current state then the path cannot reach the goal target state with zero error. To my mind the only way to reach the goal would be to break either the monotonicity condition or allow the initial direction to point into the hypersphere towards the goal, i.e. violate the constraint that travel must initially be tangential to the hypersphere. Since neither of these are possibilities when starting a single layer network at a local minimum the final state cannot be the goal. This contradicts the original assumption that the global minimum weight state produces the goal output state.

Therefore the conjecture may be formulated that for single layer networks any training problem which creates a local minimum on the goal's error weight surface has an unrealisable goal. This is echoed in Figures 5.5 and 5.6 in which the goal at $\mathbf{P_2}$ is not actually

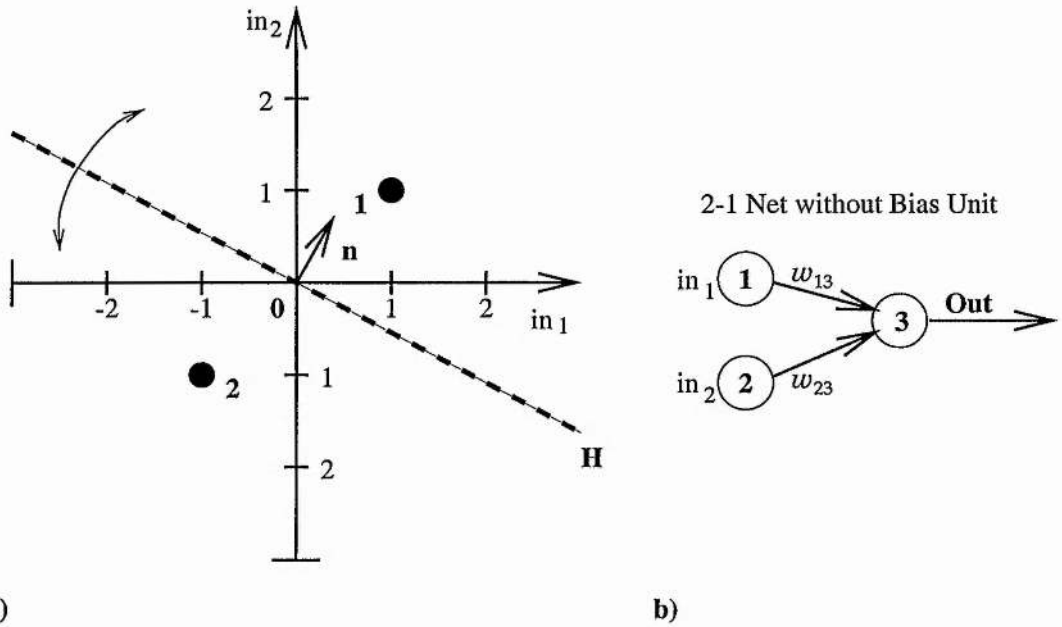
realisable itself, instead the global minimum weight state $\mathbf{W_{GM}}$ in Figure 5.5 produces an output state **GM** in Figure 5.6 with quite obviously non-zero error.

After having seen that unrealisable states quite clearly exist it is plausible to imagine that regions of unrealisable states exist and that realisable paths may be rather sparsely represented in output space. To gain a higher understanding of what realisable paths and unrealisable regions may look like more examples are examined.

The following example is for a 2-1 network without a bias unit. An schematic view of such a network can be seen in Figure 5.7**(b)**. It only has 2 weights, one for each of the input patterns' dimensions which make this kind of network very restricted in terms of what it can do. Figure 5.7**(a)** shows a training set for this 2-1 network which consists of 2 points, one at (1, 1) and the other at (-1, -1) in 2-D input space.

A 2-1 network without a bias unit is merely able to place a separating hyperplane through the origin of input space which is denoted by the separating line **H** in Figure 5.7**(a)**. This hyperplane can be rotated about the origin and the excitation achieved for the whole point set can be scaled for any rotation of the hyperplane. Training point 1 for the depicted rotation will have a positive excitation because excitation is a function of the distance from **H** to the training point in the direction of the normal to the hyperplane **n**. Training point 2 on the other hand will have a negative excitation because the distance from **H** to the training point is in the opposite direction to **n**.

Due to the symmetry in the training set the distances from the hyperplane **H** to the training points will always be the same, just in opposite directions and so the excitations will have



a) b)

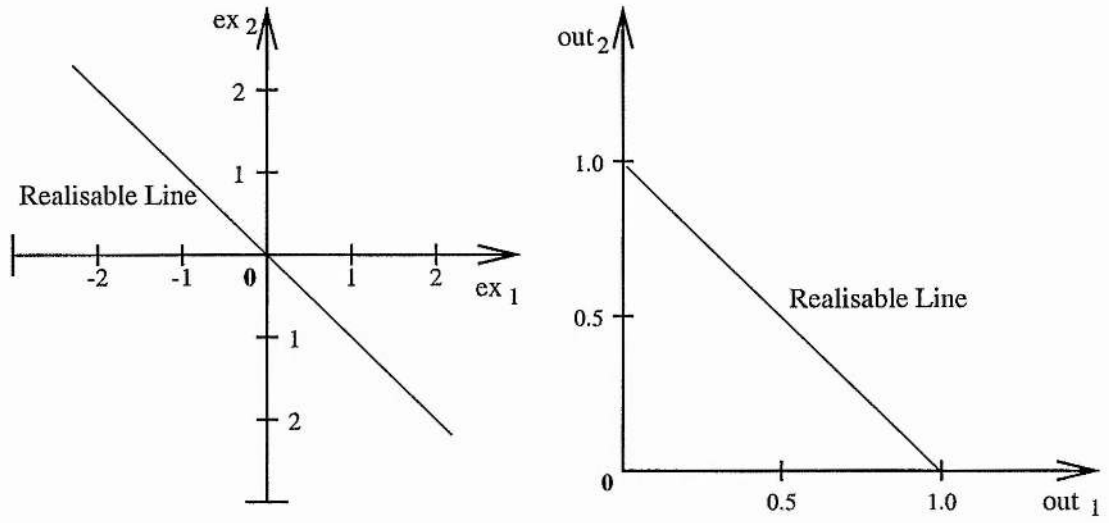
Figure 5.7: **(a)** shows input space with axes in_1 and in_2 with a training set for a 2-1 network consisting of two training patterns. **H** denotes neural separation hyperplane through the origin of input space. **(b)** is a schematic view of a single layer 2-1 network without a bias unit for which the training set in **(a)** has been designed.

opposite sign. This is depicted in Figure 5.8**(a)** which shows excitation space with its excitation axes ex_1 for pattern 1 and ex_2 for pattern 2. The only excitation states which the network can produce must lie along the line

$$ex_2 = -ex_1 \tag{5.25}$$

which defines the realisable path in output space. If an activation function $g(ex)$ is used which produces an output which is symmetric with respect to zero excitation then the realisable line in excitation space maps to a realisable line in output space. A sigmoid activation function for example produces outputs which are symmetric with respect to the zero excitation output of 0.5, which means that

$$g(ex) - 0.5 = 0.5 - g(\text{-}Ex) \tag{5.26}$$



a) b)

Figure 5.8: **(a)** shows excitation space for 2 training patterns with axes ex_1 and ex_2. **(b)** shows output space for the same 2 training patterns with axes out_1 and out_2.

such that the realisable line in excitation space in (5.25) maps to a realisable line

$$out_2 = 1 - out_1 \tag{5.27}$$

in output space as depicted in Figure 5.8**(b)**.

For the training example and network topology just described the unrealisable region is large, namely the whole of output space apart from the realisable line. This realisation prompts the view that when unrealisable regions exist in output space the realisable region is a space of lesser dimension than output space. In this case 2-D output space has a realisable 1-D line embedded in it. In what follows unrealisable regions will be examined with regard to when they are to be expected and what their dimensionality is.

For single later networks the solution manifolds for each training pattern are hyperplanes in weight space as described in section 4.1. The distance of a weight state from a solution

hyperplane for pattern p determines the excitation for that pattern p and the distance is measured along the direction of the normal to that hyperplane. Now in N-D weight space there can be at most N *linearly independent* vector directions and so at most N linearly independent normal vectors to the solution hyperplanes. A property of linear independence is that any vector belonging to a set of linearly independent vectors cannot be formed through a linear combination of the other vectors in the set. N linearly independent vectors span any N-D space, in other words they form a basis for this space. Any vector in the space can be written as a linear combination of the linearly independent basis vectors.

For a detailed discussion of linear independence, basis vectors and vector spaces see [Nob69]. If the number of training patterns P is greater than the number of weights N, for instance when $P = N + 1$, then at least one of the distances of a weight state to one of the $N + 1$ solution manifolds can be re-expressed in terms of a linear combination of the distances to the other N solution manifolds. This means that at least one pattern's excitation can be written as a function of the other patterns' excitations which means that excitation space can no longer be spanned and so unrealisable regions must exist.

To visualise this take the following example. Figure 5.9(**a**) shows a training set for a 2-1 network without bias unit. It has 3 training patterns and 2 analogue output target classes of 0.2 and 0.8, for which the global minimum separation along the in_2-axis has zero error. The solution hyperplanes for the input patterns 1, 2 and 3 are indicated in Figure 5.9(**b**) by the three lines numbered 1, 2 and 3. The units of $\ln 4$ along the w_1 and w_2 axes are for the case where the activation function is a sigmoid. Using a sigmoid activation function as defined in (4.3) and its inverse (4.5) as displayed in section 4.1, the three solution manifolds can be

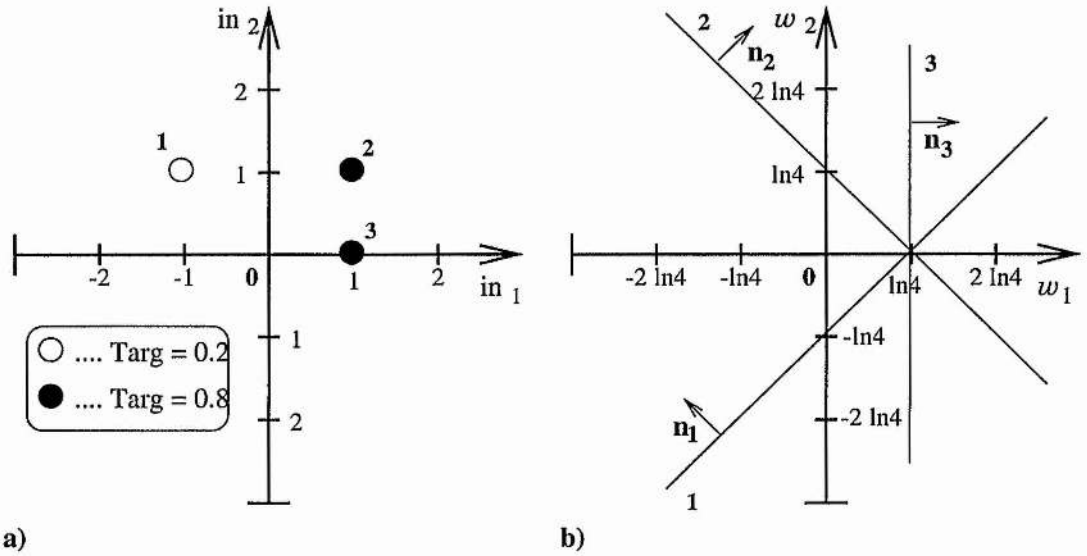


Figure 5.9: **(a)** shows input space with 3 training patterns in two target output classes for a 2-1 network without bias unit with axes in_1 and in_2. **(b)** shows 2-D weight space with 3 solution manifolds for the same 3 training patterns with axes w_1 and w_2.

written as

$$\text{ex}_1 = -w_1 + w_2 = g^{-1}(0.2) = -\ln 4 \tag{5.28}$$

$$\text{ex}_2 = w_1 + w_2 = g^{-1}(0.8) = +\ln 4 \tag{5.29}$$

$$\text{ex}_3 = w_1 = g^{-1}(0.8) = +\ln 4 \tag{5.30}$$

Each pattern's solution manifold p has a normal vector $\mathbf{n}_p$ which in Figure 5.9**(b)** point in the direction of increasing excitation for each pattern. One can see from this that the normals $\mathbf{n}_1$ and $\mathbf{n}_2$ for patterns 1 and 2 respectively span weight space such that $\mathbf{n}_3$ can be

re-expressed in terms of $\mathbf{n}_1$ and $\mathbf{n}_2$. This means that ex_3 can be re-expressed in terms of ex_1 and ex_2,

$$ex_3 = \frac{1}{2}(ex_2 - ex_1) \tag{5.31}$$

which means that the realisable region in 3-D excitation space is the 2-D plane defined by

$$ex_1 - ex_2 + 2ex_3 = 0 \tag{5.32}$$

which goes through the origin of excitation space.

Any straight line defined as the line through 2 arbitrary states on the realisable plane in excitation space can be parameterised as a function of λ such that the individual patterns' excitations ex_p along this line are functions of λ. This means that the outputs out_p for each pattern are also functions of λ. The $out_p(\lambda)$ are, similar to before, strictly monotonically increasing or decreasing functions of λ. Because this is true for any line on the realisable plane in excitation space being mapped to a curve in output space one might imagine the realisable surface in output space as a curved surface along which all outputs change monotonically. In future I will refer to the realisable manifold in output space as being monotonically changing for single layer networks.

In this case the goal is exactly realisable, i.e. the goal lies on the realisable surface in output space. If pattern 3 were changed to belong to the target class 0.2 then this would effectively shift the goal off the realisable surface.

A more general statement regarding realisable and unrealisable regions in output space for single layer networks may be formulated at this point.

1. Let P denote the number of training patterns.

2. Let I be the number of normals to P solution manifolds in N dimensional weight space which are linearly independent where $I \leq N$.

3. If $P > I$ then there are unrealisable regions and the realisable region in P dimensional output space is of reduced dimension I. The unrealisable region will be of dimension $P - I$.

4. If $P = I$ then there are no unrealisable regions, i.e. every output state for outputs between 0 and 1 may produced by a single weight state.

5. If $P < I$ then again there are no unrealisable regions but now every output state for outputs between 0 and 1 may be produced by a multiple of weight states.

As a consequence, if the number of patterns P is greater than the number of weights N, then unrealisable regions must exist because the largest number of linearly independent normal vectors to solution manifolds in weight space is $I \leq N$ which is smaller than P. The realisable region will be of dimensionality I.

5.3 Experiments with ERA

In section 4.4 the ERA method and the reasons why ERA should fail when encountering a local minimum for the goal on its travel path were described. This brief section summarises the results of testing ERA and standard unchained BP on the training examples

constructed using the techniques described in section 5.1 and discusses the results in light of the discoveries made in section 5.2 about unrealisable regions.

Both standard unchained BP and ERA were tested on 3 training sets from 25 random initial weight states in the range [-1, +1]. BP was used to train the network on ERA's subgoal chain.

The learning rate was set to be very low, with no momentum being used, in order to encourage training to follow the shape of the error-weight surfaces for each subgoal as closely as possible. This was in order to compare the success obtained with and without the use of the subgoal chains as accurately as possible by ruling out successful training caused by high learning rates or the use of momentum which may benefit one technique more than the other.

The training sets referred to in Table 5.1 as data sets 1, 2 and 3 are those depicted in Figures 5.1, 5.3 and 5.4 respectively and which can be found on page 106 and page 110. The 3 training sets are for a single layer 2-1 network.

The results summarised in Table 5.1 show ERA failing completely on all problems whereas standard BP (Std) training can find the global minimum with up to 32% success. One can see that if ERA's mean-valued output state is a local minimum for the goal ERA is bound to fail to reach the global minimum (data set 1). Failure also occurs when a local minimum for the goal lies at the end of the training path directed by the linear subgoal chain (data sets 2 and 3). Failing to reach the global minimum results in severe misclassification for the linearly inseparable problems (1 and 2). For the association problem (data set 3), failure to

Data Set	Training Method	Learning Rate	% finding Global	Average Cycles for Success
1	Std	0.1	8	906.5
1	ERA	0.1	0	fails
2	Std	0.005	4	10060.0
2	ERA	0.005	0	fails
3	Std	0.1	32	78.6
3	ERA	0.1	0	fails

Table 5.1: Table of results for standard unchained BP training (Std) and ERA.

reach the global minimum produces poor association.

As described in section 4.4, the reason for ERA's failure on data set 1 is because ERA's first subgoal consists of the mean-valued output state which constitutes a local minimum for the goal. This state is likewise a minimum for any subgoal along ERA's linear subgoal chain from the mean-valued state to the goal. Consequently ERA converges to the local minimum for the goal after training on the first subgoal in the chain and training on the remaining subgoals in the linear chain does not change the trained state thereafter. This means that ERA gets stuck at the local minimum for the goal after training on the first subgoal and is therefore unable to reach the global minimum regardless of its random starting state.

Figure 5.10 shows output space with the goal at **G**, the global minimum for **G** at **GM**, and the local minimum for **G** at **LM**. The presence of a local minimum on the goal's error-weight surface means that only a subspace of output space is realisable as discussed

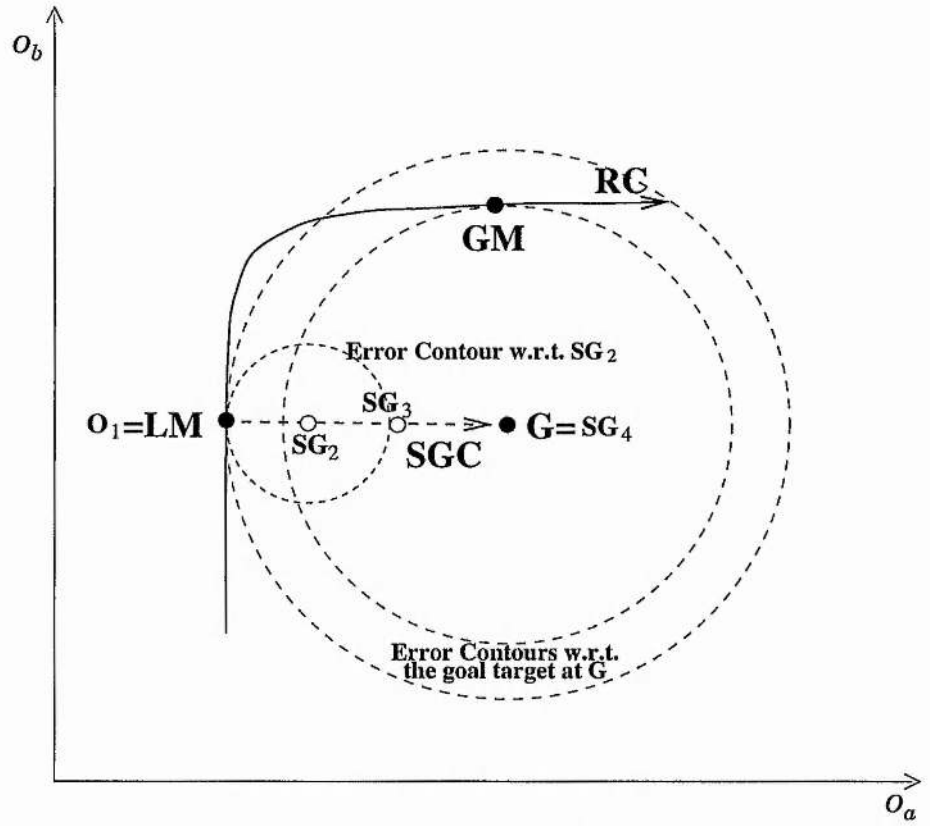


Figure 5.10: Stylised view of 2-D output space with its axes O_a and O_b. ERA gets stuck when starting at a local minimum for the goal.

in section 5.2. For single layer networks the parameterised form of any realisable output curve corresponding to a linear change in excitation is also a monotonic function of the parameter. To indicate this the realisable 1-D subspace of 2-D output space is denoted by the monotonically changing curve **RC**. The dashed concentric circles centred on **G** denote error contours with respect to the goal and the dashed circle centred on $\mathbf{SG_2}$ denotes an error contour with respect to that subgoal.

The output state $\mathbf{O_1}$ represents the mean-valued output state which is ERA's realisable 1st subgoal $\mathbf{SG_1}$ and effective starting position regardless of random initialisation. In this case $\mathbf{O_1}$ is also a local minimum for the goal. The subgoal chain is denoted by **SGC**, which leads

from $\mathbf{SG_1}$ to **G**. For simplicity, only 4 subgoals have been depicted on the subgoal chain. It should be mentioned though that ERA will get stuck at the local minimum regardless of the number of subgoals on the chain.

ERA effectively starts at $\mathbf{O_1}$ and aims to minimise the error with respect to the subgoal $\mathbf{SG_2}$. The only directions that are realisable from $\mathbf{O_1}$ are along **RC**. From the goal error contours one can see that all realisable directions represent an increase in error with respect to the goal and any subgoal along **SGC** (see the error contour around $\mathbf{SG_2}$). Consequently trying to minimise error using some form of gradient descent such as BP means that training gets stuck at the current state **LM**. This makes for 0% success for ERA on this training example which places a local minimum at ERA's mean-valued output state.

For data sets 2 and 3 ERA converges to the local minimum for the goal when trained on the final subgoal which is the goal. The reason for this can be best explained with the use of another graphic.

Figure 5.11 shows a training scenario in output space for ERA being trained on data sets 2 and 3. **RC** denotes the realisable curve, **G** is the goal, **GM** is the global minimum and the local minimum for **G** is at **LM**. The output state $\mathbf{O^*}$ denotes a state on **RC** from which any travel direction along **RC** results in decreasing the error with respect to the goal. Along one direction from $\mathbf{O^*}$ the goal error is monotonically decreased towards the global minimum at **GM**. Along the other direction from $\mathbf{O^*}$ the goal error is monotonically decreased towards the local minimum **LM**.

