NOVEL SECOND-ORDER TECHNIQUES AND GLOBAL OPTIMISATION METHODS FOR SUPERVISED TRAINING OF MULTI-LAYER PERCEPTRONS

Adrian John Shepherd

Thesis submitted for the degree of
Doctor of Philosophy in the University of London

September 1995

Department of Computer Science,
University College London

ABSTRACT

Conventional training methods for multi-layer perceptrons (MLPs), derived from the traditional backpropagation algorithm, have three serious inadequacies: convergence to a solution is frequently slow; they do not always converge to the desired global solution; and their performance is highly dependent on the setting of one or more user-defined parameters. A growing body of research indicates that second-order training methods, derived from classical optimisation theory, offer substantial improvements in training speed and a reduced sensitivity to initial parameter settings. However, experiments conducted for this research suggest that most second-order methods have worse global convergence properties than conventional methods. On the other hand, training methods that are designed to have better global convergence characteristics than conventional methods - for example, stochastic training methods - are typically as slow or slower than conventional methods.

The aim of this research is to develop MLP training algorithms that are both fast *and* 'globally-reliable' by combining second-order methods with a novel deterministic strategy for global optimisation, Expanded Range Approximation (ERA). Unlike most stochastic methods for global optimisation, the implementation of ERA with a second-order algorithm is trivial. When tested on benchmark training tasks, hybrid second-order/ERA algorithms (with appropriate parameter settings) were considerably faster and converged to a global minimum as or more frequently than conventional algorithms.

This thesis also gives practical guidelines for the efficient implementation of second-order training algorithms, with particular attention paid to factors that affect the probability of a given algorithm attaining a global minimum. In addition, a novel line-search algorithm is presented that offers an efficient compromise between the reliability of safeguarded polynomial interpolation and the speed of backtracking line searches; used as part of a second-order training algorithm, only a single function evaluation is required per training iteration in the best case.

CONTENTS

ACKNOWLEDGEMENTS

> The British postgraduate student is a lonely forlorn soul ... for whom nothing has been real since the Big Push.
>
> David Lodge[*] , *Changing Places*

I would like to thank the following for making this thesis possible, and for saving me from the narrow existence of 'the British postgraduate student':

- my first-supervisor, Dr Denise Gorse, for her enthusiastic guidance throughout the production of this thesis;
- my second-supervisor, Dr Simon Arridge, for sharing his knowledge and experience in the field of classical optimisation;
- Templer Hart, for the generous gift of computer equipment, and for acting as an unpaid PC help-desk operator;
- my wife, Wendy, and my family, for their unfailing support and encouragement.

[*] David Lodge was a postgraduate student at University College London in the late 1950s.

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

1. INTRODUCTION

Conventional training methods for multi-layer perceptrons (MLPs), derived from the traditional backpropagation algorithm, have three serious inadequacies: convergence to a solution is frequently slow; they do not always succeed in converging to a desired (and achievable) solution, irrespective of the number of training iterations allowed; and they tend to be highly sensitive to the choice of input-parameters, set heuristically by the user. A growing body of research (reviewed in section 4.1) indicates that second-order training methods, derived from classical optimisation theory, offer substantial improvements in training speed as well as a greatly reduced sensitivity to the choice of initial parameters. However, experiments conducted for this research suggest that most second-order methods have worse global convergence properties than conventional methods; tested on benchmark tests with known local minima, second-order methods failed to converge to the desired global solution as frequently as conventional methods. On the other hand, training methods that are designed to have better global convergence characteristics than conventional training methods - for example, stochastic training methods - are typically as slow or slower than conventional algorithms.

The underlying aim of this research is the development of MLP training algorithms that are both faster *and* more 'globally-reliable' than conventional training methods. The approach adopted here has been to combine fast second-order classical algorithms with a novel deterministic strategy for global optimisation - Expanded Range Approximation, or ERA for short. When tested on benchmark tasks, hybrid second-order/ERA training algorithms (with appropriate parameter settings) were considerably faster and converged to a global minimum as or more frequently than conventional training algorithms.

This thesis is in four major sections. Chapter 2 provides an overview of MLP training and the backpropagation training algorithm. A key perspective, introduced in section 2.2, is to view the training of an MLP as an optimisation process that involves the minimisation of a multi-dimensional error surface. Chapter 3 is devoted to classical optimisation methods - in particular, second-order methods for the minimisation of multi-dimensional nonlinear functions. The chapter introduces a novel line-search algorithm (section 3.2.5) that offers an efficient compromise between the reliability of safeguarded polynomial interpolation (section 3.2.2) and the speed of backtracking line searches (section 3.2.4). Chapter 4 contains detailed experimental results for traditional and classical training algorithms applied to a small number of benchmark tests (section 4.2). In contrast to earlier research in this field, the global convergence properties of different classical training algorithms are rigorously assessed, and practical guidelines given about how to maximise the probability that such algorithms will attain a global minimum (section 4.4). Finally, chapter 5 is concerned with the ERA method for global optimisation. Unlike the majority of stochastic methods for global optimisation (section 5.1.1), the implementation of ERA with a second-order algorithm is trivial; the resultant hybrid second-order/ERA training algorithms are shown to be highly effective - in terms of both training speed and global reliability - when applied to the benchmark tests used in this research.

2. MULTI-LAYER PERCEPTRON TRAINING

The class of neural network considered in this research is the multi-layer perceptron (MLP)[1]. MLPs have a wide range of applications, including pattern classification and function-learning [Lisboa, ed., 1992]. MLP training involves adjusting the network so that it is able to produce a specified output for each of a given set of input patterns; since the desired outputs are known in advance, MLP training is an example of *supervised learning.*

This chapter sets the context for the research presented in the remainder of this thesis: section 2.1 considers the physical properties and dynamics of MLP training, section 2.2 the essential characteristics of MLP training tasks and their implications for training algorithm design, and section 2.3 the properties of error-backpropagation, the dominant training paradigm for MLPs.

2.1 Introduction to MLPs

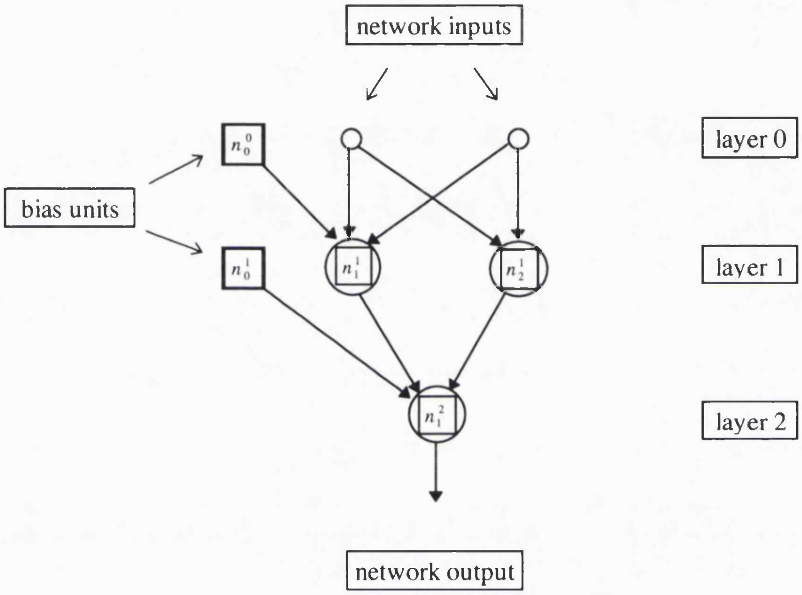
2.1.1 The MLP architecture

The MLP architecture consists of units or nodes arranged in two or more layers. (The input layer, which serves only to distribute the input from each pattern, is not counted.) Some of the nodes are connected by real-valued weights, but there are no connections between nodes in the same layer. For notational convenience, it is assumed throughout that MLP architectures are of a 'standard' form, with adjacent layers fully-connected but no connections between non-adjacent layers. Such an MLP consists of L layers with N^l nodes in each layer ($l = 0,\dots,L$), with $l = 0$ denoting the input layer. The notation for a single node is n_i^l ($i = 1,\dots,N^l$). The thresholds for the weighted sum of inputs to each node (given by Eq. 2.1 below) are treated uniformly by adding an extra node with a fixed output of 1.0 to all but the output layer. This node - called the *bias unit* - is denoted n_0^l

[1] As so often in the field of neural networks, there is little standard terminology. Alternative terms for MLPs are 'feed-forward neural networks' (various), 'multilayered neural networks' (MLNs) [Gori & Tesi, 1992], and 'feature-based mapping neural networks' or simply 'feature networks' [Hecht-Nielsen, 1990].

(for $l \neq L$). Network weights can be represented in terms of the nodes they connect, thus weight w_{ij}^{l} connects nodes n_j^{l-1} and n_i^l. However, it will often be more convenient to consider weights in terms of the weight vector **w** comprising all W weights in the network, with a single weight denoted w_i ($i = 1,...,W$).

Figure 1 - Minimal 2-2-1 MLP architecture, suitable for learning the XOR problem



The number of nodes in the input and output layers is determined by, respectively, the pattern-size and target-size of the chosen training task. MLPs are typically trained using a fixed *training set* of P training pairs, with each *training pair* comprising two real-valued vectors - a pattern $\mathbf{p}_q$ ($q = 1,...,P$) and a corresponding target (desired output) $\mathbf{t}_q$. Individual pattern and target elements are denoted $p_{i,q}$ ($i = 1,...,N^0$) and $t_{j,q}$ ($j = 1,...,N^L$) respectively. The output $y_{i,q}^0$ of input node i is simply $p_{i,q}$ for pattern q (except for y_0^0, the fixed output of the bias unit). For non-input node n_i^l, the output is given by the weighted sum

Eq. 2.1
$$y_{i,q}^{l} = s\left(\sum_{j=0}^{N^{l-1}} w_{ij}^{l} y_{j,q}^{l-1}\right) \quad \text{for } l > 0 \ ,$$

where the activation or squashing function s is typically the sigmoid:

Eq. 2.2 $$s(x) = \frac{1}{1+e^{-x}} \ .$$

The layers between the input and output layers are known as *hidden layers*. The number of hidden layers and nodes has a major impact on MLP training: too few, and the network will be unable to learn the problem; too many, and the network may take excessively long to train and have poor *generalisation* capabilities - a measure of the network's ability to classify patterns which share the same general features as, but are not identical to, patterns in the training set. Numerous schemes have been developed which calculate an appropriate number of hidden nodes from the training data or which adapt the architecture (i.e. add or 'prune' nodes) during training [Brent, 1991] [Hirose, Yamashita & Hijiya, 1991] [Santini, 1992]. (Upper and lower bounds on the number of nodes are given in [Huang & Huang, 1991].) None of these schemes have been adopted for this research, since the appropriate architecture was known in advance for the chosen benchmark tests, but all the training methods presented here could easily be modified to incorporate such schemes.

2.1.2 MLP training

MLP training is an iterative process which involves, at each iteration or *epoch*, the calculation of network outputs for each pattern in the training set and the adjustment of network weights according to the disparity between actual and desired outputs. Prior to training, the weights are initialised to small random values - small to prevent *saturation* (where one or more hidden nodes is highly active or inactive for all patterns and therefore insensitive to the training process) and random to break symmetry. The choice of initialisation range can have a significant impact on training performance (see [Kolen & Pollack, 1990]).

The degree of success at each epoch can be measured by applying an *error function* (or energy function) E of all the parameters (weights) of the network. This research uses the traditional *mean-squared error* (MSE) function, defined by

Eq. 2.3 $$E = \frac{1}{2PN^L} \sum_{p=1}^{P} \sum_{i=1}^{N^L} \left(t_{i,p} - y_{i,p}^L \right)^2 .$$

The advantages of Eq. 2.3 are consistency with the majority of MLP research and compatibility with the nonlinear least squares methods considered in sections 3.1.4 and 3.6. However, the MSE function is a 'greedy' error function; the number of misclassifications can increase from one iteration to the next despite a reduction in *E*. Other error functions, such as the exponential, frequently perform better in practice [Møller, 1993e, 65-70 & 149-161]. An investigation of the impact of different error functions on the training methods presented here will be the subject of future research.

MLP training is deemed to be successful when *E* becomes 'acceptably' small. Precisely how small is application-specific, but a close approximation to zero is generally undesirable since an MLP's ability to generalise decreases with *overtraining* [Hecht-Nielsen, 1990, 116].

The key factor in the dynamics of MLP training is the role of the hidden nodes. A hidden node that duplicates the function of another hidden node is *redundant,* i.e. makes no useful contribution to the training process. A mathematical analysis by Annema et al. of the dynamics of MLP training indicates that the build-up and dissipation of hidden-node redundancy is an integral part of the training process [Annema, Hoen & Wallinga, 1994]. The core analysis, which holds for a two-layer MLP with two hidden nodes and 'very small' initial weights, describes three distinct training phases; for MLPs of arbitrary size, this three-phase analysis can be applied iteratively to smaller and smaller clusters of redundant hidden nodes.

In phase one of the analysis redundancy builds up in the hidden layer to the point where it is 'approximately reducible to one neuron' and the entire network can be linearized, i.e. 'all neurons are activated in the approximately linear middle region' [Annema, Hoen & Wallinga, 1994]. At this stage both the attractors in weight space and the vectors of input weights are near-identical for all hidden nodes. In phase two the attractors remain near-identical, but the network can no longer be linearized. By the end of this transitional phase the cluster of redundant hidden nodes starts to split in two. Phase three consists of the division of redundant hidden-layer nodes into two distinct clusters. The input weight vectors associated with each cluster now converge towards different attractors in weight space.

The inability of an MLP to eliminate hidden-node redundancy (i.e. to proceed beyond phase two of the preceding analysis) is a frequent cause of training failure (see section 2.2.1).

2.2 Error Surfaces and Local Minima

In order to design efficient and reliable training algorithms, it is essential to gain an understanding of the principal characteristics of MLP training tasks and their implications for different training strategies. The perspective adopted here is that of function optimisation - that is, the minimisation of the MLP error function E. Viewed in these terms, MLP training is an *error-minimisation* or *optimisation* process (for which many techniques have been developed outside the neural network field), and each training task defines a multi-dimensional non-negative *error surface* (section 2.2.1), formed by plotting the value of E for all (reasonable) settings of the MLP weight vector $\mathbf{w}$. This approach has two important benefits: it aids visualisation of the training process through analogy with a three-dimensional landscape; and it leads to numerically-testable definitions for many of the conditions encountered during training.

The lowest points on the error surface are known as *global minima.* (Typically MLP error surfaces have multiple global minima, each of which is a surface rather than a single point.) If the lowest point of a 'basin' in the error surface has a higher error than that associated with a global minimum, it is termed a *local minimum.* Since the impact of local minima on MLP training is a major theme of this research, they are considered separately in section 2.2.2.

2.2.1 The MLP error surface

For any viable combination of MLP architecture, test problem, and error function, there is a corresponding error surface (or energy surface) with n+1 dimensions for an MLP with n

weights. The precise shape of the error surface is problem- and architecture-specific; since it is impractical to produce a map of an error surface that is both detailed and extensive (even for small training problems and architectures), it is no surprise that the properties of MLP error surfaces remains a subject for debate [Hecht-Nielsen, 1990, 131].

It is worth stressing that the common practice of inferring the presence of landscape features based on training error-curve information alone is highly suspect. There is, for example, a tendency to say an MLP has become trapped in a local minimum whenever the training curve flattens out at a comparatively high level, despite the fact that there are equally plausible causes for this behaviour which are unrelated to local minima. This is not simply a case of pedantry about the use of terminology; the precise cause of a given training characteristic may have profound implications for the efficacy of an attempted solution.

What reliable evidence there is (for example, from contour plots of small sections of an error surface) suggests that most MLP error surfaces share a number of broad characteristics: a high degree of *smoothness*; 'a multitude of areas with shallow slopes in multiple dimensions simultaneously' [Hecht-Nielsen, 1990, 131] ('plateaus'); convolutions and ellipses of high eccentricity ('valleys'); and many 'basins' or minima. MLP error surfaces typically have multiple global minima, owing to permutations of the weights that leave the MLP input-output function unchanged; an assessment of the prevalence of local minima is deferred to section 2.2.2.

'Plateaus', 'narrow valleys', and local minima often prove to be serious obstacles to successful training. If an MLP encounters a region with very shallow gradients, it can take many training epochs before a significant reduction in E is made - the network is said to be stuck in a *temporary minimum* [Annema, Hoen & Wallinga, 1994]. Training algorithms derived from the steepest descent method are prone to slow training in narrow valleys (see section 3.1.1). And algorithms which do not allow an increase in E at any epoch are prone to becoming trapped in local minima, i.e. no amount of additional training will enable the MLP to make further downward progress. In terms of the analysis of training dynamics presented in section 2.1.2, both local and temporary minima are closely related to the presence of redundant hidden nodes in the network. (Local minima are known to have several different physical correlates in the context of MLP training; the three most frequently encountered are redundant hidden nodes, hidden node saturation, and ‘dead regions’ of weights space where all hidden nodes are inactive [Wessels, Barnard & van Rooyen, 1990].)

The characteristics of MLP error surfaces give a good indication of the kinds of strategy that are likely to engender efficient and reliable training: the smoothness of the error surface suggests that classical optimisation with derivatives (chapters 3 and 4) will be effective; rounding errors, floating-point precision and the choice of termination criteria are likely to be important in nearly flat regions; some methods are far less prone to slow progress in ‘narrow valleys’ than others; and, if there are local minima, global minimisation strategies (chapter 5) may be necessary to ensure an acceptable probability of training success.

2.2.2 Local minima

For the purposes of this research, the term ‘local minimum’ is used in a rigorous mathematical sense. If, for a given combination of MLP architecture and training task, the error function E(**w**) is *twice-continuously differentiable* (i.e. the second-derivatives of *E* are continuous), it is possible to define useful theoretical conditions for a point $\mathbf{w}_*$ to be a *minimum* of *E* in terms of the *gradient vector* g($\mathbf{w}_*$)

Eq. 2.4
$$g_i = \frac{\partial \mathrm{E}(\mathbf{w})}{\partial w_i}$$

and Hessian matrix G($\mathbf{w}_*$)

Eq. 2.5
$$G_{ij} = \frac{\partial^2 \mathrm{E}(\mathbf{w})}{\partial w_i \partial w_j} .$$

If g($\mathbf{w}_*$) is zero, $\mathbf{w}_*$ is a *stationary point* - which means it is either a minimum, a maximum or a *saddle point.* Stationary point $\mathbf{w}_*$ is definitely a minimum if G($\mathbf{w}_*$) is *positive definite* (i.e. all the eigenvalues of **G** are strictly positive), and may be a minimum if G($\mathbf{w}_*$) is *positive semi-definite* (i.e. all the eigenvalues of **G** are non-negative). Minimum $\mathbf{w}_*$ is a *global minimum* if E($\mathbf{w}_*$) ≤ E(**w**) for all **w**, otherwise $\mathbf{w}_*$ is a *local minimum.*

The prevalence of local minima in MLP error surfaces remains a matter of debate. Local minima are known to occur with specific test problems [McInerney et al., 1989] [Lisboa & Perantonis, 1991]. On the other hand, local minima *cannot* occur if the training task is *linearly separable* [Gori & Tesi, 1992], if there are as many hidden nodes as patterns in the training set [Poston et al., 1991], or if the number of patterns is less than or equal the number of pattern elements [Yu, 1992] - assuming, in each of these cases, that the chosen architecture is capable (with some set of weights) of learning the task in question. Unfortunately, none of these results give much guidance to real-world applications, for which there is precious little hard evidence on either side of the debate. At present, it is probably reasonable to conclude that for many (if not all) realistic applications local minima present a serious obstacle to successful training.

Every minimum has an associated *basin of attraction* - a region surrounding the minimum from which it is only possible to escape by passing over higher ground (or by deforming the error surface in some way). A number of contrasting approaches have been proposed for reducing the likelihood of an MLP getting trapped in the attractive basin of a local minimum, including: stochastic methods (section 5.1.1); deterministic strategies, such as homotopic methods (section 5.1.2); changing the error function [Solla, Levin & Fleisher, 1988]; weight initialisation schemes [Wessels & Barnard, 1992]; and schemes for dynamically changing the number of hidden nodes [Hirose, Yamashita & Hijiya, 1991]. As these schemes apply to different aspects of the training process, it is likely that the 'optimal' strategy will combine several of these schemes in a single algorithm.

2.3 Backpropagation

2.3.1 Backpropagation - an overview

The vast majority of MLP research has used a version of the *backpropagation* (BP) training method (rediscovered and disseminated to a wide audience by [Rumelhart, Hinton & Williams, 1986]), and it remains the most widely used technique. BP is the benchmark against which all other training methods are judged.

In essence, BP implements *gradient* or *steepest descent* (section 3.1.1) for an MLP. At each epoch k the gradient g($\mathbf{w}_k$) is calculated and the weights updated according to the simple rule

Eq. 2.6 $$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{g}_k, \quad \text{for } \eta > 0 \quad ,$$

where η is a constant heuristically-chosen *training rate,* typically set in the range [0, 1]. A reduction in total network error E at each epoch is guaranteed so long as the gradient is greater than zero and η is sufficiently small. The calculation of the gradient by backpropagation is implemented in two phases - a forward pass and a backward pass. The *forward pass* generates the network outputs for pattern p through the calculation of each $y_{i,p}^{l}$, from layer $l = 1$ to $l = L$, according to the weighted sum of Eq. 2.1. The *backward pass* calculates the partial contribution E_p of pattern p to the total network error E, and the corresponding partial gradient $\mathbf{g}_p$, with elements $\partial E_p / \partial w_{ij}^{l}$. These elements are calculated by applying the following rule from layer $l = L$ to $l = 1$:

Eq. 2.7 $$\partial E_p / \partial w_{ij}^{l} = \delta_{i,p}^{l} y_{j,p}^{l-1} \quad ,$$

where the error term δ is given by

Eq. 2.8 $$\delta_{i,p}^{L} = \left(t_{i,p} - y_{i,p}^{L}\right) y_{i,p}^{L} \left(1 - y_{i,p}^{L}\right)$$

$$\delta_{j,p}^{l} = y_{j,p}^{l} \left(1 - y_{j,p}^{l}\right) \sum_{i=1}^{N^{l+1}} \delta_{i,p}^{l+1} w_{ij}^{l+1}, \quad \text{for } l < L \ .$$

A single training epoch consisting of P forward passes interleaved with P backward passes.

The form of BP which conforms exactly to Eq. 2.6 is called *batch* or *off-line BP.* Batch BP has proved satisfactory with many problems, but has several important drawbacks: it is often slow to reach a satisfactory error level, and particularly slow when confronted with two common features of the MLP error surface - flat regions and 'narrow valleys'; it is prone to getting trapped in local minima (and will also converge to saddle points); and training performance is sensitive to the choice of training rate. A wide variety of heuristic modifications to standard batch BP have been devised in an attempt to overcome these difficulties, such as on-line training strategies, the addition of 'momentum' to the weight updates, the adaptation of the MLP training rate, and the addition of noise to the weights. All these techniques have reported strengths when applied to specific problems, but tend to be ineffective in general. Moreover, many require an additional problem-specific parameter (or parameters) to be set heuristically by the user.

One implication of this diversity of practical BP implementations is that the choice of a 'fair' BP benchmark is extremely difficult. This thesis considers only two of the most widely-used modifications to the standard BP algorithm - on-line training (section 2.3.2) and momentum (section 2.3.3). The steepest descent algorithm, a 'classical' implementation of batch BP that sets the training rate optimally at each epoch, is described in section 3.1.1.

2.3.2 On-line backpropagation

On-line BP[2] differs from batch BP in that the weights are updated at the end of each backward pass (i.e. P times per epoch), rather than once every P backward passes (i.e. once per epoch). The weight update rule for on-line BP is

Eq. 2.9
$$\mathbf{w}_{p+1}^{k} = \mathbf{w}_{p}^{k} - \eta \mathbf{g}_{p}^{k}, \text{ for } \eta > 0.$$

Typically, the P patterns are presented to the network in random order. If the training rate η tends to zero, on-line BP can be regarded as an approximation to batch BP; however, for practical settings of η the two methods diverge.

In theory, on-line BP has several potential disadvantages compared to batch BP:

- it is not guaranteed to 'make progress' (i.e. reduce E) at each training epoch;
- it requires slightly more computational effort per epoch than batch BP;
- the optimal training rate for on-line BP is poorly understood (cf. the batch-mode steepest descent method);

[2] Alternatives to the term 'on-line' include 'stochastic' (various), 'immediate update' [Kinsella, 1992, 28], 'jump every time' [Hecht-Nielsen, 1990, 136], 'local learning' [Annema et al., 1994], and 'pattern mode' [Gori & Tesi, 1992, 78].

- it is much more difficult to analyse.

Nevertheless, on-line BP has several practical advantages over batch BP:

- on-line BP is an example of a stochastic process that can prevent an MLP from getting trapped in a local minimum;
- if the training set contains redundant information, the more frequent weight updates of on-line BP often prove more efficient [Møller, 1993b] (although it may be possible to remove redundant information by pre-processing the training set [Battiti, 1992]);
- on-line training is essential if the full complement of training patterns is not known at the start of training.

For these reasons, on-line BP can be regarded as *the* backpropagation benchmark.

There are two ways in which on-line BP can be regarded as a stochastic process. Firstly, the total network error E may rise at one or more epochs such that the network is able to escape from the basin of attraction of a local minimum. Secondly, the shape of the MLP error surface is not constant with on-line BP; local minima are meta-stable states which slowly decay to the global minimum, at a rate of $\tilde{t}/t$ for some constant $\tilde{t}$ that is 'much larger than the typical time scale $1/t$ to reach equilibrium inside an attractive region' [Kappen & Heskes, 1992, 72]. In its 'traditional' form - with weights updated after the presentation of every pattern - on-line BP makes no attempt to regulate the amount of stochastic 'noise' added to the system at each training epoch. However, the term 'on-line training' is often used in a more general sense to encompass training algorithms which update the weights after a subset n ($1 \leq n < P$) of the full training set has been presented to the network; by varying the size and membership of the subset at each training iteration it is possible to regulate the amount of stochastic noise. A wide variety of strategies have been devised for this purpose, ranging from simple heuristic schemes that gradually increase n as training progresses to complex sampling and validation schemes (see section 5.1.1).

2.3.3 Backpropagation with momentum

Backpropagation with momentum uses a modified version of the standard BP weight update formula (given by Eq. 2.6), as follows:

Eq. 2.10 $$\Delta \mathbf{w}_{k+1} = -\eta \mathbf{g}_k + \alpha \Delta \mathbf{w}_k, \quad \text{for } \eta > 0 \text{ and } 0 \le \alpha < 1 \;,$$

where α is known as the *momentum* term. (With momentum turned 'off', i.e. with α=0, Eq. 2.10 is equivalent to the standard update of Eq. 2.6.) Experience shows that the addition of momentum can significantly speed up the BP training algorithm, attributable to its impact in precisely those regions of the MLP error surface where the backpropagation algorithm performs badly - 'plateaus' and 'narrow valleys'; the momentum term accelerates convergence in flat regions by a factor that approaches $\frac{1}{1-\alpha}$ as the number of epochs (k) gets large, and reduces the number of oscillations in a narrow valley - i.e. reduces the 'narrow valley effect' (see section 3.1.1) - by averaging out the components of the gradient which alternate in sign [Watrous, 1987]. As Møller points out [1993e, 19], batch BP with momentum can be viewed as an approximation to conjugate gradient methods (section 3.5) - the important difference being that conjugate gradient methods chose parameters η and α automatically at each iteration, whereas batch BP with momentum sets η and α to fixed heuristic values.

An alternative weight update to Eq. 2.10, given by

Eq. 2.11 $$\Delta \mathbf{w}_{k+1} = -(1-\alpha)\eta \mathbf{g}_k + \alpha \Delta \mathbf{w}_k, \quad \text{for } \eta > 0 \text{ and } 0 \le \alpha < 1,$$

proved consistently slower than the Eq. 2.10 update in the experiments conducted for this research.

3. CLASSICAL OPTIMISATION

3.1 Introduction to Classical Methods

Unconstrained nonlinear optimisation is a mature branch of numerical analysis concerned with the minimisation of multi-dimensional functions[1]. For a wide class of smooth convex functions, convergence is guaranteed. However, classical optimisation is concerned only with local optimisation.

All the optimisation methods considered in this chapter share important characteristics: all derive, algebraically, from the Taylor-series expansion of a smooth function *f* in the neighbourhood of an arbitrary point **x**

Eq. 3.1
$$f(x+s) = f(x) + g(x)^T s + \frac{1}{2} s^T G(x) s + \dots ;$$

all are iterative descent algorithms (i.e. minimum $\mathbf{x}_*$ is located in a series of steps, with $f(x_{k+1}) \le f(x_k)$ at each step); and all are hybrid methods, which fall somewhere between the steepest descent method (section 3.1.1) and Newton's method (section 3.1.2).

In order to compare the theoretical performance of these methods, it will be useful to consider their *global* and *local convergence* properties. In the context of the convergence of classical algorithms, the terms 'local' and 'global' have a different meaning to that introduced in section 2.2. A method is said to be globally convergent if, for an arbitrary smooth convex function, it is guaranteed to converge (eventually) to a minimum from (almost) any starting position. (The global convergence properties of algorithms associated with convex functions are applicable to non-convex functions inside the basin of attraction of a minimum.) A method's local convergence rate, on the other hand, is its anticipated rate of convergence close to a minimum. Convergence characteristics act as a rough guide to a method's performance, but they should be treated with caution; they require conditions that do not apply in general, and the effect of rounding error is ignored.

[1] Good general surveys of the field are provided by [Fletcher, 1980], [Gill, Murray & Wright, 1981], [Luenberger, 1984], and [Wolfe, 1978].

The following survey of classical methods is necessarily selective; the emphasis is on tried and tested methods which are fast and reliable, and can be readily implemented in a neural network context.

3.1.1 The linear model and steepest descent

Optimisation methods which derive from the *linear model*

Eq. 3.2 $$f(\mathbf{x}+\mathbf{s}) \approx f(\mathbf{x}) + g(\mathbf{x})^T \mathbf{s}$$

(i.e. all but the first two terms of Eq. 3.1 are ignored) are termed *first-order methods*. The pre-eminent example is *steepest descent* (SD)[2], the longest- and most widely-known optimisation technique of all. SD sets search direction $\mathbf{s}_k$ to the negative gradient $-\mathbf{g}_k$ at each iteration, i.e.

Eq. 3.3 $$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k \ .$$

This is equivalent to the standard BP update of Eq. 2.6 except that α_k is chosen to minimise $E(\mathbf{x}_k - \mathbf{g}_k)$ rather than set to the fixed heuristic value of the BP training rate (η).

SD is easy to implement and requires, on average, the least computational effort per iteration of any classical method. However, SD is often both inefficient and unreliable. Ellipses of large eccentricity can produce the so-called ‘narrow valley effect’ (Figure 2), with the path oscillating back and forth along the local gradient. Successive SD search directions have a tendency to interfere, i.e. a minimisation in one direction can spoil the minimisation previously achieved in other directions.

If SD is applied to a quadratic function Q such as

Eq. 3.4 $$Q(\mathbf{x}) = \mathbf{x}^T\mathbf{b} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} \ ,$$

where matrix **A** is symmetric and positive definite, the theoretical upper bound on the convergence rate is given by the *convergence ratio r*,

[2] Steepest descent is also commonly referred to as 'gradient descent'.

Eq. 3.5
$$r = \left(\frac{c-1}{c+1}\right)^2$$

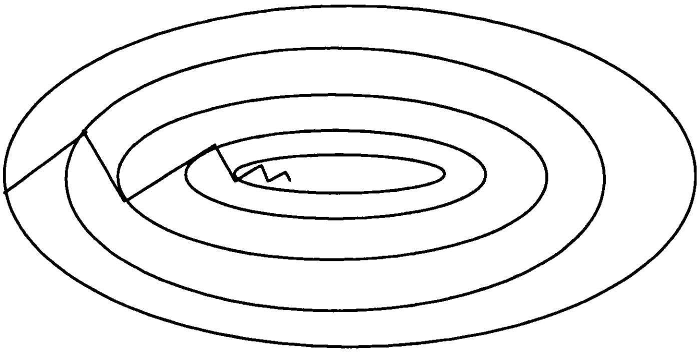
$$c = \frac{v_{\max}}{v_{\min}} ,$$

where c is the *condition number* of **A**, and $v_{\max}$ and $v_{\min}$ are, respectively, the largest and smallest eigenvalues of **A** [Luenberger, 1984, 219]. This amounts to an arbitrarily slow rate of *linear convergence* (which becomes slower as **c** increases). Moreover, SD is sensitive to rounding errors, which can cause termination far from the solution; global convergence to a stationary point of f cannot be guaranteed in practice.

Both the convergence ratio r and the steepest descent direction **s** are sensitive to the scale of **x**. The feasibility of changing the scale of **x** so that c is reduced (with a corresponding improvement in the convergence characteristics of the SD algorithm) is considered in section 3.1.5.

Figure 2 - Steepest descent and the 'narrow valley effect'

Note: successive descent directions are perpendicular to each other and to the tangent planes of the surface contours.



3.1.2 The quadratic model and Newton's method

With the exception of steepest descent, all the classical methods considered here are *second-order methods*, based on the *quadratic model*

Eq. 3.6 $$f(\mathbf{x}+\mathbf{s}) \approx f(\mathbf{x}) + g(\mathbf{x})^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T\, G(\mathbf{x})\mathbf{s} \ .$$

The theoretical convergence rate and practical performance of second-order methods are generally superior to those of first-order methods (provided *f* is sufficiently smooth). The success of the quadratic model derives from the fact that quadratic functions are a good approximation to general functions near a minimum. If successive search directions satisfy

Eq. 3.7 $$\mathbf{s}_i^T \mathbf{G}\mathbf{s}_j = \mathbf{0}, \text{ for } i \neq j \ ,$$

that is, the directions are *mutually conjugate* with respect to the Hessian matrix, they will (unlike successive steepest descent directions) be approximately non-interfering, with a correspondingly fast rate of convergence.

The straightforward implementation of the quadratic model, known as *Newton's method*, generates each search direction $\mathbf{s}_k$ as follows:

Eq. 3.8 $$\mathbf{s}_k = -\mathbf{G}^{-1}\mathbf{g}_k \ .$$

If $\mathbf{G}$ is positive definite, the $\mathbf{s}_k$ given by Eq. 3.8, commonly denoted $\mathbf{s}_\kappa^N$, has a number of important properties [Dennis & Schnabel, 1983]: $\mathbf{s}_\kappa^N$ uniquely minimises the quadratic model at $\mathbf{x}_k$ and is guaranteed to be a *descent direction*, i.e. $\mathbf{s}_\kappa^N$ satisfies

Eq. 3.9 $$\mathbf{g}_k^T \mathbf{s}_k < \mathbf{0} \ ;$$

it defines both the direction (the *Newton direction*) and step-length (the *Newton step*) to be taken at each iteration, hence

Eq. 3.10 $$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k^N \ ;$$

and, unlike the steepest descent direction, $\mathbf{s}^N$ is unaffected by the scale of $\mathbf{x}$. Moreover, the availability of the Hessian means it is possible to distinguish between saddle points and minima, and hence prevent premature termination.

For quadratic functions with a positive definite Hessian, Newton's method converges in a single iteration. For non-quadratic functions, the local convergence rate is quadratic. A sequence $\{\mathbf{x}_k\}$ that converges to the minimiser $\mathbf{x}_*$ is said to be *quadratically convergent*[3] if

Eq. 3.11 $$|\mathbf{x}_{k+1} - \mathbf{x}_*| \le c|\mathbf{x}_k - \mathbf{x}_*|^2$$

for some constant $c \ge 0$.

Unfortunately, unmodified Newton's method suffers from a number of drawbacks which make it unsuitable as a general optimisation method: it requires both first and second analytic derivatives to be available at every point $\mathbf{x}_k$[4]; it is only defined if **G** is positive definite (and is prone to failure whenever **G** is ill-conditioned), so that global convergence cannot be guaranteed; and its computational and storage costs are comparatively high - $O(n^3)$ (to solve Eq. 3.8) and $O(n^2)$ (to store **G**) respectively.

All the remaining second-order methods considered in this chapter fall somewhere between steepest descent and Newton's method. Broadly speaking, all these methods aim to: retain the guaranteed global convergence of steepest descent; generate search directions that are 'superior' to (i.e. interfere less than) the steepest descent direction when (comparatively) remote from a minimum; and approach the fast local convergence rate of Newton's method when close to a minimum. None of the methods require the prohibitively expensive calculation of second derivatives at each iteration.

In view of their 'heritage', we should expect the local convergence properties of these methods (for non-quadratic functions) to lie somewhere between those of steepest descent and Newton's method. In formal terms, this amounts to linear convergence at a faster rate than SD, or super-linear convergence. A sequence $\{\mathbf{x}_k\}$ that converges to $\mathbf{x}_*$ is said to be *super-linearly convergent* if

[3] There are two (non-equivalent) definitions of 'quadratic convergence' in general usage. The alternative definition to that used here is: convergence in (at most) n iterations for an n-dimensional quadratic function. This is often termed 'quadratic termination' - see, for example, [Wolfe, 1978, 114].

