University of Kent

Kent Academic Repository

Sirlantzis, Konstantinos (2002) *Supervisor and searcher co-operation algorithms for stochastic optimisation with application to neural network training.* Doctor of Philosophy (PhD) thesis, University of Kent at Canterbury, Canterbury, UK.

Downloaded from

https://kar.kent.ac.uk/7419/ The University of Kent's Academic Repository KAR

The version of record is available from

https://doi.org/10.22024/UniKent/01.02.7419

This document version

UNSPECIFIED

DOI for this version

Licence for this version

CC BY-NC-ND (Attribution-NonCommercial-NoDerivatives)

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in ***Title of Journal*** , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

SUPERVISOR AND SEARCHER CO-OPERATION ALGORITHMS FOR NOISY OPTIMISATION WITH APPLICATION TO NEURAL NETWORKS

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT AT CANTERBURY

IN THE SUBJECT OF OPERATIONAL RESEARCH

FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY.

By

Konstantinos Sirlantzis

March 2002

F185002

KENT
TEMPLEMAN
LIBRARY
UNIVERSITY

To my wife Eleni; without her none of these would have been possible

Contents

List of Tables

List of Figures

Abstract

In this thesis we studied a novel class of algorithms for unconstrained optimisation with particular focus to the issues arising in noisy optimisation. These algorithms were developed using an innovative framework for design of efficient and robust algorithms, namely the Supervisor and Searcher Co-operation (SSC) framework. This framework provides a systematic way to incorporate desirable characteristics of existing algorithms into a new improved scheme.

The aim was to explore the properties of the SSC-based algorithms focusing on their *behaviour in practice* under the presence of stochastic noises. To this end, first, a basic algorithm was proposed along with a number of modifications and extensions to it. Then, their properties were evaluated in a systematic way through a variety of experiments involving a wide range of non-trivial deterministic and stochastic problems. Our findings suggest that the SSC algorithms are demonstrably *efficient* in the deterministic case, but, mainly, that they are *robust* enough to successfully address the difficulties arising in the presence of stochastic noises. Also, they can easily be modified to meet specific application requirements, while the resulting algorithms retain the desirable properties of the original algorithm.

Finally, to assess the *applicability* of the SSC algorithms in real world problems an adaptation to the basic SSC algorithm was proposed for use in Multilayer Neural Network training. The corresponding evaluations were performed through statistical experiments on a number of regression and classification problems, designed to cover the complex issues associated with neural networks learning, such as overtraining, and mainly *generalisation* ability. The SSC-based algorithm exhibited significantly better performance than the two algorithms used as benchmarks for comparisons. Specifically, it was demonstrably faster with respect to reduction of the error in the training set, but more importantly, it showed increased ability to avoid overtraining and hence to generalise (perform successfully in unknown samples), which is the ultimate goal of learning in neural networks.

Acknowledgements

I would like, first of all, to thank my supervisor, Professor Wenbin Liu, for his constant and constructive guidance throughout my endeavours. He introduced me to the ideas that form the basis of this work. His continuing constructive criticism, stimulated a deeper insight into the intriguing and challenging field of optimisation. I am grateful to him for being available for discussion and valuable suggestions every time I needed it.

During this academic adventure, I was lucky and honoured to have the generous and untiring guidance of Professor Mike Fairhurst, who put his trust in me since we first met. I am enormously indebted to him for the continuous support and understanding through difficult personal circumstances. Throughout the time of our collaboration, he proved more than an academic mentor to me, he became a valuable friend.

My colleagues at the Department of Electronics and especially of the Digital Systems Research Group created a really thought–provoking environment with our discussions and long "coffee breaks"; I thank them all! My warm thanks go especially to my colleague Dr. Sanaul Hoque with whom I shared anxieties and achievements, as well as an office, during the last difficult year.

Special thanks must also go to the many friends and colleagues who provided a wonderful social context, in which it was a pleasure to live and work.

I would, also, like to acknowledge the invaluable moral support of my family. Enduring a long-term absence my father-in-law, Anestis, stayed nonetheless an inexhaustible spring of courage and support. A great and warm thanks is due to my beloved mother, Evaggelia, for being there for me during all my life, and for

putting always my welfare above her own.

Above all, in this adventure I was blessed to have as fellow traveller my companion in life, my wife Eleni. Her love and dedication were my inner power and the reason I succeeded to finish this journey. She never stopped believing in me even when, and maybe mostly when, I started losing my faith. For this, and a plethora more, I will always adore her.

Chapter 1

Introduction

This thesis studies a class of algorithms for optimisation of noisy functions developed within a novel framework, namely the Supervisor and Searcher Co-operation framework (SSC). This framework provides an effective mechanism to design new efficient optimisation algorithms by combining desirable features of two (or more) existing ones. The principle underlying the SSC framework can be viewed as a systematic way of exploring possible combinations of existing optimisation algorithms (*synthesis* of algorithms).

To verify the flexibility provided by the SSC framework a number of algorithms are introduced and experimentally evaluated in a series of benchmarking problems. Furthermore, the ability of an algorithm designed according to this framework to successfully address complex realistic tasks is explored through comparative statistical experiments in the challenging field of Artificial Neural Network (ANN) training.

In order to place the research reported in this study within the relevant cognitive contexts the remainder of the chapter is divided in two parts:

In the first part, the class iterative gradient-based deterministic optimisation algorithms is initially discussed briefly to provide an introduction to related methods used for optimisation problems in the presence of noise. Then, the noisy optimisation problem is formally defined followed by a critical review of the most

popular categories of algorithms associated with it. Consequently, the discussion is naturally led to the methods of Stochastic Approximation with which the Supervisor and Searcher Co-operation framework is more closely related.

The second part of this chapter considers concepts and theoretical aspects of the field of Artificial Neural Networks (ANNs), and discuss their relationship with noisy optimisation. Also, it demonstrates the importance of ANNs not as a single application but as a cognitive area in its own right based on both their theoretical attributes as well as the wide variety of real world problems they have been employed to address. These problems cover both the two main categories of function approximation, those of *regression* and *classification.* Subsequently, the points of the preceding discussion providing motivation to the present work are summarised and the corresponding research objectives are stated. The chapter concludes with an outline of the contents of the remaining chapters of the thesis.

1.1 Deterministic Optimisation

The classical deterministic unconstrained minimisation problem can be formally defined as follows:

$$\min_{x \in R^n} F(x).$$

In this section we limit our discussion to gradient-based iterative algorithms because they are the most closely related to the algorithms studied in the present work. Although a large number of alternative methods exist to address the above optimisation problem, e.g. the class of *trust region-based* algorithms, such an extensive survey is out of the scope of this thesis.

From a historical point of view, Cauchy was the first to apply the *steepest descent method* to solve unconstrained minimisation problems (Cauchy, 1847). The theory of local optimization provides powerful tools for the optimisation a smooth function F. A well-known method in the class of steepest descent methods (Avriel, 1976; Luenberger, 1973; Rao, 1984) for unconstrained minimisation of functions having Lipschitz continuous first partial derivatives is given by Armijo

in (Armijo, 1966). Under suitable assumptions for the function to be optimised this method always converges to a local minimum. It is, in fact, a modification of Cauchy's original method and allows for the possibility of variable step size. Furthermore, it does not require knowledge of the value of the Lipschitz constant.

The well-known Newton and Newton-type methods show superlinear convergence in the vicinity of a non-degenerate optimiser. Additionally, global convergence results for quite wide classes of problems have been stated, for example, for strictly convex functions. These methods combine robustness and the property of being locally fast. They use a (local) quadratic approximation of F, and in the case of Newton-type methods through a, possibly modified, Hessian. However, these methods assume the ability to acquire knowledge about the gradient or the Hessian, in contrast to other optimisation techniques like the Nelder-Mead (Nelder and Mead, 1965) method, or the direction set method of Powell (Powell, 1992).

Another efficient class of methods is known under the names *quasi-Newton* and *variable metric methods*, typified by the *Davidon-Fletcher-Powell* (DFP) algorithm (Avriel, 1976; Davidon, 1959; Fletcher and Powell, 1963; Luenberger, 1973; Press et al., 1992; Rao, 1984). Closely related are, also, the so-called *Fletcher-Powell* (FP), and the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) algorithms. In contrast to Newton methods they employ an approximation of the Hessian, built up iteratively. These methods are very stable and they converge superlinearly (Ortega and Rheinboldt, 1970). On the other hand, they require storage of order n^2 (where n is the dimensionality of the problem) and they need derivative calculations. Moreover, they approximate the inverse of the Hessian matrix and they involve one-dimensional subminimisation procedures.

Finally, the class of methods called *nonlinear conjugate gradient methods*, are typified by the *Fletcher-Reeves* (FR) algorithm (Polak, 1971; Press et al., 1992). Their convergence for general nonlinear problems, is linear (Luenberger, 1973; Ortega and Rheinboldt, 1970). They require storage of order only a few times n,

but they, also, involve derivative calculations as well as one-dimensional subminimisation and, finally, they exhibit high sensitivity to round-off errors (Vrahatis et al., 1996).

1.2 Noisy Optimisation

The methods mentioned previously require precise function and gradient values. In many optimisation problems of practical interest, however, the values of the objective functions are known only within some (often low) precision. For instance, when the function and gradient values are the result of numerical simulations, precise values may be difficult to be obtained. Or, when the available function values result from the numerical integration of a system of differential equations the precision of the computed values is necessarily limited.

This imprecise knowledge of the function values means that although the underlying function may be smooth, the values observed show a discontinuous behaviour. Hence, the function values corresponding to small variations around some point x do not reflect the local behaviour of the function but that of the noise. Therefore, methods employing finite difference estimators of the gradient, such as the quasi-Newton methods, often fail. Nevertheless it has been argued by many authors (Fletcher, 1987; Gill et al., 1981; Nocetal, 1992) that quasi-Newton methods using a finite difference approximation for the gradients seem to be among the more efficient for the optimisation of smooth functions when only the function values are available (Elster and Neumaier, 1997).

However, for the cases where substantial noise is present, (Gill et al., 1981, Section 8.6.2.2) suggest that a remedy could be the use of larger difference intervals in the finite difference estimation process, although this requires knowledge of the precision of the function to be optimised and hence the level of variability of the noise. The latter can also be estimated (see, for example, (Gill et al., 1981, Section 8.5.2.3)), but only at the expense of additional function evaluations, and the estimation should be repeated regularly when the noise depends on x.

1.2.1 Formulation of the Problem

In real world applications, various types of noises are present in most of the available data. Here the noises are conceived in a broad sense. In some cases, these noises can be ignored, but more often than not, they play an important role in the mathematical modelling of the application. The purpose of this thesis is to study a class of new algorithms to handle such cases. We are interested in the noisy unconstrained minimisation problem formally defined as follows:

$$\min_{x \in R^n} F(x),$$

where it is difficult or impossible to evaluate $F(x)$ exactly. Then, let $f(x)$ be an estimator of $F(x)$ in the sense

$$f(x) = F(x) + \epsilon,$$

then F is the underlying exact mathematical model, and either:
1) ϵ represents some kind of deterministic error (e.g. round-off or truncation errors due to inexact computations), or
2) ϵ is stochastic noise, which may depend on x.
In the first case, it is possible that ϵ is negligible in function value evaluation, but it becomes significant if an estimator of the gradient is needed (Mathews, 1992, pp. 28–40). In the second case, also, ϵ can rise to levels which will significantly affect the smoothness of the objective function and its gradient.

Henceforth, whenever we refer to the function value or the gradient of f at a point $x \in R^n$, we mean an estimator of the corresponding value or gradient of F at x. When we refer to a minimizer of f, we, in fact, mean a minimizer of F. Obviously, in the degenerate case where $\epsilon = 0$, the estimators are identical to the corresponding exact values.