The part of **RC** through **GM** up to $\mathbf{O^*}$ can be seen as the output space representation of the

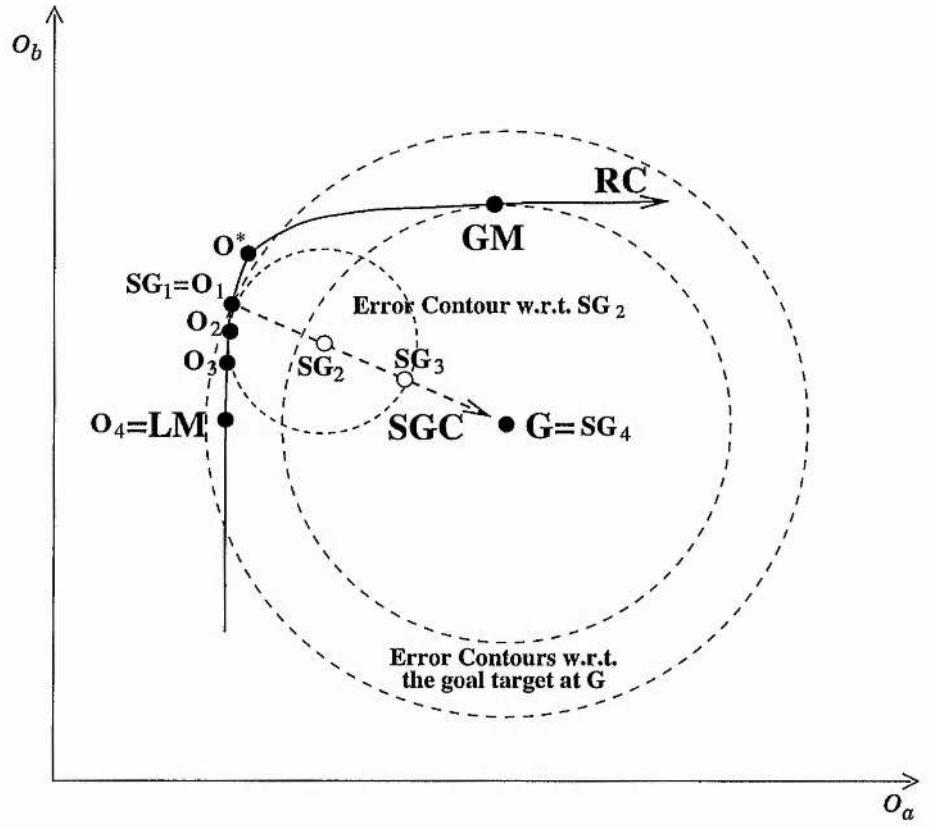


Figure 5.11: Stylised view of 2-D output space with its 2 axes O_a and O_b. ERA converges to the local minimum when its mean-valued output state is at $\mathbf{O}_1$.

global minimum's basin of attraction. Similarly the part of **RC** through **LM** up to $\mathbf{O}^*$ is the output space representation of the local minimum's basin of attraction.

ERA starts at $\mathbf{O}_1$ and aims to minimise error with respect to the subgoal $\mathbf{SG}_2$. The only directions that are realisable from $\mathbf{O}_1$ are along **RC** either towards $\mathbf{O}^*$ and **GM** or towards **LM**. As can be seen from the error contour lines, travel towards **GM** represents an increase in error with respect to the goal and any subgoal on the linear subgoal chain **SGC**. Travel towards **LM** on the other hand represents a decrease in error with respect to the goal and any subgoal on **SGC**.

Performing BP on the subgoal $\mathbf{SG_2}$ will aim to reduce the error with respect to $\mathbf{SG_2}$ and will therefore ensure travel to $\mathbf{O_2}$ which lies on the way to the local minimum **LM** and is the position on **RC** at which **RC** is tangential to the error contour surrounding $\mathbf{SG_2}$. Subsequent training on subgoals further along the chain will ensure travel continues towards the local minimum and when **G** is set as a subgoal training will converge on the local minimum **LM**.

Viewed in output space BP will converge to the local minimum for the goal when its initial output state is on the side of **RC** from $\mathbf{O^*}$ which leads to **LM**. In other words BP converges to the local minimum for the goal if its initial output state is on the output space representation of the local minimum's basin of attraction.

Although ERA sets a subgoal chain the basis for its success or failure is much the same as for standard unchained BP. That is ERA will converge to the local minimum when its mean-valued output state is in the output space representation of the local minimum's basin of attraction.

If on the other hand the training problem were different such that ERA's mean-valued output state were on the other side of **RC** towards **GM** then ERA would have had 100% success.

Success or failure for ERA depends on whether the mean-valued output state is in the output space representation of the global minimum's attractor basin or not. For single layer networks there is only one weight state which produces the mean-valued output state for its 1st subgoal. This means that ERA effectively starts at the same state regardless of its random initialisation. That is the reason for its extreme 0% success rate for the training

problems shown here. Linear subgoal chaining without the mean-value starting condition and initialised randomly may be expected to have a more variable success rate depending on the basin distribution for a problem. ERA's success rate may in general be expected to be the same as for standard BP using the mean-value weight state as its initial state.

Chapter 6

Non-linear Subgoal Chaining

Sections 4.3 and 4.4 showed how linear subgoal chaining in general and in particular as used by ERA can lead to getting trapped in local minima just as standard gradient descent on fixed error-weight travel surfaces does.

To reiterate, all travel paths which escape from local minima must initially travel towards higher LMS error with respect to the goal before being able to decrease the error again when travelling towards the global minimum. Viewed in output space the only corresponding paths which are able to escape from local minima in a continuous run and return to the global minimum are curved and initially travel away from the goal before returning towards the goal. Specifically, the only direction in which travel is possible when starting at a local minimum is initially at 90 degrees to the goal direction, or in other words tangential to a hypersphere centred on the goal and containing the current local minimum state. This is shown by (5.22) in section 5.2 and depicted in Figure 5.6 in the same section.

Linear subgoal chaining including the one used by ERA sets subgoals in a straight line towards the goal. Such a method therefore insists with each subgoal setting that progress be made towards the goal which is impossible when situated at a local minimum for that goal. This means, as stated above and explained in more detail in sections 4.3 and 4.4, that linear subgoal chaining can get trapped in local minima just as standard BP can. The empirical evidence for ERA failing to reach the global minimum when encountering a local minimum is given in section 5.3.

This observation prompted the idea to investigate methods which should not suffer from getting stuck at local minima in the same way as standard gradient descent or ERA do. One such candidate method was non-linear subgoal chaining and so the idea was to investigate non-linear subgoal chaining with respect to how it may be able to escape local minima.

The basis for the method is to start with a linear subgoal chain which should promise direct success for finding solution paths to problems which do not suffer from local minimum interference. A greater degree of freedom is then given to the travel by allowing the subgoal chain to shape itself to a non-linear shape as training proceeds. The ability to allow higher flexibility in travel and to temporarily go away from the goal in a directed fashion promises success in terms of reaching the global minimum for problems with potential local minimum interference. Starting with a linear chain but allowing the chain to shape itself suggests a greater *penetration* to the optimal problem solution, i.e. higher success rates, compared to standard gradient based techniques.

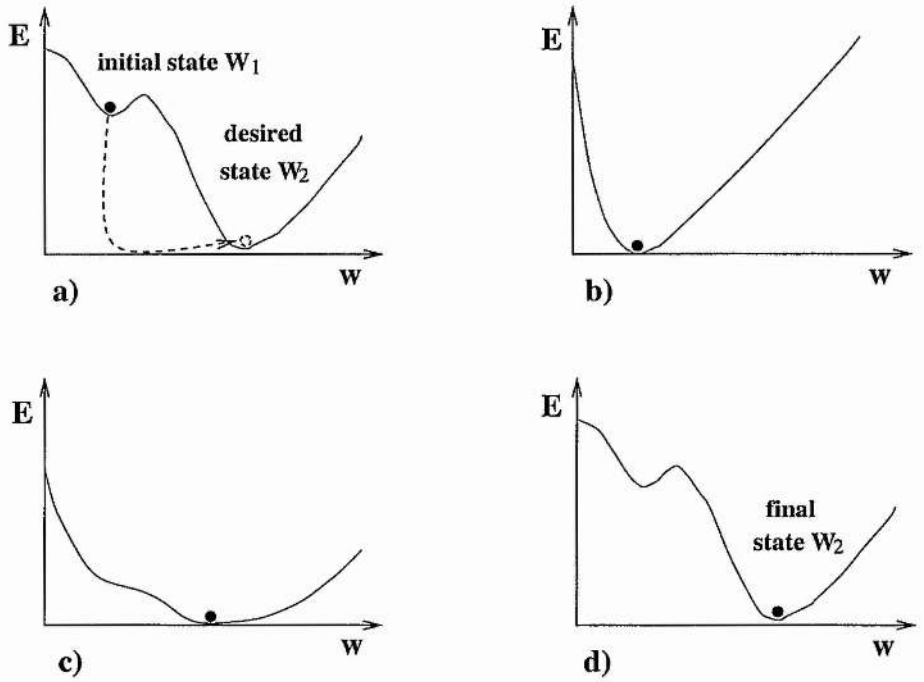


Figure 6.1: Varying the error-weight travel surface for BP. **(a)** and **(d)** symbolise the goal's error-weight surface, **(b)**, **(c)** show intermediate subgoal variations of the goal's error-weight surface.

6.1 Non-linear Subgoal Chaining with Cheat Chains

Before starting to develop a technique for non-linear subgoal chaining it was important to test how useful non-linear subgoal chains could be. If non-linear paths are made attractive for a travel technique by setting subgoals along such paths, convergence to the global minimum should always be possible if the shape of the chain is right. This should occur even when using BP to train on the individual subgoals. The use of subgoals removes the fixed error-weight surface for BP and should allow successful training to take place.

Figure 6.1 indicates how an error-weight surface would ideally have to be varied to achieve successful learning. Figures 6.1**(a)** and 6.1**(d)** show what the goal's error-weight surface

may look like, with a local minimum as the current weight state and a global minimum as the desired state. By setting the current output state to be a subgoal the error-weight surface is immediately changed to reflect this by placing the current state at zero error in error-weight space, as depicted in Figure 6.1**(b)**. By setting intermediate subgoals between the current output state and the goal state it may be possible to vary the error-weight surface such as depicted in Figures 6.1**(b)** and 6.1**(c)**, so that learning is directed towards the global minimum for the goal and will converge to the same when the error-weight surface is that of the goal again. This was a hypothesis to be tested.

One way to test this is to use training problems for which the positions of the local minimum and global minimum are known in weight space and using this information to generate subgoal chains which should lead to successful training.

The training examples used for this are for a single layer 2-1 network and were developed to show ERA failing to reach the global minimum. In fact ERA's success rate on the following training examples is 0% as shown in Table 5.1 (page 129 in section 5.3). The local and global minimum weight states were established through both convergence tests and by examining the Hessian. The method for creating the examples and testing whether local minima exist was described in section 5.1. In the following I will describe weight states for a 2-1 network as follows. A weight state is the vector containing the bias weight, the weight connected to the in_1 input of the training set and the weight connected to the in_2 input, such that the weight state can be written as the row vector $\mathbf{w} = (w_{\text{bias}}, w_1, w_2)$. Two training examples follow.

Figure 5.1 on page 106 shows the first training example. The local minimum $\mathbf{w}_{LM}$ for this test problem is found to be at the zero-valued weight state, i.e. $\mathbf{w}_{LM} = (0, 0, 0)$. The global minimum $\mathbf{w}_{GM}$ is found to be near the weight state $(1.38436,\ 0.00000,\ -2.76873)$ which corresponds to a linear separation near the line **H1** in Figure 5.1.

Figure 5.4 on page 110 shows the second training example. The local minimum $\mathbf{w}_{LM}$ for this test problem is found to be near the weight state $(0.02566,\ 0.17897,\ -0.18043)$, which corresponds to a linear separation near the line **H1**. The global minimum $\mathbf{w}_{GM}$ is found to be near the weight state $(0.03955,\ 1.16845,\ -0.23373)$, which corresponds to a linear separation near the line **H2**.

6.1.1 Experimental Results with Cheat Chains

Both test problems described above were used to test both standard BP and BP using subgoals from 100 random initial weight states in the range [-1, +1].

For the same reasons as explained in section 5.3 the learning rate was set to be very low, with no momentum being used, in order to encourage training to follow the shape of the error-weight surfaces for each subgoal as closely as possible. To reiterate, this was in order to compare the success obtained with and without the use of the subgoal chains as accurately as possible by ruling out successful training caused by high learning rates or the use of momentum which may benefit one technique more than the other.

A tolerance for determining successful training on the goal targets was set for each test problem in order to recognise sufficient convergence to the global minimum state. The

goal tolerance was set in terms of LMS error as defined in (2.3) on page 23 in section 2.1. The goal tolerances to recognise successful learning were 0.32 and 0.027 for test problems 1 and 2 respectively. If sufficient convergence to the global minimum was detected, training was terminated for that run and a success was recorded. A generous timeout allowing in the order of 10^6 cycles of BP training was set after which training that was unsuccessful up until that point was terminated.

For BP using subgoal chaining each chain was comprised of 10 subgoals. Subgoal chains were constructed for learning trials which lead to the global minimum, by using the knowledge of the global minimum weight state. Specifically 10 weight states equally spaced from the random initial weight states to the global minimum state were converted to neural output states which were used as subgoals in the *cheat subgoal chains*. This means that the subgoal chains always start at the initial random output state for each trial run and end at the outputs corresponding to the global minimum for the test problem and so form a curve of *realisable* and *progressive* subgoals leading to the global minimum. In this case all the subgoals are realisable because they are being produced by weight states. The subgoals are also optimally progressive because they lead directly towards the global minimum output state.

As a reminder, ERA as described in section 5.3 has 0% success on these two test problems because its effective starting state, the mean-valued state, is the local minimum for test problem 1, and for test problem 2 the mean-valued state lies in the local minimum's basin of attraction. In both cases ERA will always converge to the local minimum for the goal.

Test Problem	Training Method	Learning Rate	% finding Global	Average Cycles for Success
1	BP	0.1	11	940.82
1	CBP	0.1	100	910.29
2	BP	0.1	27	215.56
2	CBP	0.1	100	277.25

Table 6.1: Table of results, where BP refers to standard back-propagation and CBP refers to back-propagation using cheat subgoal chains. Test problem 1 and 2 refer to the training sets displayed in Figures 5.1 and 5.4 respectively.

Table 6.1 shows the results for standard back-propagation (BP) without subgoals and for BP using cheat subgoal chains (CBP). Standard BP converges to the global minimum for 11% and 27% of the random initialisations for test problem 1 and 2 respectively. Standard BP is not constrained to start at ERA's mean-valued state and so has a higher success than ERA on these two test problems. With cheat chains however, 100% success was obtained for both test problems when started at random initialisations, including when being started at the local minimum for each problem respectively. This is a major improvement on the standard unchained method and ERA. In terms of training cycles the non-linear chains perform similarly to standard training as can be seen from the average cycles needed for success.

Increasing the number of subgoals to 100 did not change the success rate for CBP but increased the average number of cycles for success. Using less subgoals than 10 decreased

the average number of cycles needed for success but did not affect the success rate. The variation in the average number of cycles needed was roughly $\pm 10\%$ on the results displayed in Table 6.1.

The high success rates lead to the conclusion that realisable output states along non-linear chains set as subgoals can be very successful in terms of guiding training towards the global minimum. This means that non-linear subgoal chaining can be very useful in overcoming local minima and based on the results from using cheat chains without causing any substantial loss in training speed. The question to be addressed then is how to find such non-linear chains which allow successful training.

6.2 The ZIP-model

The *ZIP-model* is a method for allowing subgoal chains to adapt during training and to organise themselves into a non-linear shape which resembles a successful cheat chain and so allows travel to the global minimum for the goal. Initially the process will start with a linear subgoal chain from the initial output state to the goal state, but will be allowed to shape itself as training proceeds through the method being described here. In the following, the ZIP-model will be introduced as a method for finding the desired shape of successful subgoal chains.

Figure 6.2 is a sketch of output space with the realisable manifold in this 2-D figure being denoted by **RM**. The only realisable states in output space are on the realisable manifold

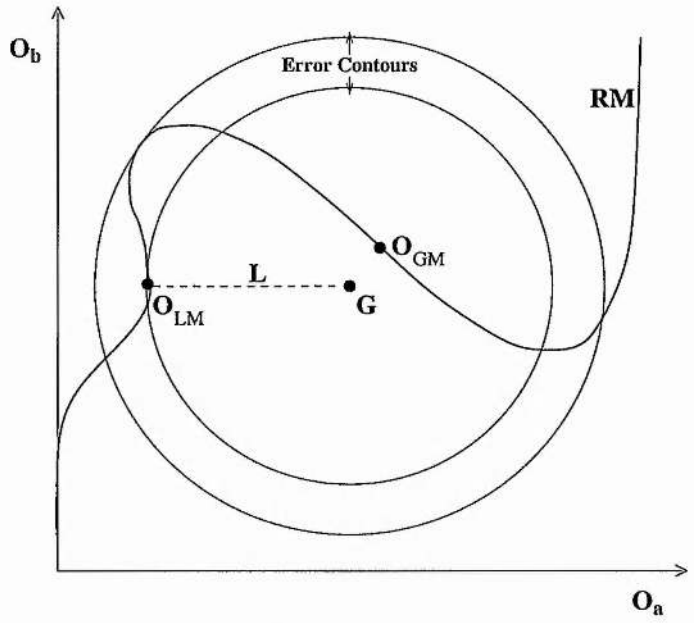


Figure 6.2: Output space with a local minimum output state $\mathbf{O}_{\mathrm{LM}}$ and the global minimum output state $\mathbf{O}_{\mathrm{GM}}$. The two concentric circles denote error contour lines with respect to the goal at **G**. The curve **RM** denotes the realisable manifold in output space that makes $\mathbf{O}_{\mathrm{LM}}$ a local minimum for the goal at **G**.

RM. The shape of **RM** is not constrained to have monotonically changing outputs here, as the monotonicity argument given in section 5.2 was only with respect to single layer networks. For multi-layer networks the realisable manifold may be more complex. The shape of **RM** creates a local minimum at $\mathbf{O}_{\mathrm{LM}}$ and the global minimum at the output state $\mathbf{O}_{\mathrm{GM}}$. The initial state in this case is at the local minimum $\mathbf{O}_{\mathrm{LM}}$ and so **L** denotes the initial linear subgoal chain.

The method for self-organisation of the subgoal chain is to find at each stage a realisable output state approximating the next subgoal as closely as possible at the same distance away from the current output state. Figure 6.3 shows $\mathbf{O}_{t+1}$ on the realisable manifold **RM** as the best realisable approximation to the subgoal $\mathbf{SG}_{t+1}$ and on the hypersphere surrounding

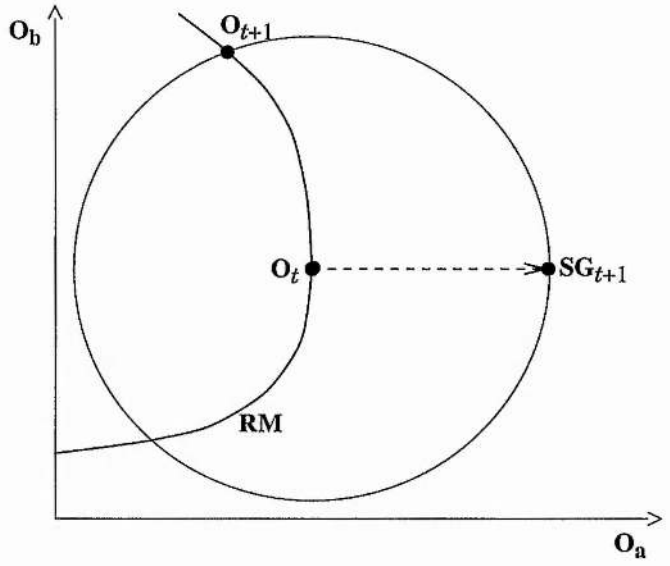


Figure 6.3: Finding a realisable state approximating the next subgoal at the same distance from the current state as the subgoal state.

$\mathbf{O}_t$. There are two reasons for attempting to find the closest realisable output state to the subgoal at the same distance as the subgoal. The first is that travelling to the hypersphere surface keeps the process moving if the current state should be a local minimum for the Goal for instance. The second is that attempting to find the closest realisable state to the subgoal minimises chain expansion which ensures that good progress is made.

After approximating the subgoal as closely as possible the last subgoal attempted is reset to be the last realised state, i.e. $\mathbf{SG}_{t+1}$ is set to be equal to $\mathbf{O}_{t+1}$ or in other words $\mathbf{SG}_{t+1}$ is *zipped* to $\mathbf{O}_{t+1}$. The sequence of such realised output states is then allowed to shape the chain.