[4] Bishop [1992] has developed an algorithm for the exact calculation of the Hessian matrix using an MLP. The algorithm requires up to $2n$ forward and backward passes per pattern for an arbitrary n-node net - an indication of the high computational costs commonly associated with analytic second derivatives.

Eq. 3.12 $$|\mathbf{x}_{k+1} - \mathbf{x}_*| \leq c_k |\mathbf{x}_k - \mathbf{x}_*|^2$$

for some sequence $\{c_k\}$ that converges to zero.

3.1.3 Line-search methods vs. model-trust region methods

Classical methods fall into two categories: line-search methods and model-trust region methods. *Line searches* share the same iterative structure, outlined in the following pseudo-algorithm:

```
1. Choose a random starting point $\mathbf{x}_0$.
2. At each iteration $k$, do the following until termination criteria are satisfied:
    2.1. compute a search direction $\mathbf{s}_k$ that is a descent direction;
    2.2. chose a step-length $\alpha_k \geq 0$ that satisfies
Eq. 3.13        $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k) < f(\mathbf{x}_k)$
    2.3. set $\mathbf{x}_{k+1}$ to $\mathbf{x}_k + \alpha_k \mathbf{s}_k$.
```

If $\mathbf{s}_k$ is a descent direction (i.e. satisfies Eq. 3.9), the existence of a positive α_k that satisfies Eq. 3.13 is guaranteed.

Whereas for line-search methods the sub-task at each iteration is to locate the minimum along the search direction from $\mathbf{x}_k$, with *model-trust region methods*[5] the aim is to find the minimum in a 'trusted' region O_k around $\mathbf{x}_k$. O_k is conveniently defined in terms of its radius α_k, and $\mathbf{s}_k$ chosen to satisfy

Eq. 3.14 $$\|\mathbf{s}_k\| \leq \alpha_k \ ,$$

where ||•|| is the Euclidean (L_2) norm[6]. The basic iterative structure of model-trust region methods is outlined in the following pseudo-algorithm:

[5] The name comes from viewing the task as defining a *region* in which it is possible to *trust* the local quadratic *model* of function *f* [Dennis & Schnabel, 1983, 130]. These methods are also known as restricted step methods [Fletcher, 1980, 113] or, simply, trust region methods.

[6] Model-trust region methods using the L_2 norm are sometimes termed Levenberg-Marquardt methods. Here the latter term is reserved for the nonlinear least squares method of section 3.6. Hypercube or boxstep methods, using the L_∞ norm, have good local but poor global convergence properties [Fletcher, 1980, 99].

```
1. Choose a random starting point $\mathbf{x}_0$ and region size $\alpha_0 > 0$.
2. At each iteration $k$, do the following until termination criteria are satisfied:
    2.1. compute *search direction* $\mathbf{s}_k$ for trusted region $O_k$ with radius $\alpha_k$;
    2.2. IF $f(\mathbf{x}_k + \mathbf{s}_k) < f(\mathbf{x}_k)$,
        2.2.1. set $\mathbf{x}_{k+1}$ to $\mathbf{x}_k + \mathbf{s}_k$;
    2.3. update $\alpha_k$ according to a *regulation scheme*.
```

Broadly speaking, schemes for regulating the radius α_k are designed to increase α at iteration k if the local model of f is accurate, but decrease α if the model is inaccurate (for example, if $f(\mathbf{x}_k + \mathbf{s}_k) > f(\mathbf{x}_k)$). In practice, it is usual to control α_k indirectly by performing the substitution

Eq. 3.15 $$\overline{\mathbf{H}}_k = \mathbf{H}_k + u_k\mathbf{I},$$

where $\mathbf{H}_k$ is the model Hessian and **I** the identity matrix. In Eq. 3.15 a change to the scalar u ($u \geq 0$) produces an inverse change in α. If $u_k = 0$, $\mathbf{s}_k$ is the same as the Newton direction; as u_k tends to infinity, $\mathbf{s}_k$ tends to the steepest descent direction - see Figure 3. Strategies for initialising and regulating u are considered in section 3.3.

An important feature of the substitution in Eq. 3.15 is that, if u_k is 'sufficiently' large, $\overline{\mathbf{H}}_k$ will be *strictly diagonally dominant* - that is, for all i ($i = 1,...,n$),

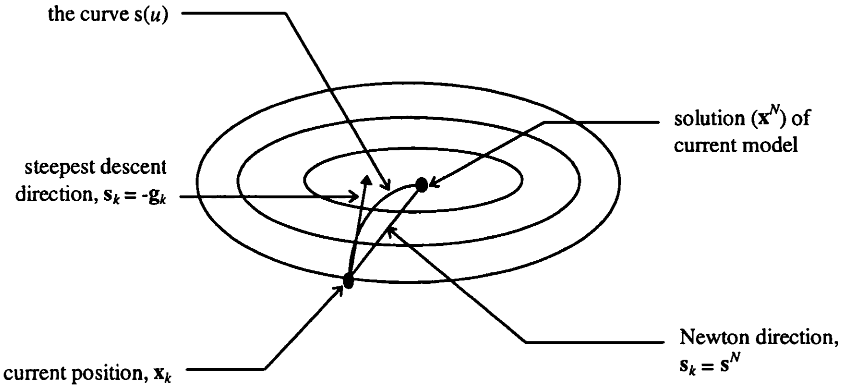
Eq. 3.16 $$\overline{\mathbf{H}}_{ii} - \sum_{j=1, j\neq i}^{n} \left|\overline{\mathbf{H}}_{ij}\right| > 0.$$

It follows, from the Gerschgorin circle theorem, that such an $\overline{\mathbf{H}}$ is positive definite [Dennis & Schnabel, 1983, 60]. Parameter u can therefore be viewed as providing a mechanism for regulating the positive definiteness of the model Hessian.

Neither line searches nor model-trust region methods are clearly superior. With line searches, the optimal accuracy with which α_k approximates the minimum along $\mathbf{s}_k$ is method- and problem-dependent. With model-trust region methods, the chosen scheme for initialising and regulating α (or u) can have a significant impact on training performance. In practice, line searches are often the first choice because they are, in general, simpler and easier to understand than model-trust region methods.

Figure 3 - Parameter *u* and the model-trust region search direction

Note: curve s(u) plots the points $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ for $0 < u \leq \infty$



the curve s(u)
steepest descent direction, $\mathbf{s}_k = -\mathbf{g}_k$
current position, $\mathbf{x}_k$
solution ($\mathbf{x}^N$) of current model
Newton direction, $\mathbf{s}_k = \mathbf{s}^N$

3.1.4 Special methods for nonlinear least squares

When using the 'traditional' sum-of-squares error function

Eq. 3.17 $$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{N^L}\left(t_{i,p} - y_{i,p}^L\right)^2$$

(cf. Eq. 2.3), the MLP training task is equivalent to a special category of problem known as *nonlinear least squares.* Such problems occur when fitting model functions to experimental data; typically the number of data values m (equivalent to PN^L for an MLP) is greater than the number of free parameters n (equivalent to the number of weights), i.e. the corresponding system of equations is *over-determined.*

The gradient and Hessian of Eq. 3.17 have a special structure with respect to the *residual vector* **r** and $m \times n$ *Jacobian matrix* **J**:

Eq. 3.18 $$r_i = t_i - y_i^L$$

Eq. 3.19 $$J_{ij} = \frac{\partial\, \mathrm{f}(x_i)}{\partial x_j} \quad .$$

In terms of **r** and **J** at iteration *k*,

Eq. 3.20 $$E_k = \frac{1}{2}\mathbf{r}_k^T\mathbf{r}_k$$

Eq. 3.21 $$\mathbf{g}_k = \mathbf{J}_k^T\mathbf{r}_k$$

Eq. 3.22 $$\mathbf{G}_k = \mathbf{J}_k^T\mathbf{J}_k + \mathbf{S}_k$$

$$\mathbf{S}_k = \sum_{i=1}^{m} r_{i,k}.\nabla^2 r_{i,k} \quad ,$$

where ∇^2 is the matrix of second derivatives such that $\nabla^2 r$ is the Hessian of *r*.

Eq. 3.22 is unsuitable as the basis of a general nonlinear least-squares algorithm because the second-order term **S** is typically unavailable. One option is to ignore **S** altogether on the assumption that the first-order term of Eq. 3.22 dominates the second-order term near the solution - a reasonable assumption so long as the residuals at the solution are small or zero [Dennis & Schnabel, 1983, 222]. Nonlinear least-squares algorithms which approximate **G** according to

Eq. 3.23 $$\mathbf{G}_k \approx \mathbf{J}_k^T\mathbf{J}_k$$

are considered in section 3.6. A second alternative to Eq. 3.22 is to approximate **S** by a secant approximation **A**, i.e.

Eq. 3.24 $$\mathbf{G}_k \approx \mathbf{J}_k^T\mathbf{J}_k + \mathbf{A}_k \ .$$

Methods derived from Eq. 3.24, which are superior to those derived from Eq. 3.23 for large-residual problems, are considered in [Fletcher, 1980] and [Dennis & Schnabel, 1983, 228-233].

The significance of Eq. 3.23 and Eq. 3.24 is that, given only $\mathbf{r}_k$ and $\mathbf{J}_k$, it is possible to approximate the Hessian matrix $\mathbf{G}_k$ immediately at each iteration, whereas with general unconstrained minimisation strategies (such as the quasi-Newton methods of section 3.4) it may take *n* iterations to calculate a satisfactory approximation of the (*n* × *n*) Hessian. For this reason, least squares methods are generally preferred to general unconstrained minimisation methods for functions of the form Eq. 3.17 on grounds of convergence speed.

3.1.5 Scaling and preconditioning

Although none of the classical algorithms considered in the remainder of this chapter are as sensitive to 'sub-optimal' scaling as the steepest descent algorithm (see section 3.1.1), all are prone to numerical problems - ranging from a general degradation in performance and loss of stability to premature termination of the multivariate algorithm - if the scale of either the independent variables **x** or the function *f* is sufficiently poor[7]. Scaling schemes aim to prevent these problems from arising by improving the scale of **x** (so that the independent variables are of a similar order of magnitude in the 'region of interest') and/or the scale of *f* (so that the norm of the model Hessian is of a similar order of magnitude to that of the Hessian itself). A convenient way of approaching the issue of scaling is in term of the condition of the Hessian matrix; a scheme that ensures a problem is 'well-scaled' by minimising the condition number (see Eq. 3.5) of the Hessian - so that similar changes in **x** lead to similar changes in *f* - is known as a *preconditioning scheme.*

Most scaling schemes modify the scale of **x** according to the linear transformation

Eq. 3.25 $$\hat{\mathbf{x}} = \mathbf{L}\mathbf{x} \ ,$$

where matrix **L** is fixed and non-singular. The optimal **L**, which transforms the model Hessian at $\mathbf{x}_*$ to the identity matrix (assuming $\mathbf{G}(\mathbf{x}_*)$ is positive definite), is

Eq. 3.26 $$\mathbf{L} = \mathrm{G}(\mathbf{x}_*)^{-1/2} \ ,$$

where **L** is a $n \times n$ matrix. Assuming $\mathbf{G}(\mathbf{x}_*)$ is not known, the **L** in Eq. 3.26 can be approximated using $\mathbf{G}(\mathbf{x}_0)$ (or, if second derivatives are unavailable, a finite-difference approximation of $\mathbf{G}(\mathbf{x}_0)$). However, unless **G** is positive definite and remains relatively constant in the region of interest - properties which cannot be guaranteed in general - there is a risk that such a scaling will actually degrade the performance of the multivariate algorithm. Although it is possible to overcome this problem by recalculating **L** periodically (*dynamic scaling* or *adaptive preconditioning*), the high cost of evaluating

[7] Certain methods, such as the DFP and BFGS quasi-Newton methods of section 3.4, are theoretically 'scale-invariant' under certain strict conditions, including the use of exact arithmetic. However, scale-invariance cannot be achieved in practice; finite floating-point arithmetic is scale-dependent - so that, for example, the error associated with the sum $x_1 + x_2$ is not related in a straightforward way to that of the sum $ax_1 + bx_2$, if $a \neq b$ - and termination, step-length and other criteria rely on implicit definitions of 'large' and 'small' [Gill, Murray & Wright, 1981].

G(**x**) or computing its approximation means that this approach cannot be recommended in a neural network context. Moreover, for methods which have, as one of their main competitive advantages, O(n) storage requirements (such as the conjugate gradient methods of section 3.5 and memoryless quasi-Newton method of section 3.4.3), the O(n^2) storage cost for matrix **L** is a significant disadvantage.

The most widely-used scaling schemes for multivariate optimisation represent **L** in Eq. 3.25 by a diagonal matrix (**D**). Given a suitable estimate of the condition number of the Hessian (calculated, for example, by the Power method [Møller, 1993d], or from the Cholesky factors of the Hessian matrix[8] [Gill, Murray & Wright, 1981, 320-322]), **D** can be used as a simple preconditioning matrix. Møller [1993d] has devised an efficient adaptive preconditioning scheme for **D**, based on an extension to the Power method, which significantly increases MLP training speed with the steepest descent algorithm under most circumstances. However, Møller reports only a modest improvement for conjugate gradient methods, and concedes that convergence may actually be degraded in some situations (for example, when the Hessian is indefinite).

A simpler alternative is to initialise **D** according to a set of n user-defined *scale factors,* representing the approximate ranges of the elements of **x**. Although this depends on the availability of useful prior knowledge about the problem structure, which cannot be guaranteed for minimisation tasks in general, Rigler et al. [1991] suggest a natural set of scale factors for MLP training derived from the gradient calculation by Eq. 2.1 and Eq. 2.7. When node output y is in the interval [0, 1] (as is the case with the standard sigmoid squashing function of Eq. 2.2), the derivative $y' = y^*(1 - y)$ (see Eq. 2.8) is constrained so that $0 \leq y^*(1 - y) \leq 1/4$. Given that the factor $y^*(1 - y)$ is used in the derivative calculation at the preceding layer by Eq. 2.7, Rigler et al. propose that the compensatory factors 6, 36, 216,... are applied as a multiplier of each partial derivative calculated at layers L-1, L-2, L-3,.... Such a scheme can be modified to take account of the scale of function f. For example Dennis and Schnabel [1983, 209] recommend that the model Hessian is initialised according to:

Eq. 3.27 $$\mathbf{H}_0 = \max\{|f(\mathbf{x}_0)|, t\}.\mathbf{D}^2 \ ,$$

[8] The Cholesky factors of the Hessian are available with certain (efficient) implementations of quasi-Newton methods - see section 3.4.2.

where t is a user-supplied estimate of the 'typical' size of f. (In the absence of any useful information, t is initialised to 1.0.)

The impact of different scaling schemes on the first- and second-order MLP training algorithms implemented for this thesis is the subject of on-going research. Preliminary experiments suggest that the scale factors proposed by Rigler et al. significantly improve the training speed of first-order algorithms, but not second-order algorithms (although the improvement for first-order methods was not sufficient to bring them up to the speed of any second-order method).

3.2 Line Minimisation

For classical methods with line searches, the task of locating the minimum along search direction $\mathbf{s}_k$ (line minimisation) is equivalent to finding the minimum of a function with a single variable (univariate minimisation). For clarity, all the line-minimisation strategies considered here are presented in terms of an arbitrary smooth univariate function f with scalar minimum x_*. (Since line minimisation is an iterative process, the suffix m will be used for line-search iterations to prevent confusion with the k-iterations of the multivariate algorithm.)

3.2.1 Line minimisation strategies

There are two broad strategies commonly used to locate minimum x_* of univariate function f: function comparison and function approximation (polynomial interpolation).

Given two initial values of x (x_a and x_b) which bracket x_*, *function comparison methods* iteratively reduce the interval in which x_* lies - the *interval of uncertainty* - by a fixed ratio. Linear convergence is guaranteed for unimodal functions. (Function $f(x)$ is unimodal in the interval $[a, b]$ if, given any $x_1, x_2 \in [a, b]$ with $x_1<x_2$, there is a unique $x_* \in [a, b]$ such that $f(x_1)>f(x_2)$ if $x_2<x_*$, and $f(x_1)<f(x_2)$ if $x_1>x_*$ [Gill, Murray & Wright, 1981, 88-89].) The technique adopted here is *golden section search*, which ensures that the interval at iteration m+1 is approximately 0.618 (the golden section) times the size of the interval at *m*. In terms of the maximum reduction of the interval for a given number of function definitions, golden section search is almost as efficient as the 'optimal' strategy, Fibonacci search. (The latter is considered impractical as it requires the storage or generation of *p* Fibonacci numbers for *p* function evaluations, where *p* is generally not known in advance.)

Function comparison methods are reliable, but make no attempt to exploit the smoothness of function *f*. The second strategy, *polynomial interpolation*, approximates *f* by a simple function $\hat{f}$ and uses $\hat{f}$ to estimate the minimum of *f*. Typically $\hat{f}$ is a *parabolic* (quadratic) or *cubic* polynomial. The former requires three pieces of data about *f* (typically $f(\mathbf{x}_a)$, $f(\mathbf{x}_b)$ and $f(\mathbf{x}_c)$) and the latter four (typically $f(\mathbf{x}_a)$, $g(\mathbf{x}_a)$, $f(\mathbf{x}_b)$ and $g(\mathbf{x}_b)$). If $\hat{f}$ is an accurate approximation of *f*, the theoretical convergence rate is super-linear (parabolic $\hat{f}$) or quadratic (cubic $\hat{f}$). If, on the other hand, $\hat{f}$ inaccurately approximates *f*, polynomial interpolation is likely to be slow and unreliable; lower-order polynomials may actually prove more accurate than higher-order polynomials in regions where *f* is comparatively non-smooth[9].

3.2.2 Safeguarded polynomial interpolation

In practice, it is possible to combine the strengths of both the above strategies in a single line-search algorithm with a convergence rate that approaches that of polynomial interpolation under favourable conditions, but remains close to the guaranteed rate of unmodified interval-reduction in the worst case. Such algorithms are termed *safeguarded polynomial interpolation* algorithms.

Designing such an algorithm is a non-trivial task, requiring efficient and robust mechanisms for detecting how 'co-operative' *f* is and for switching strategies when appropriate. The method adopted here - *Brent's method* [Brent, 1973] - is widely-used and well-regarded [Fletcher, 1980, 29] [Press et al., 1988]. Brent's method can be implemented with either parabolic or cubic interpolation. Although the latter is likely to

[9] For a visual explanation of the relative merits of high- and low-order polynomial interpolation, see [Press et al., 1988, 87].

take fewer m-iterations on average, it requires the calculation of derivatives (thereby approximately doubling the computational cost at each m-iteration for an MLP).

3.2.3 Inaccurate line searches

When a line search is used as part of a multivariate minimisation strategy, a key issue is the accuracy with which α_k is chosen to approximate the minimum along $\mathbf{s}_k$. The trade-off between the effort expended to determine an α_k of a given accuracy and the corresponding benefit (in terms of the overall reduction in E) to the multivariate algorithm is problem- and algorithm-dependent. Given a sufficiently robust multivariate algorithm, current opinion clearly favours inaccurate line searches on grounds of efficiency.

A practical and popular termination criterion for controlling the accuracy of α_k is

Eq. 3.28
$$\left|\mathbf{g}_{k+1}^T\mathbf{s}_k\right| \leq -q\mathbf{g}_k^T\mathbf{s}_k \ ,$$

where $\mathbf{g}_{k+1}$ is the gradient vector at $\mathbf{x}_k + \alpha_k\mathbf{s}_k$, and q a scalar in the range $0 \leq q < 1$. If q is small, an accurate line minimisation is performed, with $q = 0$ giving an 'exact' line search. (For exact line searches, the limiting factor is the floating-point precision available; owing to rounding error, it is a waste of effort to evaluate $\mathrm{f}(\mathbf{x}_m)$ if point $\mathbf{x}_m$ is closer than the square-root of the machine accuracy to a previously evaluated point [Press et al., 1988, 300].)

To guarantee global convergence it is important that α_k produces a 'sufficient' reduction in E. Since Eq. 3.28 takes no account of the actual reduction in E, it is usual to supplement it with the condition

Eq. 3.29
$$\mathrm{E}(\mathbf{x}_k) - \mathrm{E}(\mathbf{x}_k + \alpha_k\mathbf{s}_k) \geq -u\alpha_k\mathbf{g}_k^T\mathbf{s}_k \ ,$$

where u is in the range $0 < u \leq 0.5$. Setting $q > u$ guarantees that Eq. 3.28 and Eq. 3.29 can be satisfied simultaneously.

Algorithms which satisfy both Eq. 3.28 and Eq. 3.29 at each iteration are globally convergent (under the mild assumptions that E is bounded below and the angle between $\mathbf{s}_k$ and $\mathbf{g}_k$ is bounded away from 90 degrees) [Dennis & Schnabel, 1983, 125]. Moreover, since $\mathbf{s}_k^N$ will satisfy both conditions simultaneously when $\mathbf{x}_k$ is close to $\mathbf{x}_*$ (assuming **G** is positive definite), Eq. 3.28 and Eq. 3.29 are compatible with fast rates of local convergence.

To test condition Eq. 3.28 at each *m*-iteration requires first-derivatives; Eq. 3.28 is, therefore, inappropriate for line minimisation without derivatives. An alternative condition, proposed in [Gill, Murray & Wright, 1981, 102], replaces the left-hand side of Eq. 3.28 by a finite-difference approximation, i.e.

Eq. 3.30
$$\frac{\left|\mathrm{E}(\mathbf{x}_k + \alpha_k \mathbf{s}_k) - \mathrm{E}(\mathbf{x}_k + \nu \mathbf{s}_k)\right|}{\alpha_k - \nu} \leq -q\mathbf{g}_k^T \mathbf{s}_k \ ,$$

where ν is a scalar satisfying $0 \leq \nu < \alpha_k$. For the non-derivative line searches used in this research, Eq. 3.30 was adopted with ν=0, so that no additional function evaluations were required to test this condition. Alternatively, Eq. 3.28 can be ignored altogether [Møller, 1993a], or replaced by a heuristic mechanism for controlling line-search accuracy - for example, Kinsella [1992] places an upper limit on the number of *m*-iterations performed at a each *k*-iteration. However, neither of these alternatives have the theoretical justifications of Eq. 3.30.

3.2.4 Backtracking line search

Recent results (for conjugate gradient algorithms) published by Møller [1993a] suggest that *inaccurate* safeguarded polynomial interpolation - entailing a minimum of three function evaluations per epoch - may be less efficient than the model-trust region approach. However, there is a class of line-search algorithm (not considered by Møller) which requires only a single function evaluation per epoch in the best case - *backtracking line searches.*

For many second-order methods, the Newton step (i.e. $\alpha_k = 1$) is a 'natural' step to take at each iteration. If the Hessian is positive definite, there is a good chance that the Newton step will produce an acceptable decrease in *E*. Near the solution, allowing the full Newton step is a key to fast convergence [Dennis & Schnabel, 1983, 117].

If the error at $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ is unacceptable, backtracking algorithms iteratively 'backtrack' (i.e. reduces α_k) until an acceptable E is found. The algorithm developed here, based on the backtracking algorithms in [Dennis & Schnabel, 1983], uses parabolic interpolation for the first step and cubic interpolation thereafter. (The latter is performed without expensive derivative calculations by storing $f(\mathbf{x}_k)$, $g(\mathbf{x}_k)$ and the two most recent test values for $f(\mathbf{x}_k + \alpha_{k,m}\,\mathbf{p}_k)$.) Parameter u in Eq. 3.29 is set to a small value (10^{-04}) so that a small reduction in E is sufficient for the acceptance of a given α_k.

Dennis and Schnabel present two versions of their backtracking algorithm; one version - the 'modified' version - implements Eq. 3.28 and requires derivative calculations, the other does not. The authors give theoretical and practical reasons for implementing Eq. 3.28 as well as Eq. 3.29 with algorithms that use quasi-Newton approximations to the Hessian matrix. With the architectures, training problems and multivariate algorithms considered in this research, the Dennis-Schnabel unmodified backtracking algorithm displayed worse average convergence characteristics than the modified algorithm. The non-derivative backtracking algorithm developed here, which implements Eq. 3.30 rather than Eq. 3.28 (with a corresponding saving in derivative calculations), retained the improved performance of the modified Dennis-Schnabel algorithm.

3.2.5 Hybrid Brent/backtracking line search

In trials conducted for this research it was observed that low-accuracy non-derivative Brent's method frequently made better progress than the backtracking strategy in very flat regions - a common feature of MLP error surfaces (see section 2.2.1). In response to this observation, a novel *hybrid Brent/backtracking algorithm* has been developed. This algorithm uses the efficient backtracking strategy under 'average' conditions but switches to Brent's method under unfavourable conditions, i.e. whenever the number of backtracking iterations exceeds a user-defined limit or the multivariate algorithm generates a search direction that fails to satisfy Eq. 3.9. A few k-iterations of Brent's method are often sufficient to find a position in a more favourable region of weight-space, so that backtracking can be resumed without further interruption.

3.2.6 Line search implementation

Multivariate vs. univariate implementation. Line-search algorithms can be coded either using vector-valued points or (as here) scalar points. The second alternative requires some mechanism for evaluating the multi-dimensional function F in a single dimension; the simple device adopted here (described in [Press et al., 1988, 317]) is to provide an 'artificial' uni-dimensional function f(α) which evaluates F at $\mathbf{s}_k + \alpha\mathbf{x}_k$.

Handling non-descent directions. A prerequisite for all the line searches considered above is that $\mathbf{s}_k$ satisfies Eq. 3.9 (i.e. $\mathbf{s}_k$ is a descent direction). $\mathbf{s}_k$ is guaranteed to satisfy Eq. 3.9 with the SD algorithm, so long as the gradient is greater than zero. However, many multivariate algorithms do occasionally generate an $\mathbf{s}_k$ which is not a descent direction (for the various reasons considered in section 3.1). In these circumstances, probably the only solution (in general) is to restart the multivariate algorithm at the current position, with $\mathbf{s}$ reset to the steepest descent direction. An unfortunate by-product of *resetting* the algorithm is that any useful derivative information from previous iterations will be automatically discarded.

3.3 Model-Trust Region Strategies

The model-trust region approach has become an increasingly popular alternative to the traditional line-search approach. For all the strategies considered here, it is assumed that the step length (radius) α_k is controlled indirectly by parameter u via the substitution in Eq. 3.15. There is no 'natural' choice for u_0; recommended settings range between 0.001 [Press et al., 1988] and $\leq 10^{-06}$ [Møller, 1993a].

3.3.1 A simple model-trust region algorithm

The simplest model-trust region strategy is given by the following pseudo-algorithm (based on the Levenberg-Marquardt algorithms in [Press et al., 1988] and [Nash, 1990]):

```
1. Set $u_0 > 0$;
2. While termination criteria are not satisfied
      2.1. calculate $\mathbf{s}_k$;
```

```
2.2. IF f($\mathbf{x}_k$ + $\mathbf{s}_k$) < f($\mathbf{x}_k$),
        2.2.1. set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$;
        2.2.2. divide $u_k$ by a reduction constant;
     ELSE
        2.2.3. set $\mathbf{x}_{k+1} = \mathbf{x}_k$;
        2.2.4. multiply $u_k$ by a growth constant.
```

For this research, the reduction constant was set to 2 or 4 and the growth constant to 4 or 10. This strategy has proved satisfactory when used with the Levenberg-Marquardt method of section 3.6, but is prone to inefficient, oscillatory behaviour in regions where the appropriate value of u remains relatively constant for a number of iterations. (This behaviour is particularly apparent if the reduction and growth constants are set to the same value, as advocated by [Press et al., 1988].)

3.3.2 Fletcher's method

A better strategy - sometimes called *Fletcher's method* [Wolfe, 1978] - is to chose a u_k that ensures 'sufficient' agreement is maintained between the actual and predicted quadratic error change at each iteration (ΔE_k and ΔQ_k respectively). This is conveniently measured in terms of the ratio r, given by

Eq. 3.31

$$r_k = \frac{\Delta E_k}{\Delta Q_k}$$

The predicted error change ΔQ_k can be calculated as follows:

Eq. 3.32

$$\Delta Q_k = E_k - Q(\mathbf{s}_k)$$

$$Q(\mathbf{s}_k) = \frac{1}{2}\mathbf{s}_k^T \mathbf{G}_k \mathbf{s}_k + \mathbf{g}_k^T \mathbf{s}_k$$

This leads to the following pseudo-algorithm (based on the Levenberg-Marquardt algorithm in [Fletcher, 1980]):

```
1. Set $u_0 > 0$;
2. While termination criteria are not satisfied,
     2.1. calculate $\mathbf{s}_k$;
     2.2. evaluate f($\mathbf{x}_k$ + $\mathbf{s}_k$) and calculate $r_k$;
     2.3. IF $r_k$ is less than lower ratio limit $v$,
          2.3.1. multiply $u_k$ by a growth constant;
```

```
        ELSE IF $r_k$ is greater than *upper ratio limit w*,
            2.3.2. divide $u_k$ by a *reduction constant*;
        ELSE
            2.3.3. set $u_{k+1} = u_k$;
    2.4. IF $r_k \leq 0$
            2.4.1. set $\mathbf{x}_{k+1} = \mathbf{x}_k$;
        ELSE
            2.4.2. set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
```

The lower and upper ratio limits v and w are typically chosen so that $0 < v < w < 1$. This algorithm is relatively insensitive to changes in the various constants; those used in this research are the same (arbitrary) constants advocated in [Fletcher, 1980, 96]: v=0.25 and w=0.75. (Settings for the growth and reduction constants were the same as for the simple strategy of section 3.3.1.)

When **G** is not available (as, for example, with conjugate gradient methods), $\mathbf{G}_k\mathbf{s}_k$ (in Eq. 3.32) can be approximated by a one-sided finite-difference approximation or calculated exactly with P forward and backward passes using an algorithm described in [Møller, 1993c]. Both schemes have O(PN) time and storage costs; on average, the former yields a faster convergence rate than the latter (which is prone to numerical instability) [Møller, 1993e, 39].

3.3.3 Modern model-trust region algorithms

More recent model-trust region strategies choose u_k so that Eq. 3.14 is satisfied explicitly at each iteration, i.e. a u_k that satisfies

Eq. 3.33 $$\|\mathbf{s}_k\| \approx \alpha_k$$

whenever the length of $\mathbf{s}_k^N$ is greater than α_k [More, 1983]. Two strategies for iteratively approximating the u_k that satisfies Eq. 3.33 - the *locally constrained optimal ('hook') step* and the *double dogleg step* - are presented in [Dennis & Schnabel, 1983, 134]. (None of these more sophisticated strategies have been implemented for this research.)

3.4 Quasi-Newton Methods

Quasi-Newton (QN) *methods*[10] differ from Newton's method in that an approximation of the Hessian matrix (or its inverse) is built up iteratively, rather than calculated afresh at each epoch. Since QN algorithms do not require analytic second derivatives, they are more suitable for MLP implementation than straight Newton-type methods.

3.4.1 The Hessian update formula

In generating the Hessian approximation $\mathbf{H}_{k+1}$ from $\mathbf{H}_k$ using the derivative information collected during iteration *k*, all QN methods satisfy the so-called *quasi-Newton condition*[11]

Eq. 3.34 $$\mathbf{H}_{k+1}\mathbf{p}_k = \mathbf{y}_k \ ,$$

where $\mathbf{y}_k$ and $\mathbf{p}_k$ are respectively the gradient change and the change in position $\mathbf{x}$ during iteration *k*, i.e.

Eq. 3.35 $$\begin{aligned} \mathbf{y}_k &= \Delta\mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k \\ \mathbf{p}_k &= \alpha_k \mathbf{s}_k = \Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \ . \end{aligned}$$

Where methods differ is in the choice of updating formula that satisfies Eq. 3.34. $\mathbf{H}_0$ is typically set to the identity matrix (**I**) as a 'neutral' first approximation, making the first iteration equivalent to steepest descent.

QN methods are categorised in terms of the simple equation

Eq. 3.36 $$\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{C}_k \ ,$$

[10] There is some confusion in the numerical analysis literature about the use of the terms 'quasi-Newton methods', 'variable metric methods', and 'secant methods'; some authors maintain a distinction between these terms (see, for example, [Dennis & Schnabel, 1983]), others do not. In this research, the term 'quasi-Newton methods' is used throughout.

[11] For an explanation of the quasi-Newton condition in terms of a Taylor series expansion of f(**x**), see [Fletcher, 1980, 39].

where $\mathbf{C}_k$ is a *correction* or *update matrix.* The best-known methods use a rank-two matrix for $\mathbf{C}_k$: the *Davidon-Fletcher-Powell* (DFP) update (most concisely written in terms of the inverse model Hessian, $\mathbf{H}^{-1}$)

Eq. 3.37 $$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{y}_k} - \frac{\mathbf{H}_k^{-1}\mathbf{y}_k\mathbf{y}_k^T\mathbf{H}_k^{-1}}{\mathbf{y}_k^T\mathbf{H}_k^{-1}\mathbf{y}_k} ,$$

and the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) update (the complement of the DFP update) given by

Eq. 3.38 $$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{p}_k} - \frac{\mathbf{H}_k\mathbf{p}_k\mathbf{p}_k^T\mathbf{H}_k}{\mathbf{p}_k^T\mathbf{H}_k\mathbf{p}_k} .$$

Under various fairly stringent conditions (including the strict convexity of *f*) both DFP and BFGS methods are globally convergent with a super-linear rate of local convergence. However, there is overwhelming theoretical and experimental evidence that the BFGS update is superior to the DFP update (and probably all other updates) [Dennis & Schnabel, 1983] [Dixon, 1972] [Fletcher, 1980]. (The DFP update has, for instance, the reputation of being highly sensitive to the choice of line-search accuracy.)

3.4.2 Representing the Hessian approximation matrix

If the Hessian approximation **H** is represented directly, QN methods - like Newton's method - require the solution of Eq. 3.8 at the cost of O(n^3) multiplications per iteration. By storing an approximation of the inverse Hessian $\mathbf{G}^{-1}$ rather than **G** itself, the cost falls to O(n^2) multiplications. In terms of the inverse Hessian, the BFGS update Eq. 3.38 becomes

Eq. 3.39 $$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \left(1 + \frac{\mathbf{y}_k^T\mathbf{H}_k^{-1}\mathbf{y}_k}{\mathbf{p}_k^T\mathbf{y}_k}\right)\frac{\mathbf{p}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{y}_k} - \frac{\mathbf{p}_k\mathbf{y}_k^T\mathbf{H}_k^{-1} + \mathbf{H}_k^{-1}\mathbf{y}_k\mathbf{p}_k^T}{\mathbf{p}_k^T\mathbf{y}_k} .$$

Although the inverse Hessian approach is highly effective with many problems, there is no convenient mechanism for regulating the positive-definiteness of the inverse Hessian (in contrast to the regular Hessian - see section 3.1.3). As a consequence, wasteful resetting of the QN algorithm may be unavoidable.

A more complex alternative, developed by Gill and Murray, is to represent the Hessian approximation by its *Cholesky factorisation*

Eq. 3.40 $$\mathbf{H}_k = \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^T$$

where $\mathbf{L}_k$ and $\mathbf{D}_k$ are, respectively, a unit lower-triangular matrix (i.e. all diagonals are one) and a positive diagonal matrix. This representation is convenient for detecting and correcting an indefinite Hessian, with computational costs that are of the same order as the inverse Hessian strategy [Gill, Murray & Wright, 1981]. (Algorithms for calculating and iteratively updating the Cholesky factorisation of the QN Hessian are given in [Dennis & Schnabel, 1983].)

3.4.3 Modified quasi-Newton methods

O(n) memory storage. With large-scale problems, the O(n^2) memory cost of storing the Hessian matrix may be prohibitive. This has led to the development of a *'memoryless'* (i.e. O(n)) *quasi-Newton method* (NQN) in which the BFGS formula is applied to **I** rather than $\mathbf{H}_k$. Successive search directions are generated iteratively according to the expression

Eq. 3.41 $$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{y}_k \mathbf{p}_k^T \mathbf{g}_{k+1} + \mathbf{p}_k \mathbf{y}_k^T \mathbf{g}_{k+1}}{\mathbf{p}_k^T \mathbf{y}_k} - \left(1 + \frac{\mathbf{y}_k^T \mathbf{y}_k}{\mathbf{p}_k^T \mathbf{y}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{y}_k} \; .$$

The algorithm is equivalent to the Polak-Ribiere conjugate gradient method (section 3.5) when exact line searches are used, and is reputedly superior in practice with inaccurate line searches [Luenberger, 1984, 280].

Reset every n iterations. One inelegant but effective way of ensuring a QN algorithm is globally convergent is to reset the algorithm every n (or n+1) iterations. This strategy may be appropriate for difficult problems when using a QN algorithm that makes no attempt to regulate the condition of the model Hessian.

3.5 Conjugate Gradient Methods

Conjugate gradient (CG) *methods* are a class of second-order methods which, unlike quasi-Newton methods, require only O(*n*) storage. They are thus particularly well-suited to large-scale problems for which quasi-Newton methods may be impractical.

3.5.1 The conjugate gradient formula

CG methods exploit the fact that a sequence of search directions that are mutually conjugate (i.e. satisfy Eq. 3.7) can be generated iteratively without using the Hessian **G** according to the expression

Eq. 3.42 $$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{s}_k \ ,$$

with the first iteration equivalent to steepest descent (i.e. $\mathbf{s}_0 = -\mathbf{g}_0$). Where CG methods differ is in the formula - the so-called *conjugate gradient formula* - used to calculate scalar β_k.