1.2.2 Some Examples of Noisy Optimisation

It is possible to identify to main categories of problems where stochastic models arise in optimisation. In the first belong cases where we have to consider a quantity that is only measurable with error. For instance, we may wish to find a combination of controls maximising the output of an industrial plant, but we can only estimate the output from an experiment. Such an example occurs when attempting to optimise settings to achieve the maximum yield from a chemical reaction. The objective function is, consequently, evaluated by carrying out chemical experiments which are subject to random errors. For a given set of parameter values the experiment can be repeated many times over and then the average yield over the whole set of experiments can provide an improved estimate of the objective function.

In the second category the interest is focused on parameter estimation in complex statistical models. Such problems are the maximisation of a likelihood or the minimisation of a goodness-of-fit statistic when the appropriate distributions are so complex that they are not tractable analytically. Examples of this type of problems are given by (Diggle and Gratton, 1984) and (Ruppert et al., 1984). In maximum likelihood estimation methodology, for instance, part of the likelihood (e.g. the normalising constant) may be needed to be estimated by simulation methods or in a moment-based estimator the theoretical moments could be found by simulation. Another example of this type occurs when the evaluation of the function to be optimised involves the numerical solution of a system of Partial Differential Equations (PDE), where the accuracy depends on the grid size used. Finally, in this category can be classified also the increasingly important area of simulation optimisation (see (Fu, 1994) for a recent review).

In all the above cases finding the best choice of parameter values requires the judicious balancing of the time spent on improving the accuracy of the objective function estimator at a single point against the time spent in evaluating the function at different parameter settings.

Desired Features of the Algorithms

Noisy optimisation (and especially the case of stochastic noises) introduces requirements which are beyond the scope of most popular algorithms for deterministic function optimisation.

On one hand, many deterministic methods rely heavily on the objective function's derivatives calculation. However, if, for example, the objective function value is the expected response of a simulation process, the relationship between the simulation input and the expected response is often too complicated to find analytical expressions for the derivatives (Glasserman, 1991b; Glynn, 1987). On the other hand, the behaviour of the optimisation method should be insensitive to small perturbations in the observed function values, since these are subject to uncertainty due to either finite precision calculations or random errors.

Hence, these variations in the function values and in the corresponding estimators of the derivatives limit the applicability and the effectiveness of deterministic gradient-based algorithms. We anticipate that a desirable algorithm will combine efficiency in the easier problems with robustness and reliability when the problems become noisier, since it is difficult to control the level of noise in real world applications.

1.2.3 Categories of Methods

One possible classification of the methods proposed to tackle stochastic optimisation problems is reflected in the following two categories:

1. Methods based on *functional estimation*, which construct an estimate of the objective function over its entire domain, and then optimise the estimate.

2. *Iterative* methods, which start at some initial design point, and move by (usually small) steps at successive iterations, based on local information about the objective function, e.g. by using a gradient estimator at each iteration.

3. *Direct-search* methods, which do not require gradient estimates but utilise only evaluations of the objective function to approach the optimum.

Approaches that belong in the first category are sometimes called *stochastic counterpart* methods, because they optimise a stochastic estimate of the actual objective function. Any deterministic optimisation algorithm can be used once the estimate has been constructed. In principle, the methods of nonparametric regression, which aim to approximate an unknown function over its entire domain solely on the basis of noisy data from selected parts within the domain, are applicable to the stochastic counterpart approach. For example, in (Müller, 1985; Müller, 1989) this idea is used within the framework of kernel regression techniques. Newer approaches such as the so-called *sample path optimisation* (Plambeck et al., 1996), or the *retrospective optimisation* (see (Andradóttir, 1998) and the references therein), belong also to this category.

In the second category, usually, an "efficient" gradient estimator is needed, e.g. unbiased and/or with bounded variance (see (L'Ecuyer and Yin, 1998) for details). For certain classes of functions, methods such as perturbation analysis (Fu, 2001a), score function, or likelihood ratio, or finite differences with common random numbers, can provide gradient estimators with the required efficiency (see (Glasserman, 1991b; Glynn, 1990; L'Ecuyer, 1991; Rubinstein and Shapiro, 1993)). However, due to the fact that these methods are often hard to implement, one should often rely on the less efficient but straightforward finite differences and their variations (Spall, 1992).

Algorithms that are often characterised as direct-search methods constitute another category of techniques. These are based on deterministic analogs directly adapted to the stochastic setting. Examples include the well-known *Nedler-Mead* method (Nelder and Mead, 1965) and its variants (see (Barton and Ivey, 1996)), in which rules that control movements and possible expansions or contractions guide the updates on a simplex set of points at each iteration. The *Hookes-and-Jeeves* method (Jacobson and Schruben, 1992) belongs also to this category.

Stochastic search methods for global optimisation, such as Simulated Annealing (Kirkpatrick et al., 1982) and Genetic Algorithms (Goldberg, 1989; Holland, 1975), have often been adapted to address noisy optimisation problems with continuous parameters. These algorithms can also be categorised as direct-search methods since they typically do not employ gradients. Randomness is used to explore the set over which a function is to be optimised in an efficient manner. It has been reported that these methods generally do well in very complex optimisation problems (e.g. see (Sirlantzis et al., 2001a) for an example in Multiple Classifier Systems for handwritten character recognition). However, as pointed out in (Schoen, 1991) these algorithms are typically several orders of magnitude more computationally expensive to achieve the same accuracy as the "traditional" local optimisation methods. The task in which they can really excel is "to detect regions of attraction of new local optima, and/or provide confidence about the fact that the global optimum has already been found".

Finally algorithms based on techniques drawn from the area of Response Surface Methodology (RSM), in Statistics, developed originally for the analysis of factorial Experimental Designs can be applied in either of the first two classes.

In the following we shall review in more detail the response surface methodology and the direct-search algorithms, as well as some adaptations of deterministic gradient based algorithms, because these are the methods more often suggested in the literature to be appropriate to tackle noisy optimisation problems. Finally, the class of stochastic approximation methods will also be reviewed, as it is closely related to the algorithms proposed in this work. Recent surveys for these methods have been given by (Azadivar, 1992; Jacobson and Schruben, 1989; Safizadeh, 1990).

1.2.4 Response Surface Methodology (RSM)

Response Surface Methodology (RSM) is based on statistical field of design of experiments methodology. Since exploring the entire feasible region of the objective function, which is wasteful and in many cases impractical, RSM explores small

subregions successively based on their potential for improvements. The approach adopted in RSM is straightforward (Khuri and Cornell, 1987): the behaviour of the objective function is estimated using points in the neighbourhood of the current point x to form some kind of *factorial* experiment. Then, a low order polynomial (usually linear) is fitted to these points. Subsequently, a line search in the negative gradient direction defines the size of the step to be taken, and the whole process is repeated until the linear response surface becomes inadequate. The latter is indicated when the slope becomes "approximately" zero, which means that at that point the interaction effects are larger than the main effects. The optimum is determined analytically from the fit of a higher order response surface (e.g. a quadratic).

A related method, using a quadratic function which provides a more accurate fit to the objective function values and chooses the descent direction accordingly has been proposed by (Glad and Goldstein, 1977). For this method convergence results have been established for bounded noises. Similar approaches are adopted in (Elster and Neumaier, 1995) in which a grid-based algorithm is introduced, and in (Elster and Neumaier, 1997) where an algorithm based on the notion of *trust regions* is proposed. Both were shown to be superior to the Nelder-Mead method on a variety of noisy test problems.

Since the first phase of the algorithm is iterative, usually an attempt is made to carry out fewer function evaluations if possible. However, in the final phase it is important to acquire accurate information about the function topology, so a large number of evaluations is required. Recognising the significance of this disadvantage a method is suggested in (Karidis and Turns, 1984) which utilises the previous function evaluations for the least squares fit, instead of extra function evaluations at points in a fixed pattern. Also, the search part of this type of algorithms is often identical to Stochastic Approximation methods. Taking into consideration the above observations it is not difficult to realise that the methods in this category are, in general, more computationally expensive than the standard Stochastic Approximation algorithms. Details and references can be found in (Fu,

2001b; Safizadeh, 1990).

1.2.5 Direct-Search Methods

In the last few years there has been an increasing interest in direct search methods (Anderson and Ferris, 2001) for unconstrained optimization. Direct-search techniques rely only on the function values to find the location of the optimum. These methods do not, usually, require first- or second-order Taylor approximations of the function for their choice of search direction. Additionally they do not make gradient estimates, and involve relatively few function evaluations at each iteration. The most commonly used method in this class is due to Nelder and Mead (Nelder and Mead, 1965). The method uses repeated operations of *reflection, expansion and contraction* applied to a simplex of $n + 1$ points in R_n. Invented more than 30 years ago, the Nelder-Mead method, or some version of it, is still the most common when the function value estimators are the result of separate experiments, despite the fact that other direct-search methods, such as the one due to Powell (Brent, 1973; Powell, 1964), often exhibit superior performance.

The Nelder-Mead simplex method is also the most popular direct-search method, based on published applications (Barton and Ivey, 1996). The range of application is very broad: use of the method has been reported for problems in analytical chemistry, biology, neurology, statistics, engineering, quality control, fishery management, and fusion technology, to name but a few. In (Fletcher, 1987) this technique is considered "the most successful of the methods which *merely* compare function values". Other popular direct-search methods include the complex method by Box (Box, 1966) and the method introduced by Hooke and Jeeves (Hooke and Jeeves, 1961).

However, the situations in which the direct-search methods can prove more useful are appropriately pointed out in the following quotation from (Swann, 1972), as referred in (Barton and Ivey, 1996):

> [Direct-search methods are] particularly useful for cases where the

> function is non-differentiable, where the first partial derivatives of the function are discontinuous, and where the function value depends on some physical measurement and may therefore be subject to random error, all of which are problems which can cause difficulties using the more theoretically based gradient methods. In practice they have generally proved to be robust and reliable. In addition, the relative simplicity of the direct search approach can prove advantageous since it generally means that the methods can be easily and quickly programmed.

Very limited knowledge exists about the convergence properties of the Nelder-Mead method on deterministic functions. Moreover, general proof of convergence of the iterates (to any point, optimal of not) has yet to be reported for the cases where the expected value of stochastic functions is to be optimised. Not even for the simple case of a quadratic function. In (Barton and Ivey, 1996) it is claimed that the paper presents the first formal analysis of the behaviour of Nelder-Mead type of methods on stochastic functions.

Nevertheless, the most commonly recommended method for optimising noisy functions is the simplex (or polytope) method of Nelder and Mead (see (Nocetal, 1992, p. 30) and (Powell, 1988, p. 230) for some examples). This is due to the fact that the method is controlled solely by the relative size (ranks) of the function values and does not make any assumption about the function's continuity. Therefore, it is considered to have the ability to cope with noise. It has been argued that this method proved to be a useful in many applications, despite the lack of theoretical convergence statements (Elster and Neumaier, 1997). The argument is based on the idea that in the presence of noise more complex methods which approximate the function with some polynomial base on recent function evaluations may be led seriously astray (Powell, 1994).

On the other hand, the original Nelder-Mead method was not designed for such (stochastic) applications, and in a nondeterministic context, can terminate

inappropriately, possibly at a solution that is far from the true optimum (Anderson and Ferris, 2001). This has been recognised for some time and has led suggestions for modifications aiming to remedy the situation in practice (Hedlund and Gustavsson, 1992). Furthermore, in (Humphrey and Wilson, 1998) the Nelder-Mead procedure has been found to be extremely sensitive to user-specified starting values. Another important restriction of methods in this category is that their computation times depend heavily on the dimension of the problem. They are, usually, considered suitable for small to medium sized problems (Dennis and Torczon, 1991).

Finally, while in some cases it is possible to apply methods that use stochastic derivatives in conjunction with direct-search methods, the latter are more important when stochastic gradient techniques cannot be applied, or require greater computational effort, due, for example, to expensive calculations in order to improve the accuracy of the gradient estimators.