Specifically, a cubic spline is passed through the output state sequence, consisting of all output states realised so far, and the goal, to form a non-linear approximation of the remaining path to be attempted. This approximation is taken to be the subgoal chain. Figure 6.4 shows

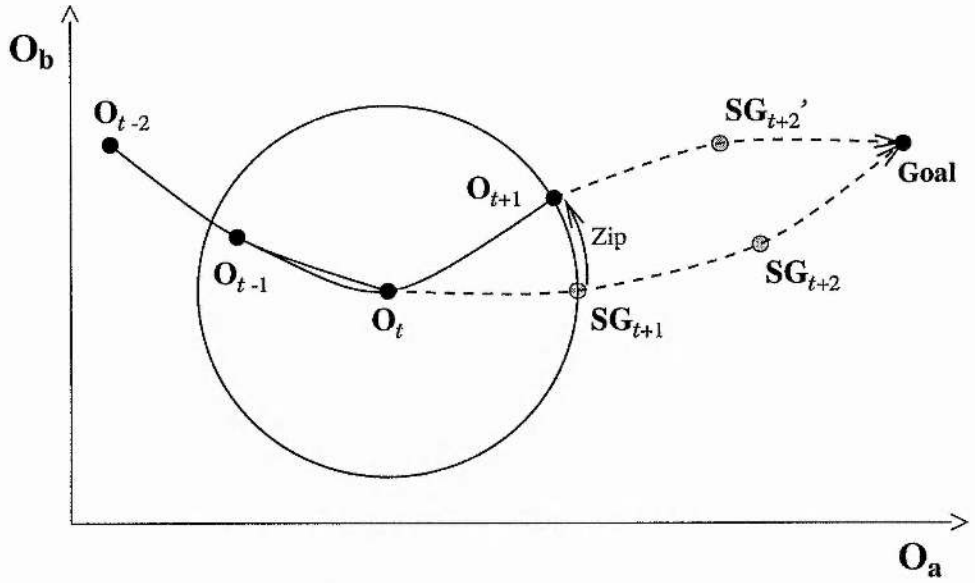


Figure 6.4: Output space with $\mathbf{O}_t$ as the output state at time t and $\mathbf{SG}_{t+1}$ as the estimated position of the subgoal to be realised by the end of time step t. $\mathbf{O}_{t+1}$ denotes the output state actually realised at the end of time step t. The dashed lines indicate the spline's approximation of the remaining path to be attempted.

how the spline is set through all output states realised up to time step t and how the spline then determines the position of the next subgoal $\mathbf{SG}_{t+1}$ to attempt and all future subgoals to be attempted such as $\mathbf{SG}_{t+2}$. As training proceeds and the subgoal chain is re-shaped the subgoals to be attempted succeeding the current subgoal are re-shaped along this subgoal chain as well such that the estimated position of the subgoal $\mathbf{SG}_{t+2}$ at time t changes to be the subgoal $\mathbf{SG}_{t+2}'$ at time $t+1$.

Such a subgoal chain can in theory set subgoals outside of the achievable bounds of 0 and 1 for output values due the spline forming its interpolation in output space. In order to avoid this the spline actually forms its interpolation based on the excitation values corresponding to the output values for each pattern. This does not change the fact that at each stage the

subgoal chain passes through all output states realised so far and the goal, the only difference is that the shape of the spline is constrained to stay within the realisable boundaries between 0 and 1 in output space.

As the output state sequence develops, the spline may form an increasingly accurate interpolation of a realisable path to the global minimum compared to the original linear subgoal chain. This accuracy may be gained because, at each stage, the training process aims for the most realisable and progressive states on an adaptive curved subgoal chain leading to the goal. The idea is that the remaining subgoal chain and the actually realised path, as well as the last output and subgoal, get zipped together as training proceeds, hence the name of the model.

Figure 6.5 shows a possible scenario where the first output state is a local minimum. Initially the subgoal chain forms a straight line from $\mathbf{O}_0$ towards the goal along which there are 8 equally spaced subgoals and the goal. For this case any output states realised when starting at the local minimum must initially lead tangentially away from the goal before being able to return to the goal. A further design feature of the method is that the distance between the current output state and the next subgoal to be attempted is not allowed to exceed the spacing between subgoals on the initial linear chain. This ensures that subgoals remain relatively close to the current state in order to facilitate training towards them. As depicted in Figure 6.5 a training run may start out with 8 subgoals on the linear chain for instance but may use more subgoals over the whole run due to chain expansion.

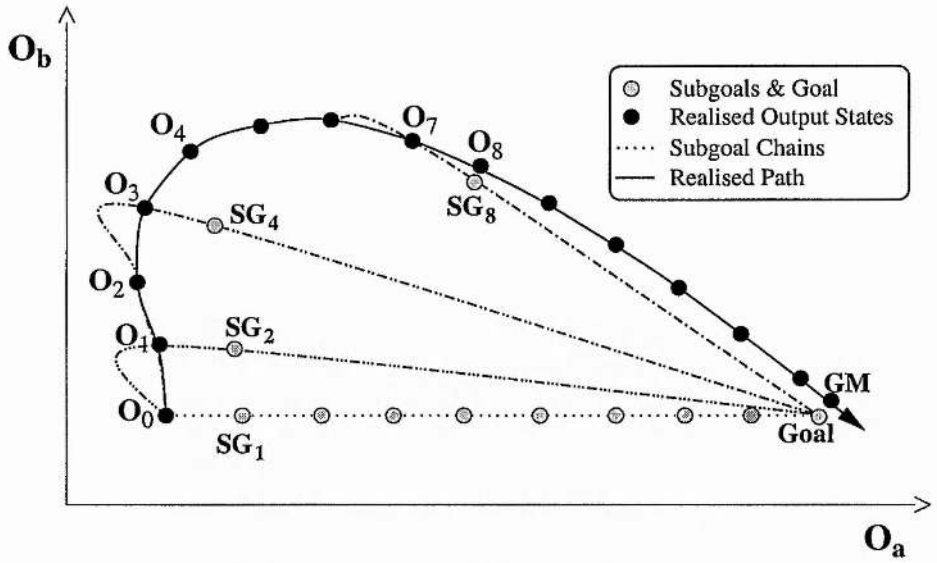


Figure 6.5: Output space depicting a scenario for non-linear subgoal chain directing learning out of the local minimum state $\mathbf{O}_0$ to the global minimum at **GM**. At each step t a spline is put through all output states up to $\mathbf{O}_t$ and the goal output state. This produces an estimate of the remaining path to be attempted. The next subgoal to be attempted is placed along the newly updated path near to the current output.

6.3 Realisable and Progressive Directions

The remaining key component of the ZIP-model to be explained is how to obtain a realisable output state from the current state which is also as progressive as possible w.r.t. the current subgoal. There are undoubtedly many ways to do this, but output-weight derivatives are a simple method to obtain directions in output space which are realisable from the current state.

The idea behind output-weight derivatives is that a change in each of the neural weights will effect a change in output state, which can be estimated by calculating the derivative of the output function with respect to each weight. The output state is made up from the

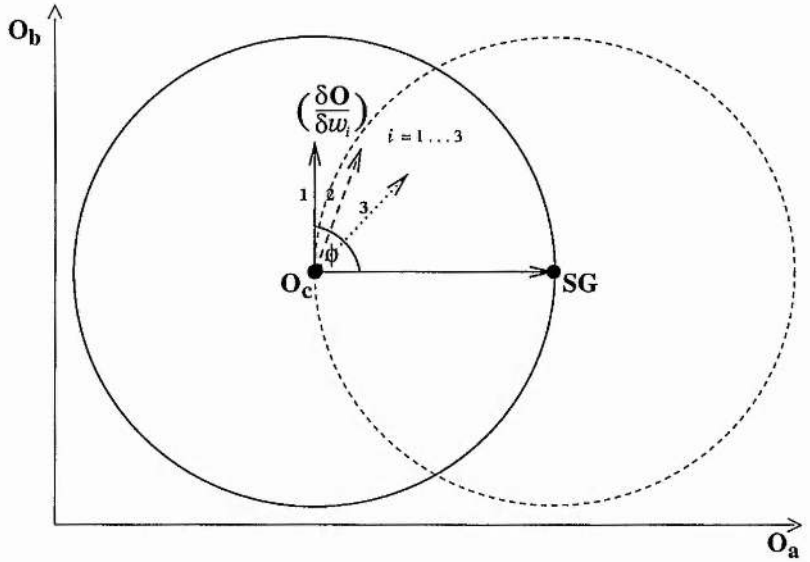


Figure 6.6: A view of output space with $\mathbf{O}_c$ as the current output state and **SG** as the current subgoal being attempted. $\left(\frac{\partial \mathbf{O}}{\partial w_i}\right)$ represents output-weight derivatives for the three neural weights 1, 2 and 3. These derivatives indicate realisable directions in output space. ϕ for example denotes the angle between $\left(\frac{\partial \mathbf{O}}{\partial w_1}\right)$ and the direction towards **SG**.

outputs obtained from each training input pattern fed through the network with a particular weight state. It can be written as a vector of outputs for each training pattern such that the state $\mathbf{O} = (O_1, O_2, \ldots, O_P)$, where P is the number of training patterns. Each of the outputs O_p for pattern p is a function of the training set inputs for pattern p and the weight state and the output state is a vector function of the same. One may then define an output-weight derivative for each neural weight w_i as the derivative of the output vector function with respect to that weight as follows

$$\left(\frac{\partial \mathbf{O}}{\partial w_i}\right)_{\mathbf{W}=\mathbf{W}_c} = \left(\frac{\partial O_1}{\partial w_i}, \frac{\partial O_2}{\partial w_i}, \ldots, \frac{\partial O_P}{\partial w_i}\right) \tag{6.1}$$

where the derivative is evaluated at the current weight state $\mathbf{w}_c$ which produces the current output state $\mathbf{O}_c$.

In this way, each neural weight can be associated with an expected direction of change in output state. Figure 6.6 shows the example for a 2-1 network with bias unit which has 3 neural weights and so 3 output-weight derivatives in output space.

Some of the realisable directions will be more progressive than others, i.e. they will pass closer to **SG** than others in terms of the angle they form with the subgoal direction. This angle can be used to grade how progressive each weight change is in terms of moving the output state towards the subgoal. The grading can be used to establish a combination of individual weight changes whose associated output change ends close to **SG** by favouring weight changes which promise to lead the output close to the subgoal.

Specifically the method assigns a fitness value to each individual weight depending on how closely the direction of its associated output-weight derivative matches the direction towards **SG**. In this implementation, fitness values range between 0 and 1. A weight w_1 for example is given a fitness value of $\frac{1+\cos(\phi)}{2}$ where ϕ is the angle between $\left(\frac{\partial \mathbf{O}}{\partial w_1}\right)$ and the direction towards **SG**. The vector containing the fitness values for each weight describes a direction in weight space. The weight state is updated along this direction until a state on the hypersphere surface in output space is realised. The weights with greatest fitness will therefore be given the largest changes, so that the state realised is close to **SG**.

The output-weight derivatives were found to provide a fairly accurate realisable direction locally for each weight individually. This was tested for single layer 2-1 networks by updating individual weights until the hypersphere surrounding the current state was reached. It was clear though, that a single weight update can promise to achieve the subgoal or the

closest realisable state to the subgoal on the hypersphere surface in only very few cases. These cases are when the desired state in weight space lies along the weight axes from the current weight state. For the general case of wanting to find the closest realisable state to the subgoal a combined weight update is needed and was therefore adopted.

It should be noted that the heuristic for the combined weight update by changing the weight state in the direction specified by the fitness values is a weak one in terms of not guaranteeing to find the closest realisable state to the subgoal. Although it may be a rather crude heuristic for finding a realisable state close to **SG** on the hypersphere surface it was nevertheless found to work well in practice for simple problems and network architectures.

6.4 Experimental Results

For testing the ZIP-model with output-weight derivatives the same test problems were used as for testing the cheat subgoal chains as described in sections 6.1 and 6.1.1, i.e. test problems 1 and 2 corresponding to the training sets depicted in Figures 5.1 and 5.4 respectively. The important difference to the previous experiments worth mentioning is that the knowledge of the global minimum weight state is not being used here to create successful subgoal chains.

In accord with the experiments in sections 5.3 and 6.1.1 for ERA, standard BP and cheat chains, non-linear chaining without the cheat was started at 100 random initialisations where each weight was given a value in the range [-1, +1]. Non-linear chaining was run

Test Problem	ERA % Success	BP % Success	OW % Success
1	0	11	63
2	0	27	94

Table 6.2: Table of results for non-linear subgoal chaining using output-weight derivatives. OW refers to output-weight derivatives and BP to standard back-propagation and ERA to expanded range approximation. Test problem 1 and 2 refer to the training sets displayed in Figures 5.1 and 5.4 respectively.

with 20 subgoals initially set in a straight line towards the goal from each random initial output state. Unlike linear subgoal chaining the number of subgoals used over the runs may vary here as the technique roughly maintains the same Euclidean distance between successive subgoals on the curved chain as the distance was between subgoals on the initial linear chain.

A generous time out was set for the non-linear subgoal chaining technique depending on the problem such that further training did not produce any increase in success. This was in order to compare the full percentages of success over the various runs. Tolerances with respect to the goal targets were set for each problem as quoted before in section 6.1.1 in order to recognise sufficient convergence to the global minimum.

Table 6.2 shows the results for non-linear subgoal chaining using output-weight derivatives and for ease of comparison reiterates the success rates for standard BP and ERA as quoted in sections 5.3 and 6.1.1 for the same problems. The results confirm that non-linear sub-

goal chaining can have higher *penetration to solution*, i.e. achieve a significantly higher success rate in reaching the global minimum state than standard training and ERA can. In addition non-linear subgoal chaining has been shown to converge to the global minimum weight state when started at the local minimum weight state for both of these test problems, something which standard BP and ERA completely fail to achieve.

6.5 Summary

It has been shown that non-linear subgoal chaining can be a feasible approach for overcoming the problems encountered when travelling through gradient descent.

A greater penetration to solution from random initialisations compared to standard training and ERA is observed. Although a slowness of convergence to local attractors with respect to BP is also observed, this is not surprising considering the inherent design of the non-linear chaining technique. That is non-linear chaining has been designed to be able to do the opposite of convergence to local attractors in order to have the chance of converging to the global attractor.

In particular it has become clear that the accuracy of the weight state transitions in aiming for the closest realisable output state to the subgoal output using output-weight derivatives has scope for improvement. Nonetheless non-linear chaining has been shown to be feasible in its approach and a promising way to overcome local minima.

Chapter 7

Hyperspherical TH

7.1 The Need for Better Direction

The use of non-linear subgoal chains is a powerful technique to extend the flexibility of neural training by allowing training to temporarily go away from the goal in order to return to the goal later on during the training process. Such flexibility in travel is necessary in order to escape a local minimum for the goal. The feasibility of using non-linear chains to direct training temporarily away from the goal to escape a local minimum was shown by using cheat subgoal chains as described in section 6.1. The ZIP-model, as described in section 6.2, is a method for allowing the subgoal chain to organise itself into a shape which will allow successful training to the global minimum. The model enables this self-organisation of the subgoal chain by attempting to *zip* the subgoal chain and the actually realised path in output space together. The idea is that this zipping should enable the

subgoal chain to become increasingly realisable as training proceeds.

One of the key components of the ZIP-model is a heuristic which aims to find a realisable state close to the subgoal on a hypersphere centred on the current state and through the subgoal. Output-weight derivatives (as described in section 6.3) formed the basis of such a heuristic, by relating each individual neural weight change to a change in the realised output state.

For each weight individually, the output-weight derivatives were found to provide an accurate prediction of a realisable direction in output space. However, a method which simply updates one single neural weight until the hypersphere in output space is reached will not in general be able to find the closest realisable state to the subgoal on the hypersphere surface (see section 6.3). A combined weight update mechanism is needed. The summative heuristic combining the weight changes along all weight axes is a weak one in carrying no guarantees to produce the closest realisable direction towards the subgoal. This is understandable due to the simple nature of the heuristic as was explained in section 6.3.

Consequently non-linear subgoal chaining using output-weight derivatives can fail even when the subgoal is realisable, due to the simplicity of the heuristic combining the weight updates.

Although the method was shown in fact to work well for simple training sets and simple architectures, as described in section 6.4, an improvement on the realisable direction taken to approximate the closest realisable direction to the subgoal direction was desirable. In the following, Hyperspherical TH is proposed as a method to provide accurate directions

for the combined weight update.

7.2 Hyperspherical TH: The Method

In the following, the method of Hyperspherical TH (HTH) is described which is intended to provide accurate directions for the combined weight update within the ZIP-model. Specifically HTH is intended to replace the part of the model which in its initial implementation used output-weight derivatives.

TH is a powerful technique to direct neural training as described in sections 4.1 and 4.2. As described there, TH approximates solution manifolds for each training pattern in weight space by hyperplanes tangent to them and then finds the closest approximation to a mutual intersection of these hyperplanes. If all the solution manifolds intersect then the weight state at the intersection is a global minimum with zero-valued error. For single layer networks, the solution manifolds are hyperplanes so that the approximation to the solution manifolds by hyperplanes is perfect and TH finds the intersection in one single application of SVD.

If there is no mutual intersection of the solution manifolds then TH becomes an iterative process if LMS (squared output) error is to be minimised (see section 4.1). Multiple iterations are generally needed for solution when dealing with multi-layer networks as well, due to the solution manifolds being non-linear.

TH was shown to consistently penetrate to optimal solution states in relatively few itera-

tions compared to standard BP. This indicates that TH's training direction obtained via SVD may be superior to BP's direction. This inspired the development of HTH as a method for improving the direction for the combined weight update in the ZIP-model.

The primary difference between HTH and TH lies in the search space which HTH examines compared to TH and the effect on training caused thereby. One of the design features for the output-weight derivative method was to avoid getting stuck by ensuring movement to a hypersphere surface surrounding the current output state. This feature is incorporated into HTH as well.

The idea behind HTH is to retain the ability of output-weight derivatives to avoid getting stuck while improving the search direction by introducing a TH like quality in terms of good direction and speed.

The initial design of HTH is to improve on output-weight derivatives in terms of being able find subgoals which are realisable, i.e. subgoals for which a weight state exists that produces the subgoal's outputs for all patterns.

7.2.1 Direction Angle Hyperplanes

In order to aim for a subgoal it has to be represented in some space. The desired subgoal output state can be expressed as an excitation state which comprises the subgoal excitations of output units for all patterns. For the initial implementation of the ZIP-model, subgoals were set in excitation space and a spline was passed through all currently achieved subgoals and the goal (see section 6.2). The representation of subgoals for the HTH implementation

is now altered slightly.

The subgoal can be interpreted as a desired change in excitation state with respect to the current excitation state. In this way the search is centred about the current excitation state and with respect to the subgoal excitation state. This is equivalent to shifting the origin of excitation space to the current excitation state. To denote this shift of the origin the space is called *delta excitation space.*

Figure 7.1 is a stylised view of delta excitation space for the output unit and 3 patterns. The state HTH is to find is the subgoal state $\mathbf{D} = (D_1, D_2, D_3)$. The search for the subgoal $\mathbf{D}$ can be re-interpreted as finding a weight change which primarily produces any state on the ray from the origin which passes through $\mathbf{D}$. The reason for this is that once a state on this ray has been found, a suitable scaling of the weight change on connections to the output unit will produce any state along this ray and including the subgoal state. This is because the change in excitation state is a linear function of the weight changes on connections to the output unit. The ray passing through the origin and intersecting the sphere surrounding the origin at the desired subgoal state $\mathbf{D}$, is called the *desired ray.*

The subgoal $\mathbf{D}$ can then be represented in terms of the radius r of the hypersphere containing the desired state and the direction angles γ_p for each pattern p between the desired ray and the coordinate axes Δex_p. The components of the subgoal can be written as $D_p = r \cos(\gamma_p)$, where p ranges from 1 to P with P denoting the number of patterns.

For each direction angle γ_p a hyperplane $\mathbf{H}_p$ can be defined in delta excitation space which forms that angle γ_p with the coordinate axis Δex_p and passes through the origin and $\mathbf{D}$.

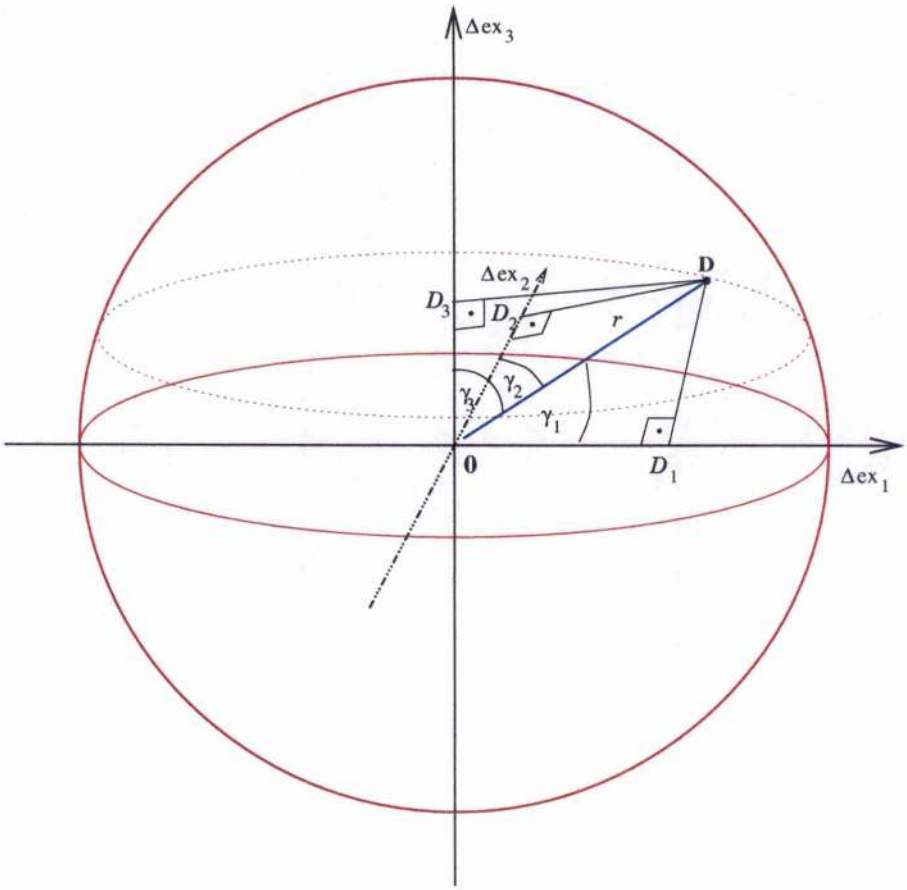


Figure 7.1: 3-D stylised view of delta excitation space with its axes Δex_p for patterns $p = 1 \ldots 3$. The desired ray passes through the origin and the desired subgoal state $\mathbf{D}$.