In terms of the Hessian matrix, β_k is defined by

Eq. 3.43 $$\beta_k = -\frac{\mathbf{g}_{k+1}^T \mathbf{G} \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{G} \mathbf{s}_k} \ .$$

In CG methods, where the aim is to avoid evaluating **G**, Eq. 3.43 is reformulated using first derivative information. The three most popular alternatives are the *Fletcher-Reeves* (FR) formula

Eq. 3.44 $$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \ ,$$

the *Hestenes-Stiefel* (HS) formula

Eq. 3.45 $$\beta_k = \frac{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}}{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{s}_k} \ ,$$

and the *Polak-Ribiere* (PR) formula,

Eq. 3.46 $$\beta_k = \frac{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \ .$$

All three formulae are equivalent for quadratic functions (with exact line searches), but the PR formula is widely preferred in practice and has been adopted here. (One reason why the PR formula may be more effective is that it tends to reset automatically to the steepest descent direction whenever the algorithm fails to make much progress.)

3.5.2 Conjugate gradient restarts

Under strict conditions, CG methods are capable of minimising quadratic functions in at most n iterations for n free parameters (MLP weights), but generally take more than n iterations for non-quadratic functions. Although it is possible to use the same update formula at every iteration (often with acceptable results in the case of the PR update [Wolfe, 1978]), it is usually much more efficient to restart the CG algorithm roughly every n iterations.

A variety of CG restart schemes have been devised, differing in the choice of restart direction and the interval between restarts. The simplest and most popular option (adopted here) is to reset **s** to the steepest descent direction every n (or n+1) iterations. (In addition, it is common practice to restart a CG algorithm whenever a search direction that fails to satisfy Eq. 3.9 is generated.) A drawback with the traditional, steepest descent restart procedure is that curvature information from previous iterations is automatically discarded. Restart schemes which aims to retains some curvature information (such as the Powell restart [Powell, 1977]), are worth considering, but have not been implemented for this research.

The traditional SD restart is important for the theoretical convergence characteristics of CG methods. It guarantees global convergence, and affords a super-linear rate of local convergence for a wide class of functions (assuming exact line searches and exact arithmetic) [Gill, Murray & Wright, 1981, 149-50]. (For a more detailed consideration of the convergence properties of CG methods in terms of the distribution of the eigenvalues of the Hessian matrix, see [Luenberger, 1984, 247-252] and [Møller, 1993e, 33-36].)

3.6 Levenberg-Marquardt Method

3.6.1 From Gauss-Newton to Levenberg-Marquardt

Section 3.1.4 introduced a class of nonlinear least-squares algorithms which approximate the Hessian matrix **G** according to the simple Eq. 3.23. The *Gauss-Newton* (GN) *method*, which implements Eq. 3.23 without modification, has the update

Eq. 3.47
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{J}_k^T \mathbf{J}_k\right]^{-1} \mathbf{J}_k^T \mathbf{r}_k \ .$$

The convergence properties of the GN method depend on the size of **S** (from Eq. 3.22), a measure of the nonlinearity and residual size associated with the chosen problem. If $\mathbf{S}_k$ is small relative to $\mathbf{J}_k^T \mathbf{J}_k$, the method is locally quadratically convergent. However, an increase in either the relative residual size or nonlinearity of the problem increases the relative size of **S**, with a corresponding decrease in convergence speed; if **S** is too large the method may fail altogether, even in the neighbourhood of a minimum [Dennis & Schnabel, 1983, 224] [Fletcher, 1980, 113]. Moreover, the method is ill-defined whenever **J** does not have full column rank, a condition that is guaranteed to occur if $m<n$. (Since $m<n$ is equivalent to $PN^L<W$ for an MLP, this makes the GN method inherently unsuitable for the XOR task used in this research; otherwise, this condition rarely arises.)

The 'straight' Gauss-Newton method can be improved by combining it with a line-search algorithm so that Eq. 3.47 becomes

Eq. 3.48
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \left[\mathbf{J}_k^T \mathbf{J}_k\right]^{-1} \mathbf{J}_k^T \mathbf{r}_k \ ,$$

where α_k is the familiar step length. This method - the *damped Gauss-Newton method* (or Hartley method [Wolfe, 1978]) - is more reliable than the unmodified version, but otherwise suffers from similar drawbacks [Dennis & Schnabel, 1983, 227] [Fletcher, 1980, 115].

The preferred modification of the Gauss-Newton method, based on the model-trust region approach considered in section 3.3, is the *Levenberg-Marquardt* (LM) *method* (or Marquardt method). The LM update is given by

Eq. 3.49
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{J}_k^T \mathbf{J}_k + u_k \mathbf{I}\right]^{-1} \mathbf{J}_k^T \mathbf{r}_k \ .$$

The LM method has several advantages over the damped Gauss-Newton method: it is well-defined when **J** does not have full column rank; several version of the LM algorithm have been proved to be globally convergent (see, for example, [Osborne, 1976]); and, when the step length is too long, the LM update (which tends to the steepest descent direction) is often superior. The theoretical local convergence characteristics of the GN, damped GN, and LM methods are broadly similar: quadratic convergence for zero-residual problems; fast linear convergence for problems that are not too nonlinear and have fairly small residuals; and slow linear convergence for problems that are sufficiently nonlinear or have comparatively large residuals [Dennis & Schnabel, 1983, 225-228].

3.6.2 Neural implementation

The Levenberg-Marquardt method, in common with all nonlinear least squares methods based on Eq. 3.23 or Eq. 3.24, requires the components of Jacobian matrix **J** to be available at each iteration. An MLP can calculate the components of **J** as follows [Battiti, 1992, 160]:

Eq. 3.50
$$\frac{\partial r_{a,p}}{\partial w_{ij}^{l}} = \delta_{i,a,p}^{l} y_{j,p}^{l-1} \ ,$$

where the term δ is given by

Eq. 3.51
$$\delta_{i,a,p}^{L} = y_{i,p}^{L}\left(1 - y_{i,p}^{L}\right), \qquad \text{for } i = a$$

$$\delta_{i,a,p}^{L} = 0, \qquad \text{for } i \neq a$$

$$\delta_{i,a,p}^{l} = y_{i,p}^{L}\left(1 - y_{i,p}^{L}\right)\sum_{h}^{N^{l+1}} \delta_{h,a,p}^{l+1} w_{hi}^{l+1}, \qquad \text{for } l < L \ .$$

To calculate $\mathbf{G}_k$ and $\mathbf{g}_k$ (by Eq. 3.23 and Eq. 3.21 respectively) requires P standard BP forward passes and PN^L 'modified' backward passes based on Eq. 3.50 (as opposed to Eq. 2.7 for the standard BP backward pass).

Kollias and Anastassiou [1989] propose several modifications to the standard LM algorithm when used to train an MLP, including the representation of the Hessian $\mathbf{G}_k$ by a near-diagonal matrix and an adaptive distributed scheme for selecting the step-length parameter. (These modifications have not been implemented for this research.)

3.7 Comparison of Methods

With respect to convergence speed, conventional wisdom ranks the preceding methods in the following order[12] - LM (ranked first for zero-residual problems only), QN, CG and NQN, SD. This ordering is fairly intuitive, as it reflects the extent to which the various methods exploit the problem structure and store useful curvature information, but is somewhat misleading; although SD is consistently rated as the poorest method, the choice between the others is less clear-cut. In practice, the fastest second-order method for a given problem can only be determined by experimentation.

On balance, an efficient implementation of the BFGS quasi-Newton algorithm probably deserves the highest recommendation for its combined speed and robustness. BFGS QN has the reputation for being the most stable method with inaccurate line searches, does not require potentially-wasteful resetting to the steepest descent direction every n iterations (cf. conjugate gradient methods), and is not sensitive to the presence of residuals at the solution (cf. the Levenberg-Marquardt method).

Unfortunately, the $O(n^2)$ storage requirements of both the QN and LM methods - where n is the number of MLP weights - make them impractical for large-scale tasks. Under these circumstances, there is little to choose between the PR CG and NQN methods, both of which have only $O(n)$ storage costs.

[12] Newton-type methods with analytic second derivatives are generally preferred to all other unconstrained minimisation strategies, but rely on the ability to evaluate $\mathbf{G}$ efficiently at each iteration. For the reasons given in section 3.1.2, such methods are inappropriate for MLP training, and are therefore ignored in the current discussion.

4. CLASSICAL MLP TRAINING METHODS

This chapter compares the experimental performance of classical optimisation algorithms (adapted for supervised learning with an MLP) with that of traditional MLP training methods, focusing on two key aspects of training algorithm performance - speed and frequency of convergence to local, rather than global, minima. (Throughout the remainder of this thesis, the terms 'local' and 'global' are used to distinguish between local and global minima - cf. chapter 3.) Of the available research papers on this subject (reviewed in section 4.1), all address - if often inadequately - the comparative performance of classical training methods and backpropagation with respect to training speed, but only a single paper gives detailed consideration to the susceptibility of different training algorithms to getting trapped in local minima.

4.1 Research Review

The following review focuses on research papers that compare, experimentally, the performance of classical and traditional training methods:

Barnard [1992]

- **Training methods**: QN, BFGS update; CG with Powell restarts; SD; 'stochastic' BP with periodic line search to set training rate.
- **Line minimisation**: no details given.
- **Test problems**: XOR; artificial set (383 patterns); aircraft data (1890 patterns).
- **Architectures**: 2-3-2 *(sic)* (XOR); 2-5-3 (artificial); 32-9-3 (aircraft); weight initialisation range [-1, 1]; sum-of-squares error function.
- **Results**: 5 training runs with each problem/algorithm combination; QN and CG consistently fewer iterations than SD; performance of QN and CG very similar; 'stochastic' strategy fewest iterations with artificial and aircraft problems.

Battiti and Masulli [1990]

- **Training methods**: NQN, with SD restart every *n*; 'bold driver' batch BP, with training rate 'growth' and 'shrink' factors of 1.1 and 0.5 respectively.
- **Line minimisation**: method based on quadratic interpolation (requiring 'a small number of energy evaluations' per epoch).
- **Test problems**: arbitrary dichotomy problems (i.e. two classes of randomly generated patterns) with between 5 and 100 patterns in the range [0,1]; recurrent logistic $x_{k+1} = 4x_k(1-x_k)$, with 10 pattern set.
- **Architectures**: 2-*n*-1 (dichotomy) where *n* is the number of patterns divided by 2; 1-5-1 (logistic); weight initialisation range [-0.1, 0.1] (dichotomy problem - unspecified for logistic problem).
- **Results**: 10 runs (logistic); significantly fewer 'learning cycles' with NQN; both methods encountered 'local minima' with dichotomy problem (frequency not documented).

Berggren [n.d.]

- **Training methods**: CG, FR update; batch BP with unspecified training rate (claimed to be 'optimal').
- **Line minimisation**: 'modified Rosenbrock success-failure line-minimisation' (attributed to F James, "MINUIT", Computer Physics Communications 10, 343, 75) with undocumented learning parameter. Average 4.2 function evaluations per epoch.
- **Test problem**: distinguish positively- from negatively-sloping smeared lines in range [-0.005, 0.005] (1,000 patterns).
- **Architecture**: 25-15-1; sum-squared error function.
- **Results**: CG fewer iterations than BP; (the author suggests that CG is more likely to get trapped in local minima than BP with a large training rate, but no evidence is presented.)

Johansson, Dowla & Goodman [1992]

- **Training methods**: 4 CG methods (FR, PR, HS and Shanno[1]) with SD restart every n; SD; batch BP with 7 different combination of training rate and momentum in the range [0.1, 0.9].
- **Line minimisation**: safeguarded cubic interpolation with accuracy (q in Eq. 3.28) 0.01, 0.1, 0.5 and 0.9, and u (Eq. 3.29) of 10^{-04}.
- **Test problems**: 3-, 4- and 5-parity; mean-squared error function with termination criterion of 10^{-06}.
- **Architectures**: n-n-1 and n-n-n-1; weight initialisation range [-0.5, 0.5].
- **Results**: single run with each problem/algorithm combination; SD fewer function evaluations than batch BP and all CG methods fewer evaluations than SD; the authors conclude that the choice of CG method and line search accuracy is highly problem-dependent, but the HS or PR update with 0.1 accuracy considered most satisfactory; only CG with the Shanno update converged successfully with all problems and line-search settings.

Kinsella [1992]

- **Training methods**: CG, Fletcher-Reeves-Polak-Ribiere update - i.e. presumably the PR update (line search and model-trust region versions); on-line BP with training rate 0.1, and momentum 0.0 or 0.5.
- **Line minimisation**: Brent's method (with the number of iterations per epoch used as a crude mechanism for controlling accuracy).
- **Test problems**: distinguish between circles and rectangles of differing sizes and positions (3 training sets of increasing size and difficulty).
- **Architecture**: 16384-2-2; sum-squared error function.

[1] The aim of the Shanno CG update is to generate, at every iteration, a search direction that is guaranteed to be a descent direction, even with inaccurate line searches [Johansson, Dowla & Goodman, 1992, 295].

- **Results**: single run with each problem/algorithm combination; CG methods of growing superiority to BP as difficulty increases; with 'middle' training set, BP got 'stuck' and CG methods encountered 'local minima'; with largest training set, line-search CG performed poorly if Brent iterations (i.e. m-iterations) limited to 5 per epoch (k-iteration), but better than model-trust region CG when limit raised to 10; (author concludes that 'no advantage appears to be gained by using the Levenberg-Marquardt [i.e. model-trust region] approach'.)

Kollias & Anastassiou [1989]

- **Training methods**: modified LM (two versions - with and without adaptive distributed selection of step-length parameter); batch BP with training rate in range [0.1, 0.5] and momentum 0.9.
- **Line minimisation**: backtracking line search (based on [Dennis & Schnabel, 1983]).
- **Test problems**: digital image halftoning (16,000 samples); XOR (training set with 5 copies of each pattern).
- **Architecture**: single neuron (halftoning); 2-2-1 (XOR); sigmoid squashing function; sum-squared error function; weight initialisation range [-0.3, 0.3] (XOR).
- **Results**: 1 run (halftoning); 10 runs (XOR); modified LM methods consistently more accurate than BP and required fewer iterations; adaptive version of LM required fewer iterations than non-adaptive version.

Møller [1993a]

- **Training methods**: CG, PR update[2] (line search and model-trust region versions) with SD restart every n; NQN (with line search); batch BP with training rate 0.2 (3- to 6-parity), 0.05 (7-parity) or 0.01 (8- and 9-parity), and momentum 0.9.
- **Line minimisation**: safeguarded quadratic interpolation with termination condition u=0.25 in Eq. 3.29.

[2] The Polak-Ribiere update is called the Hestenes-Stiefel update by Moller [1993a, 80].

- **Test problems**: 3- to 9-bit parity; termination criterion of 10^{-04} ('average error').
- **Architecture**: *n*-*n*-1.
- **Results**: 20 runs with each problem/algorithm combination (10 runs for BP with 8- and 9-bit parity); line-search CG and NQN required 2.5-12 times fewer function evaluations than BP (with, in general, a greater improvement for higher *n*); model-trust region CG required 2-3 times fewer function evaluations than line-search CG and NQN; average failure-rate with BP about twice that of other methods.

Møller [1993b]

- **Training methods**: CG, PR update (2 model-trust region versions: a 'standard' off-line algorithm, and an on-line algorithm with subset update validation scheme - see section 5.1.1); on-line BP with training rate 0.1 and momentum 0.9.
- **Test problems**: randomly generated sets, of varying degrees of redundancy, with 12-bit inputs and 3-bit outputs; 1,000 word NETtalk (5,438 patterns); exchange rate prediction (set of 4,476 daily rates for DM vs. US$).
- **Architectures**: 12-8-3 (random sets); 203-30-26 (NETtalk); 20-10-1 (exchange rate).
- **Results**: 10 runs (NETtalk); on-line BP required fewer epochs than 'standard' CG with highly redundant problems (such as NETtalk), although the latter achieved greater accuracy; on-line CG required fewer epochs and achieved greater accuracy than on-line BP with highly redundant problems.

Møller [1993d]

- **Training methods**: CG, PR update (3 model-trust region versions, with and without adaptive preconditioning) with SD restart every *n*; batch BP (6 versions, with and without adaptive preconditioning - training rate 0.25 with standard version).
- **Test problems**: XOR; 5-parity; two spirals.

- **Architectures**: 5-5-1 (5-parity); non-standard MLP with 3 hidden layers of 5 nodes each (two spirals); termination criterion of < 0.8 error for each pattern and output (all test problems).
- **Results**: 30 runs (XOR, batch BP methods - unspecified for CG methods), 20 runs (5-parity) and 10 runs (two spirals, CG methods only); preconditioning schemes produce substantial speed-up (measured in epochs) with batch BP, slight speed-up with CG, but 'barely enough to justify the extra computation'; higher tendency to converge to a saddle point with batch BP and symmetric preconditioning scheme (5-parity).

Pattichis et al. [1991]

- **Training methods**: CG, PR update with SD restart every *n*; BP with momentum (details in unavailable reference).
- **Line minimisation**: details in unavailable reference.
- **Test problem**: EMG data (740 patterns).
- **Architectures**: variety of 2- and 3-layer MLPs with 5,000 to 26,000 weight for BP and 45 to 260 weights for CG; sigmoidal outputs in (non-standard) range [-1, 1]; least-square error function.
- **Results**: CG significantly better than BP in terms of training time and successful network size.

Van der Smagt [1990]

- **Training methods**: CG, PR update with Powell restarts; BP (no details given).
- **Line minimisation**: details in unavailable reference.
- **Test problem**: XOR.
- **Architecture**: 2-2-1; sigmoid squashing function; sum-squared error function.
- **Results**: CG fewer iterations than BP; (CG error curve shows two sudden increases in *E*, which are not explained by the author).

Van der Smagt [1994]

- **Training methods**: QN, DFP update (inverse Hessian); CG, FR update; CG, PR update with Powell restarts; SD; batch BP with training rate 0.1 and momentum 0.9.
- **Line minimisation**: Brent's method (average 3-5 function evaluations per epoch).
- **Test problems**: XOR; continuous function sin(x)cos($2x$) for $0 \leq x \leq 2\pi$ (20 samples); discontinuous function tan(x) for $0 \leq x \leq \pi$ (20 samples); termination condition <0.025 per pattern (all test problems).
- **Architectures**: 2-2-1 (XOR); 1-10-1 (sin(x)cos($2x$)); 1-5-1 (tan(x)); sum-squared error function.
- **Results**: 10,000 runs with each problem/algorithm combination; success rates with average function evaluations (if given by author) in parentheses:
    - XOR: BP 91% (332), SD 38% (3,662), FR CG 81% (523), DFP QN 34% (2,141), and PR CG 82% (79);
    - sin(x)cos($2x$): BP 0%, but 15% with adaptive training rate ('over two million function evaluations'), SD 90% (4.10^{06}), FR CG 49%, DFP QN 36%, and PR CG 100%;
    - tan(x): BP 0%, SD 0%, FR CG 4%, DFP QN 40%, and PR CG 85%.

Watrous [1987]

- **Training methods**: QN, DFP and BFGS updates (both inverse Hessian); SD; batch BP.
- **Line minimisation**: no details given.
- **Test problems**: XOR (with 0.1 and 0.9 targets); multiplexor.
- **Architectures**: 2-1-1 with 7 weights (XOR); 6-4-1 (multiplexor); sum-squared error function.

- **Results**: BFGS fewest function and gradient evaluations with both problems; SD and DFP failed to converge with multiplexor problem; BP considerably slower than other methods for XOR.

Before considering the usefulness (or otherwise) of these papers in terms of the issues addressed by this research, it is worth stressing that sensitivity to initial conditions is well-recognised in optimisation theory in general [Gill, Murray & Wright, 1981, 324-330] [Murray, 1972c] and MLP training in particular [Kolen & Pollack, 1990]. The performance of a training algorithm - and, perhaps more importantly, the relative performance of different training algorithms - may be sensitive to a range of factors, including the choice of floating-point precision, termination criteria, weight initialisation range, error function, training rate (backpropagation), or accuracy (line minimisation). As a consequence, the value of published results is diminished when this type of information is omitted from the written account. Of the thirteen papers summarised above, nine fail to specify the weight initialisation range, eight the termination criteria, seven the line search accuracy, five the BP training rate, and four the chosen error function. None of the authors specify the floating-point precision of their programs. Several papers even leave us in doubt as to the algorithm being tested; four fail to give any details about the line search method, and two neglect to mention whether the BP algorithm was on- or off-line.

Perhaps an even more significant issue is the number of training runs performed with a given combination of test problem and algorithm. Under otherwise identical training conditions, the relative performance of training algorithms - in terms of both the length of training time and the final error level - can vary dramatically with different sets of starting weights (even when initialised within the same range). Our confidence that a given set of results represents the 'typical' performance of an algorithm with a particular problem is proportional to the number of runs undertaken. Of the research papers reviewed above, five fail to mention the number of training runs performed, and, of the remainder, two have results for no more than a single run per problem/algorithm, and only three have results for more than ten runs per problem/algorithm.

A further aspect of these papers worthy of comment is the choice of training-speed metric. What counts as a satisfactory metric depends on which methods are being compared. The traditional metric for MLP training - the number of training epochs - is suitable for methods which evaluate f and g (i.e. perform p forward passes and p backward passes) exactly once per epoch (k-iteration), assuming that the cost of evaluating f and g dominates the total computational cost of each algorithm[3]. The number of epochs is, therefore, an acceptable metric for comparing the performance of traditional fixed-training-rate BP methods with certain implementations of model-trust region methods for unconstrained optimisation (e.g. [Møller, 1993b]) and LM methods (e.g. [Kollias & Anastassiou, 1989]), but inadequate for comparisons with line search methods, which require, on average, more than a single function (and, in some cases, gradient) evaluation per epoch. Four of the papers reviewed above use the unsuitable number-of-epochs metric to compare the performance of traditional BP with classical line search methods. (Performance metrics are discussed further in section 4.3.1 and 4.4.2.)

Of those papers which do not contain serious methodological flaws and omissions, only a single paper - [van der Smagt, 1994] - investigates, in detail, the tendency of different training methods to get trapped in local minima.

4.2 Benchmark Training Sets

4.2.1 Benchmark criteria

The choice of appropriate benchmark test problems is a difficult but crucial aspect of MLP research. Desirable properties of an MLP benchmark include: widespread usage (enabling comparisons with earlier research); small size (allowing a large number of training runs); and similarity to 'real-world' MLP problems (giving a degree of confidence that a successful training method can be extended to practical applications). There are no MLP training problems which meet all these criteria, mainly because there is an approximate trade-off between the size of a problem and its applicability to the 'real world'. Given finite computational resources and time, this leaves a difficult choice.

[3] For practical applications with large training sets, this is a reasonable assumption to make with all the algorithms considered here. However, if two algorithms evaluate f and g roughly the same number of times per training run, it is worth taking into account the computational complexity of their respective updates.

Furthermore, an additional criterion - the presence of *known* local minima - is essential to this research, but (as noted in section 2.2.2) currently hard to satisfy.

The approach adopted here has been to perform a large number of training runs with small but non-trivial test problems. This approach has two key advantages: many important trends and characteristics in the comparative performance of different training methods only emerge if sufficient runs are performed; and much useful mathematical analysis (of the error surface and network behaviour) is currently feasible only with small network architectures. The two main training tasks used in this research - the XOR and sine problems - are described in the following sections.

4.2.2 XOR

XOR is the simplest and most widely-used MLP benchmark, but is criticised for having little in common with real-life applications[4]. For present purposes, the fact that MLPs frequently get caught in local minima when learning the XOR problem is its main recommendation. The minimal standard MLP that can learn XOR is 2-2-1.

When learning the XOR problem with any standard 2-*m*-1 architecture ($m \geq 2$) and using the error function Eq. 2.3, local minima may be found at two distinct error levels - $E \approx 0.08333$ and $E=0.0625$. That these are true local minima has been demonstrated by a numerical analysis of the XOR problem by Lisboa and Perantonis [1991][5]. It has often been assumed that local minima may occur at a third error level of $E=0.125$ (see, for example, [Hirose, Yamashita & Hijiya, 1991]); Lisboa and Perantonis appear to support

[4] 'The generalised parity problem [including XOR] is just the type of problem which distributed training by back error propagation is not suited to. Every bit of information in the data is conflicting, and there is absolutely no redundancy. Solutions to it are reached, not by generalising from sample data, but rather by reasoning about the problem as a whole at a much higher level.' [Lisboa ed., 1992, 252].

[5] The numerical simulations of Lisboa and Perantonis gave different error-levels to those in this research, attributable to their use of a different error function (the 'cross-entropy' error function

$$E = -\sum_{p}\sum_{n} \ln\left[\left(y^{L}_{p,n}\right)^{t_{p,n}}\left(1 - y^{L}_{p,n}\right)^{1-t_{p,n}}\right] ,$$

rather than Eq. 2.3) and different target outputs (0.1 and 0.9, rather than 0 and 1). The authors' main analytic findings about the existence and characterisation of XOR local minima are unaffected by these differences.

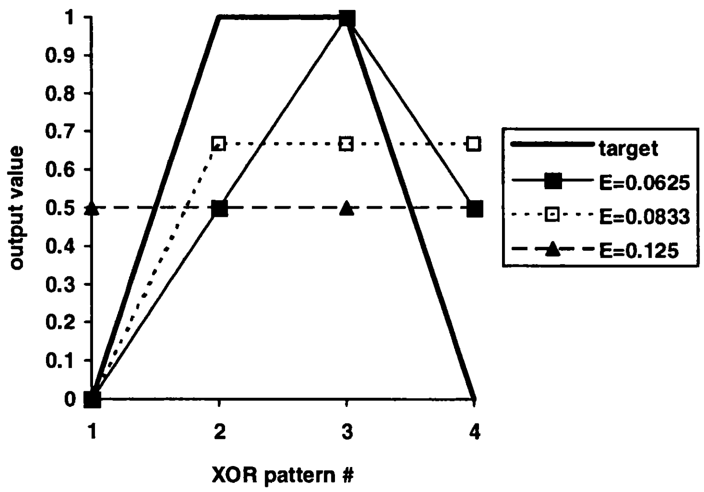
this assumption, but point out that saddle points frequently occur at the same error level. However, a recent analysis of the XOR error surface for an MLP with two hidden nodes indicates that local minima *cannot* occur at *E*=0.125 [Hamey, 1995]. The configuration of pattern classifications and misclassifications for each XOR stationary point is as follows:

- *E*=0.125 saddle points: 4 patterns 50% correct
- *E*=0.0625 local minima: 2 patterns 100% correct, 2 patterns 50% correct
- *E*≈0.0833 local minima: 1 pattern 100% correct, 1 pattern 66.7% correct, 2 patterns 33.3% correct.

Sample MLP output values for each of these stationary points are given in Graph 1. The XOR global minima are at *E*=0, i.e. there are no residuals at the solution.

In practice, MLPs are often trained using a modified XOR training set with targets 0.1 and 0.9 (rather than 0 and 1) to prevent network saturation. This option is ignored in the current chapter, but is related to the Expanded Range Approximation (ERA) strategy of section 5.2.

Graph 1 - Sample location of XOR stationary points in terms of pattern classification/ misclassification



4.2.3 The sine problem

The sine problem is a small but non-trivial example of a function-learning task. The version used in this research - based on that in [McInerney et al., 1989] - has a training set of 64 patterns, with each pattern (p_q) and target (t_q) defined as follows:

Eq. 4.1 $$p_q = (q-1)\left(\frac{3\pi}{2}\right)\left(\frac{1}{64}\right),\ \text{for } q = 1,\ldots,64$$

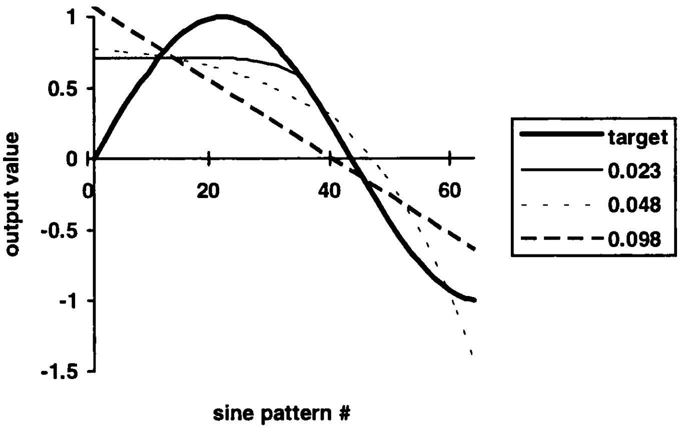
Eq. 4.2 $$t_q = \sin\left(p_q\right)\ .$$

The minimal standard MLP that can learn the sine function is 1-2-1. This architecture is prone to getting trapped in local minima, but with a much smaller frequency than XOR with a minimal architecture. McInerney et al.'s investigation (combining a smart raster scan of the error surface with numerical analyses of candidate minima) of a limited region of weight space found local minima 'like pinholes in a large flat board' [McInerney et al., 1989, 9]; these local minima occur at an error level of $E\approx0.023$, using the architecture and training set of this research[6]. With certain training methods used in this research, a small number of runs converged to stationary points at other error levels; whether these are local minima or saddle points is not known. The location of the sine stationary points, in term of the classification and misclassification of patterns in the training set, are shown in Graph 2.

McInerney et al.'s analysis indicated that the sine global minima are at $E>0$, i.e. there are residuals at the solution; the lowest error level attained for the sine problem with any of the training methods used in this research was 3.0^{-06}.

[6] The error-levels reported by McInerney et al. for both the sine and XOR local minima are different to those given here. In the absence of full details about McInerney et al.'s implementation, the precise cause of these differences is uncertain. However, the differences in error levels do not appear to be of any great significance; when the MLP used for this research was initialised using the sample location of the sine local minima given in [McInerney et al., 1989, 8], it rapidly became trapped at $E\approx0.023$, irrespective of the (non-global) training algorithm used.

Graph 2 - Location of sine stationary points in terms of pattern classification/ misclassification



4.3 Experimental Results

4.3.1 General training conditions

This section specifies the general conditions which remained constant for all the results presented later in this chapter.

MLP architecture and initialisation. All the results presented below are for an MLP with the minimal standard architecture capable of learning the given benchmark training set (on the assumption that larger networks are likely to yield fewer local minima), using the mean-squared error function (Eq. 2.3) and sigmoid squashing function (Eq. 2.2). The same 200 sets of random starting weights were used for all the tests, with a weight initialisation range of [-1, 1].

As the sine problem defined by Eq. 4.1 and Eq. 4.2 has a number of target values outside the range [0, 1], the corresponding MLP architecture must, of necessity, have *linear output nodes,* i.e. the sigmoid squashing function is not applied to the nodes in layer *L*. With binary problems, such as XOR, the architecture may have either linear output nodes or *sigmoid output nodes*. XOR with linear output nodes and XOR with sigmoid output nodes are treated as separate training tasks in this research, on the grounds that the corresponding error surfaces have different shapes and properties[7].

Termination conditions. Three common-sense termination criteria are generally applied to unconstrained minimisation: '"Have we solved the problem?" "Have we ground to a halt?" or "Have we run out of money, time, or patience?"' [Dennis & Schnabel, 1983, 159]. All three criteria are used in this research, but the specific tests performed are somewhat different to those advocated in the optimisation literature [Dennis & Schnabel, 1983, 159-61] [Gill, Murray & Wright, 1981, 305-12]; in order to accurately record the susceptibility of each training algorithm to becoming trapped in local minima, priority was given to the prevention of premature termination under all reasonable circumstances (at the expense of some - perhaps many - additional iterations with those training runs that *did* get trapped). A training run was deemed to have 'ground to a halt' only if there was no reduction in E for 10 epochs, or a high epoch limit - 100,000 for BP methods, 10,000 otherwise - exceeded. (For second-order methods with Brent's line search algorithm, this upper epoch limit was sufficiently high to ensure that it was reached by only a small fraction of runs.)

The optimal accuracy with which E should approximate a global minimum in the MLP error surface is problem- and application-specific, although an exceedingly accurate solution is rarely desirable (see section 2.1.2). The tabulated results below are for E=0.01 only; a consideration of training performance across a range of different error tolerances is deferred until section 4.4.2.

Training speed metric. The main training-speed metric used here is the number of *equivalent function evaluations* (EFEs), defined as follows:

Eq. 4.3
$$\text{EFEs} = \frac{\text{function evaluations} + \text{gradient evaluations}}{2},$$

where a function evaluation and gradient evaluation comprise P forward passes and P backward passes respectively (for a training set with P patterns). The computational effort associated with an EFE is, therefore, roughly equivalent to that associated with a traditional BP epoch. For a given training algorithm and choice of parameters, the number

[7] Every training algorithm tested here displayed a higher failure-rate for the XOR problem when the architecture had a sigmoid output node. This is attributable to the greater degree of freedom allowed in the values of network outputs with linear output nodes.

of EFEs required to achieve a given error tolerance is characterised by three main statistics - the mean, the standard deviation and the median (ignoring, in all three cases, runs that did not achieve that tolerance).

Implementation and settings. The code for this research was written in the C++ programming language. The programs were compiled using Borland C++ v3 and tested on two IBM-compatible PCs: a Gateway 2000 75MHz Pentium computer running MS-DOS 6.22 and a Zenith Data Systems 286 computer running MS-DOS 5.0. All training algorithms were implemented with double (15-digit) precision arithmetic.

The classical algorithms tested for this research were as follows:

- **BA** and **OL** - batch and on-line BP (section 2.3). The momentum parameter is implemented as in Eq. 2.10 and set either 'on' (0.9) or 'off' (0.0). With OL BP, training patterns were presented in random order.
- **SD** - the steepest descent algorithm (section 3.1.1).
- **CG** - the Polak-Ribiere (PR) conjugate gradient method (section 3.5.1).
- **QN** - the BFGS quasi-Newton method (section 3.4.1). For QN implemented with line-search methods but without positive definiteness enforced, the model Hessian was represented by the inverse Hessian; for QN implemented with a model-trust region strategy or with positive definiteness enforced, the model Hessian was represented directly (section 3.4.2).
- **NQN** - the 'memoryless' quasi-Newton method (section 3.4.3).
- **LM** - the Levenberg-Marquardt nonlinear least-squares algorithm (section 3.6).
- **BR** and **DBR** - Brent's line-search method, with and without derivatives (section 3.2.2).
- **BT** and **DBT** - backtracking line search, with and without derivatives (section 3.2.4). Accuracy parameter q (Eq. 3.30 for BT, Eq. 3.28 for DBT) was set to 0.9 and parameter u (Eq. 3.29) to 0.001.
- **DBT-DBR** - the hybrid Brent/backtracking algorithm with derivatives (section 3.2.5). Accuracy parameter q (Eq. 3.28) was set to 0.9 for the backtracking line search and 0.5 for Brent's method. Parameter u (Eq. 3.29) was set to 0.001 for the backtracking

line search and 0.25 for Brent's method. Both Brent's method and the backtracking strategy were implemented with derivatives. The DBT-DBR algorithm switching parameters were set as follows: a) switch to Brent's method whenever the backtracking strategy executes 4 or more EFEs during a single epoch; b) switch to the backtracking strategy after 3 epochs of Brent's method.

- **ST** - the 'simple' model-trust region strategy (section 3.3.1).
- **RT** - Fletcher's model-trust region strategy (section 3.3.2). The lower and upper limits for ratio r (Eq. 3.31) were set to 0.25 and 0.75 respectively.

Tabulated results. The following information will aid interpretation of the tables in sections 4.3.2, 4.3.3 and 4.3.4:

- *Global minima percentage column.* In a small number of cases the percentage of runs converging to known local and global minima does not total 100 - marked by one or more asterisks in the global minima column. This is an indication that one or more training runs converged to a stationary point or points not given a separate column (marked *), that one or more training runs failed to converge to any stationary point within the allowed number of epochs (marked **), or both (marked ***).
- *'EFEs per run' columns.* The mean, standard deviation (s.d.) and median number of EFEs are given for successful runs terminated at E=0.01.
- *'Resets / run' column.* With line-search methods, this column is reserved for the mean number of SD resets per run caused by a failure to generate an $\mathbf{s}_k$ that satisfies Eq. 3.9; no other resets are counted. With model-trust region methods, the mean number of times per run that $E(\mathbf{x}_k + \mathbf{s}_k) > E(\mathbf{x}_k)$ is recorded. In all cases, figures are for successful runs terminated at E=0.01.
- *'EFEs per k' column.* For line-search methods, this column contains the mean number of EFEs per k-iteration (epoch) for successful runs terminated at E=0.01. With model trust region methods, the number of EFEs per k for a given run is calculated as follows: EFEs / (EFEs - resets). (The 'EFEs per k' column is omitted for BP methods as a single EFE is performed at every epoch.)

Results for the BT and DBT backtracking line-search strategies are omitted for the XOR task with sigmoid output nodes on the grounds that, with all the multivariate algorithms tested, a significant percentage of runs failed to converge to any stationary point within 10,000 training epochs. Given that training times for the sine task were much longer than those for XOR - attributable, primarily, to the larger number of patterns in the sine training set - a slightly smaller range of test results are presented for the sine task (section 4.3.4) than for XOR (sections 4.3.2 and 4.3.3).