1.2.6 Stochastic Approximation

The name Stochastic Approximation covers a great number of methods used to tackle mainly optimisation problems involving stochastic errors. These techniques range from the classical Robbins-Monro/stochastic gradient method to recent methods involving scaling of the gradient estimates (Poljak and Tsypkin, 1973)

We begin with one of the most popular algorithms in engineering computations, namely the classical Stochastic Approximation algorithm (denoted henceforth as SA), which has been widely used in various applications with strong (stochastic) noises since the 1950's. It was introduced in the cornerstone papers (Kiefer and Wolfowitz, 1952) and (Robbins and Monro, 1951), and there exists an extensive body of references about it in the literature—see, for example, (Benveniste et al., 1990; Kushner and Clark, 1978; Kushner and Yin, 1997; Ljung, 1986), as well as the references cited therein. Stochastic approximation methods are, in general, the stochastic versions of gradient-based deterministic search

algorithms. Their general form can be, in most cases, defined as follows:

$$x_{k+1} = \Pi_\Theta(x_k - t_k \hat{\nabla} f(x_k)) \tag{1.1}$$

where, in the same notation as previously, $\hat{\nabla} f(x_k)$ is an estimate of $\nabla F(x_k)$ from iteration k, and Π_Θ is a (not always present) projection onto the set of feasible solutions Θ. In this algorithm, there is no line search at all. It uses a pre-assigned positive sequence of step sizes $\{t_k\}$ (often called *gain sequence*), e.g.

$$t_k = 1/k, t_k = 1/k^{0.5}, \text{ or } t_k = 0.001.$$

SA proves very robust, and its global convergence has been established under various assumptions for the noises. The most common criticisms are that SA is in general very slow, and that it is difficult to select suitable $\{t_k\}$, as it will be shown later on in some examples. For deterministic problems without noises, SA has been known to be very slow in comparison with efficient gradient algorithms like the conjugate gradient (CG) method. (Nevertheless, this algorithm and its variants are the most widely used optimization algorithms in the training of Feed-forward Artificial Neural Networks, (which, as stated previously, is the real world application area of particular interest in this study). This is probably due to its simplicity and strong robustness.)

There have been many improvements on SA since the 50's. A number of adaptive or second order SA methods were proposed (see (Benveniste et al., 1990; Kushner and Clark, 1978; Kushner and Yin, 1997; Ljung, 1986)), though most of them seem either to be expensive or to bring only marginal improvements, see also Uryas'ev's work in (Uryase'ev, 1992).

Considerations about the convergence of stochastic approximation algorithms place conditions on the following:

1. the objective function;

2. the gain sequence;

3. the bias and variance of the gradient estimators;

4. the choice of stopping rules.

For the objective function, in general, some degree of differentiability is required and either convexity or unimodality. Additionally, when the projection operator is not used, supplementary conditions are needed (e.g. Lipschitz continuity). For the gain sequence a very fast rate of decrease may lead the algorithm to premature convergence to a (possibly) wrong value, while a very slow decrease may prevent the algorithm from convergence to any value at all. For the gradient estimate is required that the bias should go to zero, and the variance must be, usually, uniformly bounded. Unlike the conditions for the gain sequence, those corresponding to the function and gradient estimators may not be possible to be verified directly.

A set of commonly used assumptions, which clearly satisfy the above gain sequence conditions, and allow convergence w.p. 1 of the algorithm to be established is the following: $\sum_k t_k = \infty, \sum_k t_k^2 < \infty$. The harmonic series $t_k = a/k$ (for some constant a), for instance, satisfies these conditions. However, in practice, to avoid slow convergence, sequences decreasing more gradually, or even constant step sizes, are often used. The general form of gain sequences used is $t_k = a/k^\alpha$, and (in the case of finite difference estimation for the gradients) the estimation interval is $c_k = c/k^\beta$, where α, β, a and c are constants to be selected given that $\alpha \leq 1$ and $\alpha - \beta > 0.5$. Then, the optimal asymptotic convergence rates that can be achieved are: $n^{-1/2}$ for the Robbins-Monro algorithm, and $n^{-1/3}$ for the Kiefer-Wolfowitz using symmetric differences. Finally, the choice of the appropriate gain sequence is a major difficulty with SA. The algorithm performance is extremely sensitive to it and the optimal choice requires knowledge about the eigenvalues of the Hessian of the objective function, which is typically unknown and difficult to estimate.

A notable observation is that iterate *averaging* often produces improved performance over the use of a single iterate, that is, using $\overline{x}_k = \sum_{i=1}^k x_i/k$ as the

estimate of the optimum; (Pflug, 1996) is a useful resource for further discussion and references. In (Kushner and Yin, 1997, pp. 21,327–346) it is argued that such an estimating procedure, called "Polyak averaging" is preferable only when the gain sequence goes to zero slower than $O(1/k)$. There are other stochastic approximation techniques which, instead of decreasing step lengths, they use increased sampling of $f(x)$ to ensure convergence. Examples can be found in (Dupuis and Simha, 1994).

The development of methods for gradient estimation in stochastic optimisation is a very active field of research. Procedures that use estimates of the gradient with some bias but without resorting to finite differences are often called Robbins-Monro-like algorithms, while when unbiased estimators are utilised they are categorised as Kiefer-Wolfowitz algorithms. The key point to consider that procedures providing unbiased estimates are, in general, computationally expensive. For example, the symmetrical finite difference (SD) estimator needs a different pair of estimates for each parameter dimension to obtain an estimate of the gradient, thus requiring $2n$ function evaluations. On the other hand, the one-sided finite difference (FD) estimator requires $n + 1$ function values, while the simultaneous perturbation (SP) estimator of Spall (Spall, 1992) requires only two function evaluations, because it uses the same pair for every parameter dimension. SP provides at least an order of magnitude savings in computational load. Also, SP was reported to compare favorably to finite difference and random directions gradient estimating procedures (Chin, 1990; Chin, 1997). In general, approaches providing unbiased estimators of the gradients require knowledge of the underlying system. Recent methods include perturbation analysis (Fu, 2001a; Ho and Cao, 1991; Glasserman, 1991a), the likelihood ratio/score function method (Rubinstein and Shapiro, 1993), and weak derivatives (Pflug, 1996).

Finally, stopping rules are, usually, based on the progression of the iterates, the gradient estimates, or some combination. As it will be shown in Sections 2.4 and 2.5, where we discuss the stoping criteria employed in the experiments of this study, a variety of stopping rules have been used in different reports;

(Anderson and Ferris, 2001; Elster and Neumaier, 1997; Humphrey and Wilson, 1998), for instance, represent only a small sample of these. This fact, of course, place limitations on the usefulness of the reported results for valid comparisons.

1.2.7 A Class of Classical Algorithms

Under the term "classical algorithms" a number of techniques can be considered aiming to facilitate direct application, to the noisy optimisation, problems of classical deterministic gradient-based algorithms. Let us first have a closer look at the generic classical minimization algorithm:

Given x_0, t_0, v_0,

$$x_{k+1} = x_k - t_k v_k, \ k = 0, 1, 2, ..., \tag{1.2}$$

where t_k and v_k are commonly referred to as the step length and the search direction respectively. In many cases, the following algorithm:

$$x_{k+1} = x_k - v_k$$

is locally convergent. To a large extent, it actually decides the speed of algorithm (1.2), while its global convergence is normally ensured by selecting the step lengths $\{t_k\}$ via a line search procedure of, for example:

1) the exact monotone type,

2) the inexact monotone type, e.g. Armijo-Goldstein, Wolfe-Powell (see (Dennis and Schnabel, 1983) and (Fletcher, 1987)),

3) the inexact non-monotone type, e.g. Grippo-Lampariello-Lucidi (see (Grippo et al., 1986)).

However, the robustness of most existing line search procedures deteriorates when the smoothness of the objective functions is compromised either by the presence of strong noises or because the functions are in general only Lipschitz. For instance, in stochastic optimization an exact line search could prove very expensive

and unstable, due to noises. Also, most of the existing inexact line search methods, though possibly less expensive, seem to have similar problems. In stochastic optimization the estimated gradient normally contains much stronger noises than those present in the measurement of an objective function. Hence, the Wolfe type line search results are likely to be inconsistent. In fact, the existing line search procedures frequently fail, as it will be illustrated in our numerical experiments. For nonsmooth objective functions, line search methods are in general not applicable.

There has been extensive research in developing efficient algorithms for the noisy optimization problems in the literature. An obvious approach is to use some kind of averaging procedure such as the sample mean:

$$\hat{F}(x_k) = N^{-1} \sum_{1}^{N} f(x_k)$$

(for a positive integer N) to estimate the objective function value at x_k. If the noise has zero mean, this, of course, is an unbiased estimator of the expected function values. Then existing deterministic optimization algorithms may be applicable. However, this approach is, in general, computationally expensive.

There were also attempts to generalise the standard line search procedures to the stochastic case. For instance in (Yan and Mukai, 1993), the Armijo type line search with restarts is proposed. One of the basic ideas is that the line search is allowed to restart if it fails. Since the estimated objective function values are in general different each time they are re-sampled, this line search should eventually succeed to overcome poor local minima. Also, it should be assumed that the errors in the gradient estimator used in the algorithm tend to zero as the iterations proceed. This idea has not been widely used in applications yet. In our experiments, it was found that it may not work efficiently. In general, the algorithms reviewed in this section are either unstable or slow, due to the presence of line search procedures. However, the algorithms studied in this thesis can be efficient and robust without having to resort to line searches. They belong to the class of algorithms we review in the following section.

1.2.8 Combinations of Methods

Combining techniques is currently an emerging area of active research which has shown the potential to be a highly fruitful path in tackling the problems arising in noisy optimisation. For instance, the Gradient Surface Method (GSM), which was proposed in (Ho et al., 1992), combines the approaches of RSM and SA. GSM proceeds in two phases like RSM. In the first phase it fits a surface to a set of points, while, at the same time, implicitly using a second-order design by considering also the corresponding gradient surface. A single replication is used, as usual, to obtain the gradient estimator defining the first-order least-squares fit. However, unlike SA, at every iteration in the first phase multiple such points (in essence a "window") are used. In the second phase the algorithm switches to SA when the optimum is approached. This is because SA is considered to be much more efficient than curve fitting in the neighbourhood of the optimum. Gradient estimation is used in two ways in the GSM procedure: first, to guide the search of the gradient surface least-squares fit, and second, to provide the search direction, as usual, after switching to the stochastic approximation part. This procedure offers gains of the order of n^2, where n is the number of parameters, in the cases where n is large enough. The procedure is sequential in spirit and attempts a fast exploration of each region in the fitted surface, similarly to the SA algorithms. However, it shares with RSM the property of using more than just the current iteration information. Finally, in GSM the problem of choosing an appropriate window size, replaces that of the step length choice in SA.

The algorithms studied in the present work are developed by exploiting the flexibility offered by a framework which can by assigned to this category of methods, namely the Supervisor and Searcher Co-operation framework. However, unlike the example previously presented, which is specialised on the combination of two particular algorithms the SSC framework makes and additional step and provides a generic mechanism for the *synthesis* of different methods.

1.3 Artificial Neural Networks (ANNs)

We shall now proceed to introduce concepts and discuss issues of significant importance in the field of Artificial Neural Networks, which, as explained previously, will be the area where the Supervisor and Searcher Co-operation algorithm will be evaluated for its applicability and effectiveness in real world applications. The aim of this review is to *place in context* our discussion of learning algorithms for neural networks in Chapter 5.Before moving further, the basic question to be answered is whether Artificial Neural Networks (ANNs) training is still an important problem and if the response is affirmative why it forms a challenging area for research.

1.3.1 Some Definitions

In order to form a basis to our discussion, it is useful to regard some of the definitions about them.

Definition 1. (Cheng and Titterington, 1994) Neural Networks are the mathematical models represented by a collection of simple computational units interlinked by a system of connections.

Definition 2. (Müller and Reinhardt, 1990) A Neural Network Model is defined as a directed graph with the following properties:

1. The existence of nodes and a state variable associated with each one of them.
2. The existence of links between nodes and a real-valued weight (w) associated with each link.
3. The existence of inputs associated with each node.
4. The existence of a transfer function for each node that determines its state as a function of its input.