The hyperplanes' unique orientations with respect to the coordinate axes are defined by their normal vectors $\mathbf{C}_p$. These are obtained as vectors orthogonal to the desired ray and in the plane formed between the origin, the subgoal $\mathbf{D}$ and D_p on the Δex_p axes. After some geometry the components of these normal vectors $\mathbf{C}_p$ can be written as:

$$C_{pj} = \begin{cases} -D_p D_j & , \forall j \neq p \\ |\mathbf{D}|^2 - {D_p}^2 & , j = p \end{cases} \tag{7.1}$$

So we now have P hyperplanes $\mathbf{H}_p$ which all go through the origin and $\mathbf{D}$ and form the

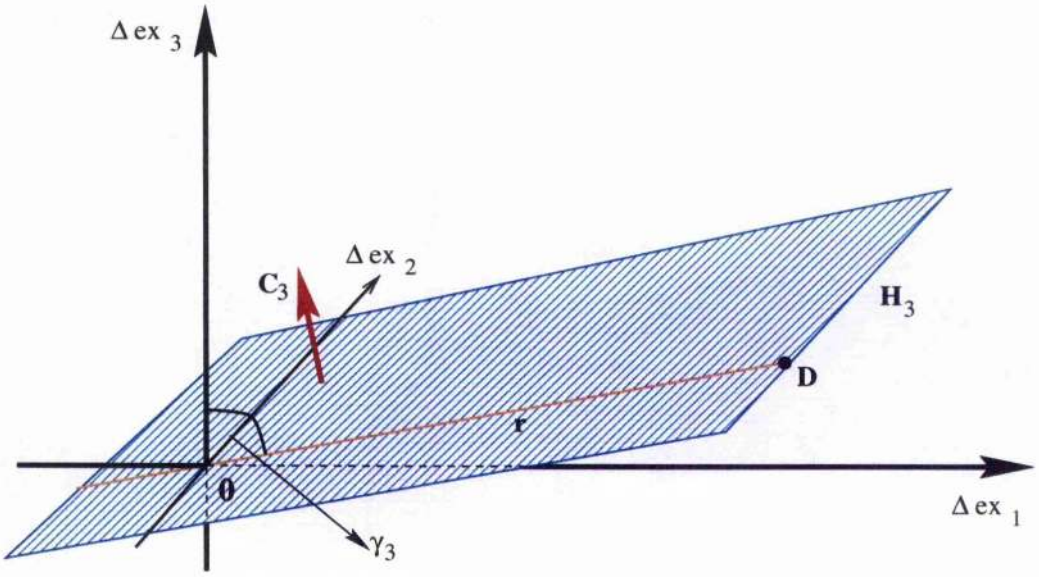


Figure 7.2: Stylised view of 3-D delta excitation-space with its axes Δex_p for patterns $p = 1 \ldots 3$ and an angle based hyperplane $\mathbf{H}_3$ containing the desired ray and subgoal state $\mathbf{D}$ at radius r from the origin. $\mathbf{C}_3$ is the normal to the hyperplane.

angle γ_p with the Δex_p axis. These hyperplanes $\mathbf{H}_p$ can be written as

$$\mathbf{C}_p \cdot \Delta \mathbf{ex} = 0 \tag{7.2}$$

or by expanding the scalar product in (7.2) as follows

$$\sum_{j=1}^{j=P} C_{pj} \, \Delta ex_j = 0 \tag{7.3}$$

where the Δex_j are the components of the vector $\Delta \mathbf{ex}$ for pattern j.

All of these hyperplanes contain the desired ray and their intersection is the desired ray which goes through the subgoal state. Figure 7.2 is a stylised view of such a hyperplane $\mathbf{H}_3$ for angle γ_3 in 3-D delta excitation space.

The objective is to find a change in weight state producing a change in excitation which lies on the desired ray. In other words the objective is to find weight changes which correspond

to changes in excitation which lie on the intersection of all the $\mathbf{H}_p$ hyperplanes.

7.2.2 Hyperplanes in Delta Weight Space

One way to establish the intersection of the hyperplanes is to establish a hyperplane expression for changes in weight state which produce states on the hyperplanes in delta excitation space and to then form an intersection of the weight space hyperplanes.

The use of delta excitation space brings with it the notion of delta weight space. Delta weight space is simply the translation of weight space to shift the current weight state to the origin of the space. The output unit's change in excitation for each pattern Δex_j can be expanded in terms of a neural weight change on connections to the output unit. For simplicity the equations below are for a single layer network with 1 output.

$$\Delta \text{ex}_j = \sum_{i=1}^{i=N} \Delta w_i \, x_{ij} + \Delta w_\beta \tag{7.4}$$

where N is the number of weights excluding the bias weight, Δw_i represents the change in the weight connecting from input i to the output unit and Δw_β is the change in the bias weight connecting to the output unit. Inserting this into (7.3) we now get the hyperplanes $\mathbf{H}_p$ defined as

$$\sum_{j=1}^{j=P} \left\{ C_{pj} \left(\sum_{i=1}^{i=N} \{ \Delta w_i \, x_{ij} \} + \Delta w_\beta \right) \right\} = 0 \tag{7.5}$$

for which we can interchange the order of the summation to obtain

$$\sum_{i=1}^{i=N} \left\{ \Delta w_i \sum_{j=1}^{j=P} \{ x_{ij} \, C_{pj} \} \right\} + \Delta w_\beta \sum_{j=1}^{j=P} C_{pj} = 0 \tag{7.6}$$

By grouping certain terms together one can rewrite (7.6) as

$$\sum_{i=1}^{i=N} \{\Delta w_i \, h_{pi}\} + \Delta w_\beta \, k_p = 0 \tag{7.7}$$

where

$$h_{pi} = \sum_{j=1}^{j=P} \{x_{ij} \, C_{pj}\} \tag{7.8}$$

and

$$k_p = \sum_{j=1}^{j=P} C_{pj} \tag{7.9}$$

From (7.7) it becomes clear that the hyperplanes $\mathbf{H}_p$ for each of the P angles γ_p pass through the origin of delta weight space. This is because the hyperplane equation (7.7) is satisfied for a zero-valued change in weight state.

So, by linking a change in excitation state to a change in weight state a hyperplanar expression in delta weight space can be found for each hyperplane in delta excitation space.

For a realisable subgoal a change in weight state exists which produces the desired change in excitation. Because the relation between weight change and excitation change is a linear one for single layer networks any change in excitation on the straight line formed between the origin of delta excitation space and $\mathbf{D}$ must be realisable as well. In other words the hyperplanes in delta weight space intersect in a line and this line of weight state changes corresponds to the straight line intersection of the hyperplanes in delta excitation space, i.e. the desired ray.

Every point on the desired ray, i.e. the intersection of the $\mathbf{H}_p$ hyperplanes, generally corresponds to a unique value along each weight change axis. Hence setting one of the weight

changes to a specific value will identify a unique point on the ray. Singular Value Decomposition (SVD) can be used to find this specific point on the intersection by setting the bias weight change to +1 for example. Specifically, setting $\Delta w_\beta = 1$ establishes a subspace of each of the hyperplanes $\mathbf{H}_p$ which in turn is a hyperplane $\mathbf{SH}_p$ in the subspace of delta weight space without the Δw_β dimension. SVD then finds the intersection point of the $\mathbf{SH}_p$ hyperplanes in the dimensionally reduced delta weight space at $\Delta w_\beta = 1$.

Using this procedure a change in weight state $\Delta\mathbf{w}$ is obtained. The weight change $\Delta\mathbf{w}$ corresponds to a change in excitation state $\Delta\mathbf{ex}$ which will either point towards the subgoal $\mathbf{D}$ on the desired ray or in the opposite direction away from it.

As mentioned above and according to (7.4), the change in excitation for an output unit is a linear function of the change in weight state for weights connecting to the output unit. If the resulting direction $\Delta\mathbf{ex}$ points away from $\mathbf{D}$ then the sign of $\Delta\mathbf{w}$ is negated, which corresponds to the negative change in excitation state $-\Delta\mathbf{ex}$ and will point towards $\mathbf{D}$.

After ensuring the excitation change is towards $\mathbf{D}$, its length is adjusted to be the radius of the hypersphere surface in order to realise $\mathbf{D}$ exactly. To do this the weight change $\Delta\mathbf{w}$ is scaled according to (7.10) such that

$$\Delta\mathbf{w}' = \frac{r}{|\Delta\mathbf{ex}|}\Delta\mathbf{w} \tag{7.10}$$

where $r = |\mathbf{D}|$. The weight state transition $\Delta\mathbf{w}'$ produces the desired subgoal value $\mathbf{D}$ exactly, if $\mathbf{D}$ is realisable.

7.3 Good Direction Not Enough

The implementation of this initial design was intended for finding the subgoal on the hypersphere surface exactly for realisable subgoals.

In order to assess whether the implementation of HTH was successful it was tested on a simple problem for a 2-1 network, for which all of output space was known to be realisable. The training set used was formed by removing the training point at (0,0) in input space from the training set for AND. The training set is depicted in Table 7.1.

This simple test training set provides a set of input points which are positioned such that a 2-1 network can always find a zero error global minimum regardless of the subgoal or goal targets. This means that for any subgoal every state on the desired ray through the origin of delta excitation space and through the subgoal **D** is realisable. For any chosen subgoal the hyperplanes $\mathbf{H}_p$ generated by HTH must all intersect to form a line in delta weight space which maps to the desired ray in delta excitation space.

In over 100 trials for this test problem from random weight states in the range [-1, +1], HTH was able to find the solution weight state in as many cycles as there were subgoals for each trial. That is HTH needed 1 cycle to realise each subgoal precisely in one shot for all the trials. This result showed that HTH's initial implementation was successful in terms of finding the subgoal on the hypersphere surface exactly for realisable subgoals.

HTH was also tested with cheat subgoal chains on problems which have a local minimum and a non-zero error global minimum. For these problems the goals are in unrealisable

Pattern	in_1	in_2	Target Output
1	0.0	1.0	0.2
2	1.0	0.0	0.2
3	1.0	1.0	0.8

Table 7.1: Test training set for HTH obtained by removing a point from the AND training set.

regions of output space. These problems are the same ones used to test the output-weight derivative method and are depicted in Figures 5.1 and 5.4 on page 106 and page 110 respectively. The cheat subgoal chains produce realisable subgoals for the problems. The results showed that HTH could attain each realisable subgoal in one shot and the global minimum was reached in 100% of trials.

The next stage was intended to deal with unrealisable subgoals by approximating the closest realisable ray to the desired ray. If a subgoal is realisable then all states on the desired ray through the origin and the subgoal are likewise realisable. As described before this is due to the linear relation between the change in weights to the output unit and a resulting change in the excitation state. Consequently, the desired ray on the hyperplanes in delta excitation space cannot be realisable for unrealisable subgoals because the ray contains this subgoal. That is, only subspaces of the hyperplanes in delta excitation space are realisable and the $\mathbf{H}_p$ hyperplanes will not intersect in the desired ray. This means in turn that the hyperplanes $\mathbf{SH}_p$ in the dimensionally reduced delta weight space at $\Delta w_\beta = 1$ will not have a mutual intersection for unrealisable subgoals.

Similar to how TH works, as described in section 4.1, it was decided to allow SVD to find the closest approximation to an intersection of the hyperplanes in delta weight space in order to find a ray which is close to the desired ray.

At an early stage in testing however, the results showed that oscillation in terms of output state occurred on successive cycles for unrealisable subgoals. This situation is depicted for a single unrealisable subgoal **SG** in Figure 7.3. The output state $\mathbf{o}_0$ is a local minimum and **GM** is the global minimum for the subgoal. For simplicity the figure explaining oscillation is in terms of a single subgoal which happens to be repeatedly set for each transition as the closest subgoal along the spline to the goal. However, oscillation is just as likely to occur when different subgoals are set.

When starting at $\mathbf{o}_0$ a state $\mathbf{o}_1$ on the hypersphere surface surrounding $\mathbf{o}_0$ and close to **SG** is obtained. Successive transitions produce the output states $\mathbf{o}_2$ and then $\mathbf{o}_3$, the latter reversing the direction of motion along **L2**. Successive transitions will oscillate between states close to $\mathbf{o}_2$ and $\mathbf{o}_3$ and **GM** is unlikely to be ever achieved.

It became clear that searching the hypersphere surface prevents the technique from getting stuck from one transition to the next, but can get the technique stuck over 2 transitions by introducing an oscillating behaviour into the search. In essence this is because the process gets stuck when no local progress towards the next subgoal can be made.

Tests with setting a *cheat goal* target, i.e. one which corresponds to the output state at the global minimum state, for the training sets displayed in Figures 5.1 and 5.4 showed that this made for successful training in 100% of all trials. This lead to the conclusion that cheat

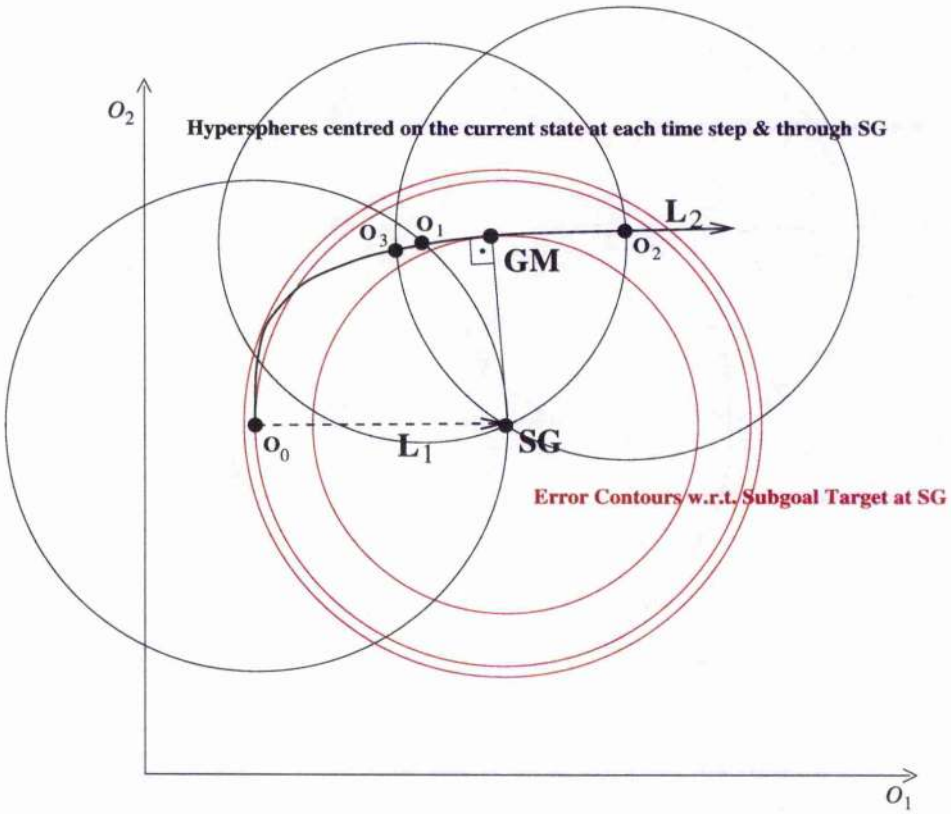


Figure 7.3: Stylised view of output space with axes O_1 and O_2. The line **L1** denotes an unrealisable path while **L2** denotes the only realisable path in output space. The global minimum for the subgoal **SG** is at **GM** at which point **L2** is tangent to the error contour surrounding the subgoal. Each successive transition is to the intersection of the realisable path with the hypersphere centred on the current state. Transitions go backwards and forwards along **L2** thus generating oscillation along **L2**.

goals induced realisable subgoals which lead to successful training. Consequently it was decided that subgoals and including the goal, which are set on the subgoal chain, should be realisable or nearly realisable. In turn the above realisation led to the design of a technique where the subgoal chain is projected along realisable directions.

Chapter 8

The Global Positioning System

Sections 4.4 and 5.3 showed how a linear subgoal chaining technique such as ERA must fail to reach the global minimum when encountering local minima because the reasons for success and failure are the same for ERA as for standard unchained training. That is, when within the basin of attraction of a local minimum for the goal it will converge towards it and be unable to penetrate to the global minimum. The reason for this is that it makes use of a linear subgoal chain which always requires progress to be made towards the goal. These subgoals are set along the chain regardless of whether they are realisable or not. If indeed subgoals along the linear chain are unrealisable, progress towards them is very limited and in the worst case no progress is possible. Essentially ERA's training always converges to the attractor of the basin it finds itself in when linear subgoal chaining commences. This basin can be either the local minimum basin or the global minimum basin. Which basin ERA will converge to depends on within which basin ERA's mean valued weight state, its

effective starting weight state, is contained.

Section 5.2 showed when unrealisable regions are to be expected in output space. Furthermore it argued that, for single layer networks at least, when local minima are present the goal is unrealisable.

Chapter 6 showed how allowing the subgoal chain to be more flexible can aid training. By enabling the subgoal chain to shape itself on the basis of progress and realisability criteria during training the travel can be directed to temporarily go away from the goal. This feature prevents training from getting stuck and may enable it overcome local minima in order to reach the global minimum. The ZIP-model was developed as a means for allowing this self organisation of the subgoal chain to take place. The heuristic devised for finding realisable and progressive output states was based on output-weight derivatives. It was found to be weak in terms of not guaranteeing to find the closest realisable state to the desired subgoal.

HTH (see chapter 7) was developed as a heuristic to replace output-weight derivatives in the ZIP-model and was successful in terms of finding subgoals which were realisable. As described in section 7.3 it became clear however that aiming for unrealisable subgoals introduced an oscillating behaviour into the training process regardless of the precision with which subgoals were being targeted. This oscillation was shown to be able to prevent training from reaching the global minimum.

In contrast, HTH's training results using cheat goal targets lead to the conclusion that cheat goals induced realisable subgoals which prevented oscillation and in turn lead to successful training. In fact for the tests performed with single layer networks, weight states producing

realisable goals were established in one single shot.

It can be shown for single layer networks that if a realisable goal is set no local minima will exist for that goal. This is due to the fact that travelling from any current weight state $\mathbf{w}_c$ to the zero error global minimum weight state $\mathbf{w}_{GM}$ in a straight line corresponds to a straight line path in excitation space, due to the linear relation between weight change and excitation change for single layer networks. Along this straight line excitation path all patterns' excitations move towards their goal excitations. Consequently all patterns' outputs must also move towards the goal, i.e. error must monotonically decrease. Such a path of monotonically decreasing error must always exist for single layer networks if the goal is realisable. Local minima can by definition not exist along such a path as they represent states from which any travel direction results in increases in error locally with respect to the goal.

Although the above reasoning without alteration may not extend to multi-layer networks, it has been found empirically that if the goal is nearly realisable, i.e. the global minimum output corresponds to almost zero error, it can generally be found using a powerful training technique such as TH. For instance TH was able to obtain 100% success on the 2-spirals problem (see section 4.1) with a single hidden layer network topology to produce the correct classification and low error at the global minimum.

The motivation then for developing a global positioning system (GPS) is to make use of the above by calculating an estimate for the global minimum output state. This is to allow training to take place and subgoal chains to be developed with respect to the nearly realis-

able estimate of the global minimum output state and thereby overcome local minima for the goal.

The knowledge of the global minimum output state allows the optimal state to be recognised during training and training to be terminated when it is optimal. If GPS's estimate for the global minimum output state is sufficiently close to the actual global minimum output state then the hope is that training can be directed to and terminated at a state close enough to the actual global minimum such that additional training on the original goal will cause convergence to the actual global minimum state.

8.1 Realisable Manifolds in Excitation Space

For a specific network topology and a training set there will be a realisable manifold in excitation space which contains the set of excitation states which are realisable, i.e. excitation states which are producible by at least one neural weight state.

It will now be shown that the dimensionality of the realisable manifold in excitation space can be established. In the treatment of realisable manifolds and their mapping to output space all equations presented here will only deal with single layer networks. As an added simplification they will only deal with single layer networks with one output unit, but all concepts presented here are readily extendable to single layer networks with multiple output units. The equivalent representations for multi-layer networks will be discussed in chapter 9.

For an output unit in a neural network, each pattern has a manifold of weight states which produce zero excitation for this pattern. For a single layer network the weight states on this manifold for training pattern p satisfy the following equation

$$\mathrm{ex}_p = \mathbf{w} \cdot \mathbf{in}_p = \sum_{i=1}^{N} w_i \, \mathrm{in}_{pi} = 0 \tag{8.1}$$

where ex_p is the excitation for pattern p, $\mathbf{w}$ is the neural weight state including the bias weight, $\mathbf{in}_p$ is the vector containing the training inputs for pattern p including the bias unit's input of 1 and N is the number of weights leading to the output unit including the bias weight.

One can see from (8.1) that the manifold of weight states producing zero excitation for a pattern p is a hyperplane through the origin of weight space defined by its normal vector in the direction of the input vector $\mathbf{in}_p$. I will refer to these hyperplanes as zero-excitation hyperplanes and denote them as $\mathbf{H}_p$. In order to reflect the property of the $\mathbf{in}_p$ vectors being normal vectors to the hyperplanes, they will be referred to as $\mathbf{n}_p$.

This situation is depicted for 2 neural weights and 3 patterns in Figure 8.1. Because the depicted weight space is 2-dimensional it is spanned by 2 linearly independent vectors. A set of vectors which spans a space is referred to as a *basis* for this space. Any vector in 2-D weight space can be expressed in terms of the basis vectors. Similarly for any N-dimensional vector space the basis consists of N linearly independent vectors.