4.3.2 XOR results, sigmoid output nodes

Table 1

Method: batch (BA) or on-line (OL) BP, with training rate (η) and momentum (α in Eq. 2.10)

	minima (%)			EFEs per run		
method: η / α	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**
BA: 0.5 / 0.0	84.0	15.5	0.5	5,801.6	3,754.0	4,873.5
BA: 0.5 / 0.9	83.5	16.0	0.5	3,042.2	2,054.9	2,545.0
BA: 1.0 / 0.0	83.5	16.0	0.5	2,883.2	1,892.5	2,415.0
BA: 1.0 / 0.9	83.5	16.0	0.5	1,531.4	1,119.4	1,276.0
BA: 3.0 / 0.0	83.5	16.0	0.5	967.9	688.9	808.0
BA: 3.0 / 0.9	83.5	16.0	0.5	542.1	717.1	430.0
OL: 0.5 / 0.0	84.0	15.0	1.0	5,445.3	2,124.2	4,957.0
OL: 0.5 / 0.9	86.0	13.0	1.0	2,970.6	1,474.0	2,672.0
OL: 1.0 / 0.0	82.5	16.0	1.5	2,883.2	2,524.7	2,483.0
OL: 1.0 / 0.9	85.5	13.0	1.5	1,526.5	1,167.0	1,313.0
OL: 3.0 / 0.0	83.0	16.0	1.0	1,300.4	2,506.2	869.5
OL: 3.0 / 0.9	83.5	16.0	0.5	1,593.7	8,642.8	469.0

Table 2

Method: SD
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29)

	minima (%)			EFEs per run			EFEs
line search: *q* / *u*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**per *k***
BR: 0.9 / 0.001	83.0	16.0	1.0	282.8	144.2	253.0	3.51
BR: 0.9 / 0.25	83.0	16.0	1.0	284.5	144.0	254.0	3.51
BR: 0.9 / 0.4	82.5	16.5	1.0	282.0	133.3	255.0	3.52
BR: 0.5 / 0.001	82.0	17.0	1.0	400.4	214.3	343.3	5.16
BR: 0.5 / 0.25	82.0	17.0	1.0	399.3	215.2	340.8	5.18
BR: 0.1 / 0.001	80.5	18.5	1.0	509.8	1,225.8	319.5	7.24
DBR: 0.9 / 0.001	83.0	16.0	1.0	281.1	124.9	254.3	3.52
DBR: 0.9 / 0.25	82.5	16.5	1.0	284.2	125.7	257.0	3.57
DBR: 0.9 / 0.4	82.0	17.0	1.0	281.7	126.1	256.5	3.59
DBR: 0.5 / 0.001	82.5	16.5	1.0	277.0	122.9	252.5	3.53
DBR: 0.5 / 0.25	82.5	16.5	1.0	280.5	123.0	256.0	3.57
DBR: 0.1 / 0.001	83.0	16.0	1.0	410.0	628.9	342.3	4.32

Table 3

Method: PR CG with SD reset every n
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	60.5	28.5	11.0	92.6	56.2	77.0	0.83	3.92
BR: 0.9 / 0.25	60.0	27.5	12.5	90.7	54.1	76.3	0.62	3.97
BR: 0.9 / 0.4	62.5	26.5	11.0	152.7	619.1	73.5	0.32	4.93
BR: 0.5 / 0.001	60.0	30.5	9.5	141.3	109.8	108.3	1.20	5.55
BR: 0.5 / 0.25	59.5	30.5	10.0	138.0	100.6	108.0	0.65	5.74
BR: 0.1 / 0.001	65.0	27.5	7.5	166.2	144.3	122.5	0.08	7.13
DBR: 0.9 / 0.001	63.5	27.0	9.5	320.4	2,323.3	78.0	0.76	10.26
DBR: 0.9 / 0.25	62.5	28.5	9.0	402.1	2,488.0	78.0	0.68	11.54
DBR: 0.9 / 0.4	64.5	26.5	9.0	132.0	79.1	115.0	0.10	5.99
DBR: 0.5 / 0.001	61.0	26.0	13.0	179.4	611.7	82.8	0.45	6.41
DBR: 0.5 / 0.25	61.5	26.5	12.0	177.1	604.4	83.5	0.44	6.47
DBR: 0.1 / 0.001	62.5	27.0	10.5	258.8	311.4	179.5	0.08	10.68
DBT-DBR:	70.0	21.0	9.0	338.7	1,555.2	97.3	0.71	2.90

Table 4

Method: NQN with SD resets every n
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	67.0	24.5	8.5	146.4	153.3	109.0	0.11	3.51
BR: 0.9 / 0.25	68.5	25.0	6.5	135.6	91.2	109.0	0.11	3.51
BR: 0.9 / 0.4	66.5	26.5	7.0	139.9	142.6	106.5	0.11	3.62
BR: 0.5 / 0.001	61.0	28.0	11.0	231.6	353.9	141.0	0.12	5.32
BR: 0.5 / 0.25	60.5	28.0	11.5	192.5	205.2	137.0	0.13	5.36
BR: 0.1 / 0.001	56.0	29.0	15.0	173.0	124.6	133.3	0.14	6.89
DBR: 0.9 / 0.001	62.5	30.5	7.0	136.1	105.2	106.5	0.12	3.63
DBR: 0.9 / 0.25	64.0	29.5	6.5	155.7	259.1	112.8	0.16	3.74
DBR: 0.9 / 0.4	69.0	24.0	7.0	159.2	214.8	121.5	0.19	4.10
DBR: 0.5 / 0.001	66.0	26.5	7.5	151.7	137.6	116.0	0.13	3.78
DBR: 0.5 / 0.25	65.5	27.0	7.5	166.3	283.3	115.0	0.14	3.98
DBR: 0.1 / 0.001	58.0	31.5	10.5	250.7	180.1	213.5	0.12	8.39
DBT-DBR:	77.5	18.5	4.0	151.2	120.7	117.0	5.8	1.59

Table 5

Method: BFGS QN (no resets)
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	34.0	29.5	36.5	89.9	51.5	76.8	0.04	3.64
BR: 0.9 / 0.25	35.0	29.0	36.0	87.2	43.2	76.8	0.06	3.72
BR: 0.9 / 0.4	34.5	29.5	36.0	90.9	43.6	79.0	0.09	3.90
BR: 0.5 / 0.001	34.0	32.0	34.0	114.0	54.2	104.8	0.03	4.96
BR: 0.5 / 0.25	32.0	31.5	36.5	120.9	86.1	99.3	0.03	5.12
BR: 0.1 / 0.001	33.5	32.0	34.5	146.2	92.1	119.5	0.15	6.71
DBR: 0.9 / 0.001	35.5	29.5	35.0	91.5	41.0	78.0	0.10	3.75
DBR: 0.9 / 0.25	36.5	28.5	35.0	95.3	41.5	83.5	0.12	3.95
DBR: 0.9 / 0.4	35.5	30.5	34.0	106.8	57.3	87.0	0.10	4.42
DBR: 0.5 / 0.001	35.0	29.5	35.5	89.4	40.9	77.3	0.06	3.94
DBR: 0.5 / 0.25	36.0	29.0	35.0	89.7	38.1	77.3	0.08	4.02
DBR: 0.1 / 0.001	30.5	35.0	34.5	124.7	52.3	116.0	0.02	6.42
DBT-DBR:	44.5	28.5	27.0	78.7	42.9	66.5	1.57	1.89

Table 6

Method: BFGS QN (no resets) with positive definiteness enforced
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)			EFEs per run			indef.	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**H / run**	**per k**
BR: 0.9 / 0.001	34.5	28.0	37.5	90.1	54.3	77.0	0.43	3.59
BR: 0.9 / 0.25	35.0	26.5	38.5	89.1	52.4	76.8	0.21	3.75
BR: 0.9 / 0.4	34.0	29.0	37.0	90.0	52.7	78.3	0.24	3.92
BR: 0.5 / 0.001	34.5	31.5	34.0	118.0	58.7	105.5	0.09	4.98
BR: 0.5 / 0.25	32.5	31.5	36.0	116.6	65.5	99.5	0.17	5.04
BR: 0.1 / 0.001	33.0	32.0	35.0	148.8	103.9	117.8	0.30	6.72
DBR: 0.9 / 0.001	35.5	30.5	34.0	94.8	55.6	78.0	0.41	3.79
DBR: 0.9 / 0.25	36.0	29.0	35.0	96.6	51.0	82.0	0.25	3.98
DBR: 0.9 / 0.4	33.5	31.5	35.0	102.1	71.8	83.5	1.07	4.35
DBR: 0.5 / 0.001	35.5	28.5	36.0	90.1	44.2	77.0	0.38	3.94
DBR: 0.5 / 0.25	35.5	28.5	36.0	88.7	40.6	77.0	0.17	4.03
DBR: 0.1 / 0.001	30.5	34.0	35.5	134.5	91.9	116.0	0.23	6.60
DBT-DBR:	38.5	29.0	32.5	74.0	30.7	67.0	2.06	1.77

Table 7

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	42.5	28.0	29.5	74.1	42.4	62.0	21.25	1.40
RT: 0.001	36.5	28.5	35.0	66.9	60.8	55.0	17.22	1.35
RT: 0.01	34.0	32.5	33.5	60.4	24.6	56.0	14.38	1.31
RT: 0.1	34.5	31.0	34.5	63.1	29.8	58.0	14.20	1.29
ST: 0.0001	36.5	30.0	33.5	61.8	25.3	57.0	17.99	1.41
ST: 0.001	36.5	28.5	35.0	66.5	31.7	59.0	18.45	1.38
ST: 0.01	34.5	30.5	35.0	67.7	38.1	58.0	17.49	1.35
ST: 0.1	34.5	29.5	36.0	65.6	29.0	61.0	15.90	1.32

Table 8

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 4.0/10.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	38.5	28.0	33.5	70.5	34.2	60.0	22.31	1.46
RT: 0.001	39.0	30.0	31.0	66.9	36.4	59.0	19.43	1.41
RT: 0.01	37.5	28.5	34.0	66.2	33.8	60.0	18.75	1.39
RT: 0.1	36.5	34.0	29.5	65.4	28.6	60.0	18.08	1.39
ST: 0.0001	37.5	29.0	33.5	76.1	40.5	61.0	25.76	1.51
ST: 0.001	40.0	26.0	34.0	74.8	44.2	64.0	24.20	1.48
ST: 0.01	39.5	28.5	32.0	70.9	39.1	60.0	21.46	1.43
ST: 0.1	38.5	31.5	30.0	68.9	28.2	64.0	20.58	1.43

Table 9

Method: BFGS QN with SD resets every *n*
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per** k
BR: 0.9 / 0.001	65.0	27.0	8.0	127.4	142.7	96.3	0.08	3.55
BR: 0.9 / 0.25	65.0	27.5	7.5	143.6	227.5	99.5	0.09	3.61
BR: 0.9 / 0.4	61.0	28.5	10.5	119.8	110.7	95.8	0.11	3.66
BR: 0.5 / 0.001	59.0	31.0	10.0	154.0	121.7	121.3	0.08	4.92
BR: 0.5 / 0.25	58.0	30.5	11.5	153.1	116.4	120.8	0.08	5.02
BR: 0.1 / 0.001	50.5	33.0	16.5	159.2	72.0	147.0	0.09	6.46
DBR: 0.9 / 0.001	61.0	30.0	9.0	112.9	88.7	91.8	0.07	3.63
DBR: 0.9 / 0.25	63.5	29.0	7.5	128.2	121.3	96.0	0.10	3.80
DBR: 0.9 / 0.4	60.0	31.5	8.5	128.7	118.8	101.5	0.07	4.07
DBR: 0.5 / 0.001	63.0	30.0	7.0	147.8	219.0	93.3	0.10	4.10
DBR: 0.5 / 0.25	62.5	30.5	7.0	141.1	170.9	96.0	0.08	4.13
DBR: 0.1 / 0.001	56.5	31.0	12.5	201.8	137.0	168.5	0.12	7.25
DBT-DBR:	71.5	22.0	6.5	266.0	843.6	96.0	41.5	1.93

Table 10

Method: LM
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per** k
RT: 0.0001	82.0	17.5	0.5	14.1	7.1	12.0	2.74	1.24
RT: 0.001	85.5	14.5	0.0	10.5	2.7	10.0	0.57	1.06
RT: 0.01	90.5	9.5	0.0	12.4	2.8	12.0	0.41	1.03
RT: 0.1	92.0	8.0	0.0	15.7	4.4	15.0	0.40	1.03
ST: 0.0001	82.0	17.0	1.0	14.6	8.5	12.0	3.16	1.28
ST: 0.001	86.0	14.0	0.0	10.6	2.7	10.0	0.70	1.07
ST: 0.01	90.5	9.5	0.0	12.4	2.9	12.0	0.47	1.04
ST: 0.1	92.0	8.0	0.0	15.8	4.4	15.0	0.45	1.03

Table 11

Method: LM
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 4.0/10.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	81.5	17.0	1.5	12.9	4.9	12.0	2.44	1.23
RT: 0.001	86.0	13.5	0.5	11.5	3.9	11.0	1.30	1.13
RT: 0.01	91.5	8.5	0.0	12.3	4.3	11.0	1.04	1.09
RT: 0.1	92.5	7.5	0.0	13.8	3.7	13.0	0.98	1.08
ST: 0.0001	81.0	18.0	1.0	13.9	7.0	12.0	3.01	1.28
ST: 0.001	86.5	13.0	0.5	12.0	4.9	11.0	1.71	1.17
ST: 0.01	91.5	8.5	0.0	12.6	4.9	11.0	1.26	1.11
ST: 0.1	92.5	7.5	0.0	13.9	3.7	13.0	1.14	1.09

4.3.3 XOR results, linear output nodes

Table 12

Method: batch (BA) or on-line (OL) BP, with training rate (η) and momentum (α in Eq. 2.10)

	minima (%)			EFEs per run		
method: η / α	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**
BA: 0.1 / 0.0	93.0	7.0	0.0	8,026.2	8,541.2	5,746.5
BA: 0.1 / 0.9	94.5	5.5	0.0	5,067.9	8,799.1	3,104.0
BA: 0.25 / 0.0	94.0	6.0	0.0	3,678.6	6,031.9	2,335.0
BA: 0.25 / 0.9	95.0	5.0	0.0	1,988.1	3,370.9	1,269.0
BA: 0.5 / 0.0	95.0	5.0	0.0	3,088.2	13,916.6	1,206.0
OL: 0.1 / 0.0	93.5	6.5	0.0	7,254.6	7,407.1	5,570.0
OL: 0.1 / 0.9	95.5	4.5	0.0	4,017.2	4,441.4	2,909.0
OL: 0.25 / 0.0	94.5	5.5	0.0	3,396.2	7,168.0	2,286.0
OL: 0.25 / 0.9	93.5	6.5	0.0	1,928.9	4,812.3	1,108.0
OL: 0.5 / 0.0	93.0	7.0	0.0	1,611.5	2,986.1	1,038.5

Table 13

Method: SD
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29)

	minima (%)			EFEs per run			EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**per k**
BR: 0.9 / 0.001	95.5	4.5	0.0	1,385.3	1,412.3	1,048.5	2.72
BR: 0.9 / 0.25	95.0	5.0	0.0	1,464.1	2,086.3	1,074.0	2.76
BR: 0.9 / 0.4	94.0	6.0	0.0	1,265.3	3,025.9	675.3	3.17
BR: 0.5 / 0.001	95.5	4.5	0.0	2,529.1	3,565.4	1,753.0	4.23
BR: 0.5 / 0.25	94.0	6.0	0.0	2,472.2	2,841.7	1,766.8	4.27
BR: 0.1 / 0.001	94.5	5.5	0.0	1,564.1	5,084.1	720.0	5.69
DBR: 0.9 / 0.001	94.5	5.5	0.0	1,283.0	1,149.5	1,018.0	2.75
DBR: 0.9 / 0.25	93.5	6.5	0.0	1,285.9	1,546.3	924.5	2.77
DBR: 0.9 / 0.4	94.5	5.5	0.0	1,229.3	1,977.4	710.0	3.69
DBR: 0.5 / 0.001	93.5	6.5	0.0	1,212.4	1,334.4	924.5	2.79
DBR: 0.5 / 0.25	93.5	6.5	0.0	1,285.2	1,538.3	924.5	2.77
DBR: 0.1 / 0.001	93.0	7.0	0.0	1,128.5	1,869.5	662.5	4.10

Table 14

Method: PR CG with SD reset every n
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); BT, DBT, DBT-BR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	93.5	6.0	0.5	226.0	609.0	145.0	27.8	2.31
BR: 0.9 / 0.25	93.0	6.0	1.0	317.9	2,130.9	95.5	2.72	2.97
BR: 0.9 / 0.4	91.5	8.5	0.0	133.8	180.9	99.5	0.44	3.21
BR: 0.5 / 0.001	94.0	5.0	1.0	369.6	975.8	167.0	56.67	2.38
BR: 0.5 / 0.25	90.5	9.5	0.0	266.0	1,056.5	131.5	2.02	4.22
BR: 0.1 / 0.001	90.5	8.5	1.0	215.0	233.8	162.0	7.51	4.56
DBR: 0.9 / 0.001	92.5	7.5	0.0	354.3	1,667.6	134.0	39.36	2.41
DBR: 0.9 / 0.25	90.0	9.5	0.5	138.8	206.6	103.3	1.61	3.12
DBR: 0.9 / 0.4	91.0	8.0	1.0	256.9	1,298.0	100.3	0.27	3.37
DBR: 0.5 / 0.001	90.5	8.5	1.0	206.6	715.2	100.0	2.14	3.15
DBR: 0.5 / 0.25	90.0	9.0	1.0	201.9	641.6	104.0	1.96	3.16
DBR: 0.1 / 0.001	89.5	9.0	1.5	189.6	160.0	151.5	0.01	5.67
BT:	89.0**	5.0	1.5	179.0	191.3	125.3	26.98	1.58
DBT:	86.0**	4.5	0.0	194.4	388.3	112.0	21.56	1.40
DBT-DBR:	94.5	5.0	0.5	139.8	125.1	110.5	13.43	1.49

Table 15

Method: NQN with SD resets every *n*
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); BT, DBT, DBT-DBR

	minima (%)			EFEs per run			resets	EFEs
line search: *q* / *u*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
BR: 0.9 / 0.001	95.0	4.5	0.5	354.5	1,377.4	106.5	15.60	2.70
BR: 0.9 / 0.25	94.0	6.0	0.0	270.1	1,100.9	107.5	0.91	3.06
BR: 0.9 / 0.4	93.5	6.0	0.5	279.2	1,256.4	103.5	0.76	3.13
BR: 0.5 / 0.001	89.5	9.0	1.5	244.2	511.7	129.0	5.78	3.80
BR: 0.5 / 0.25	90.5	8.0	1.5	184.0	195.2	136.0	0.34	4.29
BR: 0.1 / 0.001	89.5	8.0	2.5	288.6	738.5	171.0	7.95	4.83
DBR: 0.9 / 0.001	94.0	6.0	0.0	205.9	457.0	105.3	2.35	3.06
DBR: 0.9 / 0.25	92.0	8.0	0.0	303.6	1,103.3	104.8	2.46	3.10
DBR: 0.9 / 0.4	91.5	7.5	1.0	417.7	2,320.9	111.0	1.06	3.29
DBR: 0.5 / 0.001	91.5	6.5	2.0	145.2	160.4	101.5	0.56	3.29
DBR: 0.5 / 0.25	90.0	7.5	2.5	139.1	133.8	101.5	0.48	3.29
DBR: 0.1 / 0.001	90.5	8.0	1.5	353.4	1,309.0	160.0	0.31	5.23
BT:	89.0**	6.5	0.0	226.6	441.3	114.8	18.84	1.72
DBT:	88.0**	6.5	0.5	269.8	950.8	92.0	7.69	1.35
DBT-DBR:	92.5	7.0	0.5	249.3	826.7	96.5	4.44	1.30

Table 16

Method: BFGS QN (no resets)
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); BT, DBT, DBT-DBR

	minima (%)			EFEs per run			resets	EFEs
line search: *q* / *u*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
BR: 0.9 / 0.001	65.0	22.0	13.0	90.2	111.3	71.0	0.08	3.23
BR: 0.9 / 0.25	66.5	19.5	14.0	95.4	122.5	73.5	0.09	3.32
BR: 0.9 / 0.4	62.0	24.0	14.0	84.1	85.5	70.0	0.06	3.39
BR: 0.5 / 0.001	63.5	23.0	13.5	123.6	126.5	92.5	0.02	4.71
BR: 0.5 / 0.25	67.0	20.5	12.5	134.2	137.6	99.0	0.06	4.94
BR: 0.1 / 0.001	64.5	23.0	12.5	144.9	111.6	115.0	0.06	6.26
DBR: 0.9 / 0.001	64.5	23.5	12.0	97.1	116.5	75.0	0.16	3.50
DBR: 0.9 / 0.25	68.0	20.5	11.5	103.5	120.7	73.8	0.09	3.55
DBR: 0.9 / 0.4	64.5	24.5	11.0	89.8	47.7	77.5	0.02	3.62
DBR: 0.5 / 0.001	69.5	19.0	11.5	119.3	192.2	79.5	0.10	3.70
DBR: 0.5 / 0.25	69.0	21.0	10.0	114.2	162.0	78.5	0.15	3.75
DBR: 0.1 / 0.001	64.0	23.5	12.5	147.5	132.6	116.0	0.13	5.82
BT:	76.0	16.0	8.0	109.8	133.0	68.0	3.74	1.95
DBT:	74.5**	17.5	7.5	78.6	80.0	52.0	2.09	1.54
DBT-DBR:	73.0	19.0	8.0	89.0	100.7	55.3	2.03	1.90

Table 17

Method: BFGS QN (no resets) with positive-definiteness enforced
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); BT, DBT, DBT-DBR

	minima (%)			EFEs per run			indef.	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**H / run**	**per k**
BR: 0.9 / 0.001	63.0	23.0	14.0	91.0	101.9	72.3	0.07	3.24
BR: 0.9 / 0.25	65.5	20.5	14.0	111.4	194.6	74.0	0.23	3.31
BR: 0.9 / 0.4	62.0	24.5	13.5	85.9	90.0	71.5	0.09	3.43
BR: 0.5 / 0.001	65.0	22.0	13.0	142.2	246.5	95.5	0.04	4.70
BR: 0.5 / 0.25	67.5	19.5	13.0	140.8	155.3	99.0	0.04	4.92
BR: 0.1 / 0.001	64.5	23.0	12.5	153.1	143.2	115.5	0.11	6.30
DBR: 0.9 / 0.001	62.0	23.5	14.5	89.8	51.8	75.3	0.03	3.42
DBR: 0.9 / 0.25	66.0	22.0	12.0	93.8	76.1	76.3	0.05	3.50
DBR: 0.9 / 0.4	65.5	23.5	11.0	94.5	64.3	79.0	0.05	3.62
DBR: 0.5 / 0.001	68.0	20.0	12.0	102.5	96.6	77.3	0.03	3.60
DBR: 0.5 / 0.25	66.0	21.5	12.5	92.1	56.9	76.8	0.02	3.61
DBR: 0.1 / 0.001	64.0	22.0	14.0	139.2	105.1	116.0	0.04	5.83
BT:	68.0**	21.5	7.5	185.9	908.0	72.5	5.24	2.84
DBT:	70.0**	18.0	10.5	147.1	618.3	56.5	7.53	2.63
DBT-DBR:	70.0	20.5	9.5	77.3	122.4	56.3	1.19	1.59

Table 18

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	74.0	19.5	6.5	72.1	69.0	53.0	21.57	1.43
RT: 0.001	70.5	20.0	9.5	63.9	42.9	52.0	17.84	1.39
RT: 0.01	70.0	18.0	12.0	69.7	72.8	53.5	19.09	1.38
RT: 0.1	67.0	20.5	12.5	58.4	33.5	49.0	14.16	1.32
ST: 0.0001	72.5	21.0	6.5	65.7	40.4	54.0	20.99	1.47
ST: 0.001	72.0	18.5	9.5	63.1	37.3	54.5	18.76	1.42
ST: 0.01	66.0	20.5	13.5	66.8	50.7	52.0	19.15	1.40
ST: 0.1	66.5	23.0	10.5	66.2	75.7	50.0	17.95	1.37

Table 19

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 4.0/10.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	72.5	19.5	8.0	87.1	167.6	55.0	28.28	1.48
RT: 0.001	72.0	19.0	9.0	70.9	62.6	56.0	22.15	1.45
RT: 0.01	71.5	17.5	11.0	65.3	39.9	55.0	19.58	1.43
RT: 0.1	68.5	19.0	12.5	66.1	48.7	54.0	19.56	1.42
ST: 0.0001	72.0	19.5	8.5	68.7	41.4	60.5	23.54	1.52
ST: 0.001	73.0	19.0	8.0	69.8	52.5	59.0	23.16	1.50
ST: 0.01	70.0	20.0	10.0	69.7	47.0	59.5	22.96	1.49
ST: 0.1	68.0	19.5	12.5	67.4	51.2	55.0	21.54	1.47

Table 20

Method: BFGS QN with SD resets every n
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); BT, DBT, DBT-DBR

	minima (%)			EFEs per run			resets	EFEs
line search: q / u	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	94.0	5.5	0.5	362.7	2,221.0	85.5	0.09	3.48
BR: 0.9 / 0.25	91.5	7.5	1.0	127.6	212.3	81.5	0.04	3.25
BR: 0.9 / 0.4	90.0	8.5	1.5	140.5	375.5	82.0	0.03	3.34
BR: 0.5 / 0.001	89.0	9.5	1.5	159.7	167.4	107.3	0.03	4.49
BR: 0.5 / 0.25	88.5	9.0	2.5	175.0	299.0	105.5	0.02	4.67
BR: 0.1 / 0.001	89.0	10.0	1.0	208.4	507.3	125.8	0.05	6.18
DBR: 0.9 / 0.001	93.5	6.0	0.5	121.4	150.0	85.0	0.02	3.33
DBR: 0.9 / 0.25	92.5	6.5	1.0	119.7	140.9	88.0	0.04	3.37
DBR: 0.9 / 0.4	91.0	8.0	1.0	124.4	166.7	89.8	0.04	3.55
DBR: 0.5 / 0.001	91.5	7.5	1.0	115.2	142.0	87.0	0.04	3.48
DBR: 0.5 / 0.25	92.0	7.0	1.0	116.5	150.2	87.5	0.04	3.49
DBR: 0.1 / 0.001	88.5	10.0	1.5	216.1	322.7	135.0	0.12	5.72
BT:	95.0**	3.5	1.0	221.4	557.0	95.0	8.44	1.74
DBT:	92.5**	4.5	1.5	403.0	2,072.7	63.0	3.11	1.11
DBT-DBR:	92.0	6.5	1.5	124.7	255.4	63.0	2.75	1.59

Table 21

Method: LM
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	90.5	9.5	0.0	67.7	461.4	12.0	15.59	1.30
RT: 0.001	96.0	4.0	0.0	11.8	10.7	10.0	1.60	1.16
RT: 0.01	99.0	1.0	0.0	30.1	277.0	10.0	4.49	1.18
RT: 0.1	99.5	0.5	0.0	13.4	8.3	12.0	0.48	1.04
ST: 0.0001	89.5	10.5	0.0	143.9	948.1	12.0	46.94	1.48
ST: 0.001	97.0	3.0	0.0	11.7	11.9	10.5	1.76	1.18
ST: 0.01	99.0	1.0	0.0	42.5	451.1	10.0	11.16	1.36
ST: 0.1	99.5	0.5	0.0	13.4	8.1	12.0	0.54	1.04

Table 22

Method: LM
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 4.0/10.0

	minima (%)			EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	92.0	8.0	0.0	19.7	54.3	12.0	5.18	1.36
RT: 0.001	94.5	5.5	0.0	12.9	14.6	10.0	2.22	1.21
RT: 0.01	98.0	2.0	0.0	15.7	63.4	10.0	2.35	1.18
RT: 0.1	98.5	1.5	0.0	28.7	221.1	11.0	3.21	1.13
ST: 0.0001	92.5	7.5	0.0	28.7	130.3	12.0	9.05	1.46
ST: 0.001	95.0	5.0	0.0	12.8	13.1	10.0	2.38	1.23
ST: 0.01	97.0	2.5	0.5	18.7	112.7	10.0	4.02	1.27
ST: 0.1	98.5	1.5	0.0	52.1	551.4	11.0	16.06	1.45

4.3.4 Sine results

Table 23

Method: batch (BA) or on-line (OL) BP, with training rate (η) and momentum (α in Eq. 2.10)

	minima (%)		EFEs per run		
method: η / α	**global**	**0.023**	**mean**	**s.d.**	**median**
BA: 0.1 / 0.0	100.0	0.0	3,543.2	3,797.1	2,714.5
BA: 0.1 / 0.9	100.0	0.0	1,914.9	2,013.3	1,444.5
BA: 0.25 / 0.0	100.0	0.0	1,429.5	1,456.8	1,097.5
BA: 0.25 / 0.9	100.0	0.0	752.0	689.3	580.5
BA: 0.5 / 0.0	100.0	0.0	798.4	936.4	565.0
OL: 0.1 / 0.0	100.0	0.0	3,576.6	4,208.9	2,635.5
OL: 0.1 / 0.9	100.0	0.0	1,930.8	2,376.0	1,386.5
OL: 0.25 / 0.0	100.0	0.0	1,499.8	2,100.1	1,053.5
OL: 0.25 / 0.9	100.0	0.0	901.7	1,800.1	556.0
OL: 0.5 / 0.0	100.0	0.0	1,012.9	3,195.5	527.0

Table 24

Method: SD
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29)

	minima (%)		EFEs per run			EFEs
line search: *q* / *u*	**global**	**0.023**	**mean**	**s.d.**	**median**	**per *k***
BR: 0.9 / 0.001	100.0	0.0	1,529.0	1,710.5	1,209.5	3.10
BR: 0.9 / 0.25	100.0	0.0	1,489.9	999.6	1,306.5	3.22
BR: 0.9 / 0.4	100.0	0.0	1,543.6	1,161.7	1,302.3	3.33
BR: 0.5 / 0.001	100.0	0.0	1,971.4	1,636.3	1,562.0	3.76
BR: 0.5 / 0.25	100.0	0.0	2,188.0	1,347.5	1,773.0	4.10
BR: 0.1 / 0.001	100.0	0.0	1,711.8	2,671.6	1,394.3	5.50
DBR: 0.9 / 0.001	100.0	0.0	1,628.7	1,528.2	1,216.8	3.29
DBR: 0.9 / 0.25	100.0	0.0	1,675.4	1,264.2	1,314.0	3.33
DBR: 0.9 / 0.4	100.0	0.0	1,575.0	1,260.5	1,313.0	3.74
DBR: 0.5 / 0.001	100.0	0.0	1,716.2	1,458.4	1,349.0	3.33
DBR: 0.5 / 0.25	100.0	0.0	1,680.3	1,302.8	1,311.0	3.34
DBR: 0.1 / 0.001	100.0	0.0	1,447.6	1,569.2	1,002.0	4.20

Table 25

Method: PR CG with SD reset every *n*
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); DBT-BR

	minima (%)		EFEs per run			resets	EFEs
line search: *q* / *u*	global	0.023	mean	s.d.	median	/ run	per *k*
BR: 0.9 / 0.001	99.5	0.5	234.4	233.7	167.5	21.92	2.58
BR: 0.9 / 0.25	100.0	0.0	131.4	83.1	111.5	0.68	2.99
BR: 0.9 / 0.4	100.0	0.0	137.6	233.8	106.8	0.12	3.07
BR: 0.5 / 0.001	100.0	0.0	267.0	565.2	176.8	17.78	3.13
BR: 0.5 / 0.25	100.0	0.0	147.5	77.5	127.8	0.56	4.03
BR: 0.1 / 0.001	99.5	0.5	190.8	93.5	160.0	4.64	4.75
DBR: 0.9 / 0.001	100.0	0.0	219.5	286.1	145.5	10.72	2.89
DBR: 0.9 / 0.25	100.0	0.0	137.6	99.8	116.5	0.60	3.18
DBR: 0.9 / 0.4	99.5	0.5	129.2	76.3	113.0	0.10	3.39
DBR: 0.5 / 0.001	100.0	0.0	161.2	228.2	116.3	0.58	3.26
DBR: 0.5 / 0.25	100.0	0.0	143.6	116.4	114.3	0.47	3.28
DBR: 0.1 / 0.001	99.5	0.5	156.3	105.6	135.0	0.0	4.74
DBT-DBR:	99.5	0.5	145.6	380.4	84.0	3.43	1.28

Table 26

Method: NQN with SD resets every *n*
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); DBT-BR

	minima (%)		EFEs per run			resets	EFEs
line search: *q* / *u*	global	0.023	mean	s.d.	median	/ run	per *k*
BR: 0.9 / 0.001	98.5	1.5	165.2	184.1	125.5	8.84	2.71
BR: 0.9 / 0.25	98.5	1.5	178.7	706.1	101.5	0.77	2.94
BR: 0.9 / 0.4	98.5	1.5	205.6	1,104.9	97.0	0.25	3.06
BR: 0.5 / 0.001	99.0	1.0	258.1	941.4	146.0	7.91	3.65
BR: 0.5 / 0.25	99.5	0.5	323.1	1,558.0	129.0	0.50	4.19
BR: 0.1 / 0.001	99.0	1.0	255.8	911.9	166.8	3.37	5.78
DBR: 0.9 / 0.001	100.0	0.0	261.2	1,277.2	118.5	3.30	2.96
DBR: 0.9 / 0.25	99.5	0.5	270.8	1,096.0	107.0	0.78	3.07
DBR: 0.9 / 0.4	99.5	0.5	275.5	1,115.6	108.0	0.29	3.26
DBR: 0.5 / 0.001	99.5	0.5	222.2	615.5	113.5	0.47	3.19
DBR: 0.5 / 0.25	99.5	0.5	236.5	737.8	113.5	0.47	3.18
DBR: 0.1 / 0.001	99.0	1.0	660.6	4,414.6	134.8	0.05	4.74
DBT-DBR:	100.0	0.0	132.9	490.4	71.3	1.34	1.34

Table 27

Method: BFGS QN (no resets)
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); DBT-BR

	minima (%)		**EFEs per run**			**resets**	**EFEs**
line search: *q* / *u*	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per** *k*
BR: 0.9 / 0.001	89.0*	8.0	88.7	38.9	76.3	0.01	2.98
BR: 0.9 / 0.25	89.0*	8.0	90.9	44.4	77.0	0.02	3.11
BR: 0.9 / 0.4	89.5*	7.5	96.0	51.1	79.0	0.04	3.23
BR: 0.5 / 0.001	91.5*	5.5	131.5	71.7	108.0	0.01	4.51
BR: 0.5 / 0.25	90.0*	7.0	133.5	68.3	106.8	0.01	4.76
BR: 0.1 / 0.001	90.0*	7.0	159.5	82.5	134.8	0.03	6.06
DBR: 0.9 / 0.001	90.0*	7.5	100.1	76.4	82.3	0.01	3.18
DBR: 0.9 / 0.25	88.5*	9.5	97.4	48.7	83.0	0.02	3.29
DBR: 0.9 / 0.4	88.5*	9.5	104.4	55.8	85.5	0.01	3.52
DBR: 0.5 / 0.001	88.0*	9.0	96.5	44.1	83.5	0.02	3.39
DBR: 0.5 / 0.25	88.0*	9.5	97.5	44.7	84.3	0.02	3.41
DBR: 0.1 / 0.001	88.5*	8.0	134.4	72.3	110.5	0.03	5.01
DBT-BR:	89.5*	9.5	65.4	56.2	48.0	0.76	1.55

Table 28

Method: BFGS QN (no resets) with positive-definiteness enforced
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); BT, DBT, DBT-DBR

	minima (%)		**EFEs per run**			**resets**	**EFEs**
line search: *q* / *u*	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per** *k*
BR: 0.9 / 0.001	88.5*	8.0	89.5	38.6	76.5	0.02	2.98
BR: 0.9 / 0.25	89.0*	8.0	91.4	45.0	77.0	0.02	3.09
BR: 0.9 / 0.4	89.5*	7.5	96.7	52.9	79.0	0.04	3.23
BR: 0.5 / 0.001	90.5*	6.0	129.0	64.1	108.0	0.02	4.51
BR: 0.5 / 0.25	90.0*	7.0	132.7	67.7	106.3	0.01	4.76
BR: 0.1 / 0.001	90.0*	7.0	161.3	84.7	134.8	0.04	6.08
DBR: 0.9 / 0.001	89.0*	8.5	96.4	48.1	82.0	0.02	3.19
DBR: 0.9 / 0.25	88.5*	9.0	104.0	95.5	83.0	0.19	3.30
DBR: 0.9 / 0.4	89.5*	7.5	107.5	59.1	88.0	0.03	3.53
DBR: 0.5 / 0.001	89.0*	7.5	99.8	50.5	84.5	0.02	3.41
DBR: 0.5 / 0.25	89.5*	7.5	101.6	51.5	85.5	0.01	3.42
DBR: 0.1 / 0.001	88.5*	7.5	135.3	72.6	110.5	0.04	4.98
BT:	88.5***	5.5	1,337.6	10,876.2	69.5	5.38	6.97
DBT:	85.5***	8.5	61.9	47.6	45.0	0.99	1.40
DBT-BR:	89.5*	8.0	66.1	56.9	50.5	0.83	1.50