According to this definition the building elements of ANNs are:

- The *computational units* (neurons or nodes) that can be characterized as *input* units if their input is fed outside the system, *output* units if they

provide their state variable outside the system, and *hidden* units if their input is a function of other units and their output serves as input to other units.

- The *link weights* that must be determined in order for the task of the network to be achieved.

It is often convenient to consider the neurons organized in layers containing nodes that share the same task. So, we have input, output, and hidden layers. Networks that have no hidden layer are called *single-layer* networks while those with one or more hidden layers are called *multilayer* networks (see *Multilayer Perceptrons* below).

The transfer function can be linear or nonlinear one. A Neural Network, therefore, is characterized by, a) its pattern of connections between the neurons (called architecture) and, b) the method of determining the weights of the connections (called learning algorithm). The question of determining these characteristics for specific applications is not a straightforward task.

On the other hand, depending on the topologies of their connections, there exist various types of networks. For example, *Feedforward Neural Networks* (FNNs), are the ones with connection topologies admitting no closed paths, in contrast to *Recurrent Networks* which have topologies admitting closed paths. The most popular examples of FNNs are the so-called Multilayer Perceptrons (MLPs).

1.3.2 ANNs as Universal Approximators

To answer the question about the importance of ANNs training we must follow a path beginning with Kolmogorov's "mapping network existence theorem"((Kolmogorov, 1957) as cited in (Ripley, 1993), and (Fausett, 1994)), its refinement by Sprecher (Sprecher, 1965), the casting in neural nets terminology of Sprecher's theorem by Hecht–Nielsen (Hecht-Nielsen, 1987), to the Hornik, Strinchcombe and White (Hornik et al., 1989) theorem stating that neural networks having

"squashing[1]" activation functions, are universal approximators of arbitrary functions. In other words, "standard feedforward networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well with respect to a metric, regardless of the squashing function ψ being continuous or not, regardless of the dimension of the input space, and regardless of the input space environment μ". Where μ is a probability measure and ρ_μ a metric defined on it.

Furthermore, White (White, 1992, chapter 8) gives theoretical results in order for these approximation procedures to be consistent, that is the probability that the approximation error exceeds any specified level ϵ as measured by a metric ρ tends to zero, depending on a trade off between the growth of network complexity q (q=number of hidden units) and the sample size n. He states subsequently that the only errors ultimately made by these consistent procedures are the inherent unavoidable errors arising from any fundamental randomness or fuzziness in the true relations between the example pair. In essence, White describes here, albeit from a different perspective, the well-known bias/variance *dilemma* which we shall discuss further below. However, he notices that although the above results provide *asymptotic* guidelines on the network complexity, they say nothing about determining an adequate complexity in a specific application given the 'examples' (training set) size n, and he quotes: "It is apparent that methods developed by statisticians will prove helpful in this search". For a formal treatment of the above issues we refer to White (White, 1992).

1.3.3 Relations with Statistics

There is an emerging belief that artificial neural networks have strong relationship with statistical inference (Cheng and Titterington, 1994; Ripley and Hjort, 1984; Smith, 1993). A remark of Aharonian cited in (Ripley, 1993, p. 105), comes as

[1] A function $\psi : R \to [0, 1]$ is a squashing function if is non–decreasing, $\lim_{\lambda\to\infty} \psi(\lambda) = 1$ and $\lim_{\lambda\to-\infty} \psi(\lambda) = 0$ (Hornik et al., 1989, definition 2.3). Later, White (White, 1990) extended these results to include the set of functions $\Psi : R \to R$ such that Ψ is bounded, satisfies a Lipschitz condition and is either squashing or l–finite.

natural judgement on the advantages and the usefulness of the former against the latter:

> You will often see someone claim some great breakthrough in using neural network for financial analysis; check to see if the author compares his results to traditional statistical analysis - if not, then (s)he probably has not stumbled onto anything significantly (in the statistical sense) new.

In fact, there exist a number of studies comparing their results with conventional statistical techniques having though contradictory conclusions. Some of them turn in favour of neural networks (e.g. (Weigend et al., 1990), (Weigend et al., 1991) compare them to the TAR model (Tong, 1983; Tong and Lim, 1980) over the sunspot numbers and exchange data, while in (Lachtermacher and Fuller, 1995) a comparison is made with ARMA and ARIMA models over stationary and non-stationary series), and others in favour of traditional statistical techniques (e.g. in (Tang et al., 1991) ANNs are used as alternatives to Box-Jenkins models with unsatisfactory results).

The strong relations and similarities of Artificial Neural Network modeling and statistics have been highlighted by an increasing number of authors (Cheng and Titterington, 1994), (Ripley, 1993), and (White, 1992). Additionally, Levin *et al.* (Levin et al., 1990) present a rigorous account of the relations between learning in artificial neural networks and the field of statistical mechanics, especially the notions of *entropy*, *stochastic complexity* and the *minimum description length principle (MDL)*. Especially concerning nonparametric regression, Refenes (Refenes, 1995) has shown that neural network models can be expressed as nonparametric nonlinear additive regression models. Consequently, he proposed that all the known results about the latter should be exploited, in order to expand our understanding of the former. In the light of this proposition are most of the following notes.

1.3.4 ANN Training as an Optimisation Problem

A really intriguing field for applications of optimization algorithms is the area of neural networks training. In particular the design of training algorithms for the most commonly used type of Feedforward Neural Networks (FNNs) the Multilayer Perceptrons (MLPs).

The challenge in this area of applications arises from the fact that, depending on the type of data available, as we shall see below, it incorporates both the deterministic and stochastic cases of unconstrained optimization. That is, in some cases it can be considered an optimization problem with deterministic noises (which in certain situations can become negligible), while in other cases it is an optimization problem with stronger noises of stochastic nature.

1.3.5 Deterministic vs Stochastic Training

In general the problem of learning in neural networks can be considered either as deterministic or stochastic depending on the definition of the criterion function or more generally the goal to be achieved (Battiti, 1992). If one considers the minimisation of an error function with respect to a particular **fixed** data set, it leads up to a deterministic optimisation problem which is related to memorisation of the specific data. However, when we view the training set as random samples drawn from an unknown distribution, we arrive at a stochastic setting corresponding to generalisation over unseen examples. These two definitions of the training problem are closely related to the duality between interpolation and extrapolation in statistics (see (White, 1992)).

As a consequence a large number of the authors using and comparing deterministic optimisation techniques directly adjusted to neural networks training from the field of numerical non-linear optimisation (some of which we will discuss in the following section) tend to report on the observed error over the training set and the corresponding convergence properties of their algorithms (see for example (Johansson et al., 1992; Denton and Hung, 1996)). In other cases only

a limited exploration of the generalisation capabilities is included concerning a small subset of the example problems used (see for example (Magoulas et al., 1999; Vrahatis et al., 2000b)). Even in these cases a note of caution in (Reed and Marks, 1999, p. 168) warns us that the quantities reported and compared should be comparable, taking into account additional forward and backward steps often required by these algorithms as compared to simpler training algorithms and their variants. In many such cases large initial differences in training speed were either considerably reduced or overturned when an appropriately comparable measure (e.g. CPU time) was considered.

The view adopted in this study is that the issue of generalisation is fundamental prerequisite in the concept of learning, so, although performance on a training set is important when comparing *learning* algorithms for neural networks, monitoring also their behaviour on an unknown data set is crucial for any comparative assessment. As a consequence, we chose in our experiments to measure the evolution of the performance on both the training and an independent test set at two different points in time during the simulation runs. Furthermore, to account for the already discussed sensitivity of neural networks to the initial weights, we perform a number of runs from different starting points and subsequently perform statistical comparisons, including testing of alternative hypotheses on the averaged outcomes of the runs. Finally, we compliment these analyses with alternative measures of performance which shall be discussed in Chapter 6.

1.3.6 Memorisation vs Generalisation

When dealing with the issue of learning we have to address the issues of *memorisation* and *generalisation* (Ripley, 1993, p. 42). This distinction becomes crucial when the models of our interest are characterised by broad generality, in terms of the classes of relationships that can represent, like the neural networks models do. Memorisation, can be considered as the ability to fit the features of the given data optimally with respect to a certain error criterion. Generalisation, on the other hand is, in essence, the ability of performing well for items not in the training

set, or in other words, to perform equally well with respect to both the 'seen' and the 'unseen' cases. To achieve generalisation "it is important to find a suitable compromise between overfitting and underfitting" (Cheng and Titterington, 1994, p. 20). That is, learning the 'details' of a given data set as opposed to extracting only very general characteristics from it.

The relative weight between generalisation and memorisation is problem dependent, though. For classification tasks with fully determined classes, such as the **XOR** or the **parity** problem, the issue of generalisation does not arise at all. Gemman *et al.* in (Geman et al., 1992, pp. 18–33) characterise this case as "degenerate", and further illustrate and discuss the total generalisation error and its bias and variance components as functions of the number of hidden nodes—i.e. the degree of modelling flexibility—of a neural network. They also demonstrate that there exists a characteristic difference with the case of an 'ambiguous', as they call it, classification task due to noise inherent in the available data. In general, for any task which is subject to 'uncertainty', generalisation ability is the most desirable feature, dictating the choice of which method to use. "In forecasting, nobody cares how well a model fits the training data – only the quality of future predictions counts", as very clearly Gershenfeld and Weigend (Gershenfeld and Weigend, 1993, p.19) pointed out. To successfully address the issue of generalisation is not a straightforward task. Bienenstock and Geman (Cheng and Titterington, 1994, Discussion, p. 36) remark:

> Statisticians know that generalisation (good performance on samples not in the training set) depends almost entirely on the extent to which the training set is representative, and/or the structure of the problem happens to accommodate the models used.

In summary, the ability of neural network to generalise depends on:

- the network architecture and size,
- the quality and quantity of the data, and

- the learning algorithm,

An increasing number of studies in neural networks literature are attempting to tackle the above issues in both the theoretical and practical grounds.

1.3.7 Network Architecture and Training Data

To begin with, White in (White, 1990) provides theoretical results for the choice of proper activation functions, number of hidden nodes q_n and bounds for the values of the weights Δ_n, in order to ensure that the sequence of his 'connectionist sieves' $\{\Theta_n\}$, will have a rate of growth to guarantee increasing flexible networks. In addition, these choices are such that $\Theta_n(\psi)$ can be sufficiently 'big', to avoid underfitting, while preventing it from becoming too big too fast, to avoid overfitting. Other theoretical results presented by Baum and Haussler (Baum and Haussler, 1989), address the issue of the network capacity, that is, essentially, the number of classes of functions representable by the network. They also pose the question of the appropriate training set size in order to achieve a specified level of generalisation with a network of fixed size. Their results being implemented in a 'rule of thumb', have been referred to by various authors (Fausett, 1994); (Ripley, 1994). More specifically, they suggest that Baum and Haussler's results imply that for a two–layer network with W number of weights, the number of training examples N required to guarantee a success rate on a set of unknown examples (drawn from the same sample space) of at most ϵ worst than that on the training set, is approximately equal to W/ϵ.

Finally, the quality of the data and its inherent complexity depend on the phenomenon observed and the measurement mechanism. In realistic situations the experimenter normally has not (absolute) control over the noise intrinsic in the data. Hence even when experiments with artificial or simulated data are conducted, it is advisable to emulate these conditions inducing appropriate types of noises. These could be a bounded variate for the case of deterministic inaccuracies—such as round-off errors —an additive random component to simulate measurement

noises, or, finally, a noise variate which is involved in the dynamics of the modelled system. Obviously the two latter cases are the most important to address and they are both explored in our experiments included in Chapter 6.

1.3.8 Learning Algorithms

The learning algorithm is nothing more than an efficient way of determining the weights so that our objective can be reached. Learning is usually divided in *supervised* where the adjustments to weights (model parameters) are made according to the closeness of the network output to a desired target example, and *unsupervised* where no such target examples exist, so that updates are based on other information provided by the training set. In the case of supervised learning the closeness of the network output to target is defined by a suitably chosen error function E.