In the 2-D case depicted in Figure 8.1, $\mathbf{n}_1$ and $\mathbf{n}_2$ are linearly independent of each other, i.e. $\mathbf{n}_2$ cannot be expressed solely in terms of $\mathbf{n}_1$. The vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ form a basis for weight space. This means that $\mathbf{n}_3$ can be expressed as a linear combination of the basis vectors $\mathbf{n}_1$

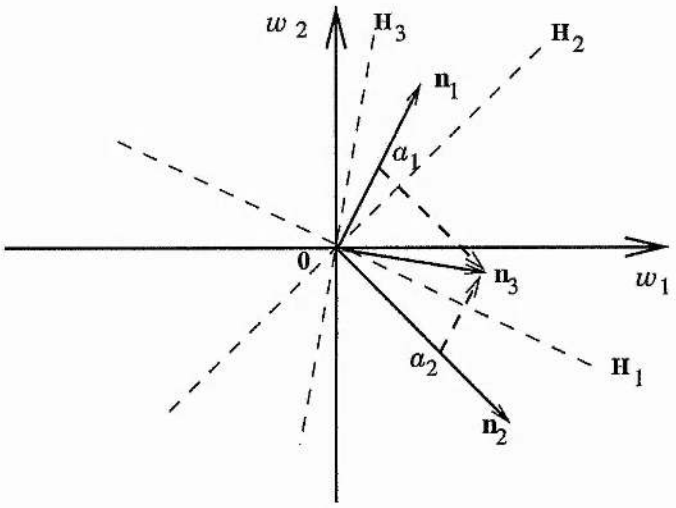


Figure 8.1: 2-D weight space with its axes w_1 and w_2 containing 3 zero-excitation hyperplanes $\mathbf{H}_1$, $\mathbf{H}_2$ and $\mathbf{H}_3$ for patterns 1 to 3 respectively. The vectors defining the hyperplanes are denoted by $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$ respectively.

and $\mathbf{n}_2$. Specifically $\mathbf{n}_3$ can be written as

$$\mathbf{n}_3 = a_1\,\mathbf{n}_1 + a_2\,\mathbf{n}_2 \tag{8.2}$$

where a_1 and a_2 are the coefficients.

The excitation for each pattern p is the scalar product of the vector $\mathbf{n}_p = \mathbf{i}\mathbf{n}_p$ and the weight state $\mathbf{w}$, i.e.

$$\mathrm{ex}_p = \mathbf{n}_p \cdot \mathbf{w} \tag{8.3}$$

such that for our 2-D example ex_3 can be written and subsequently expanded as

$$\begin{aligned} \mathrm{ex}_3 &= \mathbf{n}_3 \cdot \mathbf{w} \\ &= (a_1\,\mathbf{n}_1 + a_2\,\mathbf{n}_2) \cdot \mathbf{w} \\ &= a_1\,(\mathbf{n}_1 \cdot \mathbf{w}) + a_2\,(\mathbf{n}_2 \cdot \mathbf{w}) \\ &= a_1\,\mathrm{ex}_1 + a_2\,\mathrm{ex}_2 \end{aligned} \tag{8.4}$$

by first using (8.3) then (8.2) then the linearity property of the scalar product (see Theorem 9.4 on page 285 in [Nob69]) and finally (8.3) again. This describes what the excitation value for pattern 3 has to be in terms of what the excitation values for patterns 1 and 2 are. By using an additional coefficient $a_3 = -1$ this can be written as

$$a_1 \, \mathrm{ex}_1 + a_2 \, \mathrm{ex}_2 + a_3 \, \mathrm{ex}_3 = 0 \tag{8.5}$$

which defines a plane through the origin of 3-D excitation space. This plane constitutes the realisable manifold of excitation space on which all realisable excitation states lie. The normal to the realisable hyperplane in excitation space is the vector $\mathbf{a}$ containing the coefficients. The fact that the realisable plane goes through the origin will be seen for all future descriptions of realisable manifolds in excitation space. The reason for this is that the zero-valued excitation state is always realisable with a zero-valued weight state.

The above describes the situation for a single output unit with 2 weights attempting to learn 3 training patterns such that the realisable manifold is a plane in 3-D excitation space. For the more general case for a single output unit, the dimensionality of the realisable manifold in excitation space is equal to I which is the number of linearly independent normals $\mathbf{n}_p$ to the zero-excitation hyperplanes $\mathbf{H}_p$. As described above the $\mathbf{n}_p$ vectors are drawn from the the training set input vectors for a single layer network.

In general, the number I of linearly independent input vectors is at most N, i.e. the number of weights. This is because the number of linearly independent vectors in an N-dimensional vector space can be at most N. These I linearly independent input vectors are normals $\mathbf{n}_i$ to the zero-excitation hyperplanes $\mathbf{H}_i$, where i ranges from 1 to I. Because the excitations

ex_i correspond to the scalar product $(\mathbf{n}_i \cdot \mathbf{w})$, the I excitations are linearly independent of each other due to the linear independence of the normal vectors. If I excitation variables are linearly independent of each other then the dimensionality of the realisable manifold in excitation space is I.

By expressing each of the remaining $(P - I)$ patterns' normals $\mathbf{n}_p$ in terms of the linearly independent set of I normals $\mathbf{n}_i$, $(P - I)$ hyperplanar equations are found each of which express an excitation value in terms of the linearly independent set of excitations. These $(P - I)$ hyperplanar equations represent the realisable manifold in excitation space. If $(P - I = 0)$, i.e. the number of linearly independent excitation variables I is the same as the number of patterns P then the whole of excitation space is realisable and hence the whole of output space is realisable. In this case there will not be any local minima. The same is true for the case when $I > P$.

In the following it will be examined how make use of the equations determining the realisable manifold in excitation space for one case. This case is where the realisable manifold is a single hyperplane in excitation space.

8.2 Realisable Curved Manifolds in Output Space

Section 5.2 showed how the realisable line in excitation space maps to a realisable line in output space for a simple example with 2 training patterns. The reason that a realisable line in output space was obtained was due to the simplicity of the example. In general the

realisable manifold in output space is not expected to be linear which can be shown with another simple 2 pattern example.

Imagine the realisable manifold in excitation space is defined for 2 training patterns as

$$\mathrm{ex}_2 = a_1 \, \mathrm{ex}_1 \tag{8.6}$$

where ex_1 and ex_2 are the excitations for pattern 1 and 2 respectively and a_1 is the coefficient describing the realisable line. For a sigmoidal activation function the realisable manifold in output space is a curve defined by

$$\mathrm{out}_2 = \frac{1}{1 + \left(\frac{1}{\mathrm{out}_1} - 1\right)^{a_1}} \tag{8.7}$$

where out_1 and out_2 are the outputs for pattern 1 and 2 respectively.

For the specific example given in section 5.2 by (5.25) and (5.27) on page 122 the coefficient $a_1 = -1$. In this case (8.7) reduces to the straight line in output space defined by $\mathrm{out}_2 = 1 - \mathrm{out}_1$. Likewise if $a_1 = 1$, (8.7) reduces to the straight line defined by $\mathrm{out}_2 = \mathrm{out}_1$. Another special case is when $a_1 = 0$ so that (8.7) reduces to the straight line $\mathrm{out}_2 = 0.5$.

In general though, the training set may not contain a symmetry like the one shown in Figure 5.7**(a)** on page 122 which makes the coefficient $a_1 = -1$ for a 2-1 network without a bias unit and 2 training patterns. A value of $a_1 = \pm 1$ for 2 training patterns signifies that one pattern's excitation is the same or the opposite of the other pattern's excitation for all weight states. Training sets that cause $a_1 = 0$ are also not expected to occur generally. In order for a_1 to be 0 for a 2-1 network without bias, a training pattern must be at the point

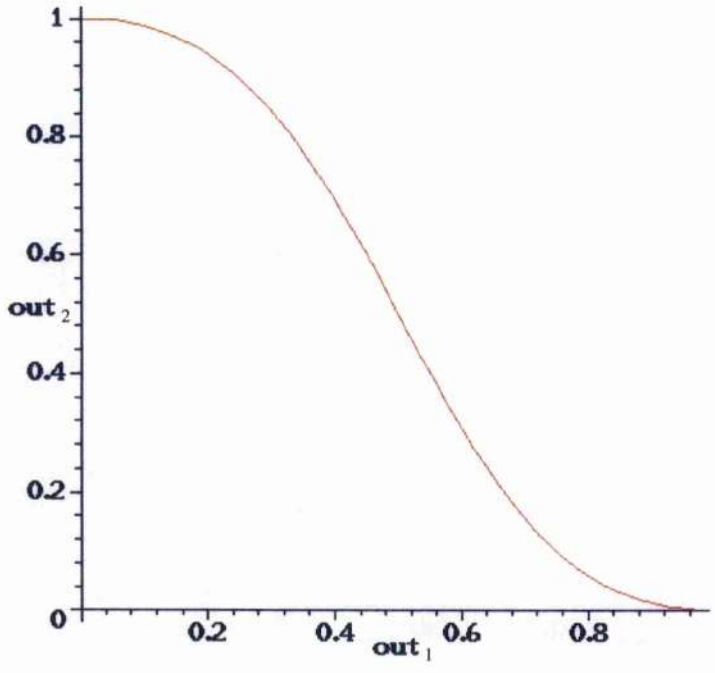


Figure 8.2: Output space for 2 patterns with axes out_1 and out_2. The realisable curve in output space for the 2 training patterns is generated by the realisable line in 2-D excitation space with a coefficient of $a_1 = -2$.

(0,0) in input space. For a setting of $a_1 = -2$ for instance, i.e. not one of the 3 cases mentioned above, the realisable manifold in output space is no longer a straight line but is the curve depicted in Figure 8.2.

For GPS the aim is to find the closest point on the realisable manifold to the goal state in output space. However, even for the simple 2-D example shown here this may prove to be non-trivial. The approach taken here is to render the task feasible by using a tractable approximation of the realisable manifold.

8.3 A Semi-linear Approximation

The Sigmoid activation function can be approximated by a semi-linear form. One such semi-linear activation function divides the relation between excitation and output into 3 linear segments. It produces an output of 0 or 1 for an excitation outside an interval of $[-k, +k]$ and a linear dependence of output on excitation within the same interval of excitation. The semi-linear activation function S may be defined as

$$\text{out} = S(\text{ex}) = \begin{cases} 0 & ,\text{if ex} < -k \\ \frac{\text{ex}}{2k} + \frac{1}{2} & ,\text{if } |\text{ex}| < k \\ 1 & ,\text{if ex} > +k \end{cases} \tag{8.8}$$

and Figure 8.3 shows the sigmoid activation function and the semi-linear activation function in the same graph.

A semi-linear activation function with only 3 linear segments as described above may produce a fairly good approximation of the sigmoid activation function. Work which was done for a project at St Andrews called Wintermute [WLC93], and is unpublished so far, was intended to investigate graphically how well a semi-linear activation function could approximate a sigmoid activation function. This was done by comparing the I/O mappings generated by the two activation functions for various network topologies and training sets. The results from this project showed that a good approximation may be obtained by using a semi-linear activation function even though just 3 line segments are used.

A specific example of an I/O mapping produced with a sigmoid activation function compared to the I/O mapping produced with a semi-linear activation function with 3 segments

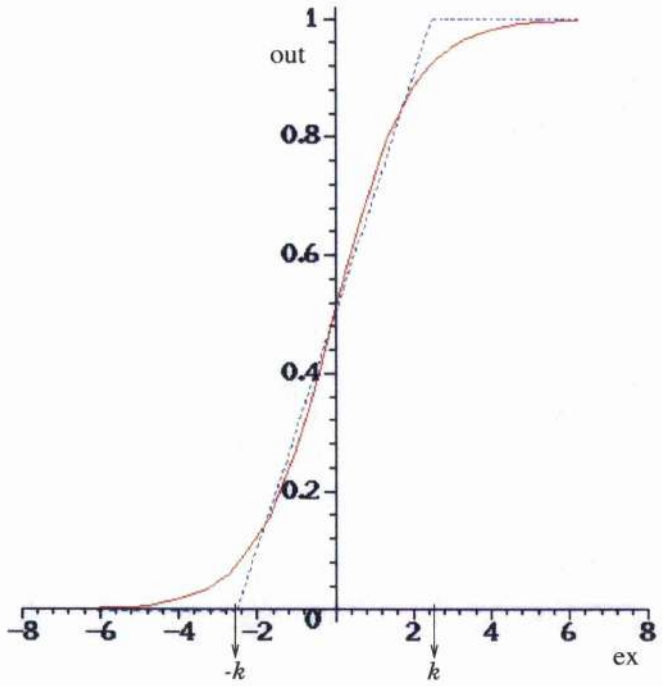
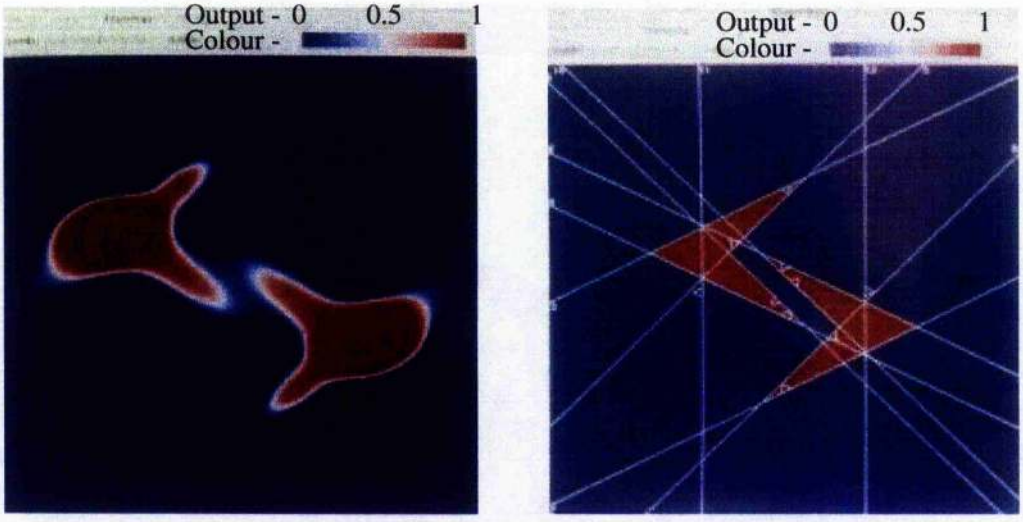


Figure 8.3: The solid curve denotes the sigmoid activation function and the dashed lines denote the semi-linear approximation to the sigmoid. The cut-off value of k for the semi-linear activation function is set to 2.5 here for example.

is displayed in Figure 8.4. Figures 8.4(**a**) and 8.4(**b**) both show the I/O space for a 2-10-1 multi-layer network for a specific weight state. The colouring at a specific point in I/O space denotes the I/O mapping, i.e. the neural output corresponding to the neural input. Figure 8.4(**a**) shows the I/O mapping for a network with a sigmoid activation function. The I/O mapping produced by a network with a semi-linear activation function is displayed in Figure 8.4(**b**). The 10 hidden unit hyperplanes producing the I/O mapping are indicated by the 10 white lines in Figure 8.4(**b**). The comparison in this case shows that the semi-linear activation function is able to produce a fairly good approximation of the true I/O mapping.



a) b)

Figure 8.4: I/O space for a 2-10-1 multi-layer network. **(a)** shows the I/O mapping for a weight state and using a sigmoid activation function. **(b)** shows the I/O mapping for the same weight state but using a semi-linear activation function. The hidden unit hyperplanes that produce the mapping are denoted as lines in I/O space.

For the purposes of GPS it was decided to make use of a semi-linear activation function to form an approximation of the realisable manifold in output space. The semi-linear form is tractable since the linear components allow highly feasible computation of distances to the approximations of the realisable manifold, as will be shown.

For the simple example shown in Figure 8.2 in section 8.2, the realisable manifold in excitation space was defined by $\mathrm{ex}_2 = a_1\,\mathrm{ex}_1$ with $a_1 = -2$. Figure 8.5**(a)** shows this realisable manifold as the line $\mathbf{RM_e}$ through the origin of excitation space. The realisable manifold $\mathbf{RM_e}$ is shown to intersect with 4 hyperplanes in 2-D excitation space which are the 4 dashed lines $\mathrm{ex}_1 = -k$, $\mathrm{ex}_1 = +k$, $\mathrm{ex}_2 = -k$ and $\mathrm{ex}_2 = +k$. The intersection points are denoted by the points **1**, **2**, **3** and **4** on $\mathbf{RM_e}$.

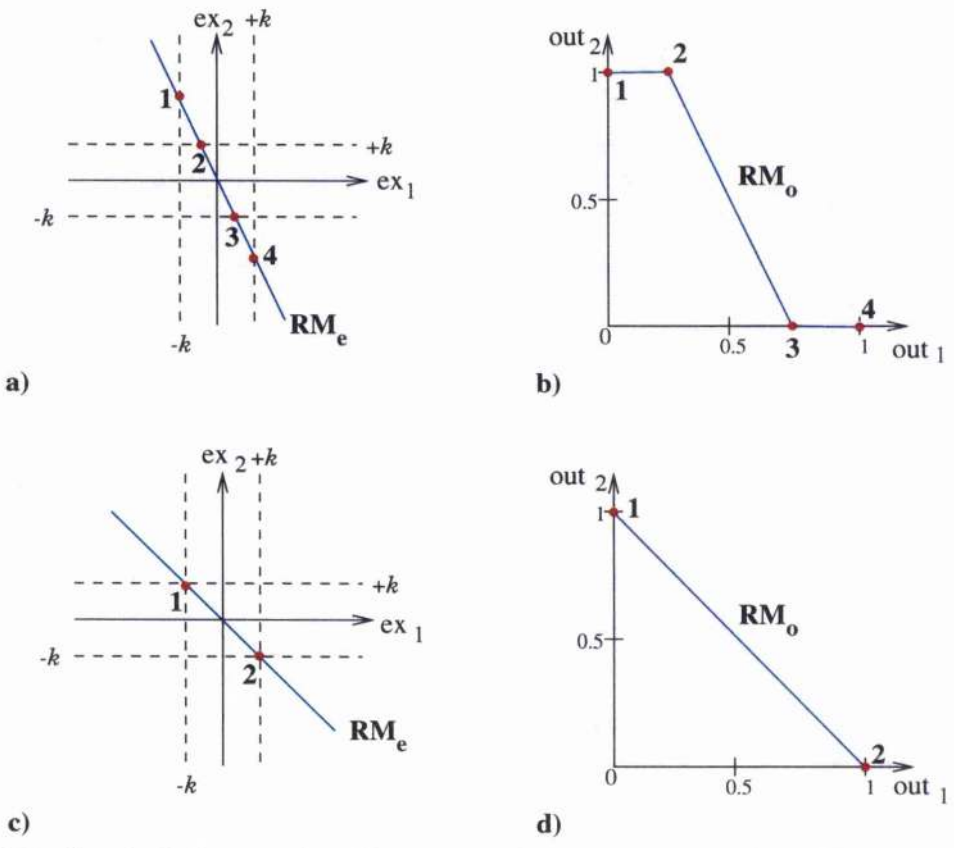


Figure 8.5: Simple 2-D examples of mapping the realisable manifold $\mathbf{RM_e}$ in excitation space to the semi-linear approximation of the realisable manifold in output space $\mathbf{RM_o}$. **(a)** shows the $\mathbf{RM_e}$ $ex_2 = -2ex_1$ in excitation space. **(b)** shows $\mathbf{RM_o}$ in output space generated by the $\mathbf{RM_e}$ in **(a)**. **(c)** shows the $\mathbf{RM_e}$ $ex_2 = -ex_1$ in excitation space. **(d)** shows $\mathbf{RM_o}$ in output space generated by the $\mathbf{RM_e}$ in **(c)**.

The intersection points **1** and **4** on $\mathbf{RM_e}$ correspond to excitation states at which the output out_1 for pattern 1 changes from being a constant value of 0 or 1 to being a linear function of ex_1. The intersection points **2** and **3** denote the same change over states for the output variable out_2 for pattern 2. The 4 intersection points on $\mathbf{RM_e}$ are shown to map to 4 points in output space in Figure 8.5**(b)**.

Since the only changes in output function occur at the intersection points the three straight

line connections between the points **1**, **2**, **3** and **4** in Figure 8.5**(b)** form the realisable manifold $\mathbf{RM_o}$ in output space. Applying the semi-linear activation function S to $\mathbf{RM_e}$ can consequently be seen to produce $\mathbf{RM_o}$ in Figure 8.5**(b)**. The realisable manifold $\mathbf{RM_o}$ displayed in Figure 8.5**(b)** is the semi-linear approximation to the realisable curve in Figure 8.2 which was obtained by using a sigmoidal activation function.

Figures 8.5**(c)** and 8.5**(d)** show how the realisable manifold in excitation space for a setting of $a_1 = -1$ maps to a single line segment in output space. This produces the line defined by $(\text{out}_2 = 1 - \text{out}_1)$ which is equivalent to the realisable line obtained with a sigmoid for that coefficient setting as displayed in Figure 5.8 on page 123 in section 5.2.

The situation depicted in Figures 8.5**(c)** and 8.5**(d)** for when the coefficient $a_1 = -1$ can be seen as a rare case which is not expected to occur generally as explained in section 8.2. Similarly, the cases where $a_1 = +1$ or $a_1 = 0$ are also not expected to occur generally, for the reasons explained in section 8.2. The case where $a_1 = 0$, which is not depicted here, would result in $\mathbf{RM_o}$ being the straight line defined by $\text{out}_2 = 0.5$. These are the only 3 cases where the realisable manifold in output space consists of one single line segment in 2-D output space.