Table 29

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)		EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	93.0*	6.0	68.4	50.5	53.5	19.63	1.37
RT: 0.001	93.0*	6.5	60.2	35.3	48.5	16.12	1.37
RT: 0.01	94.0*	5.5	59.5	37.6	48.0	14.80	1.33
RT: 0.1	93.0	7.0	59.1	34.1	49.0	13.47	1.30
ST: 0.0001	92.5*	6.5	67.8	50.5	52.0	21.32	1.47
ST: 0.001	93.5*	6.0	65.2	42.3	51.0	20.06	1.44
ST: 0.01	94.0*	5.5	61.3	32.7	49.0	17.63	1.40
ST: 0.1	93.5*	6.0	63.0	45.8	49.0	16.83	1.36

Table 30

Method: BFGS QN (no resets)
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 4.0/10.0

	minima (%)		EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	93.0*	6.0	69.4	45.9	52.0	21.58	1.46
RT: 0.001	93.5*	6.0	66.5	46.2	52.0	19.85	1.43
RT: 0.01	92.5*	6.0	65.0	45.4	50.0	18.29	1.40
RT: 0.1	92.0*	7.5	63.0	37.5	50.0	17.86	1.39
ST: 0.0001	90.0*	9.0	72.7	84.4	53.0	25.29	1.53
ST: 0.001	90.0*	9.5	68.3	82.3	52.0	22.16	1.50
ST: 0.01	91.0*	7.5	68.4	79.9	51.0	22.68	1.51
ST: 0.1	92.0*	7.0	69.0	80.6	50.0	21.99	1.47

Table 31

Method: BFGS QN with SD resets every n
Line search: BR, or DBR, with parameters q (Eq. 3.28, Eq. 3.30) and u (Eq. 3.29); DBT-BR

	minima (%)		EFEs per run			resets	EFEs
line search: q / u	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per k**
BR: 0.9 / 0.001	100.0	0.0	102.0	74.1	82.0	0.03	2.88
BR: 0.9 / 0.25	99.5	0.5	101.8	79.1	81.5	0.02	2.97
BR: 0.9 / 0.4	99.5	0.5	100.2	59.1	85.5	0.02	3.06
BR: 0.5 / 0.001	100.0	0.0	137.9	89.5	114.0	0.03	4.18
BR: 0.5 / 0.25	100.0	0.0	141.8	131.2	113.5	0.02	4.34
BR: 0.1 / 0.001	100.0	0.0	354.4	2,114.4	139.5	0.02	6.50
DBR: 0.9 / 0.001	100.0	0.0	109.3	73.1	87.8	0.02	3.08
DBR: 0.9 / 0.25	99.5	0.5	109.9	72.3	88.5	0.04	3.15
DBR: 0.9 / 0.4	98.5	1.5	143.0	480.7	93.0	0.03	3.40
DBR: 0.5 / 0.001	100.0	0.0	111.1	74.7	90.3	0.02	3.24
DBR: 0.5 / 0.25	100.0	0.0	110.8	72.6	91.0	0.02	3.25
DBR: 0.1 / 0.001	100.0	0.0	163.6	159.9	131.3	0.03	5.04
DBT-DBR:	100.0	0.0	104.1	247.9	54.5	0.48	1.22

Table 32

Method: LM
Model-trust region strategy: RT, or ST, with parameter u_0 (Eq. 3.15) and reduction/growth constants 2.0/4.0

	minima (%)		EFEs per run			resets	EFEs
trust method: u_0	**global**	**0.023**	**mean**	**s.d.**	**median**	**/ run**	**per k**
RT: 0.0001	97.0*	2.0	172.1	47.1	174.0	25.66	1.18
RT: 0.001	99.0*	0.5	165.9	48.5	165.0	21.62	1.15
RT: 0.01	100.0	0.0	152.8	45.8	141.0	14.07	1.10
RT: 0.1	100.0	0.0	146.9	43.0	134.0	10.86	1.08

4.4 Comparison of Training Methods

4.4.1 Convergence to local minima

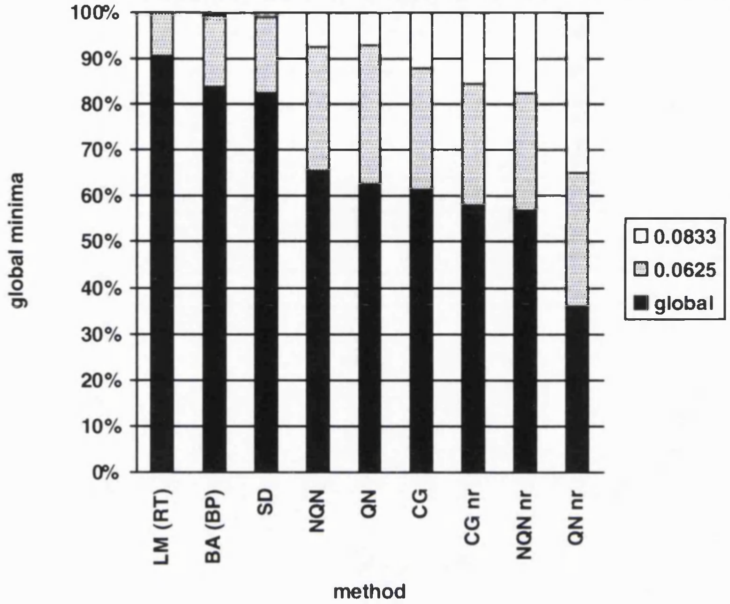
It is evident from the results in section 4.3 that there is a marked disparity between the frequency of convergence to global, as opposed to local, minima for different training algorithms. A major - and novel - finding of this research is that there is a strong correlation between, on the one hand, the frequency with which a training algorithm converges to global minima and, on the other, the frequency with which that algorithm discards derivative information accumulated during one or more previous training epochs. This effect can be traced both in the performance of different multivariate algorithms, and in the performance of different versions of the same multivariate algorithm.

Graphs 3, 4 and 5 plot, for each multivariate algorithm and training task, the percentage of runs that converged to global and local minima using (where possible) the same line-search method and parameters - Brent's method with derivatives and with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29), or **DBR: 0.5 / 0.25** for short. Two additional algorithms were tested for this purpose - the Polak-Ribiere conjugate gradients without resets **CG nr** and the memoryless quasi-Newton method without resets **NQN nr**. A clear trend emerges if we divide the training methods into three broad categories: methods which never store derivative information (i.e. generate a new model at every epoch); methods which discard accumulated derivative information approximately once every n epochs for an n-weight MLP; and methods which rarely, if ever, discard derivative information. Methods from the first category (BA[8], OL, SD and LM) consistently converge to a global minimum as frequently or more frequently than methods from the second category (CG, NQN and QN with SD resets); methods from the second category consistently converge to a global minimum as frequently or more frequently than methods from the third category (CG, NQN and QN without resets).

[8] Strictly speaking, BA and OL with momentum *never* discard all previous derivative information. However, the use of momentum with backpropagation is essentially a heuristic procedure and is not comparable to the second-order classical methods considered here. For this reason, and since it appears to have no significant effect on the percentage of runs successfully converging to a global minimum, the momentum term is ignored in the current discussion.

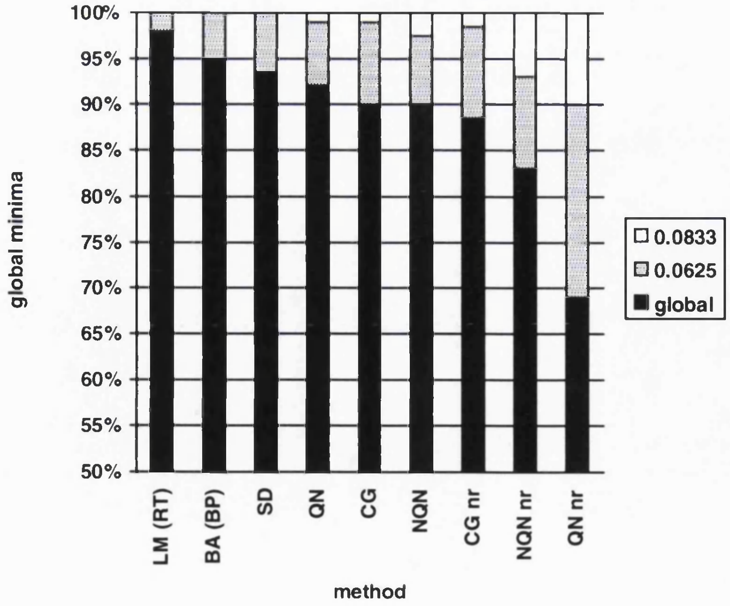
Graph 3 - Training algorithm convergence to global minima (XOR, sigmoid)

Results are for **DBR: 0.5 / 0.25**, except LM (RT, with u_0=0.01 and reduction/growth constants 2.0/4.0) and BA (η=3.0, α=0.9)



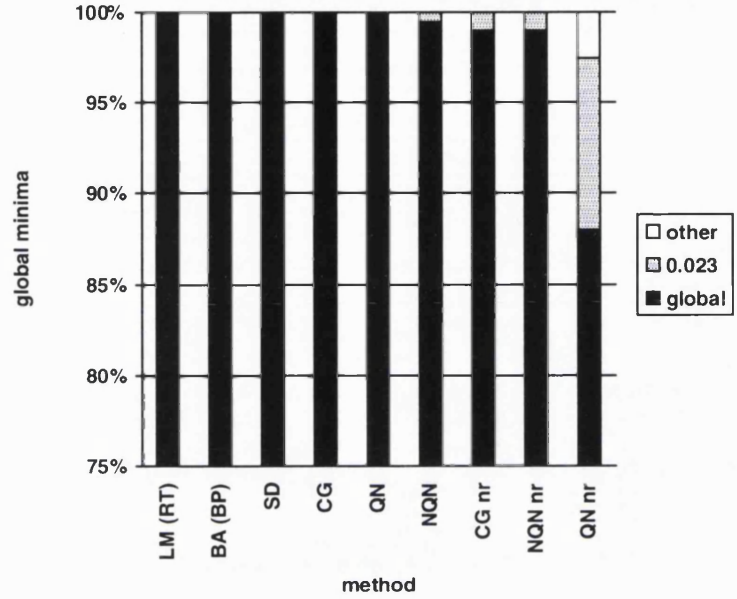
Graph 4 -Training algorithm convergence to global minima (XOR, linear)

Results are for **DBR: 0.5 / 0.25**, except LM (RT, with u_0=0.01 and reduction/growth constants 2.0/4.0) and BA (η=0.5, α=0.0)



Graph 5 -Training algorithm convergence to global minima (sine)

Results are for **DBR: 0.5 / 0.25**, except LM (RT, with u_0=0.01 and reduction/growth constants 2.0/4.0) and BA (η=0.5, α=0.0)



The simple categorisation above does not account for one of the main features of Graphs 3, 4 and 5 - the much poorer global reliability of the non-resetting QN algorithm compared with the CG and NQN non-resetting algorithms. To see how this fits in with the proposed correlation between global convergence and reset frequency, it is necessary to examine the number of resets actually performed by these algorithms - specifically, those resets performed during successful training runs at error levels of *E*>0.0625 and *E*>0.023 for the XOR and sine problems respectively (since, for a reset to have any effect on the global convergence of a classical descent algorithm, it must take place at a higher error level than that of the lowest local minimum). The appropriate data for all classical algorithms using the **DBR: 0.5 / 0.25** line search strategy is presented in Tables 33, 34 and 35; these results largely conform to the identified trend, and show, in particular, that the QN algorithm without resets has both the lowest mean reset-rate and the lowest probability of converging to a global minimum of all the algorithms tested.

Table 33 - mean resets by E=0.0625 (XOR, sigmoid)

Methods: classical methods with line search
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)

	global	mean resets by E=0.0625		
	minima	every n	Eq. 3.9	
method:	(%)	epochs	failure	total
SD:	82.5	----	----	65.69
NQN:	65.5	2.95	0.09	3.05
QN:	62.5	2.24	0.06	2.30
CG:	61.5	1.64	0.43	2.07
CG nr:	58.0	----	0.40	0.40
NQN nr:	57.0	----	0.12	0.12
QN nr:	36.0	----	0.08	0.08

Table 34 - mean resets by E=0.0625 (XOR, linear)

Methods: classical methods with line search
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)

	global	mean resets by E=0.0625		
	minima	every n	Eq. 3.9	
method:	(%)	epochs	failure	total
SD:	93.5	----	----	411.70
QN:	92.0	2.05	0.03	2.08
CG:	90.0	3.84	1.86	5.70
NQN:	90.0	2.54	0.44	2.98
CG nr:	88.5	----	1.51	1.51
NQN nr:	83.0	----	2.86	2.86
QN nr:	69.0	----	0.14	0.14

Table 35 - mean resets by E=0.0625 (sine)

Methods: classical methods with line search
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)

	global	mean resets by E=0.0229		
	minima	every n	Eq. 3.9	
method:	(%)	epochs	failure	total
SD:	100.0	----	----	433.19
CG:	100.0	4.04	0.43	4..47
QN:	100.0	3.17	0.02	3.18
NQN:	99.5	8.11	0.47	8.58
CG nr:	99.0	----	0.45	0.45
NQN nr:	99.0	----	0.69	0.69
QN nr:	88.0	----	0.00	0.00

That Tables 33, 34 and 35 do not show a perfect correlation between an algorithm's global reliability and its mean reset rate is not surprising; the latter takes no account of the dynamics of different multivariate strategies - such as the known automatic-resetting property of the Polak-Ribiere conjugate gradient algorithm (see section 3.5.1) - and may, in any case, be an imperfect measure of the number of resets that 'counted' (i.e. that changed the outcome of a training run). To eliminate the first of these factors, further analysis has been confined to different versions of the same multivariate algorithm, concentrating on a single task and method - XOR with sigmoid output nodes using the quasi-Newton algorithm, with and without SD resets. This combination of task and algorithm is the natural choice; QN is the most robust multivariate algorithm and the subject of the greatest number of tests in this research, and it produced the widest differential between highest and lowest rates of global convergence when applied to the XOR task with sigmoid output nodes. The full set of line search results for QN methods (excluding those with positive definiteness enforced) are listed in Table 36 and plotted in Graph 6.

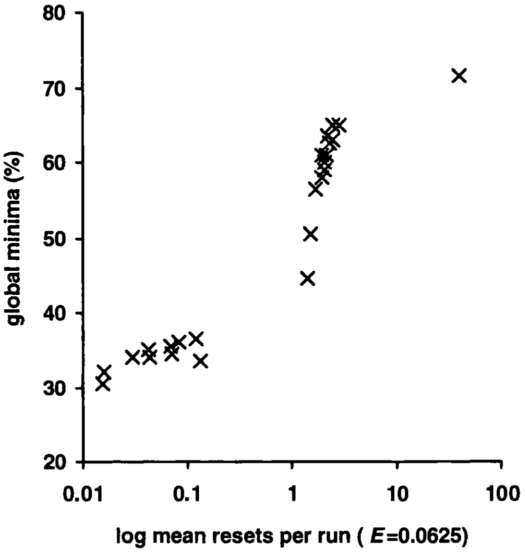
Table 36 - mean resets by *E*=0.0625, QN methods

Task: XOR, sigmoid output nodes
Line search: BR, or DBR, with parameters *q* (Eq. 3.28, Eq. 3.30) and *u* (Eq. 3.29); DBT-BR

	global	**mean resets by *E*=0.0625**		
	minima	**every *n***	**Eq. 3.9**	
method, line search: *q* / *u*	**(%)**	**epochs**	**failure**	**total**
QN, DBT-DBR:	71.5	3.60	36.27	39.87
QN, BR: 0.9 / 0.25	65.0	2.78	0.08	2.86
QN, BR: 0.9 / 0.001	65.0	2.42	0.06	2.48
QN, DBR: 0.9 / 0.25	63.5	2.19	0.06	2.24
QN, DBR: 0.5 / 0.001	63.0	2.44	0.06	2.49
QN, DBR: 0.5 / 0.25	62.5	2.24	0.06	2.30
QN, BR: 0.9 / 0.4	61.0	2.06	0.08	2.14
QN, DBR: 0.9 / 0.001	61.0	1.91	0.04	1.95
QN, DBR: 0.9 / 0.4	60.0	2.03	0.03	2.06
QN, BR: 0.5 / 0.001	59.0	2.01	0.06	2.07
QN, BR: 0.5 / 0.25	58.0	1.96	0.03	1.99
QN, DBR: 0.1 / 0.001	56.5	1.65	0.06	1.71
QN, BR: 0.1 / 0.001	50.5	1.47	0.07	1.53
QN nr, DBT-DBR:	44.5	----	1.42	1.42
QN nr, DBR: 0.9 / 0.25	36.5	----	0.12	0.12
QN nr, DBR: 0.5 / 0.25	36.0	----	0.08	0.08
QN nr, DBR: 0.9 / 0.001	35.5	----	0.07	0.07
QN nr, DBR: 0.9 / 0.4	35.5	----	0.07	0.07
QN nr, DBR: 0.5 / 0.001	35.0	----	0.04	0.04

	global	mean resets by *E*=0.0625		
	minima	every *n*	Eq. 3.9	
method, line search: *q* / *u*	(%)	epochs	failure	total
QN nr, BR: 0.9 / 0.25	35.0	----	0.04	0.04
QN nr, BR: 0.9 / 0.4	34.5	----	0.07	0.07
QN nr, BR: 0.9 / 0.001	34.0	----	0.04	0.04
QN nr, BR: 0.5 / 0.001	34.0	----	0.03	0.03
QN nr, BR: 0.1 / 0.001	33.5	----	0.13	0.13
QN nr, BR: 0.5 / 0.25	32.0	----	0.02	0.02
QN nr, DBR: 0.1 / 0.001	30.5	----	0.02	0.02

Graph 6 - Global convergence frequency vs. log mean resets per successful run, QN line-search methods (XOR, sigmoid)



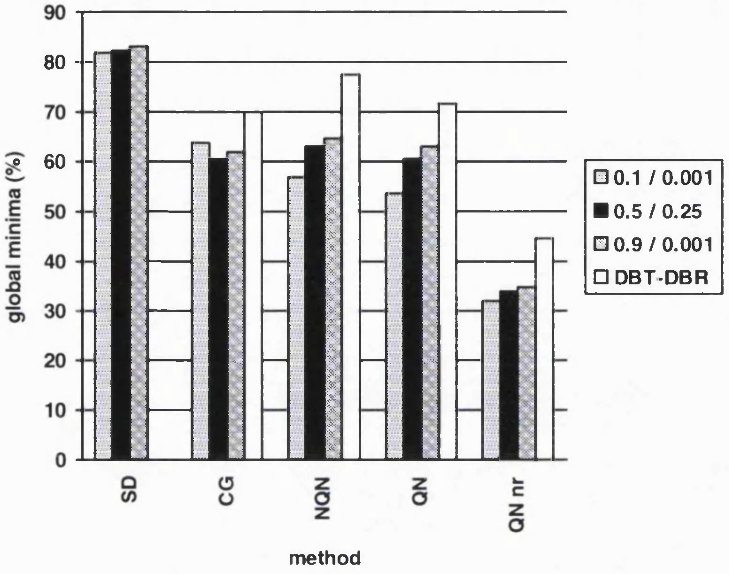
The preceding results provide convincing evidence of a strong correlation between global convergence and reset frequency for a single algorithm and task. Similar, but less detailed, analyses were performed on the sigmoid and linear XOR results for each classical algorithm, excluding LM. (With the sine task, too few training runs converged to non-global minima for meaningful comparisons to be made.) The CG, NQN and QN algorithms, with and without resets, showed the same broad correlation - over a narrower convergence range - as the QN algorithm with XOR and sigmoid output nodes; however, there was negligible change in the global reliability of SD, irrespective of the number of resets performed. Taken together, these results suggest the following rule for second-order nonlinear optimisation algorithms with line searches when applied to problems with local minima: increasing the reset-rate of an algorithm is likely to produce a steady improvement in its global reliability, so long as the reset-rate remains 'low' or 'moderate'; the benefit of increasing the reset-rate diminishes as the frequency of global convergence approaches that of the steepest descent algorithm.

An important corollary is that a given second-order algorithm (with or without SD resets every *n* epochs) is less likely to get trapped in local minima when used with an inaccurate line-search strategy, and more likely to get trapped with an accurate line search. Inaccurate line searches typically require more epochs to reach a given error tolerance than accurate ones, with correspondingly more resets performed every *n* epochs; and inaccurate line searches are, intuitively, more likely to generate an $\mathbf{s}_k$ that is not a descent direction because the local model, generated iteratively by the multivariate algorithm, is based on less-accurate information.

On balance, the hybrid backtracking/Brent algorithm, DBT-DBR, is the least-accurate line-search algorithm tested for this research - excluding the non-hybrid backtracking algorithms, BT and DBT, which failed to produce a full set of training results (see section 4.3.1). The DBT-DBR algorithm typically spends the majority of training epochs in derivative backtracking (DBT) mode, with parameters q=0.9 (Eq. 3.28) and u=0.001 (Eq. 3.29). Brent's method, with the same settings for q and u, has a tendency to achieve greater accuracy than the backtracking strategy, owing to the enforced bracketing of the minimum (see section 3.2.1). Graphs 7 and 8 plot, for the sigmoid XOR and linear XOR training tasks respectively, the global convergence frequency of each classical algorithm with line searches of varying accuracy - Brent's method with settings q=0.9 / u=0.001, q=0.5 / u=0.25 and q=0.1 / u=0.001, and the novel DBT-DBR strategy. These graphs support the thesis that the global reliability of a given classical multivariate algorithm tends to increase as the accuracy of the line search employed by that algorithm decreases; the low-accuracy DBT-DBR algorithm has the highest rate of global convergence in 6 out of 8 cases and the second-highest in the remaining 2 cases, whereas the high-accuracy Brent 0.1 / 0.001 line-search has the lowest (or equal lowest) rate in 9 out of 10 cases.

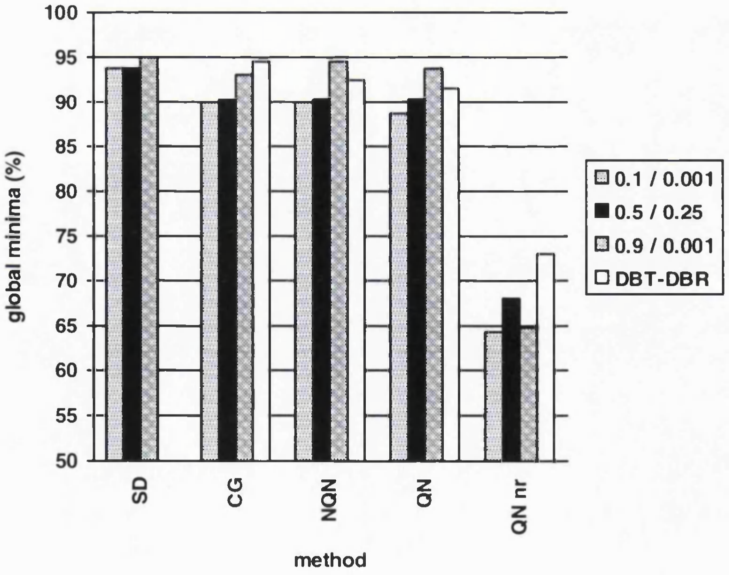
Graph 7 - Line searches and global convergence (XOR, sigmoid)

Note: the Brent global percentage - for given settings of parameters q (Eq. 3.28) and u (Eq. 3.29) - is the average of the BR and DBR percentages with those settings.



Graph 8 - Line searches and global convergence (XOR, linear)

Note: the Brent global percentage - for given settings of parameters q (Eq. 3.28) and u (Eq. 3.29) - is the average of the BR and DBR percentages with those settings.

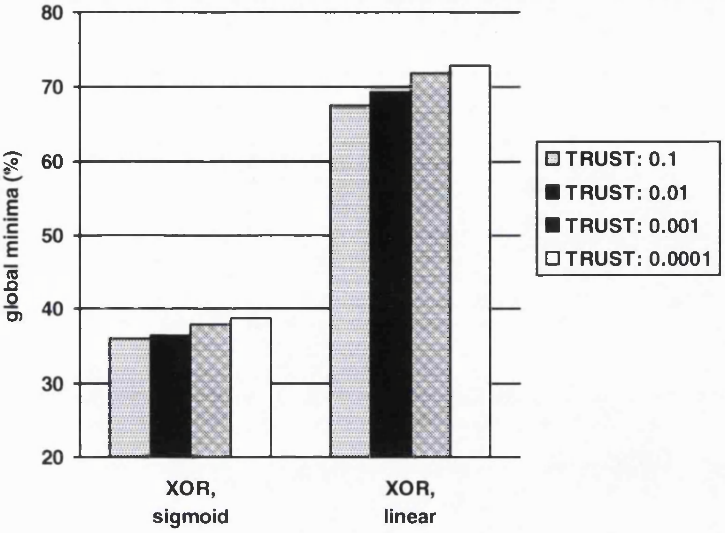


So far, nothing has been said about model-trust region strategies or the strategy for enforcing the positive-definiteness of the Hessian matrix. When a second-order classical method without resets is implemented using one of these strategies, derivative information from previous epochs is *never* wholly discarded. Rather than reset the model completely to the current steepest descent direction, these strategies perturb the model Hessian towards the linear model of steepest descent, to a degree determined by the size of scalar u in Eq. 3.15; if u is sufficiently large, such a perturbation is roughly equivalent to the traditional SD reset (see section 3.1.3). Both model-trust region strategies and the strategy for enforcing the positive-definiteness of the Hessian can be said, therefore, to 'partially reset' the multivariate algorithm.

Judging by the tabulated results of section 4.3, the adoption of the enforced positive-definiteness strategy generally has little impact on either the global convergence characteristics or the training speed of the non-resetting QN algorithm. An examination of the mean number of partial resets performed with line searches of different accuracy suggests that the correlation between global reliability and reset frequency extends to partial-reset frequency, but that the number of partial resets required to produce an equivalent effect is higher (by as much as a factor of 12 in certain cases). With model-trust region strategies, both partial-reset frequency and global reliability increased as the size of u_0 decreased (Graph 9). (For LM model-trust region algorithms, which 'reset' at every training epoch, these considerations do not apply. In fact, the global convergence frequency of the LM method given in the tables of section 4.3 show the opposite trend to those for QN methods, i.e. global reliability has a tendency to increase as the number of partial-resets decreases.)

Graph 9 - Model-trust region strategies and global convergence, QN without resets

Note: the model-trust region global percentage, for a given u_0 (Eq. 3.15), is the average of four percentages - those for RT, and ST, with reduction/growth constants of 2.0/4.0 and 4.0/10.0.



Having established the link between the global reliability of an algorithm and its reset (or partial-reset) rate, we are left with an obvious question: 'How does resetting an algorithm improve its chances of reaching a global minimum?' For an explanation, let us consider the situation where a second-order algorithm has constructed, by iteration k, a model of E which, if minimised, will trap the algorithm in a local minimum. (Such a situation may be a common occurrence in the early stages of training, owing to the build-up of hidden-node redundancy - see sections 2.1.2 and 2.2.1.) Now let us suppose that the derivative information generated at iteration k is *not* dominated by the curvature towards this (or any other) local minimum. If the algorithm is reset at iteration k, a new model will be constructed which has a much better chance of directing the algorithm towards a global minimum. However, if no reset is performed, the derivative information generated at iteration k will be incorporated in the existing model; since the old (pre-iteration k) model will tend to dominate the new information, the search direction generated at iteration k is likely to head towards the basin of attraction of the local minimum.

4.4.2 Training speed and accuracy

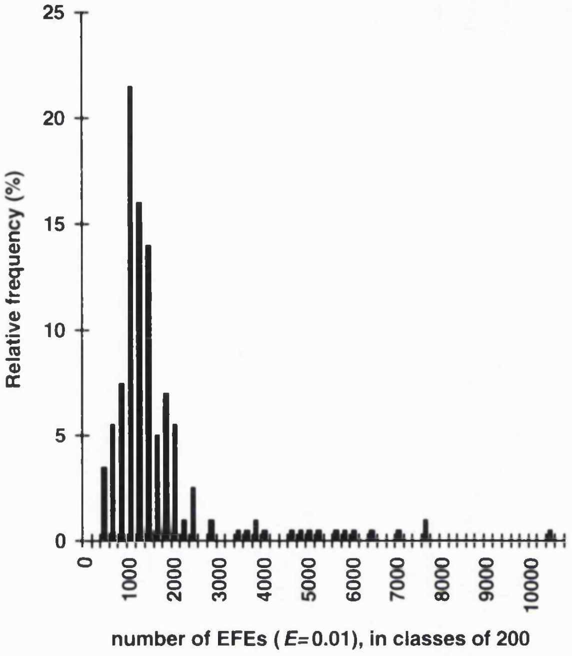
The tables in section 4.3 provide two estimates of the 'typical' speed of each training algorithm - the mean and the median EFEs per run. (In either case, training runs that fail to reach the chosen tolerance - E=0.01 for the tabulated results - are ignored.) Before embarking on an analysis of the comparative speeds of different algorithms, it is worth examining which of these two measures is superior. Graph 10 shows the distribution of the training speed data for a single combination of algorithm, task and settings. A prominent feature of this distribution - and one that is common to all the results in section 4.3 - is the broad tail, caused by a small number of outlier points (corresponding to training runs that required a disproportionately large number of EFEs to reach a given tolerance). Since the mean is sensitive to the breadth of the tail whereas the median is not, the latter is the more robust estimate of the typical speed of an MLP training algorithm[9]. For this reason, the median EFE rate is the adopted training speed metric for this analysis.

The following analysis of MLP training speed is in three stages. The first stage gives an overview of the performance of traditional and classical methods, and addresses the fundamental question, 'Do second-order classical methods speed up training sufficiently to justify the additional programming, storage and (at each training epoch) computational costs involved?' The second stage compares the performance of different second-order methods using, where possible, the same line-search strategy and settings. The final stage of the analysis examines, in detail, the performance of different versions of the same second-order algorithm. At each stage, the actual performance of the algorithms under investigation, when applied to the three tasks used in this research, will be assessed in terms of their anticipated performance based on theoretical and practical experience.

[9] There are several instances where the mean EFE rate given in section 4.3 grossly over-estimates the typical performance of an algorithm. Prime examples are to be found in Table 3 for settings DBR: 0.9 / 0.001 and DBR: 0.9 / 0.25.

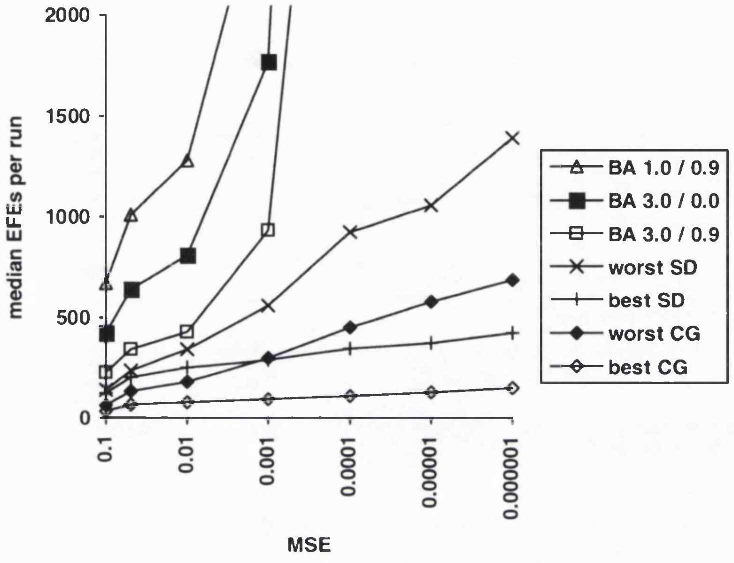
Graph 10 - Relative frequency histogram, SD DBR: 0.5 / 0.25 (sine)

Note: the corresponding mean and median are 1,680.3 and 1,311.0 respectively

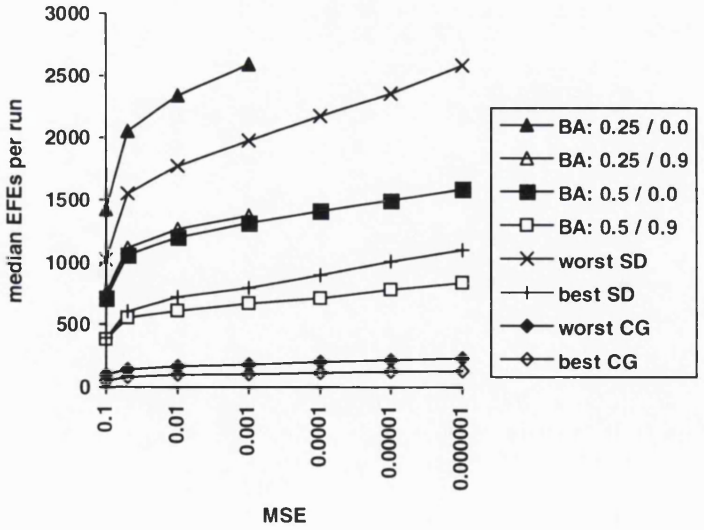


Overview of traditional and classical methods. Graphs 11, 12 and 13 plot, for each training task, the performance of traditional methods (represented by on-line BP with several different combinations of training rate and momentum), first-order classical methods (i.e. SD), and second-order classical methods (represented by the Polak-Ribiere CG algorithm with SD resets). In each of these graphs, the best curve plotted for on-line BP represents the 'near-optimal' performance of that algorithm - i.e. a modest increase in the training rate was sufficient to induce oscillatory behaviour. (In the majority of cases, batch BP was slightly slower than on-line BP with the same training rate and momentum.) The SD curves show the fastest and slowest performances of the SD algorithm using either the BR or the DBR line-search strategy; the CG curves show the fastest and slowest performances of CG using the BR, DBR or DBT-DBR line-search strategies.

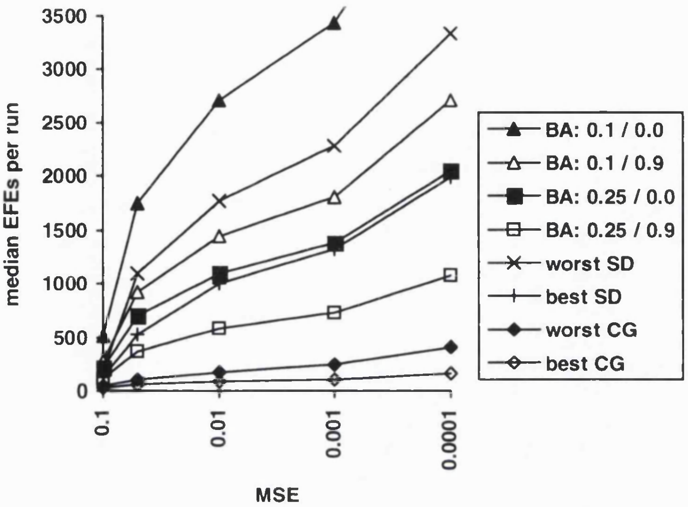
Graph 11 - Training speed, first- and second-order methods (XOR, sigmoid)



Graph 12 - Training speed, first- and second-order methods (XOR, linear)



Graph 13 - Training speed, first- and second-order methods (sine)

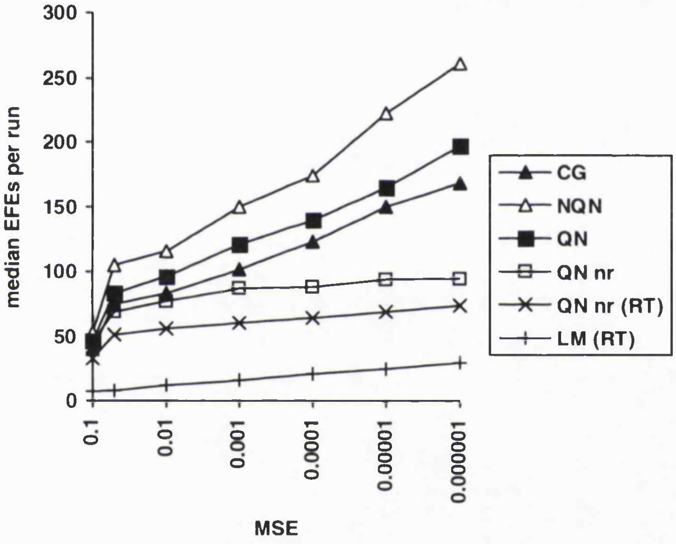


These results illustrate two key reasons why second-order classical methods are likely to produce faster solutions to many MLP training problems than traditional training methods. Firstly, traditional fixed step-length BP is highly sensitive to the choice of training rate, and the optimal (or near-optimal) training rate is task-specific; classical methods, using variable step-length strategies, are far less sensitive to the choice of initial parameter-settings, and (as the final stage of this analysis will show) certain parameter-settings tend to be consistently better than others. Secondly, the second-order CG algorithm is consistently faster than any of the first-order methods, attributable to the superiority of a quadratic model over a linear model for generating 'effective' search directions (see sections 3.1.1 and 3.1.2). With BP, the addition of the heuristic second-order momentum term (described in section 2.3.3) was highly effective in the form specified by Eq. 2.10, but of negligible benefit when used in the form given by Eq. 2.11.