Learning algorithms are, essentially, another important issue of an application set up which involves the issue of generalisation. They are, as stated above, procedures to enable a neural network reach a near optimal solution (at least including local minima) through minimization of an error function. But, in the case of forecasting, which is of our particular interest, generalisation is the ultimate goal. Most of the existing texts and prominent authors in the field (see for example, (Bishop, 1995; Reed and Marks, 1999; White, 1989; Ripley, 1993) recognise the fundamental importance of generalisation and its assessment as a crucial issue in the evaluation and comparisons of training algorithms. As Ripley (Ripley, 1993, p. 53) insightfully quotes from (Raudys and Jain, 1991, p. 47) "it remains to solve the problem of 'designing fast training algorithms which minimize the true error instead of minimizing the apparent error'. Here the 'true' error refers to the rate on future examples.". Consequently, the performance of a learning procedure, with respect to this goal, passes through the search of suitable evaluation techniques.

1.3.9 The Bias/Variance Dilemma

The bias/variance dilemma is one of the most important. In general, under the regression framework between variables Y and X, it is well known that among all functions of X, the regression $E(Y|X)$ is the best predictor of Y given X in the mean–squared–error sense. Thus, based on sample D of pairs (X_t, Y_t) of realizations of X and Y, the effectiveness of any function $f(X_t; D)$ as predictor of Y_t can be expressed:

$$\begin{aligned} E[(Y_t - f(X_t; D))^2 | X_t, D] &= E[Y_t - E[Y|X])^2 | X_t; D] + \\ & \quad (f(X_t; D) - E[Y|X])^2 \end{aligned} \tag{1.3}$$

While the first part of the Right Hand Side (RHS) of expression (1.3) does not depend on D and f, the second part can serve as a measure of effectiveness of f as predictor of Y. Furthermore, its expectation denotes f's effectiveness as an estimator of $E(Y|X)$ with respect to D:

$$\begin{aligned} E_D[(f(X_t; D) - E[Y|X])^2] &= (E_D[(f(X_t; D) - E[Y|X])^2 + \\ & \quad E_D[(f(X_t; D) - E_D[(f(X_t; D)])^2] \end{aligned} \tag{1.4}$$

This expression can be easily transferred to the neural networks framework, by simply substituting $f(X_t; D)$ by the network output function $f(X_t, W; D)$ where W is the vector of all weight strengths. The first part of the RHS of (1.4) is the bias term, while the second expresses the variance. In the ANN context, as pointed out in (Geman et al., 1992), the contribution of each term to the total error along with the expected trade–off, it is influenced by two distinct factors of the ANNs design. (These factors of the network complexity are expressed by the number of nodes and the time of training.) In (Geman et al., 1992) is illustrated, by certain examples, that a small number of nodes introduce significant bias to the resulting network, which eventually decreases as a result of a larger network. However, increasing the complexity will introduce an increasing variance component by

allowing the network to become too "faithful" to the data. For the case of the duration training, the same mechanism holds for bias and variance respectively.

1.3.10 Applications to Regression and Classification

The theoretical results presented previously establish the potential of neural networks as function approximators. Their importance in this context is that they offer a very powerful and flexible apparatus for representing non-linear mappings which can be applied in a wide range of application areas. If for example this mapping is defined on the conditional expectations of the output given the available date (input variables) — in other words, in terms of an average over a random quantity — the problem belongs in the class of "regression" (Bishop, 1995, p. 5). In this case the outputs represent values of continuous variables. On the other hand, when the task is to assign new inputs to one of a number of discrete classes or categories on the basis of known attributes, the problem belongs to the class of "classification" (Neal, 1996, p. 2). The function to be approximated in the latter case is the conditional probability function of the output to be assigned a particular class label given its attributes vector.

The regression category, broadly speaking, includes applications where the average response of a phenomenon is to be estimated based on observations of its characteristics. For example, in (Thodberg, 1996) the objective is to determine the average fat content of meat based on near infrared spectroscopy measurements. It also includes tasks which involve the prediction of the response of dynamic systems based on time dependent observation of its state, that is time series forecasting. An enormous amount of work has been published up to now concerning applications of artificial neural networks to time series forecasting. These applications extend from weather forecasting and climate research (Hu, 1964; Elsner and Tsonis, 1992) to the sunspot numbers prediction (Nowlan and Hinton, 1992). Moreover, this list can be extended from chaotic time series in the early work of Lapedes and Farber (Lapedes and Farber, 1987a) to a large number of studies concerning short–term prediction of electric load forecasting (among

others (Hwang and Moon, 1991; Lee and Park, 1992). In the field of business and finance, applications include the areas of stock market prediction—beginning with an early attempt of White (White, 1988) to predict IBM stock prices, German stocks prediction (Schöneburg, 1990), or interval forecasting for stock market indeces (Sirlantzis, 1996), to indicatively only mention some—exchange rates prediction (Refenes and Zaidi, 1992; Refenes, 1991; Refenes et al., 1993), bond rating (Garavaglia, 1991; Utans and Moody, 1991), mortgage prediction, etc., covering all possible fields during the last decades.

In our experiments, in order to cover both of the above broad areas of regression tasks, we have included two conventional regression problems with different distributions of noise components as well as two examples of chaotic time series prediction.

On the other hand, the classification class includes two–class (binary) problems as well as multi–class problems with a diverse variety of applications (Schalkoff, 1992). These extend from character recognition (Ghorbani and Bayat, 2000), speech recognition and understanding (Magoulas et al., 1999), radar/sonar signal classification (Hasenjager and Ritter, 1999), and analysis to medical diagnosis (Cho and Chow, 1999) and texture analysis (Vrahatis et al., 2000b), to name but a few.

Furthermore, in (Michie et al., 1994), the variety of the examples used extends from credit risk identification in finance to satellite image analysis and understanding. This text is also a rich source of comparative results about the effectiveness of neural network classifiers with alternative machine intelligence and conventional statistical discriminators. As pointed out by Bishop (Bishop, 1995, p. 6) "many of the key issues which need to be addressed in tackling *pattern recognition problems* are common both to classification and regression". Although the concept of pattern recognition often means different things to different people, if we define pattern recognition as the "science that concerns the description or classification (recognition) of measurements" (Schalkoff, 1992, p. 2), it comes as a natural realisation the fact that artificial neural networks were from the beginning and still

are considered mainly as efficient 'pattern recognisers'.

1.4 Motivation and Objectives

The preceding discussion about the main classes of algorithms currently employed to tackle the problems of noisy optimisation can be summarised in the following remarks which provided the motivation for the work reported in this study.

1. The algorithms that utilise gradient information compare favourably to those that do not since they can achieve faster their objective by exploiting the additional information about the (local) topology of the function in hand.

2. Gradient based algorithms that are fast and locally convergent in the noise-free cases can easily exhibit unstable behaviour in the noisy optimisation task due to the the influence of the noises to the gradient estimators used.

3. The choice of appropriate stepsizes is of paramount importance for gradient based algorithms controlling both their speed and their robustness with respect to disturbances.

4. On one hand, the choice of conservative (i.e. slowly decreasing) stepsizes in order to establish stability renders the corresponding gradient algorithms slow to converge. On the other hand, generalisation of the classical exact line search procedure for the choice of stepsizes (e.g. inexact or even non-monotone versions) offers limited improvements in the presence of strong (stochastic) noises.

5. Alternative algorithms based only on function evaluations are able to overcome in practice the adverse effects of noise in the estimators of interest and have been demonstrably robust for noisy optimisation. However, they need significant numbers of additional function evaluations and, consequently,

are slow to converge, especially when highly accurate approximations to the solutions are required.

A logical conclusion indicated by the above remarks for the design of algorithms able to successfully address the issues arising in noisy optimisation, is that a framework based on **synthesis** mechanism is needed. This should facilitate the integration of desirable characteristics of existing algorithms into a novel apparatus exhibiting qualitatively emerging behaviour different that the mere addition of its constituent parts. It should also be generic with respect to the types of algorithms that can be applied on. Finally, it should allow to adapt the behaviour of the resulting algorithms according to the characteristics of the problem at hand. As it was noted previously, a desirable algorithm should be efficient in the cases where the noise is negligible and robust when the noises are stronger. The SSC framework promises such an algorithm development tool and it is the aim of this study to explore its properties. To this end, the objectives of this research are the following:

1. First, to design and implement a basic new composite algorithm within the SSC framework based on the combination of two existing algorithms with diverse characteristics. One constituent part would be a slow but robust algorithm while the other a fast but only locally convergent scheme. To evaluate experimentally the behaviour of this algorithm on a wide range of non-trivial benchmarking function in the noise-free as well as the noisy case. Furthermore, to identify effective values for the parameters controlling its behaviour under a variety of different conditions.

2. Second, in order to demonstrate the flexibility of the SSC framework, to propose and implement a number of possible extensions and modifications of the basic algorithm aiming to address specific issues arising in realistic applications. Furthermore, to empirically verify their properties via the same set of benchmarking problems.

3. Third, in order to assess the practical applicability of the algorithms developed within the SSC framework, to adapt a representative algorithm for use in neural network learning. The important role of the ANNs' approach in a wide variety of real world applications as well as the intricate nature of the issues involved with their learning process (i.e. the issue of *generalisation*), which have already been pointed out, make them an ideally challenging benchmark for noisy optimisation algorithms. In order, finally, to validate comparative experiments assessing the performance of the proposed algorithm with respect to popular training algorithms for neural networks, to perform a series of statistical analyses evaluating the different aspects involved in the learning process.

1.5 Outline of the Thesis

The remainder of this thesis is organised as follows:

Chapter 2

This chapter is divided in two parts. In the first part, initially, the SSC framework is defined formally for the noise-free and the banded noise cases, followed by theoretical results about the convergence properties and the speed of the algorithms developed within it. Subsequently, the framework is redefined in a more general form to include the case of stochastic noises. A number of theoretical statements are also presented about the convergence characteristics of resulting algorithms. The second part of this chapter includes the formal definitions of the deterministic forms of the 23 classical benchmarking problems which are employed in the following chapters to assess the performance of the proposed algorithms. The chapter concludes by describing the stopping criteria used in this study for the deterministic as well as the stochastic cases.

Chapter 3

The subject of this chapter is the basic algorithm proposed in this work, which was designed using the SSC framework. First, formal definition along with theoretical results about its properties are presented. Then, the algorithm is experimentally evaluated in a deterministic setting followed by a similar evaluation in the presence of stochastic noises. In the latter case, the effects of different levels of noise are also explored. The sensitivity of the algorithm's behaviour with respect to different values of its controlling parameters is examined both in the deterministic and the stochastic cases and suggestions are made about effective parameter ranges for a variety of conditions. Furthermore, the use of a number of different gradient estimators is also studied, including the classical finite difference estimators as well as more recent proposals such as the Simultaneous Perturbation estimators (Spall, 1998). In an additional set of experiments the effectiveness of statistical techniques aiming to reduce the adverse consequences of strong stochastic values is considered. Finally, both in the deterministic and the stochastic setting, the performance of the proposed SSC algorithm is compared with that of well-known algorithms over the same set of testing problems.

Chapter 4

A number of extensions and modifications of the basic algorithm are introduced. The aim is to demonstrate the flexibility of the SSC framework in designing efficient algorithms suitable to address specific application requirements. These algorithms are evaluated both in the deterministic and stochastic cases using the same set of benchmarking problems employed in the previous chapter. Finally, the results obtained are discussed with respect to the particular issues each algorithm was created to address.

Chapter 5

The aim of this chapter is to provide adequate background information on concepts and issues related to neural network learning procedures and their assessment. First, a survey of existing learning algorithms is present. Next, a critical review of comparative studies in the literature is provided aiming to identify efficient algorithms to be used as benchmarks in the comparative experiments reported in the following chapter. Subsequently, an adaptation of the basic SSC algorithm introduced in Chapter 3 along with the existing algorithms used for comparisons are formally defined. The chapter concludes with a discussion of technical issues involved in the experimental evaluation of neural networks learning algorithms.