For the cases where a_1 is not equal to -1, 0 or +1 the realisable manifold in 2-D output space consists of three line segments as depicted in Figure 8.5**(b)**. This is because the realisable manifold $\mathbf{RM_e}$ goes through the origin of excitation space and must intersect the $\pm k$ lines in 4 unique points if a_1 is not equal to -1, 0 or +1. These 4 unique points map to 4 unique points in output space so that 3 line segments will be formed between these 4 points in

output space.

Two of the 3 line segments on the semi-linear realisable manifold in output space are are always situated along the edges of the square which bounds the region of output space for outputs in range from 0 to 1. This is because for two patterns i and j two of the line segments correspond to excitations defined by $|\text{ex}_i| \geq k$ and $|\text{ex}_j| \leq k$ such that out_i is 0 or 1 and out_j varies linearly depending on ex_j.

The line segment in 2-D output space corresponding to $|\text{ex}_i| \leq k$ and $|\text{ex}_j| \leq k$ will either be the only line segment if $(a_1 = 0, \text{or} a_1 = \pm 1)$ or the middle line segment of 3 for other values of a_1. In either case both outputs out_i and out_j vary linearly along this segment as functions of the excitations ex_i and ex_j. I will refer to this segment as the *centre segment.* For this centre line segment the following can be established. The realisable manifold in excitation space is defined as

$$\text{ex}_2 = a_1 \, \text{ex}_1 \tag{8.9}$$

By applying the semi-linear activation function defined in (8.8) to (8.9) one can establish that

$$\text{out}_2 = S(\text{ex}_2) = S(a_1 \, \text{ex}_1) = a_1 \, S(\text{ex}_1) - \frac{1}{2} \, (a_1 - 1) = a_1 \, \text{out}_1 - \frac{1}{2} \, (a_1 - 1) \tag{8.10}$$

This shows that the gradient or orientation of the centre line segment in output space defined by the coefficient a_1 is the same as for the realisable manifold in excitation space. The centre line segment can also be seen to go through the point (0.5, 0.5) in output space for any value of a_1. This is because the 0.5 valued output state corresponds to the the zero-valued excitation state which is always producible at the zero-valued weight state. In other

words, for any value of a_1 produced by the training set the 0.5 valued output state lies on the realisable manifold in output space.

Another observation to be made from (8.10) is that the shape of the realisable manifold in output space is independent of the semi-linear activation function's cut-off value k. This is true for all $k \neq 0$ and also holds in the limit as $k \to 0$. For this reason, specific values of k are not relevant in the following sections.

8.4 Realisable Hyper-surfaces in Output Space

The previous section 8.3 showed how the semi-linear activation function may be applied to simple examples with 2 training patterns. The description is now extended to a more general case where the realisable manifold is a hyperplane in P-dimensional excitation space and P is the number of training patterns. This realisable hyperplane will be shown to map to a hyper-surface in output space. One may define a hyper-surface in P-dimensional space as a surface which has dimension $P-1$. In this case the hyper-surface will be shown to be comprised of hyperplanar facets or in other words bounded regions of hyperplanes.

The realisable manifold in excitation space for this case is given by

$$\sum_{i=1}^{P} a_i \, \mathrm{ex}_i = 0 \tag{8.11}$$

which defines a hyperplane through the origin of excitation space with an orientation given by its normal vector **a** comprised of all the coefficients a_i. Again, the origin of excitation space is always realisable because it is producible by the zero-valued weight state.

When the excitation values for all patterns are such that $|\text{ex}_i| \leq k$, $\forall i$, all the outputs out_i are linear functions of the ex_i. For this case specifically it may be shown that by applying the semi-linear activation function S to (8.11) results in

$$\sum_{i=1}^{P} a_i \, \text{out}_i = \frac{1}{2} \sum_{i=1}^{P} a_i \tag{8.12}$$

This equation can be shown to be that of a hyperplane in output space with the normal vector **a** comprised of the coefficients a_i. The hyperplane goes through the *centre point* $\mathbf{c} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})$ in output space. In future I will refer to this hyperplane as the *centre hyperplane* in output space. The centre hyperplane is the P-dimensional generalisation of the centre line segment in 2-D given by (8.10). By deriving (8.12), it is shown that excitation states on the realisable manifold in excitation space defined by $|\text{ex}_i| \leq k$, $\forall i$ map to output states on this centre hyperplane.

In order to visualise how the realisable manifold in excitation space maps to output space some further 2-D diagrams are presented. The principle for a realisable hyperplane in P-dimensional excitation space mapping to output space is the same.

Figure 8.6**(a)** shows excitation space with the realisable manifold $\mathbf{RM_e}$ as a hyperplane in excitation space going through the origin and with the orientation defined by the vector **a**. A hypercube in the space denoted by $\mathbf{HC_e}$ goes through all points with excitation values at $\pm k$. The hypercube contains the region of excitation space within which all excitation values are in the range from $-k$ to $+k$.

The excitation states on $\mathbf{RM_e}$ and within the hypercube $\mathbf{HC_e}$ are realisable states defined by $|\text{ex}_i| \leq k$, $\forall i$, and are shown by (8.12) to map to output states in Figure 8.6**(b)** which lie

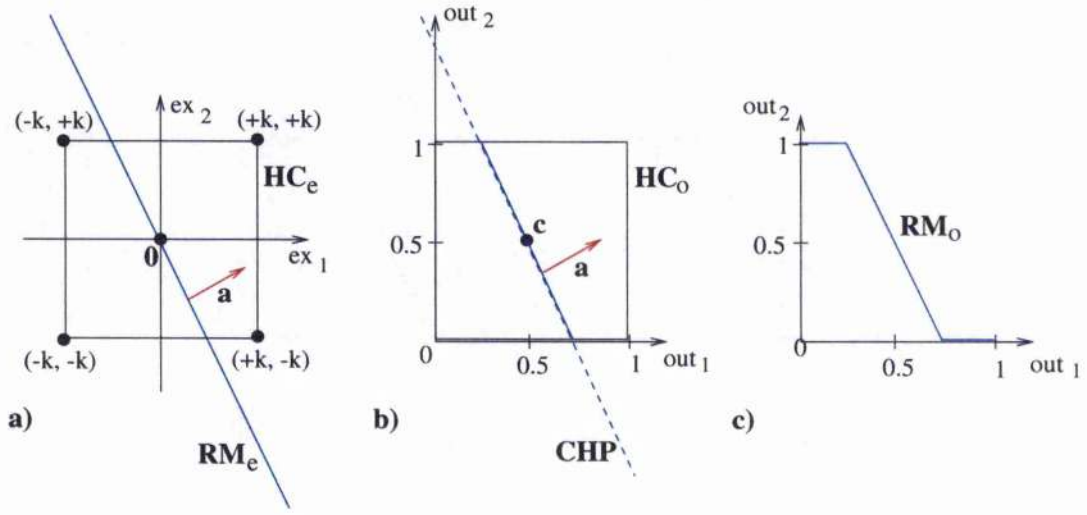


Figure 8.6: A 2-D graphical view of the realisable manifold creation in output space. **(a)** shows excitation space with the realisable manifold $\mathbf{RM_e}$ as a hyperplane through the origin and with an orientation defined by the vector **a**. **(b)** shows the centre hyperplane **CHP** through the point **c** in output space and with orientation defined by the vector **a**. **(c)** shows the semi-linear realisable manifold $\mathbf{RM_o}$ in output space which corresponds to the realisable manifold $\mathbf{RM_e}$ in excitation space in **(a)**.

on the centre hyperplane **CHP** and within a hypercube $\mathbf{HC_o}$. The hypercube $\mathbf{HC_o}$ in output space has edges of length 1, has the origin of output space as one vertex and is defined by having a second vertex at $(1, 1, 1, \ldots, 1)$. This hypercube contains the region of output space outside which no output states are realisable, because their outputs are outside the range from 0 to 1. Consequently the unrealisable regions on the centre hyperplane **CHP** are outside the hypercube $\mathbf{HC_o}$ and are denoted by dashed lines. As established by deriving (8.12) the centre hyperplane **CHP** goes through the centre point of output space which for the 2-D diagram is the point $\mathbf{c} = (0.5, 0.5)$ and has an orientation defined by the vector **a**. This is the same vector **a** that defines the orientation of the excitation hyperplane in Figure 8.6(**a**).

The region of the centre hyperplane **CHP** inside the hypercube $\mathbf{HC_o}$ which is produced by excitation states on $\mathbf{RM_e}$ and within $\mathbf{HC_e}$ forms the centre part of the semi-linear realisable manifold in output space depicted by $\mathbf{RM_o}$ in Figure 8.6(**c**).

The other parts of $\mathbf{RM_o}$ are generated by excitation states on $\mathbf{RM_e}$ outside the hypercube $\mathbf{HC_e}$. Take the states on $\mathbf{RM_e}$ above $\text{ex}_2 = +k$ for instance. These states will map to output states on $\mathbf{RM_o}$ defined by values $\text{out}_2 = S(\text{ex}_2) = 1$, and out_1 will vary linearly as a function of ex_1 while $\text{ex}_1 > -k$. Hence these states lie on the $\text{out}_2 = 1$ face of $\mathbf{HC_o}$ and towards $\text{out}_1 = 0$. Similarly, the excitation states on $\mathbf{RM_e}$ below $\text{ex}_2 = -k$ map to output states on $\mathbf{RM_o}$ defined by $\text{out}_2 = S(\text{ex}_2) = 0$ and out_1 will vary linearly as a function of ex_1 while $\text{ex}_1 < +k$. Hence these states lie on the $\text{out}_2 = 0$ face of $\mathbf{HC_o}$ and towards $\text{out}_1 = 1$.

The latter 2-D reasoning for states on $\mathbf{RM_e}$ and outside $\mathbf{HC_e}$ mapping to the faces of the hypercube $\mathbf{HC_o}$ may be extended to P dimensions as follows. The subset of states on $\mathbf{RM_e}$, defined by one of the excitations $|\text{ex}_j| \geq k$ while $|\text{ex}_p| \leq k$, $\forall p \neq j$, is mapped to a facet in output space contained within the hyperplane defined by $\text{out}_j = 0$ or $\text{out}_j = 1$ depending on whether $\text{ex}_j \leq -k$ or $\text{ex}_j \geq +k$ respectively. All further subsets of the 1st subset of excitation states with additional excitation variables $|\text{ex}_p| \geq k$ map to a set of output states with additional output variables O_p set to 0 or 1 and are on the same facet contained within the same hyperplane the 1st subset of excitation states mapped to.

In summary, realisable excitation states on $\mathbf{RM_e}$ have two types of location in output space. One is where all patterns' excitation values are in the range from $-k$ to $+k$. These states map to the centre hyperplane in output space. In contrast, excitation states on $\mathbf{RM_e}$ for

which at least one pattern's $|\text{ex}_j| \geq k$ map to the surface of the hypercube $\mathbf{HC_o}$ in output space. That is, they map to states on hyperplanes defined by $\text{out}_j = 0$ and $\text{out}_j = 1$ which contain the associated hypercube surface.

A graphical interpretation of the semi-linear realisable manifold creation in output space is to imagine squashing the centre hyperplane **CHP** onto the surface of the hypercube $\mathbf{HC_o}$ where it extends beyond the hypercube. This so called squashing may also be likened to performing an orthogonal projection of the centre hyperplane **CHP** onto the surface of the hypercube $\mathbf{HC_o}$ where it extends beyond the hypercube. This means that the only facets on $\mathbf{RM_o}$ in output space are hyperplanar facets contained either on the centre hyperplane **CHP** within the hypercube $\mathbf{HC_o}$ or on the surface of the hypercube $\mathbf{HC_o}$.

As a consequence, the realisable manifold will consist of only the centre facet when all coefficients $a_i = 0, \forall i \neq p$ and $a_p \neq 0$. The centre facet on the centre hyperplane is then defined by $\text{out}_p = 0.5$. In this case the centre facet forms an angle of 90 degrees with the hypercube faces on the hyperplanes defined by $\text{out}_i = 0$ and $\text{out}_i = 1$ for all $i \neq p$. This case is very artificial though. In practice, the realisable manifold is expected to consist of more than one facet. The joins between these facets are on the surface of the hypercube. The internal angles facing the interior of the hypercube are then between 90 and 180 degrees.

8.5 Finding Minimum Output States

The objective of GPS is to find the output state which is that of the global minimum for the goal **G**. For the semi-linear approximation of the realisable manifold in output space the following may be shown: *The shortest distance from the goal* **G** *to the closest hyperplanar facet on the semi-linear realisable manifold will always be found at right angles to that facet.*

To see this, imagine a hypersphere being expanded around **G** until it touches a hyperplanar facet on the realisable manifold. The contact point of the hypersphere with the facet is the closest point on the realisable manifold to **G**. It will now be shown that the hyperplanar facet closest to **G** is always tangential to the hypersphere at the contact point which means that the minimum distance is found at right angles to that facet.

In general a hyperplanar facet will always be tangent to the hypersphere at the contact point, except possibly for 2 cases.

1. The point on the facet closest to **G** is at an end point on the realisable manifold, i.e. the smallest hypersphere around **G** touches an end point.

2. The point on the facet closest to **G** is on a join between 2 facets, i.e. the smallest hypersphere around **G** touches a join between 2 facets.

These 2 cases will now be examined.

In order to do this, facts about the positions of the facets and their orientation are impor-

tant. This information was established in section 8.4 but is repeated briefly at this point. One facet on the semi-linear realisable manifold is on the centre hyperplane and additional facets, are on the faces of the hypercube bounding output space for values between 0 and 1. All end points of the realisable manifold and joins between facets are hence located on the faces of the hypercube. The internal angles facing the interior of the hypercube on joins between facets are between 90 and 180 degrees. This situation is depicted in Figure 8.7**(a)** where **HC** denotes the hypercube, **RM** denotes the realisable manifold and α denotes the interior angle at the joins $\mathbf{J_1}$ and $\mathbf{J_2}$. The end points of **RM** are denoted as $\mathbf{E_1}$ and $\mathbf{E_2}$. It should also be noted that the goal state **G** must always lie within **HC** or on its surface.

Touching an end point: The smallest hypersphere to touch the realisable manifold when surrounding the goal state may only touch an end point of the realisable manifold from within the hypercube for one case. This is when the goal is on a face of the hypercube which is orthogonal to the face containing the end point. Hence the hypersphere touches the facet tangentially and the minimum distance is found at right angles to the facet. This is shown in Figure 8.7**(b)**.

Touching a join between facets: The smallest hypersphere to touch the realisable manifold when surrounding the goal state cannot touch a join between facets from within the hypercube because the interior angles on joins are between 90 and 180 degrees. This situation is depicted in Figure 8.7**(c)**. The hypersphere $\mathbf{HS_1}$ with minimum radius surrounding $\mathbf{G_1}$ touches a facet before it touches the join. This is true for any goal state on the same side of the centre facet as $\mathbf{G_1}$. Similarly, the hypersphere $\mathbf{HS_2}$ with minimum radius surrounding

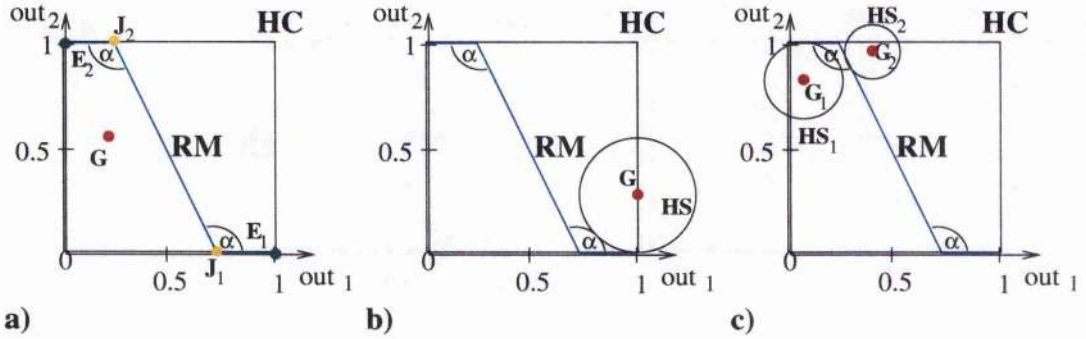


Figure 8.7: A 2-D view of the realisable manifold **RM** in output space. **(a)** shows the goal state **G** within the hypercube **HC**. The joins $\mathbf{J_1}$ and $\mathbf{J_2}$ between the centre facet and the facets on the hypercube surface have an associated interior angle α. The end points of **RM** are $\mathbf{E_1}$ and $\mathbf{E_2}$. **(b)** shows the smallest hypersphere **HS** to touch **RM** surrounding **G** and touching an end point of **RM** tangentially. **(c)** shows the smallest hyperspheres $\mathbf{HS_1}$ surrounding $\mathbf{G_1}$ and $\mathbf{HS_2}$ surrounding $\mathbf{G_2}$ touching facets tangentially rather than the joins between the facets.

$\mathbf{G_2}$ touches a facet before it touches the join. This is also true for any goal state on the same side of the centre facet as $\mathbf{G_2}$.

By demonstrating that the two cases above cannot occur it is shown that the smallest hypersphere to touch the realisable manifold and surrounding the goal must touch the manifold tangentially. This means that the minimum distance from the goal state **G** to a realisable facet is always found at right angles to the facet.

In order to establish the output state on the realisable manifold which is closest to the goal **G**, normals can be dropped from **G** to the realisable facets. The intersections of the normals with the facets yield output states, each having an associated distance to **G**. The output state associated with the minimum distance to **G** is the global minimum output state. The other output states at the base of normals from realisable facets to **G** may be shown to

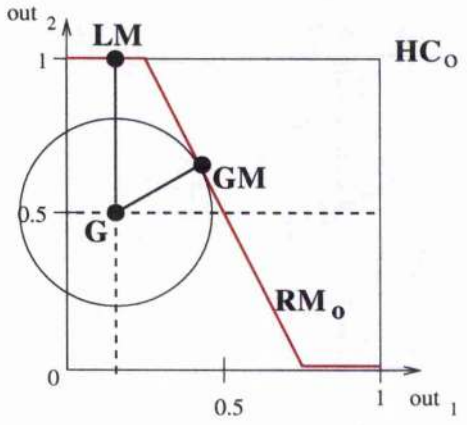


Figure 8.8: A 2-D diagram of output space showing how the minimum distance from the goal **G** to a facet on the realisable manifold $\mathbf{RM_0}$ is obtained. The circle surrounding **G** represents a hypersphere surrounding the goal and is shown to touch the centre facet at a point **GM** which is the global minimum output state. **GM** may be obtained by dropping a normal from **G** to the centre facet. One of the other normals dropped towards the hypercube $\mathbf{HC_0}$ intersects a realisable facet at the local minimum state **LM**.

be local minimum output states. This is because all output states surrounding them on the realisable manifold locally are further away from the goal. Figure 8.8 shows this scenario in output space. The realisable manifold is denoted by $\mathbf{RM_0}$. The circle surrounding the goal **G** which represents the hypersphere being expanded around **G** touches the centre facet at **GM** with a minimum radius. Consequently the global minimum output state is at **GM**.

It has been established that minima in output space may be found by dropping normals from **G** to realisable facets. It has also been established that one realisable facet always occupies the centre hyperplane through output space and additional facets may be found on the hyperplanes which contain the faces of the hypercube $\mathbf{HC_0}$. This knowledge allows the construction of a simple algorithm to find the global minimum and local minimum output states.

There are $2P$ hypercube faces which are contained on the hyperplanes defined by $\text{out}_p = 0$ and $\text{out}_p = 1$ for all p ranging from 1 to P, where P denotes the number of training patterns. Including the realisable centre facet, the maximum number of realisable facets is therefore $(2P + 1)$.

The normals dropped from the goal **G** to the $(2P + 1)$ hyperplanes, which contain the hypercube surface and the centre facet, may not all necessarily intersect realisable facets. It is therefore necessary to test whether the states at the base of the normals dropped from **G** to the hyperplanes correspond to realisable output states. This can be done by examining the realisable hyperplane in excitation space for whether states on it correspond to the output states at the base of the normals. The normals which do not intersect the realisable facets in their hyperplanes are denoted as dashed lines in Figure 8.8.

The algorithm for GPS operating on a realisable hyperplane in excitation space may be designed to establish candidate global minimum and local minimum output states on all $(2P + 1)$ hyperplanes in output space. The candidates which do not lie on a realisable facet can be eliminated. Out of the remaining candidates, the one associated with the least distance to **G** is the global minimum output state. Any remaining candidates are local minimum output states.

8.6 The GPS Algorithm

The main procedure for performing GPS on a neural training problem for which the realisable manifold in excitation space is a hyperplane may be divided into 2 stages at the top level.

1. Establish the vector **a** which defines the realisable hyperplane in excitation space and consequently the semi-linear realisable hyper-surface in output space.

2. For the goal state **G** compute the estimated global minimum position **GM** on the realisable hyper-surface in output space defined by the vector **a**.

8.6.1 Stage 1

The algorithm steps for performing stage 1 of GPS follow the descriptions given in section 8.1. The algorithm given here deals with the case where the realisable manifold is a hyperplane in excitation space.

1. Establish the normals to zero-excitation hyperplanes in weight space for all P patterns.

2. Choose a basis set of normals $\mathbf{n}_i$, consisting of N linearly independent normals, out of the P normals, where N is the number of weights.

3. Calculate the coefficients a_i for expressing the linearly dependent $\mathbf{n}_P$ as a linear combination of the basis set of normals $\mathbf{n}_i$. Group the coefficients a_i with an additional

coefficient $a_P = -1$ into a vector **a**.

8.6.2 Stage 2

The algorithm for performing stage 2 follows from the descriptions in section 8.5.

1. Establish $2\,P$ normal distances from the goal **G** to the hyperplanes containing the hypercube surface and establish the candidate output states on the hyperplanes at the base of each normal. Place the distances with their corresponding candidate output states in a list.