Second-order methods. Graph 14, 15 and 16 plot, for each training task, the performance of the following second-order methods: Polak-Ribiere CG, NQN and QN with SD resets, QN without SD resets, and the LM algorithm. The curves for CG, NQN and QN with resets represent the performance of these algorithms with the Brent line-search strategy **DBR: 0.5 / 0.25**, whereas the LM curve is for the model-trust region strategy **RT: 0.01**; both these versions of the non-resetting QN algorithm are represented.

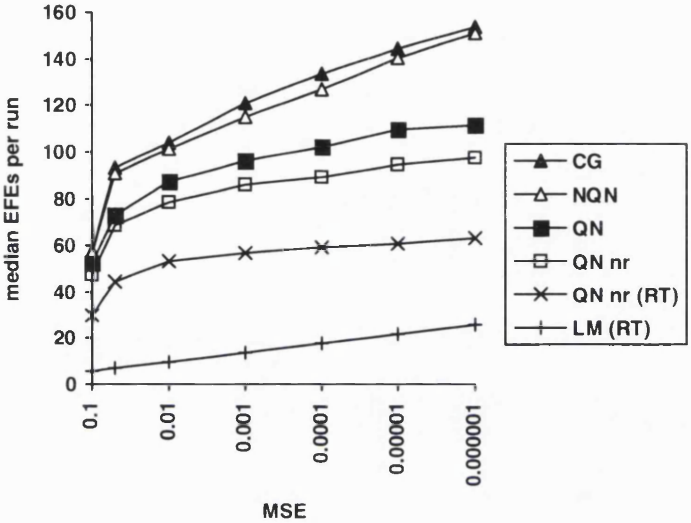
Graph 14 - Training speed, second-order methods (XOR, sigmoid)

Results are for **DBR: 0.5 / 0.25** and **RT: 0.01**



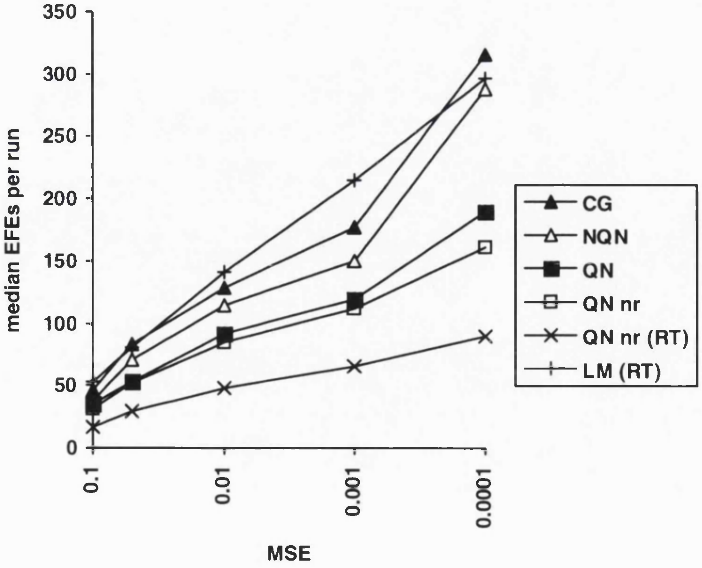
Graph 15 - Training speed, second-order methods (XOR, linear)

Results are for **DBR: 0.5 / 0.25** and **RT: 0.01**



Graph 16 - Training speed, second-order methods (sine)

Results are for **DBR: 0.5 / 0.25** and **RT: 0.01**



These graphs show that the comparative speeds of the second-order classical algorithms tested for this research are very much in line with the predictions given in section 3.7, a point made clear by the following observations:

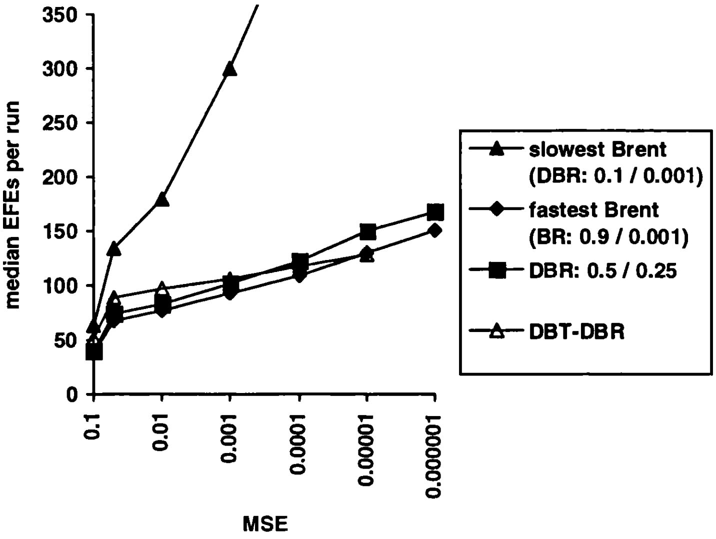
- the LM algorithm is by far the fastest algorithm with the non-residual XOR tasks, but one of the slowest second-order algorithms with the sine task, which has residuals at the solution;
- the QN algorithm is significantly faster than the CG and NQN algorithms - both of which store less derivative information than the QN algorithm - with two of the training tasks;
- the QN algorithm is consistently faster when implemented without SD resets, i.e. when (potentially) useful derivative information is not discarded every *n* epochs;
- there is little to choose between the CG and NQN algorithms, although the former is significantly faster on the XOR problem with sigmoid output nodes.

Another feature of these graphs is that the shape of the LM curves is different from the rest, being virtually straight (with the MSE plotted on a log scale) for all three tasks. This may be related to the fundamental distinction between the LM algorithm and all the other classical algorithms used here - i.e. it is a nonlinear least-squares algorithm, rather than an unconstrained nonlinear optimisation algorithm.

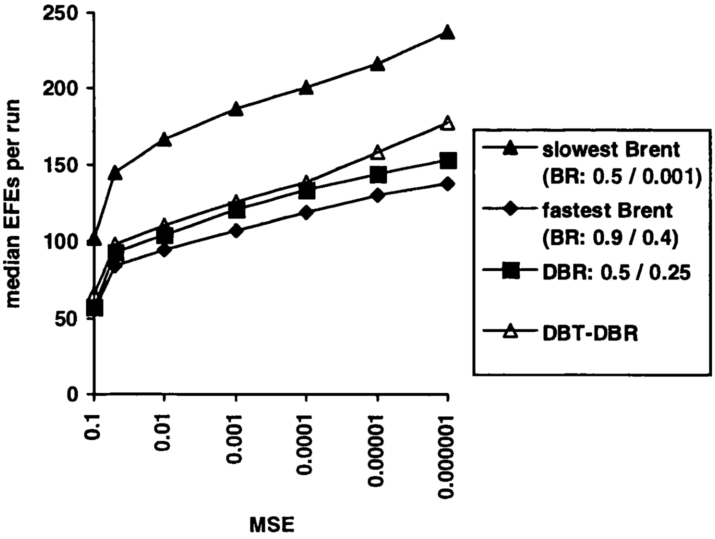
Line-search and model-trust region strategies. The analysis now turns to the performance of different versions of the same nonlinear optimisation method. Graphs 17-28 plot, for each second-order method in turn: the fastest and slowest performances of the Brent line-search strategy (with the settings used to generate the results of section 4.3); the performance of the novel hybrid Brent/backtracking line-search algorithm (DBT-DBR); and - for the non-resetting QN algorithm only - the fastest and slowest performances of either the RT or ST model-trust region strategy (MTRS) (with the settings of section 4.3).

Graph 17 - Training speed, PR CG with SD resets (XOR, sigmoid)

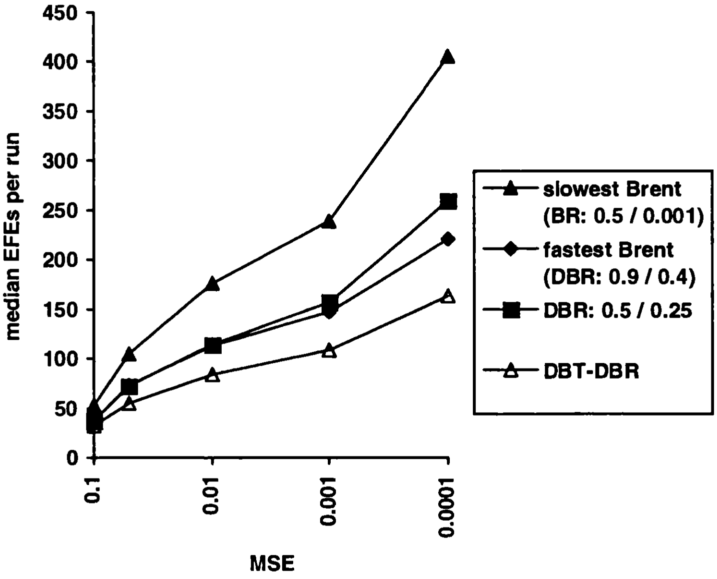
Note: no point is plotted for **DBT-DBR** at MSE 1.0^{-06} since a significant percentage of runs failed to reach this level within 50,000 EFEs.



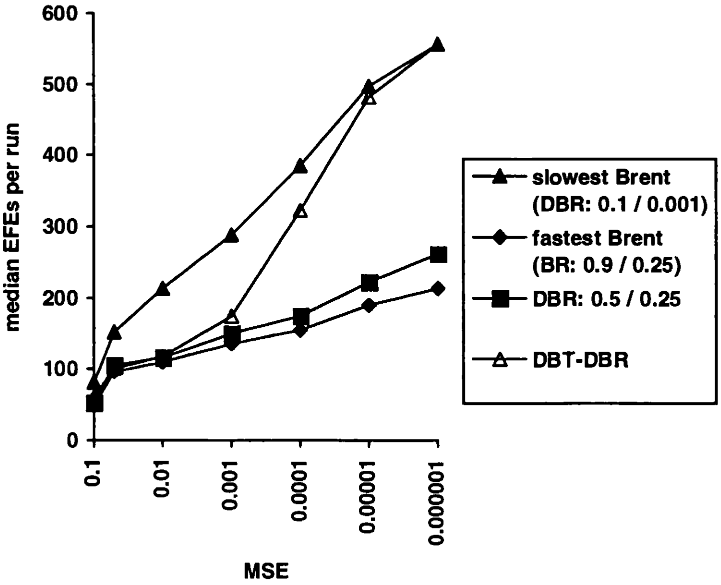
Graph 18 - Training speed, PR CG with SD resets (XOR, linear)



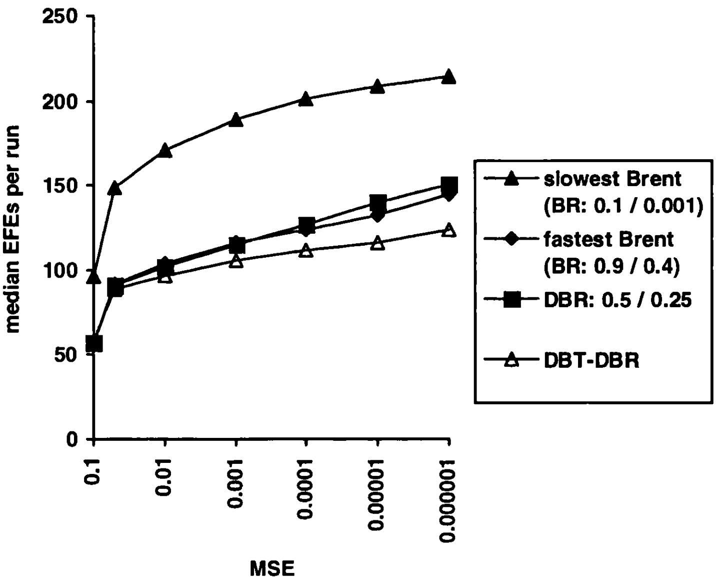
Graph 19 - Training speed, PR CG with SD resets (sine)



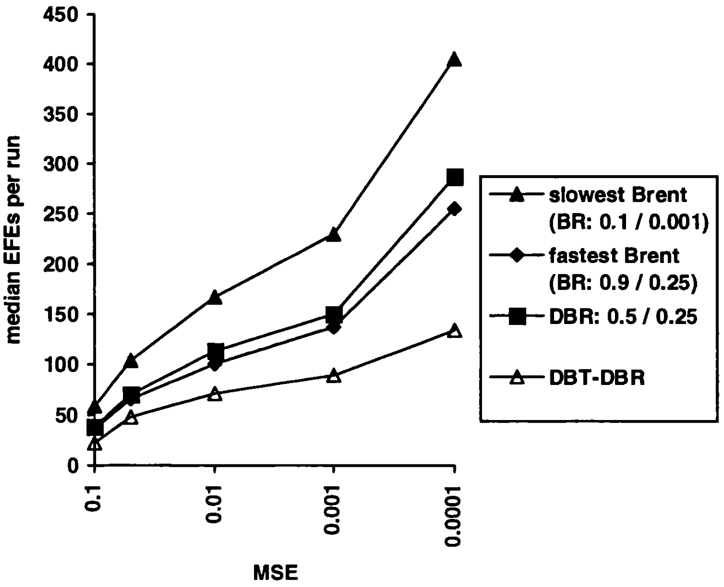
Graph 20 - Training speed, NQN with SD resets (XOR, sigmoid)



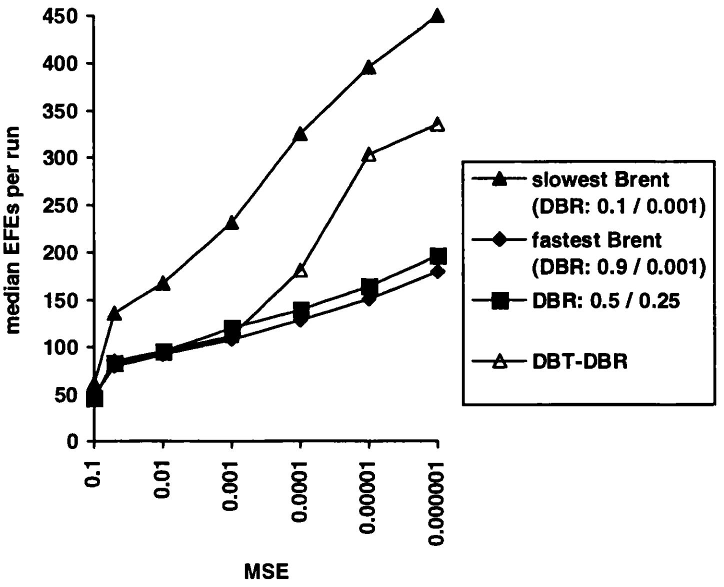
Graph 21 - Training speed, NQN with SD resets (XOR, linear)



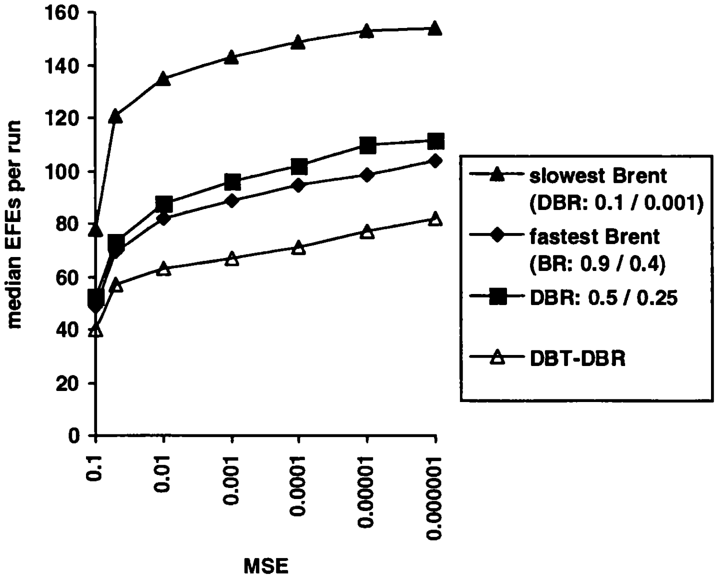
Graph 22 - Training speed, NQN with SD resets (sine)



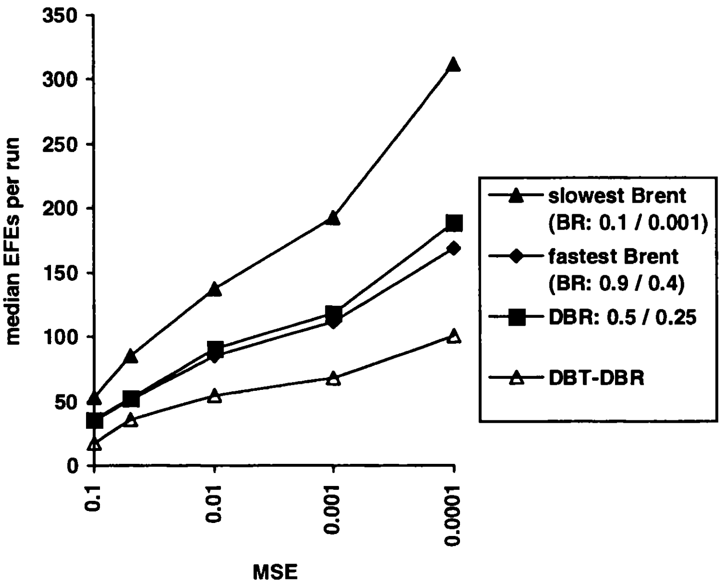
Graph 23 - Training speed, QN with SD resets (XOR, sigmoid)



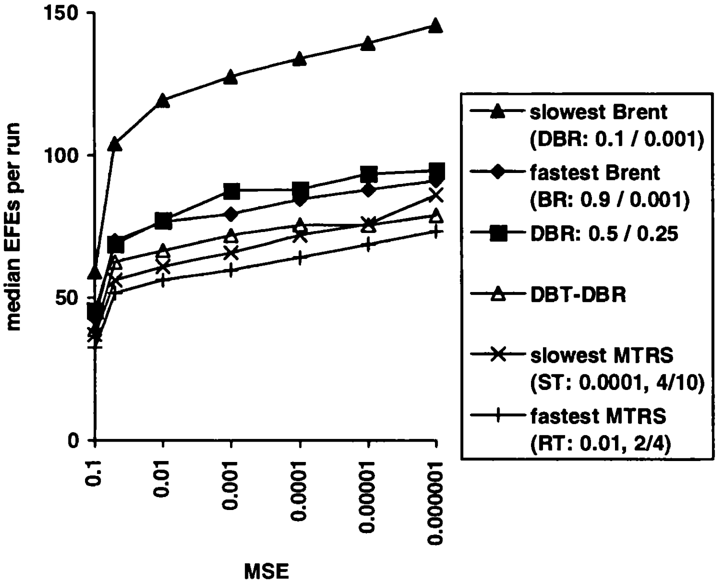
Graph 24 - Training speed, QN with SD resets (XOR, linear)



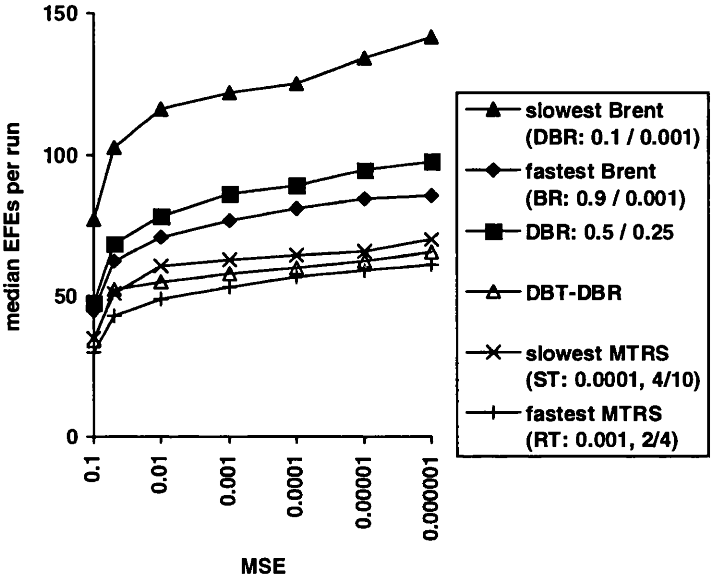
Graph 25 - Training speed, QN with SD resets (sine)



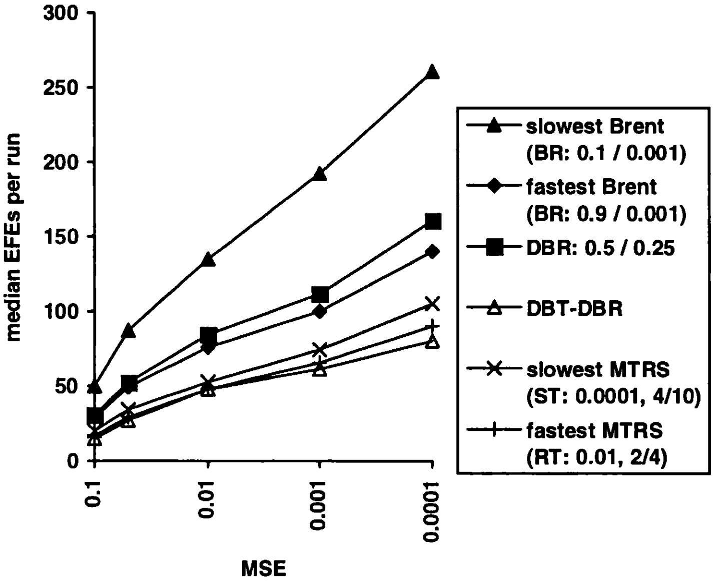
Graph 26 - Training speed, QN without resets (XOR, sigmoid)



Graph 27 - Training speed, QN without resets (XOR, linear)



Graph 28 - Training speed, QN without resets (sine)



The preceding graphs largely support the modern preference for inaccurate line searches, as the following observations make clear:

- with Brent's method, a high accuracy of q=0.1 (Eq. 3.28) accounted for 10 of the 12 slowest performances, whereas all 12 of the fastest Brent performances were produced by a low accuracy of q=0.9;
- the hybrid Brent/backtracking algorithm DBT-DBR - which, as noted in section 4.4.1, is generally less accurate than Brent's method with q=0.9 - was faster than the fastest Brent performance in 8 out of 12 cases, and approached the speed of the fastest Brent performance (at error tolerances of E>0.001) in the remaining 4 cases.

However, this endorsement of inaccurate line searches is not entirely unqualified. In addition to the problems encountered with the highly-inaccurate non-hybrid backtracking algorithms BT and DBT, two further points emerged in the preceding analysis.

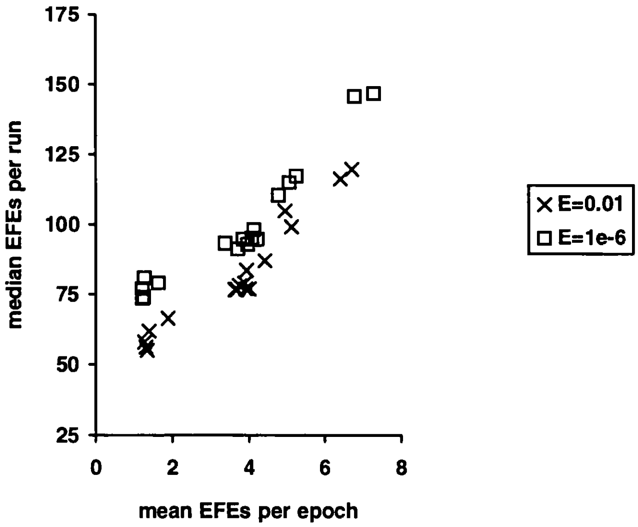
- With Brent's method, low accuracy did not guarantee fast training. All of the low accuracy settings - i.e. q=0.9 with or without derivatives and with u (Eq. 3.29) of 0.001, 0.25 or 0.4 - performed well with many, but not all, combinations of multivariate algorithm and task. On balance, the moderate-accuracy **DBR: 0.5 / 0.25** algorithm - used widely in this research as *the* representative Brent algorithm - is probably as good a choice as any, as it approached the speed of the fastest Brent algorithm in *every* case.
- With all but the non-resetting QN algorithm, the hybrid DBT-DBR algorithm proved comparatively slow at relatively low error tolerances for the XOR task with sigmoid output nodes.

These qualifications are not surprising; it is well known that the less-robust multivariate algorithms (e.g. CG and NQN) do not always perform well with inaccurate line-searches, and backtracking algorithms are designed to work with multivariate algorithms, such as the non-resetting QN algorithm with positive definiteness enforced, for which the Newton direction and Newton step are defined at each iteration. Given the lack of theoretical justification for using backtracking strategies with other methods, it is perhaps surprising that they are as effective as the results in this research suggest.

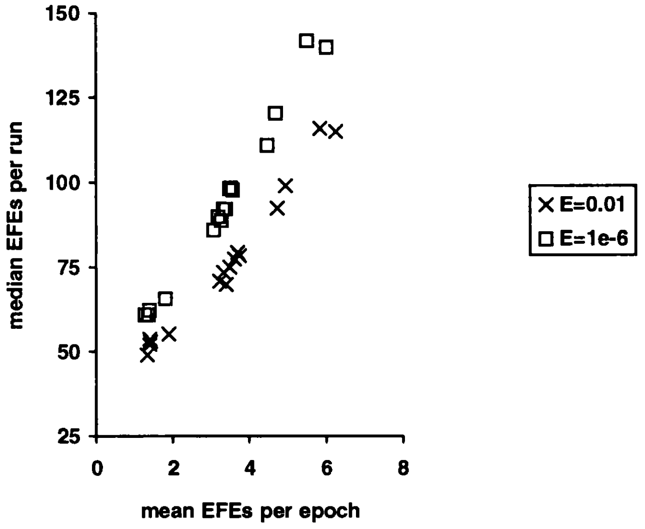
The results for the non-resetting QN algorithm indicate that model-trust region strategies are less sensitive than Brent's method to the setting of parameters. In general, the RT strategy was slightly faster than the ST strategy (with the same settings) and both RT and ST strategies slightly faster with shrink/growth constants of 2.0/4.0, rather than 4.0/10.0. These results also suggest that there is little to chose between model-trust region methods and the hybrid DBT-DBR line-search algorithm.

Finally, it is useful to compare the results for different versions of the non-resetting QN - the most robust second-order algorithm used in this research, and the subject of the greatest number of tests - in terms of the number of EFEs performed at each epoch. The mean number EFEs per epoch gives a measure of the accuracy of an algorithm that is independent of the choice of line-search/model-trust region strategy and settings. Graphs 29, 30 and 31 show that, for each training task, there is strong correlation between the mean EFEs per epoch and the median EFEs per run - the fewer EFEs per epoch, the faster the algorithm.

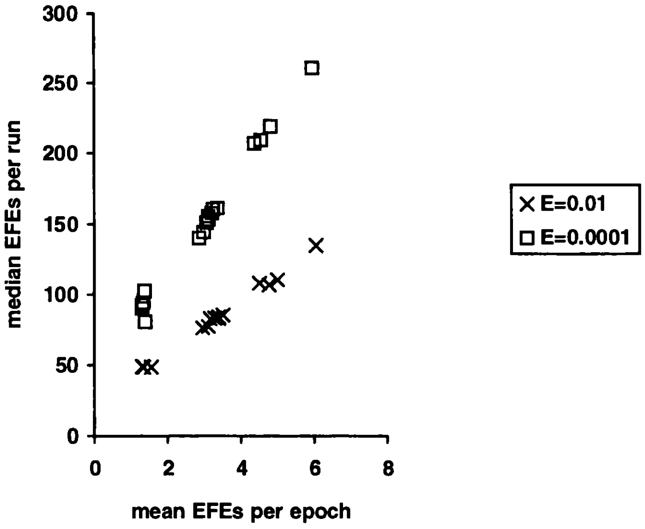
Graph 29 - EFEs per run vs. EFEs per epoch, QN methods without resets (XOR, sigmoid)



Graph 30 - EFEs per run vs. EFEs per epoch, QN methods without resets (XOR, linear)



Graph 31 - EFEs per run vs. EFEs per epoch, QN methods without resets (sine)



4.4.3 Conclusion

The results of section 4.3 and analysis of section 4.4 suggest the following practical guidelines for the selection of a fast MLP training algorithm with a reasonably high rate of global convergence, given no prior knowledge about the properties of the MLP error surface (i.e. the existence of local minima or the size of residuals at the solution):

- To maximise training speed, use second-order methods. The QN algorithm is consistently fast; if the O(n^2) storage costs of the QN algorithm is prohibitive, both the CG and NQN algorithms are viable alternatives. The LM method may be highly effective, but cannot be recommended in general because of its sensitivity to the presence of residuals at the solution.
- To maximise the chance of finding a global solution, use an algorithm that resets the model at every iteration, such as SD or LM. Of the second-order algorithms recommended on grounds of training speed, the CG and NQN algorithms are more likely to avoid local minima than the QN algorithm. With all three of these algorithms, resetting the model periodically may improve the chances of avoiding local minima.
- Provided the chosen multivariate algorithm is sufficiently robust, the adoption of an inaccurate line-search strategy or a model-trust region strategy can improve both training speed and the global convergence rate of the algorithm.

These results also suggest two important areas for future research:

- the development of 'variable' training strategies, i.e. ones which vary the accuracy and reset frequency of an algorithm at different stages of training;
- the development of nonlinear least-squares algorithms for MLP training that have O(n) storage requirements, and that are not sensitive to the size of the residuals at the solution.

5. GLOBAL OPTIMISATION - A NEW ERA?

The term 'global optimisation' covers a diverse range of strategies designed to improve the chances of converging to a global, rather than a local, minimum. Global optimisation strategies fall into two broad classes - stochastic (or probabilistic) methods, and deterministic (or classical) methods; typically, the former allow uphill motion with respect to the chosen error function, whereas the latter do not. The main purpose of this chapter is to assess the potential for combining global optimisation strategies with the fast second-order training methods of chapter 4 to produce fast global training methods - the fundamental goal of this research.

This chapter is in two parts. Section 5.1 considers global optimisation in general, and assesses the contrasting suitability of different global and second-order optimisation strategies for the development of fast global methods. The analysis conducted for this research suggests that the comparatively under-researched field of deterministic global optimisation is well-suited to second-order training. Section 5.2 presents a novel global training strategy - Expanded Range Approximation (ERA) - developed jointly with Dr D Gorse and Prof. J Taylor [Gorse, Shepherd & Taylor, 1993a, 1993b, 1994a, 1994b, 1995]. ERA is a deterministic strategy which requires modification to the MLP training set only, so that implementation with any of the training methods considered in chapter 4 is trivial.

5.1 Introduction to Global Optimisation

5.1.1 Stochastic methods

The term 'stochastic methods' covers a wide variety of techniques - of contrasting algorithmic complexity, performance and theoretical justification - for avoiding local minima (and other error-surface obstacles) through the addition of 'noise'. In contrast to the classical methods of chapter 3, all these techniques allow uphill motion with respect to the chosen error function. Stochastic methods can be divided into two broad categories, depending on whether noise is added to the system from the outset of training, or only when the MLP has converged to a local minimum. In the light of research by Gorse (using the Polak-Ribiere conjugate gradient algorithm), which suggests that random perturbations of the search direction and 'various kinds of stochastic adjustment to the current set of weights' are largely ineffective at enabling MLPs to escape from local minima [Gorse, 1992], this section concentrates on stochastic methods that add noise throughout the training process.

The fundamental problem for all stochastic methods is how to determine the appropriate level of 'noise' for an arbitrary minimisation task: too little, and the algorithm may become trapped in a local minimum; too much, and it may fail to converge to a global minimum within a reasonable number of iterations. What is needed is some scheme for adapting the level of noise; for an arbitrary task with no special features, the intuitively sensible approach is to reduce the amount of noise as the minimisation proceeds. Two widely-used strategies are to gradually reduce the level of noise as the number of iterations increases - known as *simulated annealing* (SA) [Kirkpatrick, Gelatt & Vecchi, 1983] - or as the error level falls - known as *time-invariant noise algorithms* (TINA) [Burton & Mpitsos, 1992]. (A similar effect can be achieved in the context of on-line MLP training by adopting a suitable scheme for adapting the size and composition of the subset of patterns presented to the network at each iteration.)

A number of training strategies have been developed that combine stochastic and first-order information, the most widely-used being the 'traditional' on-line BP algorithm described in section 2.3.2. In addition, algorithms that combine simulated annealing with BP [Wasserman, 1989] [Amato et al., 1991] and TINA with BP [Burton & Mpitsos, 1992] have been developed, as well as numerous strategies for combining genetic algorithms (GAs) [Holland, 1975] with BP (see, for example, [Belew, McInerney & Schraudolph, 1991]). (In fact, the algorithms of Amato et al., Burton and Mpitsos, and Belew et al. can be regarded as 'doubly stochastic' in that SA, TINA and GAs are combined with on-line BP.) All these strategies perform well on specific problems, but have potential weaknesses. For example, the SA and TINA methods are sensitive to the heuristic choice of constants[1], whereas with 'traditional' on-line BP the level of noise

[1] Burton and Mpitsos used noise constants of 0.3 for SA and in the range 2-5 for TINA, but note that 'the noise constants usually had to be decreased as the teacher task [i.e. training task] became more difficult, and when increasing the number of hidden units beyond 32' [Burton & Mpitsos, 1992, 629].

added to the system is an arbitrary function of the training set and network architecture, and may not produce a higher rate of global convergence than (non-stochastic) batch BP - as is the case with the results presented in chapter 4.

Rather than provide a detailed evaluation of the strengths and weaknesses of strategies that combine stochastic information with fixed step-length BP, our primary concern here is to assess the potential for combining stochastic information with variable step-length second-order methods to form fast global training algorithm. In this context, several fundamental difficulties present themselves.

- Second-order methods for general optimisation that rely on stored information from previous training epochs (i.e. the CG, QN and NQN methods) are likely to be severely disrupted by the addition of noise in a way that methods which compute a completely new model at each iteration (BP, SD, LM and Newton's method) are not.
- In contrast to classical optimisation, the optimal training rate (step length) at each iteration for stochastic methods is poorly understood. Results from stochastic approximation theory provide a theoretical basis for adjusting the on-line training rate η as a function of time (for a brief summary, see [Møller, 1993e, 23]). An example is the *search then converge* (STC) training rate schedule of Darken, Chang and Moody [1992], given by

Eq. 5.1

$$\eta_k = \eta_0 \frac{1 + \frac{c}{\eta_0}\frac{k}{\tau}}{1 + \frac{c}{\eta_0}\frac{k}{\tau} + \tau\frac{k^2}{\tau^2}} ,$$

where η_k is the training rate at iteration k, parameter c is set greater than a threshold of $1/2\lambda_{\min}$ (where $\lambda_{\min}$ is the smallest eigenvalue of the Hessian of E), and parameter τ relates to the number of anticipated training epochs. The significance of the STC schedule is that, when implemented with a suitable scheme for estimating c (using, for example, the Power method [Møller, 1993e, 24]), an optimal rate of asymptotic ('large time') convergence is guaranteed for on-line BP. In practice, however, the convergence rate associated with the STC schedule is highly dependent on the choice of parameters η_0 and τ. Whereas it is possible to estimate c automatically, guidance for setting parameters η_0 and τ is essentially heuristic, or requires prior knowledge about the problem. Møller rightly concludes that 'methods for the initial setting of these parameters are needed in order for the [STC] method to have any real practical use' [Møller, 1993e, 24].

- There is a fundamental mis-match between the theoretically-justified schedules for reducing noise with SA methods and the convergence rates of second-order methods. With SA algorithms, the amount of noise generated at a given iteration depends on the choice of generation function and an artificial temperature parameter *T*. With noise generated according to a Gaussian distribution - the basis of *Boltzmann annealing* - *T* should be reduced no faster than

Eq. 5.2 $T_k = T_0/\ln(k)$

to guarantee that the system will statistically find a global minimum. With noise generated according to a Cauchy distribution - known as *fast annealing* [Szu & Hartley, 1987] - *T* can be reduced at the faster rate of

Eq. 5.3 $T_k = T_0/k$.

However, both of the schedules given by Eq. 5.2 and Eq. 5.3 are much slower than the anticipated rates of convergence associated with second-order methods (see section 3.1.2).

These factors do not preclude the development of effective hybrid stochastic/second-order methods, but suggest that the direct combination of stochastic and second-order information is highly problematic - unless the stochastic component is comparatively small. Judging by available research in this field, practical implementations of hybrid stochastic/second-order methods fall into two categories - those which completely isolate the second-order method from stochastic 'noise', and those which severely restrict the level of noise added to the system. Examples of the former are:

- algorithms that have two distinct training phases - a stochastic phase and a classical second-order phase, e.g. the hybrid on-line BP/conjugate-gradient algorithm proposed in [Shepherd, 1992], which switches from on-line BP to CG after a user-defined number of training epochs have elapsed;

- hybrid GA/second-order algorithms used for 'sampling and search', by which the GA is used to chose initial weights for a population of MLPs, each of which is trained using a classical second-order method, e.g. the hybrid GA/CG algorithm in [Belew, McInerney & Schraudolph, 1991][2].