Chapter 6

In this chapter, a series of experimental comparisons are reported regarding the use of algorithms developed using the SSC framework in neural network learning. The SSC algorithm employed is evaluated in comparison with two alternative training schemes in terms of a variety of criteria including speed of convergence and generalisation ability. These comparisons are performed over a set of synthetic and real world tasks that belong to two main categories, namely regression and classification. Initially, the experimental setup adopted is described and explained. Then, a brief examination of the characteristics of the data sets used is presented, accompanied by a discussion of exploratory observations based on the so-called learning curves produced by the three algorithms in each case. Subsequently, a series of statistical analyses of the results obtained are presented and elucidated. The chapter concludes with a number of final comments about the comparative performance of the proposed SSC-based learning algorithm summarising the findings of the previous analyses.

Chapter 7

In the final chapter, an overview of the findings obtained from the experimental investigations in this study is, initially, presented, leading to general concluding remarks about the efficiency, the robustness and the practical applicability of the algorithms developed using the SSC framework, as well as the flexibility of the framework to facilitate the design of algorithms with a wide variety of characteristics highly desirable in the noisy optimisation setting. Finally, the chapter provides pointers to promising paths for fruitful future research.

Chapter 2

The Supervisor Searcher Co-operation (SSC) Framework: Theoretical and Methodological Issues of the Study

2.1 Formulation

Let $f = F + \varepsilon$, where ε is some form of noise (deterministic or stochastic), as explained in Section (1.2.1) F is a continuous function on R^n and is bounded below. We are interested in finding (local) minimizers of f (that is, of F).
For given $x_0, x_1, ..., x_l$, assume that we have an iterative algorithm, called search engine (SE):

$$x_{k+1} = x_k - se_k(x_k, x_{k-1}, ..., x_{k-l}, k, f),\ k = l, l+1, ...$$

The above notation for se_k emphasise its dependence on k and the values of $\{f(x_{k-i})\}_{i=0}^m$, $\{\nabla f(x_{k-i})\}_{i=0}^m$, $\{H(x_{k-i})\}_{i=0}^m$, etc., where $H(x)$ is the Hessian matrix of f at the point x.

Suppose that $\epsilon = 0$ and that this algorithm is convergent to a local minimizer

of f provided the starting points are very close to the minimizer. To make the algorithm convergent globally, it is classic to introduce into it a line search procedure, monotone or non-monotone, exact or inexact. However as mentioned before, a line search procedure is in general sensitive to the smoothness of the function and to the accuracy of the function value evaluation. Therefore the resulting algorithm, though convergent globally, may not be robust enough to deal with stochastic or nonsmooth optimization problems, which are becoming increasingly important in practical applications.

The essential idea adopted in (Liu et al., 1999a; Liu and Dai, 2001; Sirlantzis and Liu, 2001) is to employ an alternative globally convergent but robust iterative algorithm to supervise and therefore to safeguard the convergence of the SE algorithm. This supervising algorithm will be referred to as the supervisor (SR). Then, one may obtain a globally convergent **and** robust algorithm. Assume that this supervisor algorithm (SR) reads: Given $x_0, x_1, ..., x_l$

$$x_{k+1} = x_k - sr_k(x_k, x_{k-1}, ..., x_{k-l}, k, f),\ k = l, l+1, ...$$

In general a SR algorithm is slower but robust, and a SE algorithm is faster but only locally convergent. Therefore they have to co-operate in order to work efficiently. We proposed in (Liu and Dai, 1999; Liu et al., 1999a) a supervision principle based on the co-operation of the supervisor and the searcher (SSC). According to this principle the supervisor intervenes only when it believes that the performance of the search engine is not satisfactory while the search engine undertakes most of the (solution) searching work. For the resulting algorithm, to a large extend, global convergence may be ensured by the supervisor but the speed is decided by the search engine.

There are various ways to implement the co-operation principle. The following may be seen as one of the simplest:

Assume $f \geq 0$. Given $x_0, x_1, ..., x_l$, define $(k = l, l+1, l+2, ...)$ the following

(SSC) algorithm:

$$x_{k+1} = x_k - sr_k \text{ if } T_k f(x_k - sr_k) \leq f(x_k - se_k),$$

otherwise

$$x_{k+1} = x_k - se_k,$$

where $\{T_k\}$ is a given sequence of nonnegative real numbers.

This new algorithm may exhibit a behaviour significantly different from those of its "parent" algorithms even for problem with $\epsilon = 0$. The algorithm actually switches between its two component algorithms at a not pre-specified frequency. Assume, for example, that the Newton search engine is used and $\epsilon = 0$, then the algorithm needs only one step to find the minimizer of a convex quadratic objective function when $T_k \equiv 1$. This may be very different from the behaviour of the SR algorithm we used. On the other hand, it is not an obvious assumption that the well known n-step convergence property will still hold for the SSC algorithm when the conjugate gradient algorithm is used as search engine.

The sequence $\{T_k\}$ decides the strength of the co-operation. It is clear that the behaviours of a SSC algorithm depend not only on those of the corresponding SR and SE algorithms, but also on the degree of supervision, which is determined by the sequence $\{T_k\}$. For instance, if we take $T_k = 0$ or $T_k = \infty$, there will be no action of the search engine or the supervisor respectively (assuming $f(x_k - r_k) > 0$). Also by setting $se_k = 0$, the algorithm becomes the SR algorithm. Although such degenerate instances present little interest, they do indicate the wide range of algorithms covered by the SSC type algorithms. It seems clear that taking larger T_k will force a SSC algorithm to use more SE iterations, and therefore may increase its overall speed. However, if T_k is too large, the supervision may become too weak, and therefore the resulting SSC algorithm may not be robust enough or even convergent globally. In applications, normally one uses $T_k \equiv T > 0$. If T is smaller, then the supervisor may do most of the work, if T is larger, the search engine may be used all the way. From our experience, $T = 1$ is always the safest

value.

It is also possible to let the algorithm check for a switch only after two or more iterations, to form a multi-step SSC algorithm. Again, this could save much computational work, but weakens the supervision. These issues will be closely examined in following chapters.

Remark 2.1 *Note that as far as minimization is concerned, one can always assume that* $f \geq 0$ *by adding a positive constant to the original function. Or one can use the following SSC algorithm in the general case:*

$$x_{k+1} = x_k - sr_k \quad if\ T_k^{sign(f(x_k - sr_k))} f(x_k - sr_k) \leq f(x_k - se_k),$$

otherwise

$$x_{k+1} = x_k - se_k,$$

where $\{T_k\}$ *is a given sequence of nonnegative real numbers. All the above observations apply to the general case as well.*

There are many possible candidates for SR algorithms. In general they are expected to be simpler and robust with a global convergence property. As for SE algorithms, we may choose from a wide range of fast algorithms like Newton's algorithm, BFGS, and faster gradient methods like the conjugate gradient (CG) method.

2.2 Convergence and Speed in the Deterministic Case

Convergence

In the following we present a relevant global convergence result for $sr_k = t_k g_k$, where t_k is a positive sequence and $g_k = \nabla f(x_k)$, which holds provided $\sum_k t_k = \infty$ and $\prod_0^\infty \max(1, T_k)$ is finite as $k \to \infty$. This result is actually true for general search engines.

For ease of exposition we assume $t_k \to 0$ in the following theorem, though it is not difficult to see that it holds for the case where t_k is very small after k is large enough, as in Theorem 4.1 of (Liu and Dai, 2001).

Theorem 2.1 *(Theorem 3.1 in (Liu and Sirlantzis, 2001b) Let f be twice continuously differentiable and bounded below. Assume that ∇f is Lipschitz with a global Lipschitz constant. Let $\{x_k\}$ be generated by an algorithm based on the SSC framework defined in Section (2.1). Then $\{\sum_0^k t_k|\nabla f(x_k)|^2\}$ is convergent as $k \to \infty$ provided $\prod_0^\infty \max(1, T_k) < \infty$.*

Proof: Assume, for ease of exposition, that $f \geq 0$. It follows from the definition of the algorithms that for any $k \geq 0$

$$f(x_{k+1}) \leq \max(f(x_k - t_k g_k), T_k f(x_k - t_k g_k)) = \max(1, T_k) f(x_k - t_k g_k).$$

Let $P_k = \max(1, T_k)$. Then we have

$$f(x_{k+1}) \leq P_k f(x_k - t_k g_k) = P_k f(x_k) - t_k P_k g_k^T g_k + t_k^2 P_k g_k^T H_k g_k/2,$$

where H_k is the Hessian matrix of f at a point θ_k in the line segment $[x_k, x_{k+1}]$. Therefore there is a $C > 0$ such that

$$\begin{aligned} f(x_{k+1}) &\leq P_k f(x_k) - t_k P_k |g_k|^2 + C t_k^2 P_k |g_k|^2 \qquad (2.1) \\ &\leq P_k P_{k-1} f(x_{k-1} - t_{k-1} g_{k-1}) - t_k P_k |g_k|^2 + C t_k^2 P_k |g_k|^2. \qquad (2.2) \end{aligned}$$

Repeating this procedure we have

$$f(x_{k+1}) \leq P_0^k f(x_0) - \sum_{i=1}^{k} P_i^k (t_i - C t_i^2) |g_i|^2,$$

where $P_i^k = \prod_i^k P_m$. It follows that there is a $k_0 > 0$ such that

$$t_k(1 - C t_k) \geq c' t_k, \; k > k_0$$

where $c' > 0$ is a constant independent of k.

Let us now define $S_k = \sum_{i=k_0+1}^{k} P_i^k t_i |g_i|^2$. It follows that $1 \leq P_i^k \leq P_i^{k+1} \leq P_0^\infty$. Thus

$$0 \leq P_0^k f(x_0) \leq P_0^\infty f(x_0),$$

and

$$0 \leq P_i^k t_i |g_i|^2 \leq P_i^{k+1} t_i |g_i|^2.$$

Hence, $S'_k = P_0^k f(x_0) - \sum_{i=1}^{k_0} P_i^k (t_i - C t_i^2)|g_i|^2$, is bounded above and below and $0 \leq S_k \leq S_{k+1}$. Therefore, $\{\sum_0^k t_k |g_k|^2\}$ is convergent as $k \to \infty$, as S_k is a monotone increasing sequence and f is bounded below.

In particular we know that $x_k \to x^*$ when f is uniformly convex, where x^* is the minimizer of f. The above result is clearly a generalisation of the global convergence Theorem 4.1 in (Liu and Dai, 2001). Note that no assumption was made for any particular search engine.

In practical computation T_k is often fixed to a constant $T > 1$. In the following theorem we show that the speed of the SSC algorithms is as fast as the search engine.

Deterministic Noises

Remark 2.2 *It seems that some convergence results can be similarly established for the case where the noises are deterministic; that is, f and g_k are only some approximation of F and ∇F. For instance, assume ϵ is smaller in function value evaluation, but significant in gradient estimation, e.g. when using a finite difference approximation scheme to estimate ∇F. Then it follows from the proof that if g_k is only an approximation of $\nabla F(x_k)$, the global convergence will still hold provided that*

$$|g_k| \leq C|\nabla F(x_k)|, (g_k, \nabla F(x_k)) \geq c|\nabla F(x_k)|^2,$$

where $c, C > 0$ are independent of k. These conditions are quite light and easy to meet. In fact, g_k is allowed to be far away from $\nabla F(x_k)$, e.g., twice as long as $\nabla F(x_k)$ with a 45 degree angle between them. Note that even when the nature of ϵ is stochastic the above results still hold provided $|\epsilon|$ is bounded, for example when ϵ has some truncated distribution. This indicates strong robustness of the SSC type algorithms and their extensions.

Speed

In many cases, a SSC algorithm is as fast as the search engine. For instance, the following result can be established.