2. Establish the normal distance from **G** to the centre hyperplane and establish the candidate output state on this hyperplane at the base of the normal. Append the distance and corresponding candidate output state to the list.

3. Sort the $(2P + 1)$ elements in the list in order of increasing distance value.

4. For each candidate in the list establish whether it is realisable or not, by examining whether the realisable hyperplane in excitation space is able to produce the candidate output state. Delete unrealisable candidates and their associated distance to **G** from the list.

After running this procedure the 1st list entry contains the minimum distance obtained by dropping a normal from **G** to a realisable facet and the output state at the base of the normal. This corresponds to the global minimum output state and its associated global minimum distance to the goal **G**.

8.6.3 Complexity Issues

For the case where the realisable manifold is a hyperplane in P-dimensional excitation space the main calculations may be examined to establish whether the time and memory usage for GPS is feasible. In this sense feasible means that the procure may run in polynomial time.

Stage 1: One of the main components is to draw $(P-1)$ linearly independent vectors $\mathbf{n}_i$ from P vectors from the training set. This part may be shown to be computable in polynomial time as it involves standard methods from linear algebra. The other main component is to express the one linearly dependent vector as a linear combination of the $(P-1)$ basis vectors. Calculating the coefficients to express the linearly dependent vector in terms of the basis may also performed in polynomial time in standard linear algebra. These coefficients form the vector $\mathbf{a}$ define the realisable manifold in excitation space.

Stage 2: $(2P+1)$ normals are dropped to hyperplanar facets, which involves $(2P+1)$ vector operations. The states with their associated distances to $\mathbf{G}$ are then ordered in terms of increasing distance. The ordering of the list with $(2P+1)$ elements may be done with an upper bound of the order $O((2P+1)^2)$ operations with an ordering procedure called *Bubble Sort* (see pages 38 to 40 in [GL82]). Each state found at the base of a normal is tested for whether it is realisable or not which involves a further $(2P+1)$ operations.

Time Usage: The operations for GPS in stage 1 and stage 2 may be performed in polynomial time. Hence the GPS procedure as a whole may be performed in polynomial time.

Space Usage: The memory usage of GPS will not exceed the amount of space needed to store $(2P + 1)$ output states of dimensionality P. This in the order of $O(P^2)$.

8.7 Experiments and Discussion of Results

This first section presents two artificial test examples among many which were used to test the implementation of stage 2 of GPS in 3-D. The second part of this section shows how GPS was applied to two neural training problems which are known to have a local minimum and a global minimum state. The latter tests are in order to see whether GPS as a whole, including stage 1 of the procedure, was implemented correctly and to see whether the semi-linear approximations used in GPS are able to make sufficiently precise predictions for a neural network with a sigmoid activation function.

8.7.1 Artificial Examples in 3-D

The GPS algorithm presented was tested on a number of artificial examples which allow visualisation of the realisable hyper-surface in output space, the goal state and the obtained global minimum state. This was to establish whether the implementation of stage 2 of the GPS procedure was successful or not. Each of these artificial examples defines a realisable

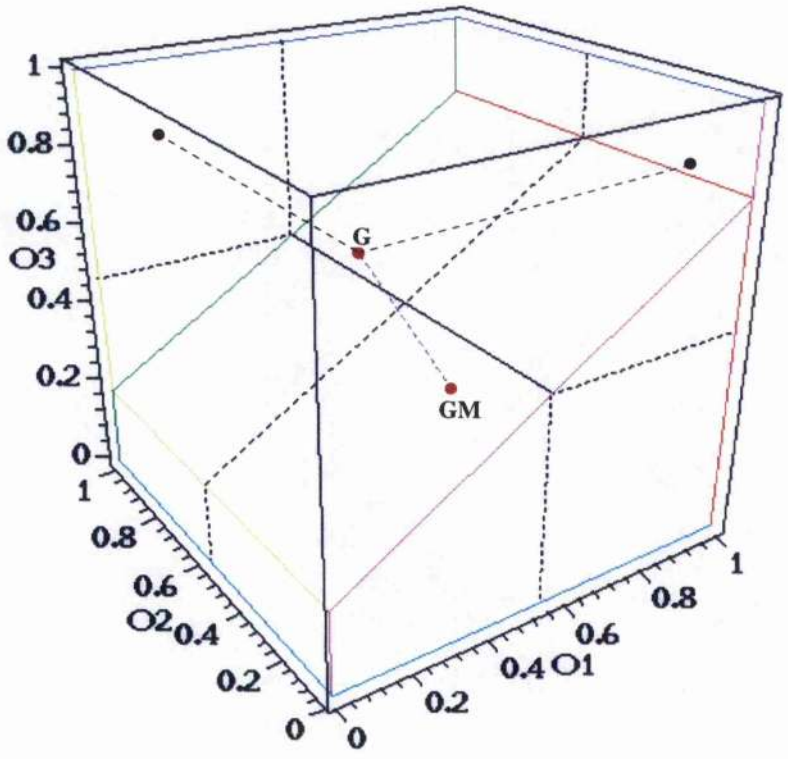


Figure 8.9: Artificial 3-D test example 1 for GPS.

2-D plane in 3-D excitation space by its normal vector **a**. GPS is then given a 3-D goal output state state **G** and returns the list of realisable candidate states in order of increasing distance to **G**.

Figure 8.9 shows the 1st test example which is defined by $\mathbf{G} = (0.2, 0.2, 0.8)$ and $\mathbf{a} = (0.6, 0.00001, -1)$. After running GPS the state $\mathbf{GM} = (0.41, 0.20, 0.44)$ on the centre facet is found to be the closest to **G** at a distance of 0.41. Two other normals from **G** are found to connect with realisable facets on the surface of the cube and at a distance of 0.8.

The 2nd test example is displayed in Figure 8.10. For this example $\mathbf{G} = (0.2, 0.2, 0.8)$ and

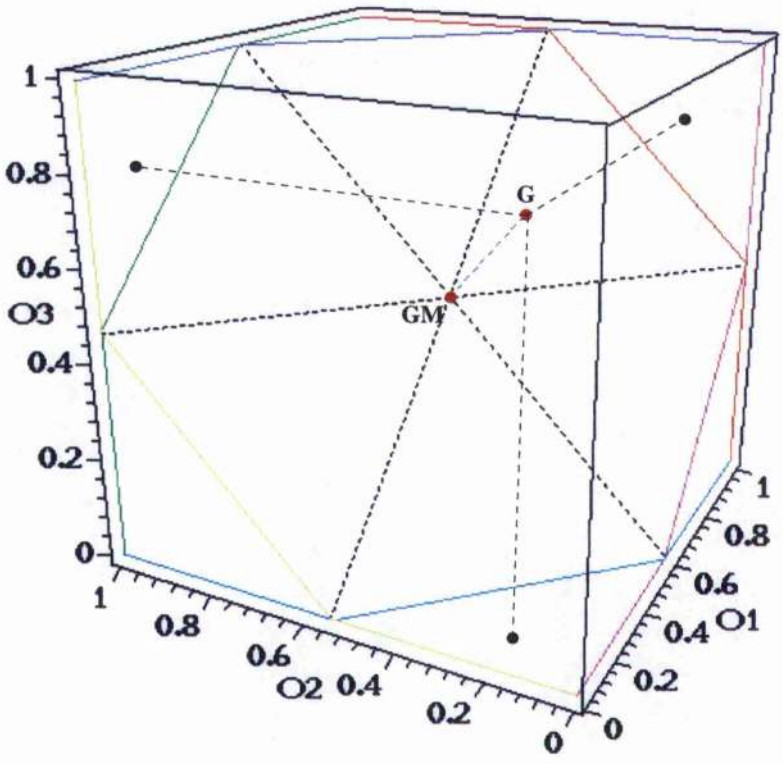


Figure 8.10: Artificial 3-D test example 2 for GPS.

$\mathbf{a} = (1, 1, -1)$. The number of facets on the realisable semi-linear surface in output space can be seen to consist of 7 facets. This is the maximum number of facets, one facet on each side of the cube bounding output space between 0 and 1, and one facet on the centre hyperplane within the cube. For this example GPS obtains the state $\mathbf{GM} = (0.5, 0.5, 0.5)$ on the centre facet as the closest state to **G** at a distance of 0.52. 3 other normals from **G** are found to connect with realisable facets on the surface of the cube and at a distance of 0.8.

When looking at the results obtained from GPS on these and many more test examples not

Pattern	in_1	in_{bias}	Goal Target
1	1	1	0.1
2	2	1	0.1
3	1.1	1	0.9

Table 8.1: Test training set 1 for GPS with known local minimum and global minimum output states and weight states for a 1-1 network.

actually shown here it became clear that GPS had been implemented successfully for 3-D at least. However, the artificial examples used here in initial tests do not relate to neural training problems. The next step was to test GPS on problems with known local minimum and global minimum states for a neural network.

8.7.2 Neural Training Problem 1

The 1st neural training problem chosen to test GPS has only 3 training patterns and is for a single layer 1-1 net. The reason for choosing such an example is that both error-weight space and output space are 3-D. Consequently it is possible to plot the error-weight surface, the realisable manifold and GPS's semi-linear approximation to the realisable manifold in output space. The test for GPS is to see how its prediction of global minimum and local minimum states for this training set matches those previously obtained through convergence tests using BP.

The training set consisting of 3 training patterns for a 1-1 net is given in Table 8.1. The input vectors for these training patterns consist of the vectors $\mathbf{in}_p$ where p refers to training

pattern p. Each training input vector $\mathbf{in}_p$ has elements in_i where i refers to input in_1 and in_{bias}. As described in section 8.1 the input vectors $\mathbf{in}_p$ for a training set are the normal vectors $\mathbf{n}_p$ to the zero-excitation hyperplanes in weight space for a single layer network and for this training set they can be written as

$$\begin{aligned} \mathbf{n}_1 &= (1,1) \\ \mathbf{n}_2 &= (2,1) \\ \mathbf{n}_3 &= (1.1,1) \end{aligned} \tag{8.13}$$

These normal vectors are linked to the excitation for a training input pattern by (8.3) which denotes that $\text{ex}_p = \mathbf{n}_p \cdot \mathbf{w}$ where $\mathbf{w} = (w_1, w_{\text{bias}})$ is the neural weight state for this 1-1 network.

The error-weight surface for this example is depicted in Figure 8.11. The positions of the global and local minimum are indicated roughly by the arrows leading to the error surface in the figure from **GM** and **LM** respectively. Using BP for convergence tests the global minimum weight state $\mathbf{w}_{\text{GM}}$ is found to be near $(-1.53, 1.51)$ and the local minimum weight state $\mathbf{w}_{\text{LM}}$ is found to be near $(43.94, -46.14)$.

The first stage in performing GPS is to establish the realisable manifold in excitation space. This is done by first establishing a linearly independent set of basis normal vectors $\mathbf{n}_p$. The remaining normals can be expressed as a linear combination of the basis set of normals. In this case $\{\mathbf{n}_1, \mathbf{n}_2\}$ is a linearly independent set of normal vectors and forms the basis in terms of which $\mathbf{n}_3$ may be expressed. For this specific example it is found that

$$\mathbf{n}_3 = a_1\,\mathbf{n}_1 + a_2\,\mathbf{n}_2 \tag{8.14}$$

where $a_1 = 0.9, a_2 = 0.1$. By using (8.3) and the linearity property of the scalar product,

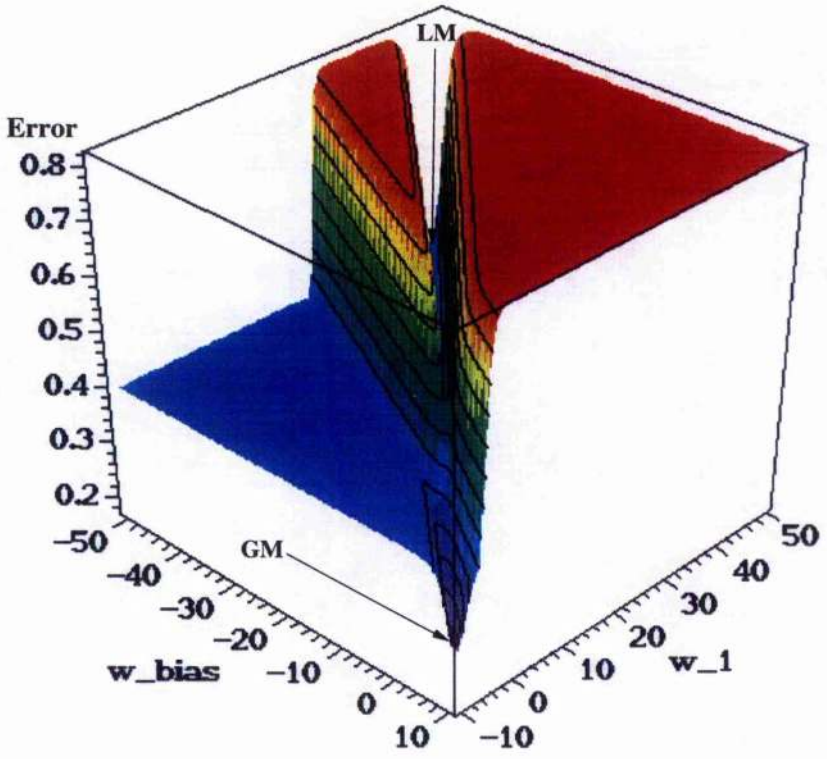


Figure 8.11: Error-weight surface for the 1-1 net test example for GPS. The positions of the global minimum and local minimum are indicated roughly by the arrows leading from **GM** and **LM** respectively.

(8.14) may be re-written as

$$\mathrm{ex}_3 = a_1\,\mathrm{ex}_1 + a_2\,\mathrm{ex}_2 \tag{8.15}$$

By defining an extra coefficient $a_3 = -1$ we can subsequently re-write (8.15) as

$$a_1\,\mathrm{ex}_1 + a_2\,\mathrm{ex}_2 + a_3\,\mathrm{ex}_3 = 0 \tag{8.16}$$

which defines a realisable manifold in 3-D excitation space as a plane through the origin

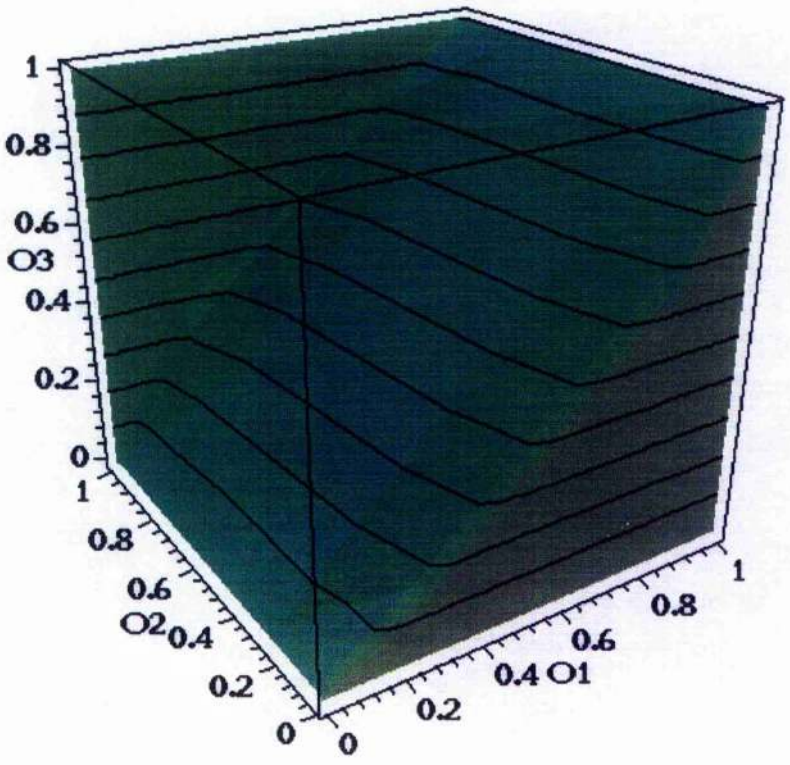


Figure 8.12: The realisable surface in 3-D output space for the 1-1 net training example and a sigmoid activation function.

and with the orientation given by its normal vector $\mathbf{a} = (a_1, a_2, a_3)$.

After GPS has established the vector **a** stage 2 of GPS is invoked which calculates the minimum output positions by dropping normals from the goal **G** to the hyperplanar facets on the realisable manifold in output space. Because output space is 3-D in this case it is possible to plot both the realisable surface in output space for a 1-1 net with a sigmoid activation function and GPS's semi-linear approximation of this realisable surface. Figure 8.12 shows the realisable surface in 3-D output space for the 1-1 net training example and a sigmoid activation function. Figure 8.13 shows GPS's semi-linear approximation to the realisable

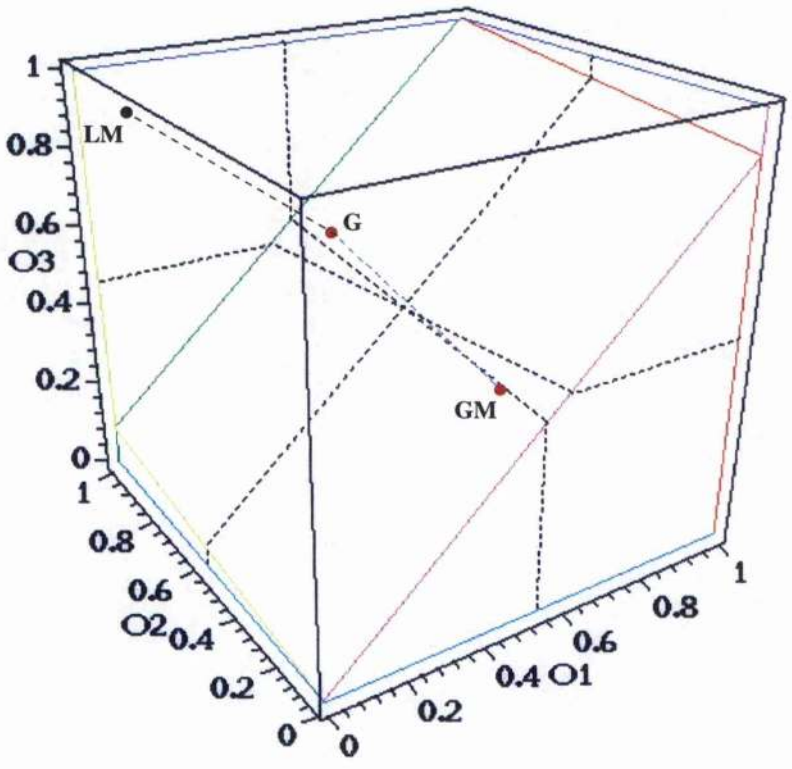


Figure 8.13: This shows GPS's semi-linear approximation to the realisable surface in 3-D output space for the 1-1 net training example. Dropping normals from the Goal **G** onto the hyperplanar facets produces GPS's approximation of the global and local minimum output states **GM** and **LM** respectively.

surface in 3-D output space for the 1-1 net training example. These two figures show that in this case GPS has indeed made a good approximation of the realisable surface.

GPS creates a list of realisable states found at the base of the normals dropped from **G** to the facets shown in Figure 8.13. The list is ordered in terms of increasing distances from these realisable output states to **G**. The first entry in the list is the global minimum output state and the remaining entries in the list are local minimum output states.

For the neural training problem described here GPS establishes 2 output states corresponding to 2 minima. The output states will be written as row vectors $\mathbf{out} = (\text{out}_1, \text{out}_2, \text{out}_3)$ where out_p is the output value for training pattern p. The output states established by GPS are as follows:

1. The global minimum output state $\mathbf{out}_{\text{GM}} = (0.50, 0.14, 0.46)$ with an associated goal distance of 0.59.

2. The local minimum output state $\mathbf{out}_{\text{LM}} = (0.1, 1.0, 0.9)$ with an associated goal distance of 0.9.

where the goal distances are the Euclidean distances from the respective output states to the goal output state $\mathbf{G} = (0.1, 0.1, 0.9)$.

In comparison, the global minimum and local minimum output states at the weight states found by using convergence tests as described above are as follows:

1. $\mathbf{out}_{\text{GM}} = (0.50, 0.18, 0.46)$ with an associated weight state
   $\mathbf{w}_{\text{GM}} = (-1.53, 1.51)$.

2. $\mathbf{out}_{\text{LM}} = (0.1, 1.0, 0.9)$ with an associated weight state
   $\mathbf{w}_{\text{LM}} = (43.94, -46.14)$.

where the weight states are written as row vectors $\mathbf{w} = (w_1, w_{\text{bias}})$.

Comparing the GPS output states to the BP output states shows that the global minimum and the local minimum output states computed by GPS match the states obtained by BP in

convergence tests to a high degree of precision. This shows that although the error-weight surface as depicted in Figure 8.11 may be quite complex with ravines and multiple attractors, even for this simple network and training problem, the realisable surface as depicted in Figure 8.12 may be relatively simple in comparison. Consequently the semi-linear approximations made by GPS make for a good approximation of the surface as depicted in Figure 8.13 and so GPS produces a good approximation to the actual global minimum and local minimum output states.

8.7.3 Neural Training Problem 2

The 2nd neural training problem chosen to test GPS is the training set shown in Figure 5.4 on page 110 in section 5.1. This training set was one of those developed to show ERA failing and showed ERA completely failing to reach the global minimum in section 5.3. The reason for ERA's failure was that it converged to a local minimum rather than the global minimum. As in section 8.7.2 the test for GPS is to see how its prediction of global minimum and local minimum states for this training set matches those previously obtained through convergence tests using BP.