An example of a hybrid stochastic/second-order method that combines stochastic and second-order information directly but severely restricts the level of stochastic noise throughout the training process is Møller's on-line CG algorithm [Møller, 1993b]. At each training epoch, Møller's algorithm uses a validation scheme to ensure, with a high probability, that the normalised error for the subset of patterns (chosen using standard sampling techniques or by an active data-selection scheme) is an approximation to that for the entire training set; 'the better the approximations are the better and more reliable will a conjugate gradient algorithm converge' [Møller, 1993e, 41]. As with the strictly classical implementation of CG methods described in chapter 3, Møller's algorithm resets to the steepest descent direction whenever a given $\mathbf{s}_k$ (i.e. the search direction at iteration k) fails to satisfy Eq. 3.9 (i.e. is not a descent direction).

Møller's algorithm has several important merits: it has a much better theoretical foundation than the majority of proposed strategies in this field, its performance does not rely on heuristically-chosen parameters, and it appears to be highly successful at eliminating redundancy in the training set. However, Møller does not quantify the global convergence properties of the algorithm - a primary issue for the current discussion. From the perspective of global optimisation, it is clear that Møller's on-line CG algorithm is very different in character to traditional implementations of on-line BP; with traditional on-line BP, wild changes in the error surface are permitted at each iteration, whereas Møller's algorithm expends considerable effort ensuring that the error surface does not change very much throughout the training process. For this reason it seems likely that the global convergence properties of Møller's on-line CG algorithm are little (if any) better than those of classical CG.

[2] Interestingly, the experiments performed by Belew et al. suggest that the tendency for CG and other classical algorithms to generate weights of much greater magnitude than those generated by BP can be problematic for GA schemes that involve encoding of the weight vector **w**. Using a uniform encoding scheme, GA/BP was able to find 'perfect solutions' to the six-bit symmetry task whereas the solutions found by GA/CG were 'poor in absolute terms' [Belew, McInerney & Schraudolph, 1991, 532].

5.1.2 Deterministic methods

As noted in the introduction, deterministic global methods - unlike stochastic global methods - share the fundamental property that an increase in network error is not required at any stage of the training process. Outlined below are two contrasting approaches to deterministic global optimisation - homotopic methods and tunnelling methods. In neither case do the problems associated with combined stochastic/second-order training methods (discussed in section 5.1.1) arise.

Homotopic methods. Two functions f(x) and g(x) are said to be homotopic to each other if f can be continuously deformed into g, or vice versa - i.e. there exists a *homotopy function* h(λ, x), continuous in both its variables, for which h(0, x) = g(x) and h(1, x) = f(x). The fundamental idea behind homotopic methods for the solution of nonlinear systems is to use a homotopic function h to progressively deform a simple function g, with a known solution, into the nonlinear function f, to which a solution is desired. A variety of such methods, of varying complexity, have been devised outside the field of neural networks (see, for example, the method in [Finhoff & Zimmerman, 1992] for the solution of nonlinear systems of equations).

The second half of this chapter is devoted to a novel homotopic training strategy, ERA, specifically designed for training MLPs. Rather than attempt to devise - for an arbitrary MLP architecture and training task - a simple function with a known solution, ERA deforms the 'normal' error surface E into a surface E' that is easier to solve; having minimised E', surface E' is progressively deformed back into the original surface E. Details of the ERA method are given in section 5.2.

Tunnelling methods. Tunnelling methods proceed in cycles, with each cycle comprising a minimisation phase followed by a tunnelling phase. In the minimisation phase, the MLP is trained in the normal manner using a given minimisation algorithm, until a minimum **w*** is located. The method then enters the tunnelling phase, in which regions of the error surface where E(**w**) > E(**w***) are 'tunnelled through' until a region is located where E(**w**) < E(**w***). The cycles are repeated iteratively until a global minimum is located.

Probably the most effective tunnelling strategy to date is the TRUST (Terminal Repeller Unconstrained Subenergy Tunnelling) method [Cetin, Barhen & Burdick, 1993][3]. At a given minimum **w***, the TRUST method defines a sub-energy tunnelling function which flattens all values of E(**w**) above a threshold of E(**w***) that lie within a specified domain of interest *D*; values of *E*(**w**) below threshold *E*(**w***) are left 'nearly unmodified' [Barhen, Fijany & Toomarian, 1994, 371]. An important characteristic of the TRUST method, as modified by [Barhen, Fijany & Toomarian, 1994], is that convergence to a global minimum is formally guaranteed within domain *D*. However, the TRUST method is much more complicated to implement than the homotopic ERA method of section 5.2, and relies on the user to specify the domain of interest *D*. (The published results in [Barhen, Fijany & Toomarian, 1994] for the TRUST method with gradient descent are based on comparatively small domains of interest, with weights restricted to the range [-10, +10], or smaller. In general, such settings would be inappropriate when using the TRUST method with classical optimisation algorithms, owing to the tendency of such algorithms to generate weights of large magnitudes.)

5.2 Expanded Range Approximation (ERA)

5.2.1 Introduction

The fundamental idea behind the Expanded Range Approximation strategy is to perform a homotopy on the target-vectors $\mathbf{t}_p$ (p=1,...,P) of a given MLP training-set, *S*. The homotopy is achieved by compressing the target-vectors to their mean values - i.e. the *mean target-vector* <**t**> with elements

Eq. 5.4 $$< t >_i \; = \frac{1}{P} \sum_{p=1}^{P} t_{i,p} \quad \text{, for } i=1,\ldots,N^L$$

- and then progressively expanding them back to their original values. The expansion of the compressed target-vectors, $\mathbf{t}_p(\lambda)$ (p=1,...,P), is regulated by a *range parameter* λ ($0 \leq \lambda \leq 1$) according to the following rule:

[3] For a useful summary of the development of tunnelling algorithms for global optimisation, see [Cetin, Barhen & Burdick, 1993, 99-102].

Eq. 5.5 $$\mathbf{t}_p(\lambda) = <\mathbf{t}> + \lambda\left(\mathbf{t}_p - <\mathbf{t}>\right).$$

A parameter value of λ=0 sets each $\mathbf{t}_p(\lambda)$ to <**t**>, a value of λ=1 gives the original training set *S*. In place of the error function *E* given by Eq. 2.3, the modified training set S(λ) is evaluated using a corresponding error function, E(λ), defined by

Eq. 5.6 $$\mathrm{E}(\lambda) = \frac{1}{2PN^L}\sum_{p=1}^{P}\sum_{i=1}^{N^L}\left[t_{i,p}(\lambda) - y_{i,p}^{L}(\lambda)\right]^2 .$$

With λ=1, Eq. 5.6 is equivalent to Eq. 2.3. By monotonically increasing the range parameter from λ=0 to λ=1 in a series of steps, the error surface E(0) is progressively deformed into the original error surface E(1).

Our analysis of how the ERA method works suggests that its success is attributable to the following: the problem defined by λ=0 appears to have only a global minimum, and can be solved trivially; a sufficiently small step η away from λ=0 (i.e. λ=η<<1) is seen to keep the system within the basin of attraction of a global minimum; and the range can be progressively expanded up to λ=1 without displacing the system from the global minimum at any step. A summary of our (on-going) analysis is presented in section 5.2.2. (Full details are given in [Gorse, Shepherd & Taylor, 1995].)

At present, ERA is underpinned by empirical evidence only - the method has been seen to give a greatly improved probability of global convergence in all the cases examined. A theoretical underpinning which delineates the conditions (for example, the rate of expansion of parameter λ) under which ERA can be guaranteed to work is obviously desirable, and an analytical study of the technique is under way. However, even if it can be shown that ERA works in theory, it does not guarantee that a practical implementation is possible; if, to successfully avoid local minima, it proved necessary to expand the range parameter from λ=0 to λ=1 in a very large number of steps and/or to approximate a global minimum of E(λ) at each step to a high degree of accuracy, it is likely that ERA would be deemed excessively slow for many practical tasks. The practical implementation and performance of the ERA method are the subjects of sections 5.2.3 and 5.2.4 respectively.

5.2.2 How ERA works

The following 3-stage description of the ERA method is applicable to its use with any classical multivariate algorithm, i.e. the only assumption made is that successive search directions satisfy Eq. 3.9. In practice, ERA can also improve the practical performance of non-classical training methods (see, for example, the results in section 5.2.4 for ERA with on-line BP).

Solving the system forλ=0. Although we have yet to derive a general proof that the error surface E(0) has only a single global minimum, we have accumulated considerable evidence - both analytical and empirical - to support this proposition. A limited proof, for the XOR task and a 2-layer MLP, is given in the appendix of [Gorse, Shepherd & Taylor, 1995]. The empirical evidence for the proposition that the error surface E(0) has only global minima comes from simulations performed with randomly generated architectures and training sets, as summarised in Table 37. In each case, the training set comprised different input pattern-vectors, but identical target-vectors (with both pattern- and target-vectors set randomly to real values in the range [0, 1]). The task of minimising such a training set (i.e. one with the same target outputs for all input patterns) using the mean-squared error function given by Eq. 2.3 is equivalent to solving E(0) for some training set which has, as its mean target-vector, those target outputs. (In all cases, weights were initialised in the range [-1, 1] and the network architecture used sigmoid output nodes.) All of the tasks summarised in Table 37 converged to $E\approx10^{-25}$ within 100 training epochs using the non-resetting QN algorithm with the DBR line-search - i.e. the training algorithm which, judging by the results in chapter 4, is most prone to getting trapped in local minima.

Table 37 - MLP architectures, training set sizes, and numbers of runs used in testing the proposition that E(0) has only a global minimum

architecture	# of training pairs	# of runs
8-1	62	30
25-1	28	24
3-1-1	41	5
5-2-1	33	41
5-3-1	21	58
5-4-1	42	16
2-2-2	36	18
3-4-2	30	22
4-1-2	32	19

To satisfy the condition E=0 for λ=0, the MLP weights must be of a special form - i.e. one that ensures that, whatever the input pattern, the output is always the same. For a single-layer N-1 MLP architecture with the target value <t> for each input pattern, the following weight values are guaranteed to satisfy E=0:

Eq. 5.7
$$w_{10}^{1} = \ln\left(\frac{<t>}{1-<t>}\right)$$
$$w_{1k}^{1} = 0, \text{ for } k = 1,\ldots,N^{0},$$

where weight w_{10}^{1} connects the output node n_1^1 to the bias unit and weight w_{1k}^{1} connects n_1^1 to the kth input element. For a 2-layer MLP architecture with target-vector <t> for each input pattern, the following weight values are guaranteed to satisfy E=0:

Eq. 5.8
$$w_{i0}^{2} = \ln\left(\frac{<t>_i}{1-<t>_i}\right) - \sum_{j=1}^{N^1} w_{ij}\, s\left(w_{j0}\right), \text{ for } i = 1,\ldots,N^{2},$$
$$w_{jk}^{1} = 0, \text{ for } j = 1,\ldots N^{1} \text{ and } k = 1,\ldots,N^{0},$$

where s(x) is the sigmoid function. The terminal network weights from the training runs performed with each training tasks in Table 37 - more than 200 sets of weights in total - were compared with these predictions; in every case the weights satisfied Eq. 5.7 or Eq. 5.8 (as appropriate) to an accuracy of at least five decimal places.

According to Eq. 5.7 and Eq. 5.8, the error surface E(0) has a single global minimum. However, with a 2-layer MLP the global minimum constitutes a surface, rather than a single location in weight-space, and one for which the network weights may have infinite values.

The first step $\lambda=\eta<<1$. To justify the assertion that a sufficiently small step away from $\lambda=0$ keeps the system within the basin of attraction of the E(0) global minimum (located by the weight-setting procedure described above), it is useful to investigate the properties of the error surface E(η). We can do this by reformulating the error function of Eq. 5.6 as follows:

Eq. 5.9 $$\mathrm{E}(\eta) = \mathrm{E}(0) + \eta E_1 + \mathrm{O}(\eta^2)\,,$$

$$E_1 = \frac{1}{2PN^L}\sum_{p=1}^{P}\sum_{i=1}^{N^L}\left[t_{i,p} - <t>_i - y'(0)\right]y_{i,p}^L(0),$$

where the value of $y'(0)$, the derivative of $y_{i,p}^L(0)$, depends on the chosen learning law, but is assumed bounded. In terms of Eq. 5.9, what we need to show is that the perturbation brought about by the term ηE_1 will not create any local minima, i.e. will only shift the location of the global minimum. Outside a small neighbourhood (N_0) of the global minimum, it can be shown that no local minima can exists whenever

Eq. 5.10 $$\eta\left|\frac{\partial E_1}{\partial \mathbf{w}}\right| < \left|\frac{\partial\,\mathrm{E}(0)}{\partial \mathbf{w}}\right| \rightarrow \frac{\partial}{\partial \mathbf{w}}\left[\mathrm{E}(0)+\eta E_1\right] \neq 0\,.$$

To visualise how a first step η that is sufficiently small guarantees the absence of local minima outside N_0, let us consider the one-dimensional case where a perturbation, in the form of a surface E_1 with multiple minima, is added to an error surface E(0) with a single, global minimum - see Figure 4. The number of stationary points in the error surface E(η) depends on the number of intersections between the linear function $\mathrm{E}'(0)$ and the function $-\eta\,\mathrm{E}_1'$ (i.e. the number of locations where the derivative $\left[\mathrm{E}(0)+\eta E_1\right]' = 0$), and the number of such intersections is determined by the size of η. This point is illustrated in Figure 5; of the three error surfaces, corresponding to three different values of η ($\eta_1<\eta_2<\eta_3$), only E(η_1) has no local minima.

Within neighbourhood N_0, we should expect the location of the global minimum to be shifted by no more than O(η) as $\eta\rightarrow 0$, owing to the smoothness of the error surface.

Figure 4 - Schematic representation of error surfaces E(0) (with a single global minimum) and E_l (with multiple minima)

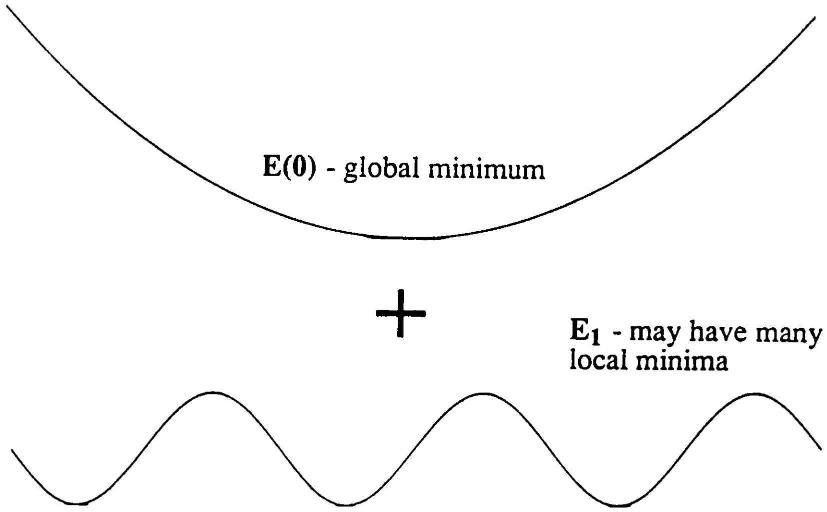
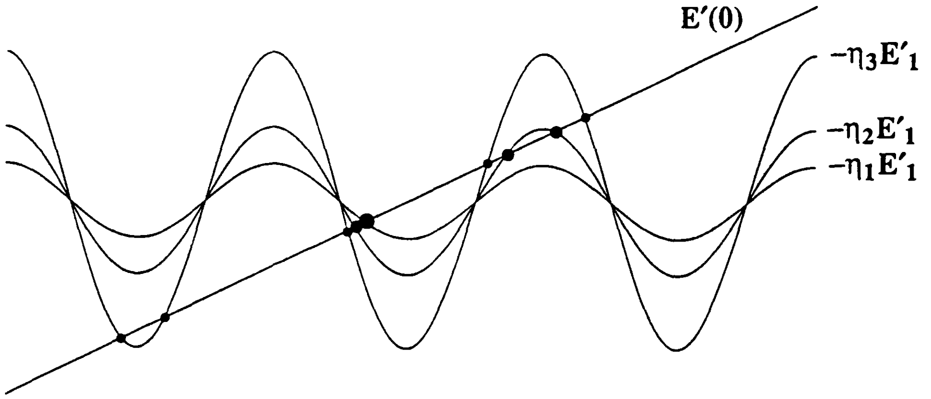


Figure 5 - Illustration of how the number of stationary points in error surface $E(\eta)$ is determined by the size of η ($\eta_1<\eta_2<\eta_3$). Stationary points in $E(\eta)$ occur where the linear function $E'(0)$ intersects function $-\eta E'_1$.



Expanding the range to λ*=1.* As with the argument developed for the first step, the smoothness of the MLP error surface is the key to showing how the system can be kept within the basin of attraction of a global minimum while progressively expanding the range parameter up to $\lambda=1$. Having located the global minimum $\mathbf{w}(\lambda)$ for the current step, we should expect a subsequent step of ε to keep the network within the basin attraction of the 'expanded' global minimum $\mathbf{w}(\lambda+\varepsilon)$ - provided the network approximates $\mathbf{w}(\lambda)$ with sufficient accuracy and ε is sufficiently small. This derives from the observation that the basin of attraction of $\mathbf{w}(\lambda+\varepsilon)$ should contain location $\mathbf{w}(\lambda)$ when $\varepsilon \to 0$, assuming that $\mathbf{w}(\lambda+\varepsilon)$ is shifted from location $\mathbf{w}(\lambda)$ by only $O(\varepsilon)$ as $\varepsilon \to 0$.

Whereas error surface $E(\lambda=0)$ has a unique global minimum, error surface $E(\lambda>0)$ typically has many global minima. The possibility that one or more global minima in the error surface $E(\lambda_k)$ (for $0<\lambda_k<1$) may cease to be global minima in surface $E(\lambda_{k+1})$ (for $\lambda_k<\lambda_{k+1}\leq 1$) is currently an obstacle to a general proof of the ERA method.

5.2.3 Implementing ERA

For this research the ERA method was implemented with three user-defined parameters: the size of the first step, $\eta>0$; an expansion-rate parameter, $\beta\geq 1$; and an error-tolerance parameter, $\varepsilon\geq 0$, controlling the accuracy with which a global minimum of $E(\lambda)$ is approximated at each step (with $\lambda < 1$). Given initial values of $\lambda_0=0$ and $\lambda_1=\eta$, subsequent values of range parameter λ are defined as follows:

Eq. 5.11 $$\lambda_{k+1} = \min[1.0, \lambda_k + \beta(\lambda_k - \lambda_{k-1})], \text{ for } k \geq 1.$$

An 'N-step ERA' method refers to the special case where $\beta=1.0$ and η is chosen such that $1/\eta$ is a whole number - i.e. range parameter λ is expanded in steps of a uniform size, so that the method requires the solution of the N problems $S(\lambda_n=n\eta)$ for $n=1,\ldots,N=1/\eta$.

Although suitable for the experimental investigation of the ERA method conducted for this research, the use of an error-tolerance parameter to terminate ERA training steps is not satisfactory in general, because the 'appropriate' level of error tolerance varies from task to task and step to step (depending on the scale of $E(\lambda)$ and the size of the residuals at a global minimum of E(λ)). One alternative is to terminate a training step (with $\lambda_k < 1$) only when the gradient drops to zero, but this approach is likely to entail a significant increase in the total number of training iterations unless the number of steps is small and the chosen training algorithm converges rapidly near a minimum. A second alternative is to terminate intermediate ERA steps on the basis of the correct classification of training patterns in the training set. For instance, a step could be terminated when $q \le P$ (for a P-pattern training set) patterns are 'correctly classified', i.e. satisfy

Eq. 5.12 $$\frac{y_{i,p}^L - <t>_i}{t_{i,p} - <t>_i} > r, \text{ for } 0 < r \le 1 \ ,$$

where parameter r determines the degree to which networks output $y_{i,p}^L$ is required to match target output $t_{i,p}$ for it to be deemed 'correct'. (With $q = P$ and $r = 1$, a step is terminated only when $E = 0$.) Schemes for automatically adjusting user-defined parameters q and r are currently under investigation.

Finally, when using the ERA method with one of the second-order classical algorithms for general optimisation considered in chapters 3 and 4, these algorithms were reset to the steepest descent direction at the start of each step, on the grounds that a descent direction for error surface $E(\lambda_k)$ is not guaranteed to be a descent direction for error surface $E(\lambda_{k+1})$.

5.2.4 Experimental results

N-step ERA. The preliminary set of ERA results presented in Tables 38, 39 and 40 are for the *N*-step ERA method - implemented with the classical and traditional training algorithms evaluated in chapter 4 - applied to the sigmoid XOR, linear XOR and sine problems respectively. (For the sine problem, results are for the non-resetting quasi-Newton algorithm **QN nr** only, on the grounds that no other algorithm had a significant failure rate on this problem without the ERA method - see results in section 4.3.4.) Selected results from these tables are presented in Graphs 32 and 33. Training conditions were identical to those used to generate the results in chapter 4 (see section 4.3.1), with the exception of the epoch limit - set to 50,000 for each ERA step. The error tolerance (at λ=1) was E=0.01.

Table 38 - *N*-step ERA, XOR with sigmoid output nodes

ERA error tolerance: $\varepsilon=1.0^{-08}$ (classical methods), $\varepsilon=1.0^{-05}$ (BP)
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)
Model-trust region strategy (LM only): RT with parameter u_0=0.01 (Eq. 3.15) and reduction/growth constants 2.0/4.0
Backpropagation settings: training rate η=3.0, momentum α=0.0

***N*-step ERA**	**minima (%)**			**EFEs per run**			**resets**	**EFEs**
method: *N*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
QN nr: regular	36.0	29.0	35.0	89.7	38.1	77.3	0.08	4.02
QN nr: 1	54.5	23.5	22.0	115.0	41.7	107.5	0.08	3.76
QN nr: 2	66.0	17.0	17.0	191.6	124.3	162.8	0.06	3.44
QN nr: 5	72.5	14.5	13.0	320.0	188.2	282.5	0.16	3.22
QN nr: 10	79.5	14.0	6.5	479.4	238.4	441.5	0.09	3.07
QN nr: 20	87.5	7.5	5.0	758.5	254.3	730.0	0.07	2.99
QN nr: 50	93.5	5.0	1.5	1,496.5	559.7	1,533.5	0.17	2.95
QN nr: 100	94.5	4.5	1.0	2,824.5	660.9	2,953.5	0.05	2.90
QN: regular	62.5	30.5	7.0	141.1	170.9	96.0	0.08	4.13
QN: 1	74.0	16.5	9.5	146.0	64.5	131.5	0.07	3.59
QN: 2	88.0	8.0	4.0	249.3	169.0	214.0	0.11	3.39
QN: 5	93.5	5.5	1.0	428.1	887.7	350.5	0.02	3.36
QN: 10	97.0	3.0	0.0	557.3	123.2	535.3	0.01	3.03
NQN: regular	65.5	27.0	7.5	166.3	283.3	115.0	0.14	3.98
NQN: 1	75.0	18.0	7.0	237.8	181.6	185.8	0.16	3.61
NQN: 2	87.5	11.5	1.0	585.6	1,984.7	310.0	0.22	3.62
NQN: 5	95.5	4.5	0.0	1,054.7	5,266.7	567.5	0.13	4.19
NQN: 10	99.0	1.0	0.0	1,242.9	2,194.5	956.8	0.16	3.32
CG: regular	61.5	26.5	12.0	177.1	604.4	83.5	0.44	6.47
CG: 1	77.5	15.0	7.5	282.1	1,314.9	139.0	0.66	6.39
CG: 2	92.0	6.5	1.5	261.8	367.5	202.0	0.61	3.85
CG: 5	94.0	5.5	0.5	361.5	81.1	340.3	0.71	3.44
CG: 10	99.0	1.0	0.0	1,220.0	9,053.8	562.3	0.86	5.69
SD: regular	82.5	16.5	1.0	280.5	123.0	256.0	-	3.57
SD: 1	87.5	12.5	0.0	9,500.3	1.8^{04}	2,360.5	-	3.54
SD: 2	90.0	10.0	0.0	1.1^{04}	2.1^{04}	3,279.5	-	3.53
SD: 5	97.5	2.5	0.0	1.2^{04}	1.9^{04}	5,688.0	-	3.53
BA: regular	83.5	16.0	0.5	967.9	688.9	808.0	-	1.0
BA: 2	88.0	12.0	0.0	3,875.2	1.1^{04}	2,672.0	-	1.0
BA: 5	95.0	5.0	0.0	9,471.6	6,927.9	7,548.0	-	1.0

N-step ERA	minima (%)			EFEs per run			resets	EFEs
method: *N*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
OL: regular	83.0	16.0	1.0	1,300.4	2,506.2	869.5	-	1.0
OL: 2	91.0	9.0	0.0	3,934.0	1,167.9	3,657.5	-	1.0
OL: 5	96.5	3.5	0.0	8,519.1	4,781.2	7,523.0	-	1.0
LM: regular	90.5	9.5	0.0	12.4	2.8	12.0	0.41	1.03
LM: 1	94.0	6.0	0.0	38.1	4.9	38.0	0.52	1.01
LM: 2	97.0	3.0	0.0	65.4	10.8	64.0	0.61	1.01
LM: 5	100.0	0.0	0.0	138.9	16.5	138.0	0.77	1.01

Table 39 - *N*-step ERA, XOR with linear output nodes

ERA error tolerance: ε=1.0^{-08}
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)
Model-trust region strategy (LM only): RT with parameter u_0=0.01 (Eq. 3.15) and reduction/constants 2.0/4.0

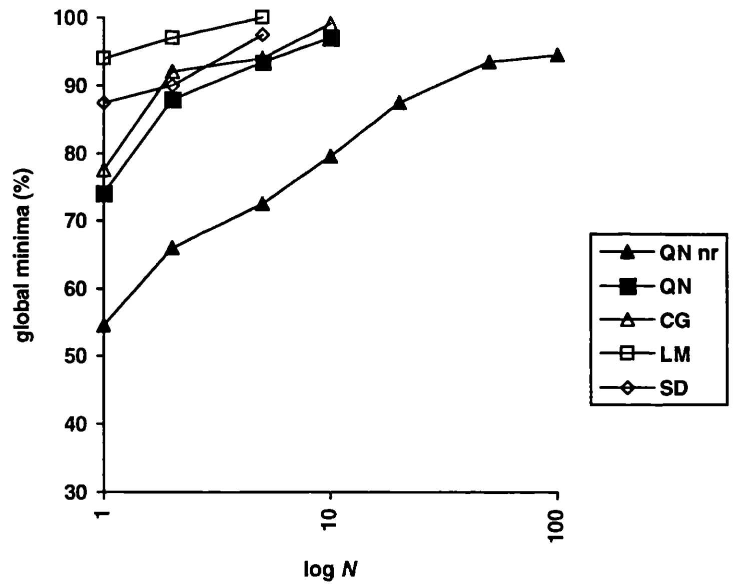
N-step ERA	minima (%)			EFEs per run			resets	EFEs
method: *N*	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
QN nr: regular	69.0	21.0	10.0	114.2	162.0	78.5	0.15	3.75
QN nr: 1	73.0	14.0	13.0	111.1	29.6	110.0	0.01	3.41
QN nr: 2	74.5	16.5	9.0	150.7	43.9	142.5	0.02	3.24
QN nr: 5	82.0	12.5	5.5	224.3	83.2	209.0	0.04	2.99
QN nr: 10	89.0	7.5	3.5	350.5	139.9	323.5	0.04	2.90
QN nr: 20	93.0	5.0	2.0	537.2	115.4	523.5	0.03	2.80
QN nr: 50	94.0	4.5	1.5	1,076.6	238.5	1,052.5	0.06	2.72
QN: regular	92.0	7.0	1.0	116.5	150.2	87.5	0.04	3.49
QN: 1	93.0	6.0	1.0	174.0	199.3	134.0	0.01	3.36
QN: 2	98.0	2.0	0.0	284.9	787.8	179.5	0.01	3.53
QN: 5	99.5	0.5	0.0	278.8	78.2	262.0	0.03	2.98
NQN: regular	90.0	7.5	2.5	139.1	133.8	101.5	0.48	3.29
NQN: 1	94.5	5.0	0.5	416.4	495.2	226.0	1.37	3.13
NQN: 2	99.0	1.0	0.0	490.3	463.4	331.5	2.83	3.07
NQN: 5	100.0	0.0	0.0	767.0	730.7	591.3	3.66	2.99
CG: regular	90.0	9.0	1.0	201.9	641.6	104.0	1.96	3.16
CG: 1	96.0	3.5	0.5	395.2	406.0	249.0	6.00	3.05
CG: 2	99.5	0.5	0.0	885.7	5,467.6	355.5	7.48	3.08
CG: 5	100.0	0.0	0.0	744.3	384.1	641.8	7.92	2.94
SD: regular	93.5	6.5	0.0	1,285.2	1,538.3	924.5	-	2.77
SD: 1	96.0	4.0	0.0	3.4^{04}	3.0^{04}	2.0^{04}	-	2.79
SD: 2	97.5	2.5	0.0	3.0^{04}	2.3^{04}	2.0^{04}	-	2.79
SD: 5	98.0	2.0	0.0	4.0^{04}	2.6^{04}	3.4^{04}	-	2.79
LM: regular	99.0	1.0	0.0	30.1	277.0	10.0	4.49	1.18
LM: 1	99.5	0.5	0.0	40.3	4.4	40.0	0.62	1.02
LM: 2	100.0	0.0	0.0	67.0	4.3	66.5	0.61	1.01
LM: 5	100.0	0.0	0.0	138.2	5.5	138.0	0.83	1.01

Table 40 - *N*-step ERA, sine task

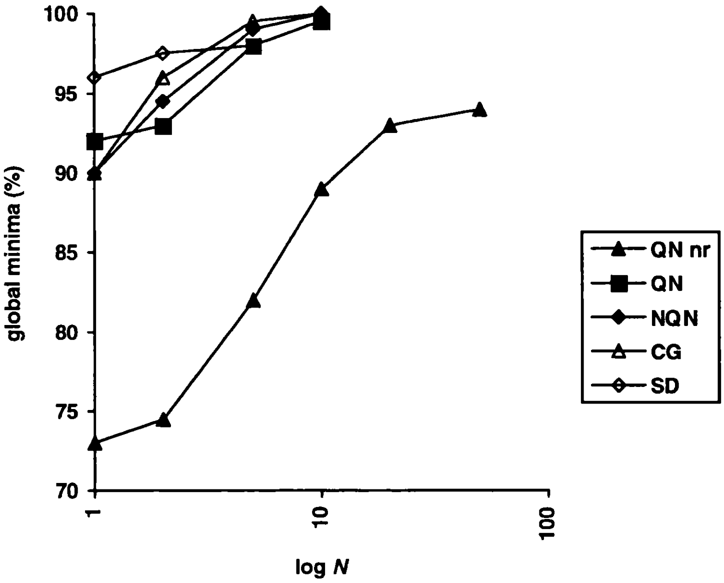
ERA error tolerance: ε=1.0^{-05}
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)

N-step ERA	minima (%)		EFEs per run			resets	EFEs
method: *N*	global	0.023	mean	s.d.	median	/ run	per *k*
QN nr: regular	88.0*	9.0	96.5	44.1	83.5	0.02	3.39
QN nr: 1	94.0	6.0	111.1	40.3	100.0	0.01	3.26
QN nr: 2	95.0	5.0	497.0	194.3	479.0	0.01	3.24
QN nr: 5	96.5	3.5	1,126.5	599.4	938.0	0.01	3.23
QN nr: 10	98.0	2.0	1,735.2	779.2	1,405.0	0.03	3.24

Graph 32 - *N*-step ERA, global convergence frequency vs. log *N* (XOR, sigmoid)



Graph 33 - *N*-step ERA, global convergence frequency vs. log *N* (XOR, linear)



The preceding tables and graphs prompt the following observations about the global convergence properties of the ERA method:

- the global reliability of the *N*-step ERA method increases with *N* (whereas the training speed - measured in median EFEs per training run - decreases with *N*);
- the optimal size of first step with respect to global convergence frequency is task- and method-specific;
- the benefit of increasing *N* for the global reliability of an algorithm is greatest for low *N* and least for higher *N*; substantial benefits accrue even with modest levels of *N*.

The role of the first step $\lambda=\eta$. An examination of the network error at each ERA step suggests that the first step has a special role:

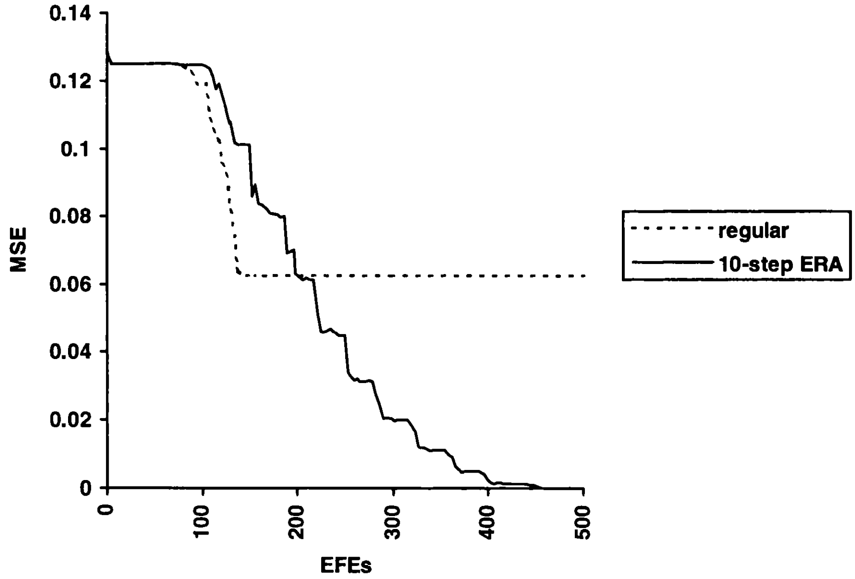
- having solved the problem $S(\lambda_1=\eta)$ at the first step, the network was predisposed to successfully solve the problem $S(\lambda_k)$ (for $k>1$) at each subsequent step;
- if the network converged to a local minimum at a given step (with range $0<\lambda_k<1$), it never succeeded in converging to a global minimum at any subsequent step;

- the number of training epochs required to find a solution at the first step is typically greater than the number required at any subsequent step (of the same size); taking as an example the 10-step ERA curve of Graph 34, the number of EFEs taken at each step was as follows: 19 (λ=0); 129 (λ=0.1); 68 (λ=0.2); 30.5 (λ=0.3); 31.5 (λ=0.4); 36.5 (λ=0.5); 39.5 (λ=0.6); 35.5 (λ=0.7); 49.5 (λ=0.8); 5 (λ=0.9); 30 (λ=1.0, with an error tolerance of 1.0^{-06}).

Graph 34 - Sample 10-step ERA training curve, QN without resets (XOR, sigmoid)

An example of a training run that converged to a local minimum (at E=0.0625) with the standard QN algorithm, but successfully converged to a global minimum with 10-step ERA. The 10-step ERA curve is plotted using the E(1) error function; although the error $E(\lambda_k)$ never increases at ERA step k, the error E(1) can - and, in this instance, does - show local increases for λ<1.

Settings: line search **DBR 0.5 / 0.25**, ERA error tolerance **$\varepsilon=1.0^{-08}$**



To gain an insight into the impact of η - the size of the first step - on the dynamics of MLP training, further experiments were conducted (with various sizes of η), focusing on the behaviour of the network outputs and weights during the solution of S(η). Graphs 35, 36, 37 and 38 plot - for η=1.0, η=0.3, η=0.2 and η=0.1 respectively - the change in network outputs for two of the four patterns in the XOR training set, $\mathbf{p}_1$=00 and $\mathbf{p}_3$=10. So that all four graphs have identical scales, the values are plotted according to the following coordinate transformation:

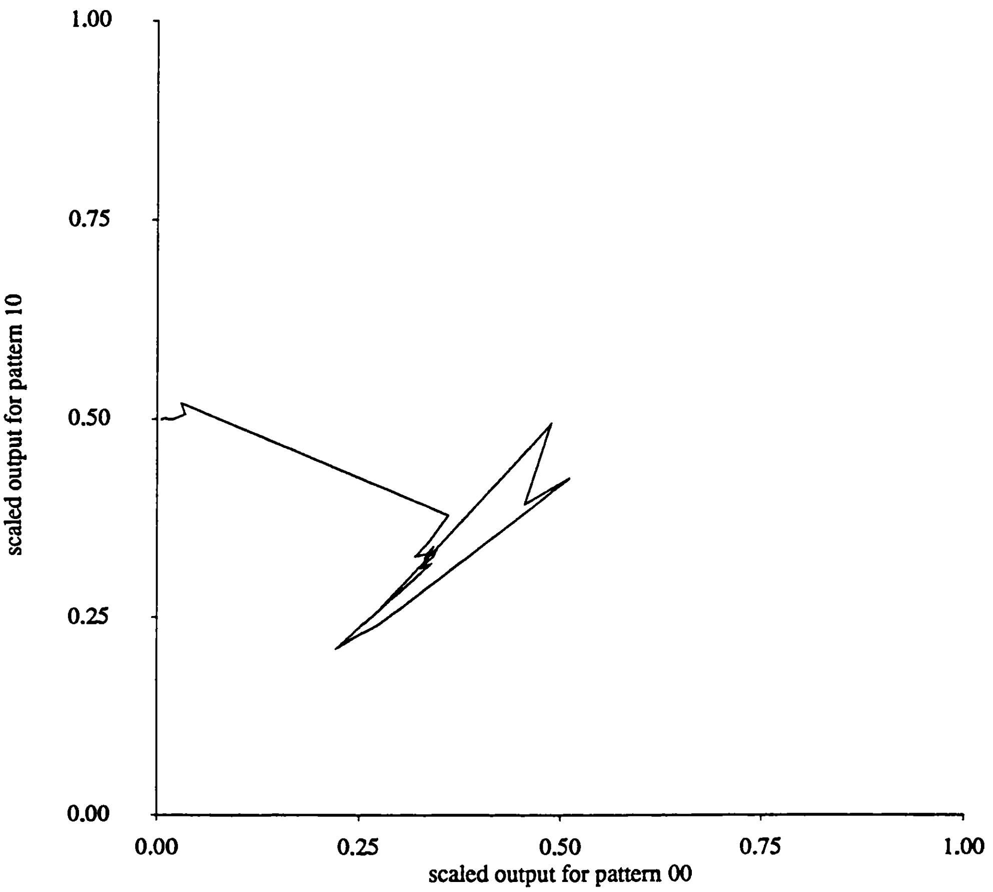
Eq. 5.13
$$(x, y) = \frac{1}{\eta}\left(y_{1,1}^{L} - 0.5(1-\eta), y_{1,3}^{L} - 0.5(1-\eta)\right),$$

where 0.5 is the mean target value $<t>$ for the XOR problem. In each case, the top left-hand corner (0,1) represents a global minimum of E(η), and the midpoint on the *y*-axis (0, 0.5) represents a local minimum of E(η) (at E=0.0625 for E(η=1)).