Theorem 2.2 *(Theorem 3.2 in (Liu and Sirlantzis, 2001b)) Let $f > 0$ be continuously differentiable. Let $\{x_k\}$ be generated by an algorithm based on the SSC principle defined in Section 2.1. Assume that Z_k is a matrix such that $se_k = Z_k g_k$, with se_k as defined in Section 2.1. Assume that $\{|Z_k|\}$ is bounded above, and $T_k \geq T > 1$ after k large enough. Let x^* be a local minimizer of f. Then the SSC algorithm is as fast as SE provided $\{x_k\}$ converges to x^*.*

Proof: As $f(x^*) > 0$, $T_k \geq T > 1$ and f is continuous, there is a $r_0 > 0$ such that

$$T_k f(x - t_k \nabla f(x)) > f(y), \ \forall k \geq 0. \tag{2.3}$$

as long as $|x - x^*| < r_0$ and $|y - x^*| < r_0$, since $\{t_k\}$ is bounded. Since $x_k \to x^*$, we can assume that there is a $k_0 > 0$ such that

$$|x_k - x^*| < r_0/2, \ \forall k > k_0.$$

There exists also a $k_1 > k_0$ such that

$$|Z_k g_k| < r_0/2, \ \forall k > k_1,$$

since $|\nabla f(x_k) - \nabla f(x^*)| = |g_k| \to 0$ as $k \to \infty$. Hence

$$|(x_k - Z_k g_k) - x^*| \leq |x_k - x^*| + |-Z_k g_k| < r_0, \ \forall k > k_1.$$

Therefore we always have

$$T_k f(x_k - t_k \nabla f(x_k)) > f(x_k - Z_k g_k), \ \forall k > k_1. \tag{2.4}$$

So, the SSC algorithm will always take the SE stepsize, as determined by the SSC switching rule, after $k > k_1$.

Although, we assumed that $t_k \to 0$, again all results hold for the case where t_k is very small after k is large enough.

The above theoretical results show that both "parent" algorithms (i.e. the supervisor and the search engine) play an important role in the overall performance of the resulting SSC algorithm. While the supervisor guarantees global convergence the searcher contributes an improved convergence rate.

2.3 Theoretical Analysis under Stochastic Noises

Convergence theory of algorithms for stochastic optimization problems has been one of the focus points of modern stochastic optimization theory, and is in general much more complicated to establish than for the deterministic problems. It seems to be relatively less involved to establish some stochastic convergence results for an algorithm if the gradient estimators used are assumed to be very accurate, see for example (Yan and Mukai, 1993). However, such an assumption is computationally very expensive to satisfy in real life problems, where the gradient estimators used are usually based on some kind of approximation procedure, like the finite difference estimation, see (Fu, 1994) for a number of examples. Consequently, more sophisticated theoretical tools, such as the Martingale theory, are needed in dealing with stochastic convergence.

The two most well-known and classical examples are the algorithm introduced

by Robbins and Monro (Robbins and Monro, 1951) and the one proposed by Kiefer and Wolfowitz (Kiefer and Wolfowitz, 1952). The first algorithm aimed to approximate the point where a regression function assumes a given value. In (Robbins and Monro, 1951) mean square convergence to the root has been proved, under certain conditions. Later Wolfowitz (Wolfowitz, 1952) demonstrated that convergence in probability still holds under much weaker than the original conditions and Blum (Blum, 1954) showed that convergence with probability 1 can be proved for this algorithm under even weaker assumptions. The second algorithm was proposed in order to approximate the point where the maximum of a regression function occurs. Kiefer and Wolfowitz proved convergence in probability of their procedure under certain conditions and Blum (Blum, 1954) extended these results to convergence with probability 1 while weakening the original assumptions. Subsequently, in (Dvoretzky, 1956) and (Wolfowitz, 1958) it was proved that under significantly weaker conditions both almost sure convergence and convergence in mean square occurs for the two classical schemes. The rate of convergence as well as the asymptotic distributions of the iterates have been studied and normality of the errors in the limit has been demonstrated for the two classical algorithms.

Finally, among the more general schemes proposed through the years notable is the one by Dvoretzky (Dvoretzky, 1956) which includes the above classical algorithms as well as numerous others as special cases. For this procedure both almost sure convergence and mean square convergence have been proved under considerably mild conditions, see (Dvoretzky, 1956). Then more systematic investigations have been carried out, and summarized, for example, in (Chen and Zhu, 1996), (Kushner and Clark, 1978), (Kushner and Yin, 1997), and (Ljung et al., 1992). Also see, (Feng et al., 2000) and (Tang et al., 1999) for some new developments.

In the literature the theoretical analysis of the properties of gradient algorithms in the presence of stochastic errors and, in many cases, non-convexity of the objective function, is mainly based on two types of methods:

(1) Methods that involve a deterministic or stochastic descent argument, which,

usually, employ either a Lyapunov function or a supermartingale convergence theorem. It worth noting at this point that the exploration of the theoretical properties of the SSC algorithms presented in the present study is based similarly on the martingale convergence theory. The majority of relevant results assume that f is bounded below, while in some cases impose also a boundedness requirement on the sequence of iterates x_k, or finally, show only that $\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0$. For example, in the analysis presented in (Luo and Tseng, 1994) for the incremental gradient method (which can be related to the "Backpropagation" training in neural networks), it is shown that $\lim_{k\to\infty} ||\nabla f(x_k)|| = 0$, but $f(x)$ is assumed bounded below. However, in (Bertsekas and Tsitsiklis, 2000) it is showed that either $f(x_k) \to -\infty$, or $f(x_k)$ converges to a finite value and $\lim_{k\to\infty} ||\nabla f(x_k)|| = 0$ while removing any type of boundedness assumption on f and x_k.

(2) Methods based on the Ordinary Differential Equations (ODE) analysis (Ljung, 1977; Kushner and Clark, 1978; Benveniste et al., 1990; Kushner and Yin, 1997), which track the evolution of the algorithm using trajectories of a differential equation $dx/dk = h(x)$. The corresponding ODE, for instance, for the stochastic steepest descent method $x_{k+1} = x_k - t_k(\nabla f(x_k) - \epsilon_k)$, is $dx/dk = -\nabla f(x)$. In this case assumptions are, usually, made to ensure that the average direction of update $h(x)$ is a well-defined function of the current iterate x (Bertsekas and Tsitsiklis, 2000). Therefore this technique cannot be applied to gradient methods involving diagonal scaling depending on the past history of the algorithm. In such cases differential inclusions should be used rather than differential equations. As a further example, an asynchronous gradient iteration can be considered which normally updates a single component at a time. The average direction of update $h(x)$ is not well-defined in this case, unless specific assumptions are made (Borkar, 1998).

One of the key conditions in establishing convergence for the classical Stochastic Approximation algorithms is, as we have already mentioned, that the step sizes used tend to zero (or become sufficiently small) as the iterations proceed.

Unfortunately, this condition cannot be met by our algorithm. Indeed, our extensive numerical observations suggest that allowing jumps in step lengths is one of the main sources of its improved efficiency, and thus it cannot be alleviated. An important observation used in establishing our convergence theory is that in some stochastic sense these larger step lengths (proposed by the search engine) will be accepted (thus the jumps will happen) only when the objective function assumes lower values by using these step sizes than by using the diminishing pre-assigned step sizes t_k—that is, only when some improvements in reducing the objective function values over the supervisor (SR) can be made. Therefore, the efficiency of the algorithms should not be lower than that of their supervisor. This critical observation has to be integrated into the existing classical techniques in order to establish convergence for our algorithm, and hence this will be one of the main focal points of the theoretical analysis presented in the remaining of this section.

2.3.1 Outline of Convergence Theory in the Stochastic Case

In the following we outline some general theoretical results for the SSC algorithms in the case the function and/or gradient estimators are corrupted by stochastic noises. We sketch and comment on the line of reasoning that leads to a convergence theory for this type of algorithms, for which a rigorous exposition can be found in (Sirlantzis et al., 2001b). Details and proofs of the relevant theorems are included in Appendix B.

To this end, we first have to reformulate the problem and the corresponding definitions for the SSC algorithm.

Formulation

Assume that $\{\xi_k\}_{\{k\geq 1\}}$, $\{\eta_k\}_{\{k\geq 1\}}$ are independent random sequences of R^n vectors and $\{\zeta_k\}_{\{k\geq 1\}}$, $\{\chi_k\}_{\{k\geq 1\}}$ are independent sequences of real-valued random variables, all defined on space $(\Omega, P, \mathcal{F})$ with $E(||\nu_k||^2) < \infty$ for $\nu_k = \xi_k, \eta_k$ and

$E(\nu_k')^2 < \infty$ for $\nu_k' = \zeta_k, \chi_k$. Furthermore $\{\nu_{k,i}\}$ —where $\nu_{k,i}$ denotes the ith component of ν_k— and $\{\nu_k'\}$ are independent for $k \geq 1$ and $i = 1, \ldots, d$. Denote sub-σ-algebra $\mathcal{F}_l = \sigma(\{\xi_k\}_{\{k \leq l\}}, \{\eta_k\}_{\{k \leq l\}}, \{\zeta_k\}_{\{k \leq l\}}, \{\chi_k\}_{\{k \leq l\}})$ for $l = 1, 2, \ldots$.

Assume that $F \in C^2(R^n)$. Let

$$\{t_k\}_{\{k \geq 0\}}, \qquad \{\beta_k\}_{\{k \geq 0\}}$$

be two positive sequences. Furthermore we sometimes assume that t_k satisfies the following conditions:

$$\begin{array}{ll} (I) & \sum_k t_k = \infty; \\ (II) & \sum_k t_k^2 < \infty \end{array}$$

For a given $T > 0$ and a random variable x_0, define, for $k = 0, 1, ...$

$$x_{k+1} = \begin{cases} z_{k+1} & \text{if } T[F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + \zeta_{k+1}] \\ & \quad \leq [F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) + \chi_{k+1}] \\ y_{k+1} & \text{otherwise} \end{cases} \tag{2.5}$$

where

$$\begin{cases} z_{k+1} &= x_k - t_k(\nabla F(x_k) + \xi_{k+1}) \\ y_{k+1} &= x_k - \beta_k(\nabla F(x_k) + \eta_{k+1}) \end{cases}$$

Thus as $F(z_{k+1}) + \zeta_{k+1}, F(y_{k+1}) + \chi_{k+1}, \nabla F(x_k) + \xi_{k+1}$, and $\nabla F(x_k) + \eta_{k+1}$ are estimators of $F(z_{k+1}), F(y_{k+1}), \nabla F(x_k)$, it is clear that $\{x_k\}$ is generated by a SSC algorithm with a given x_0. The SSC-SABB, an algorithm which provides the major example in our experimental investigations of the SSC framework, is a particular case of this class of algorithms, and will be formally introduced in Chapter 3.

Sketch of Convergence Results

Let us first define

$$S = \{x : \|\nabla F(x)\| = 0\}$$

and dis$(\cdot,\cdot)$ an appropriately chosen measure of the distance from a point in R^n to a set, and assume that the Hessian H of F is bounded over $x \in R^n$. Then we can have, for instance, a basic convergence result, such as the following theorem, for the sequence $\{x_k\}$.