The training set is for a single layer 2-1 network and consists of the 4 training patterns. These are shown in Table 8.2. The input vectors for these training patterns consist of the vectors $\mathbf{in}_p$ where p refers to training pattern p. Each training input vector $\mathbf{in}_p$ has elements in_i where i refers to input in_1, in_2 and in_{bias}. The input vectors $\mathbf{in}_p$ for the training set are the normal vectors $\mathbf{n}_p$ to the zero-excitation hyperplanes in weight space for a single layer

Pattern	in_1	in_2	in_{bias}	Goal Target
1	1	0	1	0.77
2	10	0	1	0.77
3	0	-5	1	0.77
4	-1	-1	1	0.29

Table 8.2: Test training set 2 for GPS with known local minimum and global minimum output states and weight states.

network and for this training set they can be written as

$$\begin{aligned} \mathbf{n}_1 &= (1,0,1) \\ \mathbf{n}_2 &= (10,0,1) \\ \mathbf{n}_3 &= (0,-5,1) \\ \mathbf{n}_4 &= (-1,-1,1) \end{aligned} \tag{8.17}$$

In this case $\{\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\}$ is a linearly independent set of normal vectors and forms the basis in terms of which $\mathbf{n}_4$ may be expressed. For this specific example it is found that

$$\mathbf{n}_4 = a_1\,\mathbf{n}_1 + a_2\,\mathbf{n}_2 + a_3\,\mathbf{n}_3 \tag{8.18}$$

where $a_1 = 1, a_2 = -0.2, a_3 = 0.2$. By using (8.3) and the linearity property of the scalar product as described in section 8.7.2 above, (8.18) may be re-written as

$$\mathrm{ex}_4 = a_1\,\mathrm{ex}_1 + a_2\,\mathrm{ex}_2 + a_3\,\mathrm{ex}_3 \tag{8.19}$$

By defining an extra coefficient $a_4 = -1$ we can subsequently re-write (8.19) as

$$a_1\,\mathrm{ex}_1 + a_2\,\mathrm{ex}_2 + a_3\,\mathrm{ex}_3 + a_4\,\mathrm{ex}_4 = 0 \tag{8.20}$$

which defines a realisable manifold in excitation space as a hyperplane through the origin and with the orientation given by its normal vector $\mathbf{a} = (a_1, a_2, a_3, a_4)$.

Using the vector **a** GPS calculates the minimum output positions by dropping normals from **G** to the hyperplanar facets on the realisable manifold in output space.

For the neural training problem described here GPS establishes 3 output states corresponding to 3 minima. The output states will be written as row vectors $\mathbf{out} = (\text{out}_1, \text{out}_2, \text{out}_3, \text{out}_4)$ where out_p is the output value for training pattern p. The output states established by GPS are as follows:

1. The global minimum output state $\mathbf{out}_{\text{GM}} = (0.77, 1.0, 0.77, 0.29)$ with an associated goal distance of 0.29.

2. The 1st local minimum output state $\mathbf{out}_{\text{LM1}} = (0.54, 0.82, 0.72, 0.52)$ with an associated goal distance of 0.33.

3. The 2nd local minimum output state $\mathbf{out}_{\text{LM2}} = (0.77, 0.77, 0.0, 0.29)$ with an associated goal distance of 0.77.

where the goal distances are the Euclidean distances from the respective output states to the goal output state $\mathbf{G} = (0.77, 0.77, 0.77, 0.29)$.

The global minimum and local minimum weight states obtained from 100 random initialisations of BP in the weight range [1,+1] for this training problem were performed with the following parameters. A low learning rate of 0.1 and no momentum was used to encourage training to follow the shape of the goal's error-weight surface as closely as possible and

to consequently converge to the attractor of the basin containing the initial random weight state. Each run was given a generous timeout to establish the final output states and weight states with sufficient precision. The global minimum and local minimum output states and weight states obtained are the following:

1. $\mathbf{out}_{\text{GM}} = (0.77, 1.0, 0.77, 0.29)$ with an associated weight state
   $\mathbf{w}_{\text{GM}} = (1.17, -0.23, 0.04)$.
2. $\mathbf{out}_{\text{LM1}} = (0.55, 0.86, 0.72, 0.51)$ with an associated weight state
   $\mathbf{w}_{\text{LM1}} = (0.18, -0.18, 0.03)$.

where the weight states are written as row vectors $\mathbf{w} = (w_1, w_2, w_{\text{bias}})$.

Comparing the GPS output states to the BP output states shows that the global minimum output state and the 1st local minimum output state computed by GPS match the states obtained by BP in multiple runs to a high degree of precision. As before it should be noted that GPS compared to BP only needs one single run to establish all the minimum output states. Furthermore GPS computed a local minimum output state which was not found by BP when initialised in the range [-1,+1].

To verify the 2nd local minimum output state as computed by GPS it was necessary to initialise BP in a weight range of [-2,+2]. This allowed BP to find the additional attractor predicted by GPS. The additional attractor state as found by BP is

- $\mathbf{out}_{\text{LM2}} = (0.77, 0.77, 0.0, 0.29)$ with its associated weight state
  $\mathbf{w}_{\text{LM2}} = (0.0, 2.08, 1.2)$

which again can be seen to match the state computed by GPS.

By comparing the global and local minimum output states produced by BP to the ones predicted by GPS one may surmise that GPS provides a good approximation of the global minimum and local minimum output states for this neural training problem. That is, the semi-linear approximation to the sigmoid activation function is sufficiently precise for this training example.

Using the global minimum output state predicted by GPS for this problem as the goal target for training a neural network results in 100% success in terms of sufficient convergence the global minimum weight state. This was shown for BP and HTH in sections 6.1.1 and 7.3 where training on the global minimum output state was referred to as setting a cheat subgoal chain. The element of cheat has now been removed by establishing the global minimum output state through GPS.

The fact that GPS found an additional minimum which was not encountered through BP in 100 random initialisations in the weight range [-1, +1] is also important. A plausible reason for this is that the weight range was too small to allow the initial state to be in the basin of this attractor. If this additional minimum had been the global minimum rather than a 2nd local minimum then multiple random initialisation in the range [-1,+1] would have failed to find the global minimum regardless of the number of runs. This emphasises the power of a stand-alone method such as GPS which provides a list containing the global minimum output state and other minima for a training problem in a single shot.

Chapter 9

Overall Conclusions and Further Work

9.1 Overall Conclusions

This thesis addresses an issue with regard to training feedforward neural networks with fixed architectures. The issue is that of not being able to guarantee finding the optimal training state when local minima are present on the neural error-weight surface. The reason for this is that mainstream training techniques such as back-propagation, conjugate gradient descent and Levenberg Marquardt follow methods developed in standard numerical analysis which are either based on some sort of gradient descent or on quadratic descent. Such methods may at best guarantee convergence to local minima for a finite number of random initialisations in a finite weight range.

Section 1.2 defined the local minimum problem as the issue to be addressed within this thesis. This problem is that convergence to the global minimum error-weight state may not

be guaranteed in a feasible time when using standard training methods.

Chapter 2 provided a review of standard training techniques for feedforward neural networks which for the most part are local minimisation methods. An exception is simulated annealing which was described in section 2.4 which is intended for being able to find the global optimum. Although simulated annealing is an approach to global optimisation the local minimum problem was shown to still apply because simulated annealing may only guarantee success when its run time tends towards infinity. The presence of the local minimum problem in another area of heuristic search, namely that of symbolic AI, was shown in chapter 3. This chapter also examined the use of subgoals in symbolic AI to break down large problem search spaces in order to facilitate heuristic search.

A target based use of subgoals in neural training was examined in chapter 4 and it was concluded that nearby subgoals may be used to direct training more precisely than when aiming for a more distant goal state. This was shown for a tangent hyperplanes technique (TH) which is based on original work by Antonio Fernandes and Mike Weir ([Fer97] and [WF94]) and was developed further in the neural group here at St Andrews to deal with noisy data. It was also believed though that subgoals in output space set on a linear subgoal chain such as used by ERA [GST97] would not suffice to direct training away from local minima. Chains allowing temporary travel away from the goal were thought to be needed, because local minima imply unrealisable regions in output space.

So in chapter 5, a framework was developed which explains the notion of unrealisable regions in output space. Novel methods for designing training sets which create local

minima were developed and presented and training examples for single layer networks were created. ERA was tested on these training examples and was shown to converge to local minima in 100% of trials. The claims made for ERA were rebutted both empirically and theoretically. The need was shown for more flexible non-linear subgoal chains which may allow local minima to be avoided.

The use of non-linear subgoal chains to successfully direct learning to the global minimum state was shown in chapter 6. The ZIP-model was presented as a way to allow non-linear subgoal chains to be dynamically shaped in order to allow this successful training to occur. An initial design of a key component of the model using output-weight derivatives was implemented and tested. The results showed that non-linear chaining was able to provide travel to the global minimum state for various training examples where standard BP and ERA were not. Although these results were promising it was discovered that the heuristic for aiming towards realisable and progressive output states had scope for improvement in terms of its precision.

Chapter 7 introduced HTH as a method to improve the precision with which realisable and progressive states were targeted. Although the implementation was successful it was furthermore discovered that regardless of the precision, the unrealisability of subgoals close to the goal state introduced an oscillatory behaviour into the training process which prevented HTH from converging to the global minimum state. It was also observed that setting realisable goal states made for successful training which lead to the final design of a system to overcome the local minimum problem.

The global positioning system (GPS) was presented in chapter 8 as a stand alone system which is able to compute the output states corresponding to minima on the neural error-weight surface. The implementation of GPS made use of a semi-linear approximation to the sigmoid activation function. This approximation was found to suffice in terms of the precision with which GPS was able to determine the output states corresponding to the minima for a neural training problem. The use of more segments in the semi-linear approximation to the sigmoid remains as an option to improve the approximation if it is desired. The initial implementation of GPS was for single layer networks and more specifically for the case where the realisable manifold in excitation space is a hyperplane. The design of GPS is expected to generalise to realisable manifolds of lower dimension and to multi-layer networks.

The fundamental merit of a stand alone system such as GPS is that all minimum output states may be feasibly enumerated in a single run. The time to conduct this run may be shown to be in polynomial time with respect to the number of training patterns. Training a network to achieve the weight state corresponding to the desired output state delivered by GPS may be done using whichever neural training technique is preferred. Training may occur on a first time basis with a sufficiently powerful training technique such as TH with subgoal chaining, as the realisable projection of the desired target state is a quantity estimated beforehand through GPS. Recognition of the optimum training state supplied by GPS allows training to be known to be complete when it is reached.

9.2 Further Work

9.2.1 Extending GPS

As mentioned above, the initial implementation of GPS is for the case where the realisable manifold in excitation space is a hyperplane. The first extension of GPS to be investigated would be how to cater for realisable manifolds of lower dimensionality. No reason in principle is obvious why this should not be possible but a concrete method is needed nevertheless.

In the general case for P training patterns and N weights, the realisable manifold in excitation space may be of dimension I, where I is the number of linearly independent normal vectors to the zero-excitation hyperplanes in weight space which form a basis for weight space. Expressing the $(P - I)$ linearly dependent normals in terms of the basis set of normals, $(P - I)$ expressions are obtained which each define individual hyperplanes in $(I + 1)$-dimensional orthogonal subspaces of excitation space. The $(P - I)$ subspaces share the same I excitation axes.

This may be the point at which to start investigations into how the individual hyperplane equations can be used to obtain a useful description of the realisable manifold in excitation space and subsequently in output space. One possible avenue to explore is to use the individual hyperplane equations in combination to generate a basis for the realisable manifold in excitation space. This basis may be converted to bases for the I-dimensional facets on the semi-linear realisable manifold in output space. The bases in output space can be con-

verted to orthonormal bases using Gram-Schmidt orthogonalisation. The goal state **G** may then be projected onto the facets using the orthonormal bases for the facets.

The next thing would be to extend GPS to multi-layer networks, i.e. to those including hidden units. Undoubtedly GPS will not be able to be applied without alteration. In principle though there seems to be no reason for not being able to extend GPS to multi-layer networks. The manifolds of weight states producing zero excitation for each training pattern are now no longer hyperplanes but are non-linear in shape. This means that multiple normals $\mathbf{n}_p$ to the zero-excitation manifolds exist for each pattern p depending on the position along the manifold.

It has been reported in [Fer97] that such manifolds in weight space producing one excitation value for a pattern are slightly curved but without major undulations. The fact that these manifolds are non-linear was encountered by Antonio Fernandes when developing the tangent hyperplanes (TH) technique. The non-linearities were overcome by making TH iterative rather than a one shot procedure.

This option may also be available for GPS, but other options will be initially pursued in order to retain the one shot approach for GPS as much as possible. This may be the point at which to start investigating whether the shape of the non-linear manifolds may be approximated. For a network with a single hidden layer the non-linearity in the manifolds may be seen to arise from the sigmoid activation functions of units in the additional layer. It may be possible then to approximate the shape of the manifolds by a semi-linear function similar to how the current implementation of GPS approximates the sigmoid activation function for

single layer networks. Such an approximation may allow bounded regions in weight space to be constructed within which the normals to each linear approximation of a training pattern's excitation manifold may be expressed in terms of a basis set of normals. Effectively this approach to extend GPS to multi-layer networks would enable GPS to operate on each realisable manifold defined for a region in weight space.

This is merely a broad outline of where investigations would start for extending GPS. Further work is necessary to establish the validity of these approaches.

9.2.2 GPS Applied to Function Minimisation

There is a conceptual similarity between using GPS with neural training and finding a linear least squares solution to a set of equations with SVD which is worth mentioning. The set of equations may be represented as

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{9.1}$$

where the objective is to find the $\mathbf{x}$ which minimises the residue $r = (\mathbf{b} - \mathbf{A}\,\mathbf{x})^{\mathrm{T}}(\mathbf{b} - \mathbf{A}\,\mathbf{x})$.

Part of SVD computes the orthonormal basis of the range of the linear mapping represented by the matrix $\mathbf{A}$ containing the coefficients of the linear equations. This is in order to project the equivalent of the target state $\mathbf{b}$ onto the range. The linear least squares solution for $\mathbf{x}$ is then subsequently found in terms of the projected target state. GPS treats the feedforward neural network as a non-linear mapping and computes the equivalent of a semi-linear approximation to the range of that mapping which is called the realisable manifold. The minimum output states are then found by projecting the goal target state onto the semi-

linear manifold. The non-linear least-squares solution is then found by training the network on the realisable output states found through the projection of the goal state.

Just as SVD is used to obtain linear least squares solutions, it may be seen as fruitful to expand the use of GPS to non-linear function optimisation. One approach may seek to have a neural network approximate a function in its error-weight surface. Finding the global minimum on the neural error-weight surface is then equivalent to finding the global minimum on the approximation to the function which is to be optimised. The use of GPS is then to establish the global optimum output of the neural network and to subsequently train the network to achieve the optimum weight state.

9.2.3 The Use of GPS with regard to Generalisation

As mentioned early on in the thesis in section 1.1.3, some connectionists may see reaching global minimum error on the training set as unnecessary when wishing to obtain good generalisation from a neural network. In order to claim this they may argue that weight states producing good generalisation are seldom very close to those which minimise training error. If generalisation is the main aim of using neural networks the question is then posed of why to bother finding the global minimum error-weight state.

One response to this is that the situation may occur that good generalisation requires a state to be found close to the global minimum. In this case standard techniques may not be able to supply the state and consequently good generalisation may not be achieved. GPS is able to locate all minima in output space on a semi-linear approximation to the

realisable manifold. In combination with neural training techniques this allows adequate search within each basin for generalisation purposes.

The question may also be posed as to how good generalisation may be defined. With a finite amount of data available good generalisation may be defined as using the available data as best possible to obtain the optimal output classification corresponding to the target data. Some connectionists may believe that optimal generalisation is obtained when the optimum output classification to the entire data set is obtained without having trained the neural network on the whole data set. If this is the case, GPS may supply the optimum output classification on the entire data set. This would allow recognition of optimal generalisation during training on the training set. The relation between optimum generalisation and obtaining the optimum output classification on the entire data set needs investigating both theoretically and empirically.

Another potential use for GPS lies with a generalisation technique recently developed by Polhill and Weir in [PW01]. This technique is claimed to guarantee correct generalisation in various cases, as long as the global minimum training error can be found reliably. GPS would be able to allow the global minimum training error to be achieved reliably in a feasible time.

The above points make clear that there may be scope for incorporating GPS, which is capable of calculating the global optimum output response for a neural network, into the area of generalisation. This forms a major part of future research and will aim to establish the exact framework allowing an incorporation of GPS into generalisation techniques.

List of Figures

List of Tables

Bibliography

[AHW97] P Auer, M Herbster, and M K Warmuth. Exponentially many minima for single neurons. In *Advances in Neural Information Processing*, number 7. MIT Press, 1997.

[BE60] Widrow B and Hoff M E. Adaptive switching circuits. In *IRE WESCON Convention Record*, volume 4, pages 96–104. IRE, New York, 1960.

[BF91] I Bellido and G Fernandez. Backpropagation growing networks: Towards local minima elimination. In A Prieto, editor, *Proceedings of IWANN '91*, pages 130–135, Heidelberg, Germany, 1991. Springer.

[BFG95] Monica Bianchini, Paolo Frasconi, and Marco Gori. Learning without local minima in radial basis function networks. *IEEE Transactions on Neural Networks*, 6(3):749–756, May 1995.

[BFGM98] M Bianchini, P Frasconi, M Gori, and M Maggini. *Optimization Techniques*, volume 2 of *Neural Network Systems Techniques and Applications*. Academic Press, San Diego CA, 1998.

[BH89] P Baldi and K Hornik. Neural networks and principal components analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[BL90] E B Baum and K Lang. Constructing hidden units using examples and queries. In *Proceedings of Neural Information Processing Systems*, volume 3, pages 904–910. Denver, 1990.

[BR88] M Brady and R Raghavan. Gradient descent fails to separate. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 649–656, 1988.

[Bra89] M L et al Brady. Back-propagation fails to separate where perceptrons succeed. In *IEEE Transactions on Circuits and Systems*, volume 36, May 1989.

[Fer97] A R Fernandes. *Robustness and Generalisation: Tangent Hyperplanes and Classification Trees*. PhD thesis, School of Computer Science, St.Andrews, 1997.

[FGT92] P Frasconi, M Gori, and A Tesi. Success ad failures of backpropagation: A theoretical investigation. Technical report, Dipartmento di Sistemi e Informatica, Universita di Firenze., Via di Santa Marta 3, 50139 Firenze, Italy., 1992.

[GL82] Les Goldschlager and Andrew Lister. *Computer Science, A Modern Introduction*. Prentice-Hall, 1982.

[GST97] D Gorse, A J Shepherd, and J G Taylor. The new era in supervised learning. *Neural Networks*, 10(2):343–352, 1997.

[Jef85] Alan Jefrey. *Mathematics for Engineers and Scientists*. Van Nostrand Reinhold (UK) Co. Ltd, 3rd edition, 1985.

[KGV83] S Kirkpatrick, C Gelatt, and M Vecchi. Optimisation by simulated annealing. *Science*, (220):671–680, 1983.

[LS93] G F Luger and W A Stubblefield. *Artificial Intelligence, Structures and Strategies for Complex Problem Solving*. World Student Series. Addison Wesley, 2nd edition, 1993.

[LW88] K Lang and M J Witbrock. Learning to tell two spirals apart. In *Proceedings of the Connectionist Models Summer School*, pages 52–59. Morgan Kaufamnnn, 1988.

[LW99] J P Lewis and M K Weir. Subgoal chaining and the local minimum problem. In *Proceedings of the International Joint Conference on Neural Netwrorks*, 1999.

[MP88] M Minsky and S Papert. *Perceptrons - Expanded Edition*. MIT Press, Cambridge MA, 1988.

[Nob69] Ben Noble. *Applied Linear Algebra*. Prentice-Hall, 1969.

[Pol71] E Polak. *Computational Models in Optimization*. Academic Press, New York, 1971.

[Pre94] W H Press. *Numerical Recipes*. Cambridge University Press, 2nd edition, 1994.

[PW01] J G Polhill and M K Weir. An approach to guaranteeing generalisation in neural networks. *Neural Networks*, 14(8):1035–1048, October 2001.

[RK91] Elaine Rich and Knight Kevin. *Artificial Intelligence*. McGraw Hill, 2nd edition, 1991.

[RM86] D E Rummelhart and J L McClelland. *Parallel Distributed Processing*, volume 1. The MIT Press, London England, 1986.

[RN95] Stuart J Russel and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, London, 1995.

[SKB96] Ida G Springkhuizen-Kuyper and Egbert J W Boers. The error surface of the 2-2-1 xor network: Stationary points with infinite weights. Technical report, Department of Computer Science, Leiden University, Oct 1996.

[SS89] E D Sontag and H J Sussmann. Back-propagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems*, (3):91–106, 1989.

[SS91] E D Sontag and H J Sussmann. Back propagation separates where perceptrons do. *Neural Networks*, 4:243–249, 1991.

[WC90] M K Weir and Li H Chen. Training and generalisation using continuous back-propagation. Technical report, School of Computer Science, University of St. Andrews, Scotland, 1990.

[Wei91] M K Weir. A method for self-determination of adaptive learning rates in back propagation. *Neural Networks*, 4:371–379, 1991.

[Wei00] M K Weir. Personal conversation with Mike Weir, 2000.

[WF94] M K Weir and A R Fernandes. Tangent hyperplanes and subgoals as a means of controlling direction in goal finding. In *Proceedings of the World Conference on Neural Networks*, volume 3, pages 438–443, 1994.

[WLC93] M K Weir, S Lansley, and A Clark. *Wintermute*. University of St Andrews, 1993.

[WLM00] M K Weir, J P Lewis, and G Miligan. Using tangent hyperplanes to direct neural training. In *Proceedings of Second International ICSC Symposium on Neural Computation*, 2000.