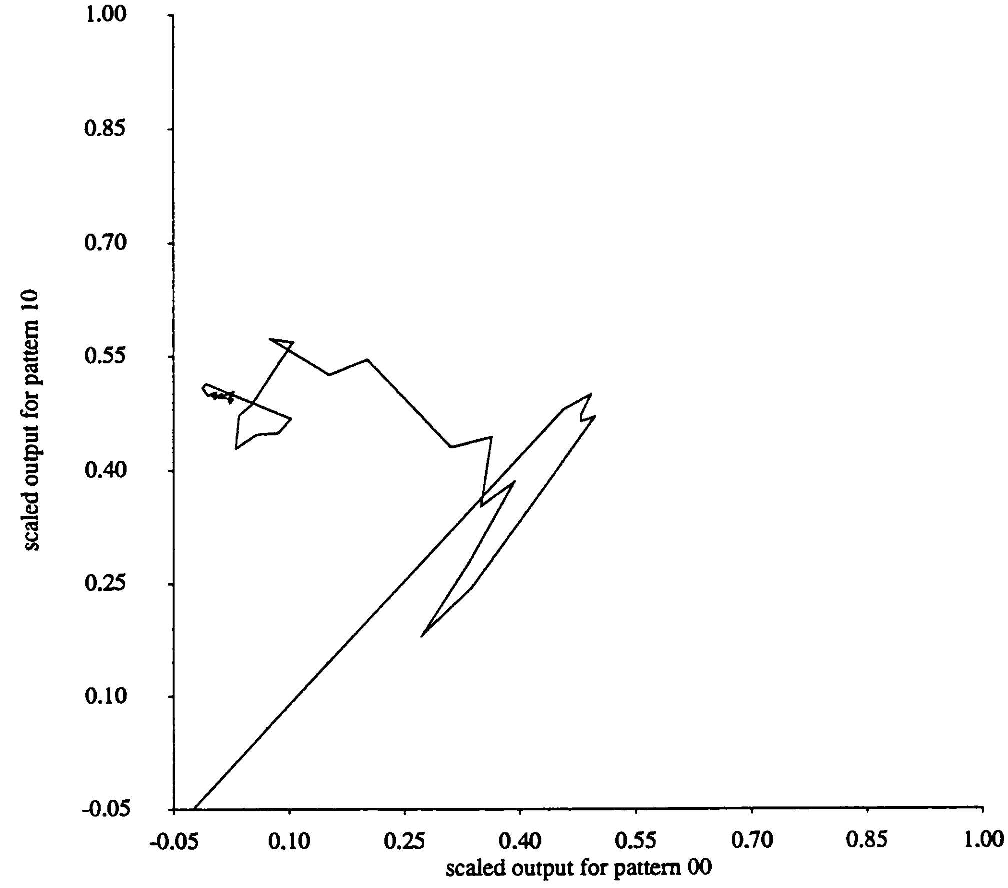
These graphs point to a progressive change in network behaviour as η is decreased. At η=1.0 (Graph 35) the output values rapidly converge to the local minimum. At η=0.3 (Graph 36) the network again converges to the local minimum, but the trajectory appears to come closer to escaping to the desired solution. At η=0.2 (Graph 37) the trajectory spends a lot of time in the vicinity of the local minimum, but ultimately succeeds in reaching the global minimum. Finally, at η=0.1 (Graph 38) the trajectory heads more-or-less directly to the global minimum.

Graph 39 plots, for different sizes of η, the changes in value of a single MLP weight during the first 60 epochs of training on the XOR task with sigmoid output nodes. The graph focuses in on the critical value of η (η_{crit}) at which the network starts to successfully converge to a global minimum. In this case, $0.13 < \eta_{crit} < 0.14$, i.e. for values of η below η_{crit} the network consistently converges to a global minimum, whereas for η greater than η_{crit} it consistently converges to a local minimum. An interesting feature of this graph is that not only are the final weight values different when the network converges to a global rather than local minimum, but the progression to these final values is also different in character; for $\eta > \eta_{crit}$ progression to the final weight values is far less smooth than for $\eta < \eta_{crit}$. This behaviour suggests that a phase change occurs in the learning system at the critical value η_{crit}.

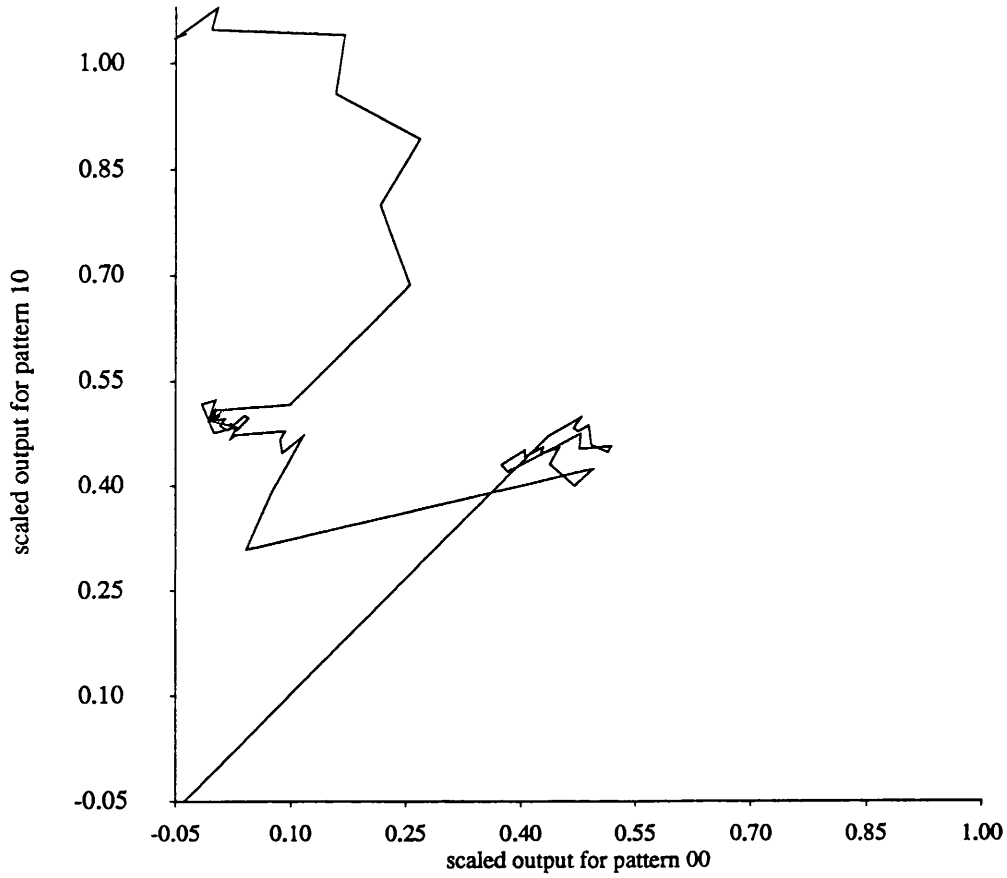
Graph 35 - Trajectory of network outputs for XOR patterns 00 and 10 with η=1.0



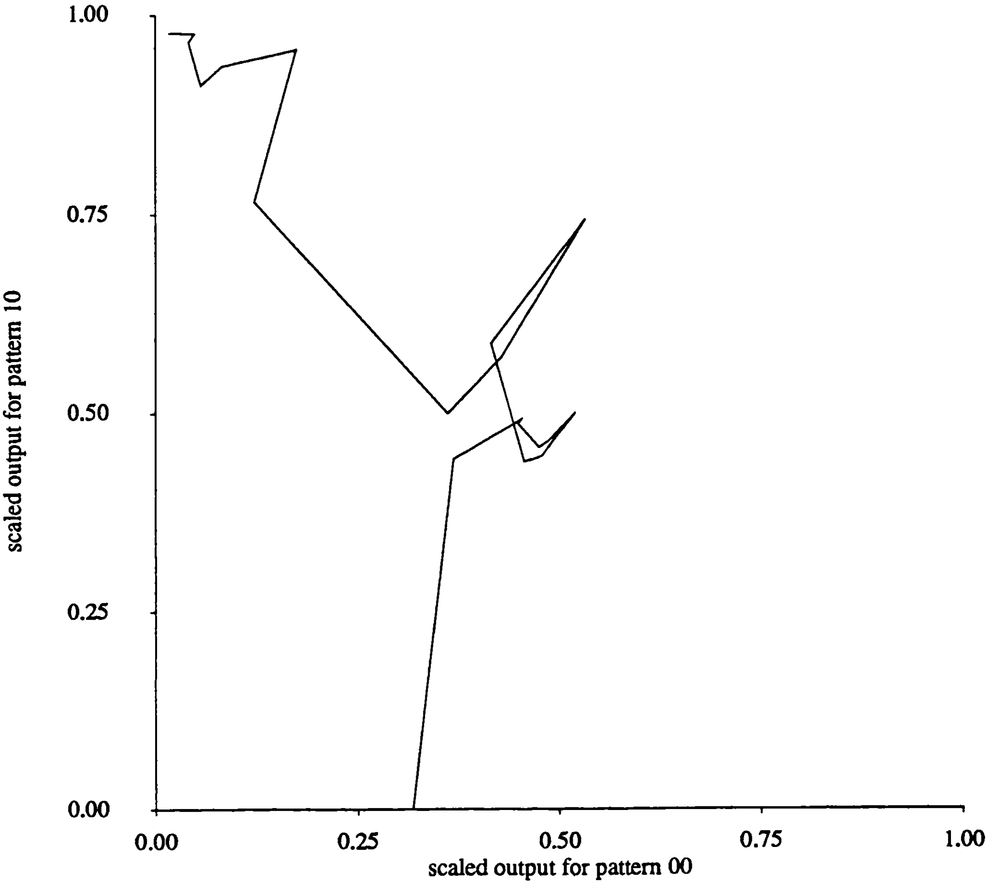
Graph 36 - Trajectory of network outputs for XOR patterns 00 and 10 with η=0.3



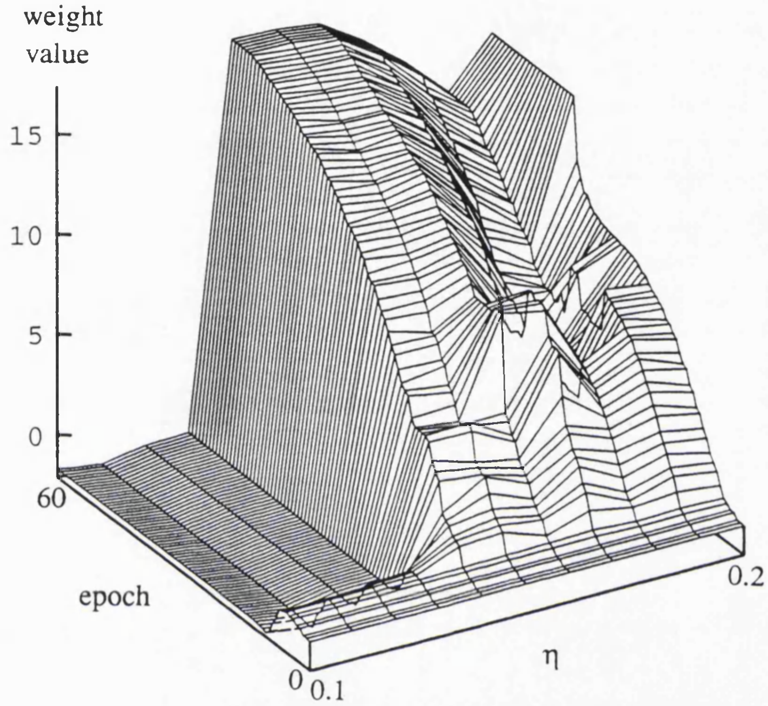
Graph 37 - Trajectory of network outputs for XOR patterns 00 and 10 with η=0.2



Graph 38 - Trajectory of network outputs for XOR patterns 00 and 10 with η=0.1



Graph 39 - The value of a single MLP weight plotted as a function of time (in epochs) and first step size (η)



Improving the speed of ERA. The chosen values of parameters ε and β used to generate the results in Tables 38, 39 and 40 are appropriate from the perspective of global convergence, in that the global convergence frequency at the first step (i.e. for surface E(λ=η)) was retained for the fully-expanded problem E(1) in the vast majority of cases. However, an obvious question is whether these 'conservative' settings for ε and β are overly-conservative with respect to training speed, i.e. is it possible to reduce the number of steps and/or the required accuracy at each step without degrading the global reliability of the ERA method? The following results for the non-resetting quasi-Newton algorithm and XOR task with sigmoid output nodes demonstrate the impact of relaxing parameters β (Table 41) and ε (Table 42) on the global reliability and speed of the ERA strategy for different sizes of first step η.

Table 41 - ERA expansion-rate parameter, β

Task: XOR, sigmoid output nodes
Method: BFGS QN without resets
Line search: DBR with parameters *q*=0.5 (Eq. 3.28) and *u*=0.25 (Eq. 3.29)
ERA error tolerance: ε=1.0^{-08}

	minima (%)			EFEs per run			resets	EFEs
first step (η) : β	global	0.0625	0.0833	mean	s.d.	median	/ run	per *k*
0.2 : 1.0	72.5	14.5	13.0	320.0	188.2	282.5	0.16	3.22
0.2 : 2.0	72.5	14.5	13.0	271.5	204.3	223.8	0.12	3.39
0.2 : 4.0	72.5	14.5	13.0	207.2	135.4	178.0	0.12	3.48
0.1 : 1.0	79.5	14.0	6.5	479.4	238.4	441.5	0.09	3.07
0.1 : 3.0	79.5	14.0	6.5	256.5	148.4	225.3	0.04	3.30
0.1 : 5.0	79.5	14.0	6.5	286.2	165.8	234.0	0.05	3.32
0.1 : 9.0	79.5	14.0	6.5	212.3	126.5	185.5	0.07	3.48
0.05 : 1.0	87.5	7.5	5.0	758.5	254.3	730.0	0.07	2.99
0.05 : 3.0	87.0	8.5	4.5	312.5	225.8	261.0	0.06	3.25
0.05 : 5.0	87.0	8.5	4.5	261.1	181.7	227.0	0.07	3.36
0.05 : 10.0	87.0	8.5	4.5	274.4	173.1	239.5	0.05	3.31
0.05 : 19.0	85.5	8.0	6.5	218.6	158.1	183.0	0.04	3.49
0.02 : 1.0	93.5	5.0	1.5	1,496.5	559.7	1,533.5	0.17	2.95
0.02 : 3.0	93.0	5.5	1.5	345.8	153.8	307.3	0.08	3.25
0.02 : 5.0	93.0	5.5	1.5	306.5	158.3	268.5	0.04	3.29
0.02 : 10.0	93.0	5.5	1.5	252.4	87.6	233.8	0.12	3.39
0.02 : 20.0	93.0	5.5	1.5	263.4	114.8	242.8	0.04	3.35
0.02 : 49.0	77.0	9.0	14.0	215.4	102.4	196.5	0.03	3.54

	minima (%)			EFEs per run			resets	EFEs
first step (η) : β	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
0.01 : 1.0	94.5	4.5	1.0	2,824.5	660.9	2,953.5	0.05	2.90
0.01 : 5.0	92.5	4.5	3.0	302.9	230.1	257.0	0.03	3.33
0.01 : 10.0	92.0	5.0	3.0	262.4	213.8	223.8	0.01	3.43
0.01 : 50.0	86.5	7.0	6.5	297.7	229.0	257.5	0.04	3.40
0.01 : 99.0	74.5	8.0	17.5	255.4	256.0	202.0	0.10	3.66

Table 42 - ERA error-tolerance parameter, ε

Task: XOR, sigmoid output nodes
Method: BFGS QN without resets
Line search: DBR with parameters q=0.5 (Eq. 3.28) and u=0.25 (Eq. 3.29)
ERA expansion rate: β=1.0

	minima (%)			EFEs per run			resets	EFEs
first step (η) : ε	**global**	**0.0625**	**0.0833**	**mean**	**s.d.**	**median**	**/ run**	**per *k***
1.0 : 1.0^{-08}	54.5	23.5	22.0	115.0	41.7	107.5	0.08	3.76
1.0 : 1.0^{-06}	46.0	33.0	21.0	108.7	40.5	100.3	0.17	4.03
1.0 : 1.0^{-04}	36.0	29.0	35.0	97.9	43.8	89.5	0.10	3.98
0.5 : 1.0^{-08}	66.0	17.0	17.0	191.6	124.3	162.8	0.06	3.44
0.5 : 1.0^{-06}	59.5	23.0	17.5	260.8	179.8	176.0	0.11	3.51
0.5 : 1.0^{-04}	55.0	26.0	19.0	146.4	64.5	132.5	0.14	3.59
0.2 : 1.0^{-08}	72.5	14.5	13.0	320.0	188.2	282.5	0.16	3.22
0.2 : 1.0^{-06}	72.5	15.0	12.5	293.9	203.4	250.0	0.18	3.30
0.2 : 1.0^{-04}	69.0	19.0	12.0	231.7	143.4	201.8	0.14	3.36
0.1 : 1.0^{-08}	79.5	14.0	6.5	479.4	238.4	441.5	0.09	3.07
0.1 : 1.0^{-06}	79.5	13.0	7.5	416.1	157.6	383.0	0.11	3.10
0.1 : 1.0^{-04}	78.0	16.0	6.0	324.9	239.8	275.5	0.14	3.17
0.05 : 1.0^{-08}	87.5	7.5	5.0	758.5	254.3	730.0	0.07	2.99
0.05 : 1.0^{-06}	84.0	11.0	5.0	672.8	397.3	617.5	0.07	3.07
0.05 : 1.0^{-04}	77.5	17.0	5.5	423.8	162.8	395.5	0.06	2.95
0.02 : 1.0^{-08}	93.5	5.0	1.5	1,496.5	559.7	1,533.5	0.17	2.95
0.02 : 1.0^{-06}	90.0	7.5	2.5	1,133.1	288.3	1,072.0	0.05	2.91
0.02 : 1.0^{-04}	86.0	10.0	4.0	634.2	449.7	574.0	7.66	2.72
0.01 : 1.0^{-08}	94.5	4.5	1.0	2,824.5	660.9	2,953.5	0.05	2.90
0.01 : 1.0^{-06}	92.5	4.5	3.0	1,855.8	492.9	1,813.5	0.11	2.87
0.01 : 1.0^{-04}	92.5	6.0	1.5	755.6	224.4	729.0	25.09	2.34

The preceding results for the hybrid QN/ERA algorithm (without resets) applied to the XOR (sigmoid) task prompt the following observations about the setting of parameters β and ε.

- When the size of the first ERA step is comparatively large ($\eta \geq 0.1$), the range parameter can be expanded up to $\lambda=1$ in a single step (with expansion-rate parameter $\beta = 1/\eta - 1$) without reducing the global reliability of the algorithm. When the size of the first ERA step is small ($\eta \leq 0.05$), setting $\beta \leq 10$ has negligible (if any) impact on global reliability. However, with $\beta > 10$ a significant deterioration in the rate of global convergence may occur.
- In terms of convergence speed, setting $\beta > 1$ is beneficial for all η less than 0.5, and most beneficial for small η. It is worth noting, however, that convergence speed does not increase consistently with the size of β; for example, with a first step of $\eta=0.01$, ERA is faster with $\beta=10$ than with $\beta=50$. This is attributable to the fact that increasing β from 10 to 50 does not reduce the number of ERA steps required (i.e. 3), but merely increases the size (and 'difficulty') of the intermediate step.
- When the size of the first ERA step is large ($\eta \geq 0.5$), increasing error-tolerance parameter ε produces only a modest improvement in convergence speed, but a substantial deterioration in the rate of global convergence. For smaller η, increasing ε brings about a greater improvement in convergence speed, and the impact on global reliability is somewhat reduced.

It is clear from the results in Tables 41 and 42 that, with appropriate settings for parameters β and ε, it is possible to make substantial improvements on the convergence speed of the default ERA method (*N*-step ERA with $\varepsilon=1.0^{-08}$), without a significant reduction in global reliability. However, although the preceding observations offer some guidance about appropriate settings for β and ε, their optimal settings are task-specific. (This reinforces the desirability of automated parameter-setting schemes, mentioned briefly in section 5.2.3.)

5.3 Conclusion

In contrast to the majority of global optimisation strategies (see section 5.1), the ERA method of section 5.2 is both easy to implement, and fully-compatible with second-order classical optimisation algorithms. Furthermore, the results in section 5.2.4 suggest that ERA represents a highly efficient compromise between global reliability and training speed; when applied to benchmark tests with known local minima, ERA (with appropriate parameter settings) proved highly effective at improving the global reliability of MLP training algorithms without excessively increasing the number of training iterations required to find a solution.

If, as anticipated, it is possible to automate the progressive expansion of range parameter λ, the ERA method will depend on a single user-defined parameter - the size of the first ERA step, η. There is no 'natural' choice of first-step size. In practice, η should reflect the 'hardness' of the training task (if known), and the priorities of the user; broadly speaking, η should be set to a 'small' value for hard problems or when a high degree of global reliability is required, but to a 'large' value if convergence speed is a higher priority. In this context, one promising feature of the results in section 5.2.4 is that a significant improvement in the rate of global convergence is still achieved when η is comparatively large (i.e. $\eta \geq 0.1$).

However, unqualified endorsement of the ERA method must be withheld, pending the outcome of our on-going investigations regarding:

- the development of a rigorous mathematical proof that ERA works in all but pathological (and rare) cases;
- the application of ERA to hard, real-world training tasks;
- the refinement of the procedure by which range parameter λ is progressively expanded.

6. CONCLUSION

This thesis has proposed a novel approach to the development of MLP training algorithms that are both faster and more globally-reliable than conventional, backpropagation-derived, training methods - that is, to combine fast second-order training algorithms (chapters 3 and 4), implemented so as to maximise their potential for global convergence, with the deterministic ERA method for global optimisation (chapter 5). When tested on the benchmark training tasks of section 4.2, hybrid second-order/ERA training algorithms, with suitable parameter settings, proved consistently faster and converged to a global minimum as or more frequently than conventional training algorithms. Moreover, one training algorithm in particular - the Levenberg-Marquardt/ERA algorithm - outperformed all conventional methods by a wide margin. For example, the LM / 5-step ERA algorithm achieved a 100% global convergence rate at a cost of only 138.0 median EFEs per run on the XOR task (with both sigmoid and linear output nodes); this compares with global convergence rates of 83.5 % at a cost of 469.0 median EFEs (sigmoid) and 93.0% at a cost of 1,038.5 median EFEs (linear) for on-line backpropagation - generally the best of the conventional options - with near-optimal parameter settings.

This research has identified several areas where the performance of these algorithms may be open to improvement, for example:

- the choice of error function (section 2.1.2);
- the adoption of an appropriate scaling or preconditioning scheme (section 3.1.5);
- the development of LM algorithms that are less sensitive to the presence of residuals at the solution and have O(n) storage requirements (section 3.6);
- the refinement of the procedure for regulating the progressive expansion of the ERA range parameter λ (section 5.2).

Furthermore, although hybrid second-order/ERA training algorithms have proved highly effective within the experimental framework adopted for this research, an assessment of their performance when applied to hard, real-world training tasks is a clear priority for the future. All of these topics are, or will be, the subject of further research by the author.

BIBLIOGRAPHY

- S Amato, B Apolloni, G Caporali, U Madesani and A Zanaboni (1991): "Simulated annealing approach in backpropagation", *Neurocomputing 3*, 207-220
- A J Annema, K Hoen and H Wallinga (1994): "Learning behaviour and temporary minima of two-layer neural networks", *Neural Networks,* **7** (9), 1387-1404
- J Barhen, A Fijany and N Toomarian (1994): "Globally optimal neural learning", *Proceedings of WCNN '94, San Diego*, June 1994, **III**-370-**III**-375
- E Barnard (1992): "Optimization for training neural nets", *IEEE Transactions on Neural Networks*, **3** (2), March 1992, 232-240
- R Battiti and F Masulli (1990): "BFGS optimization for faster and automated supervised learning", *Proceedings of the International Neural Network Conference (INNC 90), Paris, France*, 757-760
- R Battiti (1992): "First- and second-order methods for learning: between steepest descent and Newton's method", *Neural Computation*, **4** (2), 141-166
- R K Belew, J McInerney and N N Schraudolph (1991): "Evolving networks: using the genetic algorithm with connectionist learning", in C G Langton, C Taylor, J D Farmer and S Rasmussen, eds., *Artificial Life II*, SFI Studies in the Sciences of Complexity, vol. X, Addison-Wesley, Reading, Mass., 511-547
- M Berggren (n.d.): "An efficient method of training feed-forward neural networks using conjugate gradients", Laboratoire de L'Accelerateur Lineaire, Centre d'Orsay, Orsay, France, pre-print
- C Bishop (1992): "Exact calculation of the Hessian matrix for the mutilayer perceptron", *Neural Computation*, **4**, 494-501
- R P Brent (1973): *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey
- R P Brent (1991): "Fast training algorithms for multilayer neural nets", *IEEE Transactions on Neural Networks*, **2** (3), May 1991, 346-354

- R M Burton and G J Mpitsos (1992): “Event-dependent control of noise enhances learning in neural networks”, *Neural Networks*, **5**, 627-637

- B C Cetin, J Barhen and J W Burdick (1993): "Terminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization", *Journal of Optimization Theory and Applications,* **77** (1), April 1993, 97-126

- C Darken, J Chang and J Moody (1992): “Learning rate schedules for faster stochastic gradient search”, in S Y Kung, F Fallside, J A Sorensen and C A Kamm, eds., *Neural Networks for Signal Processing*, **2**, IEEE Workshop, IEEE Press, 3-13

- J E Dennis and R B Schnabel (1983): *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

- L C W Dixon (1972): “The choice of step length, a crucial factor in the performance of variable metric algorithms”, in Lootsma, ed., 149-170

- W Finhoff and H G Zimmerman (1992): "Homotopy methods in circuit analysis", pre-print, Siemens, Munich

- R Fletcher (1980): *Practical Methods of Optimization*, vol. 1, *Unconstrained Optimization*, John Wiley & Sons, New York

- P E Gill, W Murray and M H Wright (1981): *Practical Optimization*, Academic Press, London

- M Gori and A Tesi (1992): “On the problem of local minima in backpropagation”, *IEEE Transactions on Pattern Analysis and Machine Learning*, **14** (1), 76-86

- D Gorse (1992): “Classical and stochastic search in conjugate gradient algorithms”, *Proceedings of IJCNN ‘92, Beijing*, November 1992, 435-440

- D Gorse and A Shepherd (1992): “Adding stochastic search to conjugate gradient algorithms”, *Neural Network World*, **2**, 599-605

- D Gorse, A Shepherd and J Taylor (1993a): “Avoiding local minima using a range expansion algorithm”, *Neural Network World*, **5**, 503-510

- D Gorse, A Shepherd and J Taylor (1993b): "Tracking global minima by progressive range expansion", *Proceedings of IJCNN '93, Portland, Oregon*, July 1993, **IV**-350-**IV**-353
- D Gorse, A Shepherd and J Taylor (1994a): "Avoiding local minima by a classical range expansion algorithm", *Proceedings of ICANN '94, Sorrento,* **1**, May 1994, 525-528 (RN/94/14)
- D Gorse, A Shepherd and J Taylor (1994b): "A classical algorithm for avoiding local minima", *Proceedings of WCNN '94, San Diego*, June 1994, **III**-364-**III**-369 (RN/94/15)
- D Gorse, A Shepherd and J Taylor (1995): "The new ERA in supervised learning", submitted to Neural Networks
- L G Hamey (1995): "Analysis of the error surface of the XOR network with two hidden nodes", Computing Report 95/167C, Department of Computing, Macqarie University NSW 2109 Australia
- R Hecht-Nielsen (1990): *Neurocomputing*, Addison-Wesley, Reading, Mass.
- D M Himmelblau (1972): "A uniform evaluation of unconstrained optimization techniques", in Lootsma, ed., 69-97
- Y Hirose, K Yamashita and S Hijiya (1991): "Back-propagation algorithm which varies the number of hidden units", *Neural Networks*, **4**, 61-66
- J Holland (1975): *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor MI
- S Huang and Y Huang (91): "Bounds on the number of hidden neurons in multilayer perceptrons", *IEEE Transactions on Neural Networks*, **2** (1), January 1991, 47-55
- D Jacobs, ed. (1977)*: The State of the Art in Numerical Analysis: Proceedings of the Conference on The State of the Art in Numerical Analysis held at the University of York, April 12th-15th, 1976, organized by The Institute of Mathematics and its Applications,* Academic Press, London

- E M Johansson, F U Dowla and D M Goodman (1992): "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method", *International Journal of Neural Systems*, **2** (4), 291-301

- B Kappen and T Heskes (1992): "Learning rules, stochastic processes, and local minima", in I Aleksander and J G Taylor, eds., *Artificial Neural Networks, 2: Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92) Brighton, United Kingdom, 4-7 September, 1992,* 2 vols., Elsevier Science Publishers B.V., Amsterdam, **I**-71-**I**-77

- J A Kinsella (1992): "Comparison and evaluation of variants of the conjugate gradient method for efficient learning in feed-forward neural networks with backward error propagation", *Network*, **3**, February 1992, 27-36

- S Kirkpatrick, C D Gelatt, Jr. and M P Vecchi (1983): "Optimization by simulated annealing", *Science*, **220**, 671-680

- D E Knuth (1981): *Semi-Numerical Algorithms,* 2nd edition, vol. 2 of *The Art of Computer Programming*, Addison-Wesley, Reading, Mass.

- J F Kolen and J B Pollack (1990): "Backpropagation is sensitive to initial conditions", *Complex Systems*, **4**, 269-280

- S Kollias and D Anastassiou (1989): "An adaptive least squares algorithm for the efficient training of artificial neural networks", *IEEE Transactions on Circuit and Systems*, **36** (8), August 1989, 1092-1101

- P J G Lisboa and S J Perantonis (1991): "Complete solution of the local minima in the XOR problem", *Network*, **2**, February 1991, 119-124

- P J G Lisboa, ed. (1992): *Neural Networks: Current Applications*, Chapman & Hall, London

- F A Lootsma, ed. (1972): *Numerical Methods for Non-linear Optimization*, Academic Press, New York

- D G Luenberger (1984): *Linear and Nonlinear Programming*, 2nd edition, Addison-Wesley, Reading, Mass.

- J M McInerney, K G Haines, S Biafore and R Hecht-Nielsen (1989): "Error surfaces of multi-layer networks can have local minima", Technical Report No. CS89-157, Department of Computer Science and Engineering, University of California

- M Møller (1993a): "A scaled conjugate gradient algorithm for fast supervised learning", *Neural Networks*, **6** (4), June 1993, 525-533 (reprinted in [Møller, 93e], appendix A)

- M Møller (1993b): "Supervised learning on large redundant training sets", *International Journal of Neural Systems*, **4** (1), 15-25 (reprinted in [Møller, 93e], appendix B)

- M Møller (1993c): "Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time", Technical Report, Daimi PB-432, Computer Science Department, Aarhus University, 1993 (reprinted in [Møller, 93e], appendix C)

- M Møller (1993d): "Adaptive Preconditioning of the Hessian Matrix", submitted to *Neural Computation* (reprinted in [Møller, 93e], Appendix D)

- M Møller (1993e): *Efficient Training of Feed-Forward Neural Networks*, PhD Thesis, Daimi PB-464, Computer Science Department, Aarhus University, December 1993

- J J More (1983): "Recent developments in algorithms and software for trust region methods", in A Bachem, M Grotschel and B Korte, eds., *Mathematical programming: the state of the art, Bonn 1982*, Springer-Verlag, Berlin, 258-287

- W Murray, ed. (1972): *Numerical Methods for Unconstrained Optimization*, Academic Press, London

- J C Nash (1990): *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*, Adam Hilger, Bristol

- M R Osborne (1976): "Nonlinear least squares - the Levenberg algorithm revisited", *Journal of the Australian Mathematical Society*, **19** (Series B), 343-357

- D B Parker (1987): "Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order Hebbian learning", *Proceedings of the First IEEE International Conference on Neural Networks*, IEEE Press, New York, June 1987, **II**-593-**II**-600

- C S Pattichis, C Charalambous, C N Schizas and L T Middleton (1991): "EMG diagnosis using conjugate gradient backpropagation neural network learning algorithm", in T Kohonen, K Makisara, O Simula and J Kangas, eds., *Artificial Neural Networks: Proceedings of the 1991 International Conference on Artificial Neural Networks (ICANN-91) Espoo, Finland, 24-28 June, 1991*, 2 vols., Elsevier Science Publishers B.V., Amsterdam, 1621-1624

- T Poston, C N Lee, Y Choie and Y Kwon (1991): "Local minima and back propagation", in *Proceedings of IJCNN '91, Seattle, WA*, July 1991, **II**-173-**II**-176

- M J D Powell (1977): "Restart procedures for the conjugate gradient method", *Mathematical Programming*, **12**, April 1977, 241-254

- W H Press, B P Flannery, S A Teukolsky and W T Vetterling (1988): *Numerical Recipes in C*, Cambridge University Press, Cambridge

- A K Rigler, J M Irvine and T P Vogl (1991): "Rescaling of variables in back propagation learning", *Neural Networks*, **4**, 225-230

- D E Rumelhart, G E Hinton and R J Williams (1986): "Learning internal representations by error propagation", in D E Rumelhart and J L McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, MIT Press, Cambridge MA, 318-362

- S Santini (1992): "The bearable lightness of being: reducing the number of weights in backpropagation networks", in I Alexander and J G Taylor eds., *Artificial Neural Networks, 2: Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92) Brighton, United Kingdom, 4-7 September, 1992,* 2 vols., Elsevier Science Publishers B.V., Amsterdam, **I**-139-**I**-142

- R W H Sargent and D J Sebastian (1972): "Numerical experience with algorithms for unconstrained minimization", in Lootsma, ed., 45-68

- A Shepherd (1992): "Towards a hybrid conjugate gradient/ backpropagation algorithm for training feed-forward neural networks", MSc Thesis, Department of Computer Science, University College London

- S A Solla, E Levin and M Fleisher (1988): "Accelerated learning in layered neural networks", *Complex Systems*, **2**, 625-639

- H Szu and R Hartley (1987): "Fast simulated annealing", *Physics Letters*, **122**, 157-162

- P P van der Smagt (1990): "Neural implementation of Powell's conjugate gradient minimisation", Report UvA-sma-2-9012-2, Department of Computer Science and Mathematics, University of Amsterdam

- P P van der Smagt (1994): "Minimisation methods for training feedforward neural networks", *Neural Networks*, **7** (1), 1-11

- P D Wasserman (1989): *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York

- R L Watrous (1987): "Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization", *Proceedings of the International Conference on Neural Networks*, IEEE Press, New York, July 1987, **II**-619-**II**-627

- L F A Wessels and E Barnard (1992): "Avoiding false local minima by proper initialization of connections", *IEEE Transactions on Neural Networks*, **3** (6), November 1992, 899-905

- L F A Wessels, E Barnard and E van Rooyen (1990): "The physical correlates of local minima", in *Proceedings of the International Neural Network Conference* (Paris), 985

- M A Wolfe (1978): *Numerical Methods for Unconstrained Optimization: an introduction*, Van Nostrand Reinhold, New York

- X Yu (1992): "Can backpropagation error surface not have local minima" *(sic)*, *IEEE Transactions on Neural Networks*, **3** (6), November 1992, 1019-1021