Theorem 2.3 *(Theorem 4.3 in (Sirlantzis et al., 2001b)) Suppose that* $\{\xi_k, \mathcal{F}_k\}$ $(k \geq 1)$ *is a sequence of martingale differences and* $T < 1$. *Assume that the conditions (I) and (II) hold. Then if either* χ_k, ζ_k $(k \geq 1)$ *are bounded, or* $\chi_k \sim N(0, \gamma(k)), \zeta_k \sim N(0, \theta(k))$ *for* $\gamma(k) > 0, \theta(k) > 0$ *such that the following condition (III) is satisfied:*

$$\gamma(k) \leq [o(C) + (1-T)C]^2/[8\log(k+1)]$$

and

$$\theta(k) \leq [o(C) + (1-T)C]^2/[8\log(k+1)],$$

then we have for $k \geq 1$

$$\lim_{k\to\infty} F(x_k)$$

exist a.s. and

$$P(\liminf_{k\to\infty} \text{dis}(x_k, S) = 0) = 1,$$

provided C *is large enough.*

The above result requires that C is large enough, but it is not always possible in practical problems. In real computation, one may add a positive constant to the objective function, as we have done in the following sections. This strategy always works provided that the objective functions have lower bounds. Clearly the larger of the added constant, more likely that the algorithms use the supervisors's step size, thus more stable of the algorithms. However, adding a too large constant may affect efficiency of the algorithms - they may virtually always use the supervisors and then in fact become a Stochastic Approximation algorithm. Furthermore it was found through numerical experiments in (Sirlantzis et al., 2001b) that the

performance of the algorithms is not very sensitive to the value of the added constant.

Let us make a few comments on the condition (III) in Theorem 2.3. Note that the requirement that $\gamma(n) \leq [o(C) + (1-T)C]^2/[8\log(k+1)]$, $\theta(k) \leq [o(C) + (1-T)C]^2/[8\log(k+1)]$ is quite weak. In practical applications, one could take the average of $c(T,C)\log(k+1)^{1/2}$ samples of the function values to meet this condition, where c can be made smaller by taking a large C for a fixed $T < 1$. As computation normally stops after, say 10^9 iterations, it is usually safe to simply assume that $\chi_k, \zeta_k \sim N(0,1)$. Alternatively one could take the average of $O(\log(n+1)^{1/2})$ samples of the function values to meet this condition, assuming that $\{\gamma(k)\}, \{\theta(k)\}$ are bounded.

In practical applications we often use

$$\frac{1}{c_k}\left(\frac{F(x_k + c_k e_{k,i}) - F(x_k - c_k e_{k,i})}{2} + \frac{\nu_{k+1,i}^{(1)} - \nu_{k+1,i}^{(2)}}{2}\right) \tag{2.6}$$

as a gradient estimator to replace

$$\nabla F(x_k)_i + \xi_{k+1,i}$$

where ν_k, $\nu_k^{(j)}, j = 1,2$, $k \geq 1$ are i.i.d. random vector variables (e.g. $\nu_k = \chi_k, \zeta_k$ and $\nu_k^{(j)} = \xi_k^{(j)}, \eta_k^{(j)}$), $\nabla F(x_k)_i$, $\nu_{k,i}$ and $\nu_{k,i}^{(j)}$ denote the ith component of ν_k and $\nu_k^{(j)}$, and $\nabla F(x_k)$ respectively, $\nu_{k,i}$, $\nu_{k,i}^{(j)}$ for $i = 1, \ldots, d$ are also i.i.d., $e_{k,i}$ is the standard unit vector across the ith coordinate, and $c_k > 0$.

If we further assume that $F \in C^3(R^n)$ and both its second and third order derivatives are bounded functions in R^n, we have the following results.

Theorem 2.4 *(Theorem 4.6 in (Sirlantzis et al., 2001b)) Suppose that the above central difference gradient estimator is used, condition (I) holds and $T < 1$. Assume, also, that*

$$\sum_1^\infty t_k c_k^2 < \infty, \sum_1^\infty t_k^2 c_k^{-2} < \infty.$$

Then, if either χ_k, ζ_k $(k \geq 1)$ are bounded, or $\chi_k \sim N(0, \gamma(k)), \zeta_k \sim N(0, \theta(k))$

for $\gamma(k) > 0, \theta(k) > 0$, *so that condition (III) holds as well,*

$$\lim_{k\to\infty} F(x_k)$$

exist a.s., and

$$P(\liminf_{k\to\infty} \text{dis}(x_k, S) = 0) = 1,$$

provided C *is large enough.*

As we shall see in the following chapters the above convergence theory holds directly for the basic types of algorithms developed within the Supervisor Searcher Co-operation framework, as well as for some of the extensions introduced in this study. While, on the other hand, no such claim can be made for other extensions, for which, nevertheless, shall be demonstrated to have rather desirable characteristics. For the latter similar results can not be established without specifically exploring first the properties of the approximation to the function values we are using. Comments will be made on that during the development of the relevant formulations.

2.4 The Test Problems

We used up to 23 test problems in a series of experiments in order to empirically assess the properties of the algorithms developed within the Supervisor and Searcher Co-operation framework defined the previous section.

Problems 1 to 18 are drawn from (More et al., 1981). They are well known deterministic benchmarking problems and their stochastic versions are certainly non-trivial. For example, most of them are not convex. The objective function for these unconstrained optimisation problems is obtained by:

$$F(x) = \sum_{i=1}^{m} f_i^2(x), \tag{2.7}$$

where f_i are the function components defined below.

Problems 19 to 20 are stated in (Raydan, 1997). As mentioned there the Hessian of the former at x^* is the identity matrix, while the Hessian of the latter has n distinct eigenvalues, where n is the dimensionality of the problem. We adopt the initial values used in both the above works.

Problems 21–23 are a one-dimensional fourth order polynomial, a multi-dimensional second order polynomial, and a one-dimensional quadratic, which represent "well-behaved" cases.

For all the functions the format adopted for the presentation is as follows:
Name of the function (function index in the relevant reference)
(a) Dimensionality of the function domain
(b) Function or function component definition
(c) Starting Point x_0
(d) Minimum function value and/or the point x^* this value occurs

1. *Helical valley function (7)*

   (a) $n = 3, \quad m = 3$

   (b) $f_1(x) = 10[x_3 - 10\theta(x_1, x_2)]$
   $f_2(x) = 10[(x_1^2 + x_2^2)^{1/2} - 1]$
   $f_3(x) = x_3$
   where

$$\theta(x_1, x_2) = \begin{cases} \frac{1}{2\pi} \arctan\left(\frac{x_2}{x_1}\right) & \text{if } x_1 > 0, \\ \\ \frac{1}{2\pi} \arctan\left(\frac{x_2}{x_1}\right) + 0.5 & \text{if } x_1 < 0. \end{cases}$$

   (c) $x_0 = (-1, 0, 0)$

   (d) $F = 0$ at $(1, 0, 0)$

2. *Biggs EXP6 function (18)*

   (a) $n = 6, \quad m \geq n$ variable

(b) $f_i(x) = x_3 \exp[-t_i x_1] - x_4 \exp[-t_i x_2] + x_6 \exp[-t_i x_5] - y_i$
where $t_i = (0.1)i$
and $y_i = \exp[-t_i] - 5\exp[-10t_i] + 3\exp[-4t_i]$

(c) $x_0 = (1, 2, 1, 1, 1, 1)$

(d) $F = 5.65565\ldots 10^{-3}$ if $m = 13$, which is the case we used, or $F = 0$ at $(1, 10, 1, 5, 4, 3)$

3. *Gaussian function (9)*

(a) $n = 3, \quad m = 15$

(b) $f_i(x) = x_1 \exp\left[\dfrac{-x_2(t_i - x_3)^2}{2}\right] - y_i$
where $t_i = (8 - i)/2$ and

i	y
1,15	0.0009
2,14	0.0044
3,13	0.0175
4,12	0.0540
5,11	0.1295
6,10	0.2420
7,9	0.3521
8	0.3989

(c) $x_0 = (0.4, 1, 0)$

(d) $F = 1.12793\ldots 10^{-8}$

4. *Powell badly scaled function (3)*

(a) $n = 2, \quad m = 2$

(b) $f_1(x) = 10^4 x_1 x_2 - 1$
$f_2(x) = \exp[-x_1] + \exp[-x_2] - 1.0001$

(c) $x_0 = (0, 1)$

(d) $F = 0$ at $(1.098\ldots 10^{-5}, 9.106\ldots)$

5. *Box three-dimensional function(12)*

   (a) $n = 3$, $m \geq n$ variable; we used $m = 10$

   (b) $f_i(x) = \exp[-t_i x_1] - \exp[-t_i x_2] - x_3(\exp[-t_i] - \exp[-10t_i])$
   where $t_i = (0.1)i$

   (c) $x_0 = (0, 10, 20)$

   (d) $F = 0$ at $(1, 10, 1)(10, 1, -1)$
   and wherever $x_1 = x_2$ and $x_3 = 0$

6. *Variably dimensioned function (25)*

   (a) n variable, $m = n + 2$; we used $n = 6$

   (b) $f_i(x) = x_i - 1, \quad i = 1, \ldots, n$
   $$f_{n+1}(x) = \sum_{j=1}^{n} j(x_j - 1)$$
   $$f_{n+2}(x) = \left(\sum_{j=1}^{n} j(x_j - 1)\right)^2$$

   (c) $x_0 = (\xi_j)$ where $\xi_j = 1 - (j/n)$

   (d) $F = 0$ at $(1, \ldots, 1)$

7. *Watson function (20)*

   (a) $2 \leq n \leq 31$, $m = 31$; we used $n = 9$

   (b) $$f_i(x) = \sum_{j=2}^{n}(j-1)x_j t_i^{j-2} - \left(\sum_{j=1}^{n} x_j t_i^{j-1}\right)^2 - 1$$
   where $t_i = i/29$, $1 \leq i \leq 29$
   $$f_{30}(x) = x_1, \quad f_{31}(x) = x_2 - x_1^2 - 1$$

   (c) $x_0 = (0, \ldots, 0)$

   (d) $F = 2.28767\ldots 10^{-3}$ if $n = 6$
   $F = 1.39976\ldots 10^{-6}$ if $n = 9$
   $F = 4.72238\ldots 10_{-10}$if $n = 12$

8. *Penalty function I (23)*

(a) n variable, $m = n + 1$; we used $n = 8$

(b) $f_i(x) = \alpha^{1/2}(x_i - 1), \quad 1 \le i \le n$

$$f_{n+1}(x) = \left(\sum_{j=1}^{n} x_j^2\right) - \frac{1}{4}$$

where $\alpha = 10^{-5}$

(c) $x_0 = (\xi_j)$ where $\xi_j = j$

(d) $F = 2.24997\ldots 10^{-5}$ if $n = 4$

$F = 5.42150\ldots 10^{-5}$ if $n = 8$

$F = 7.08765\ldots 10^{-5}$ if $n = 10$

9. *Penalty function II (24)*

(a) n variable, $m = 2n$; we used $n = 3$

(b) $f_1(x) = x_1 - 0.2$

$$f_i(x) = \alpha^{1/2}\left(\exp\left[\frac{x_i}{10}\right] + \exp\left[\frac{x_{i-1}}{10}\right] - y_i\right), \quad 2 \le i \le n$$

$$f_i(x) = \alpha^{1/2}\left(\exp\left[\frac{x_{i-n+1}}{10}\right] + \exp\left[\frac{-1}{10}\right] - y_i\right), \quad n < i < 2n$$

$$f_{2n}(x) = \left(\sum_{j=1}^{n}(n - j + 1)x_j^2\right) - 1$$

where $\alpha = 10^{-5}$ and $y_i = \exp\left[\frac{i}{10}\right] + \exp\left[\frac{i-1}{10}\right]$

(c) $x_0 = (\frac{1}{2}, \ldots, \frac{1}{2})$

(d) $F = 3.19940\ldots 10^{-6}$ if $n = 3$

$F = 9.37629\ldots 10^{-6}$ if $n = 4$

$F = 2.93660\ldots 10^{-4}$ if $n = 10$

10. *Brown badly scaled function (4)*

(a) $n = 2,\ m = 3$

(b) $f_1(x) = x_1 - 10^6$

$f_2(x) = x_2 - 2 \cdot 10^{-6}$

$f_3(x) = x_1 x_2 - 2$

(c) $x_0 = (1, 1)$

(d) $F = 0$ at $(10^6, 2 \cdot 10^{-6})$

11. *Brown and Dennis function (16)*

(a) $n = 4, \quad m \geq n$ variable; we used $m = 20$

(b) $f_i(x) = (x_1 + t_i x_2 - \exp[t_i])^2 + (x_3 + x_4 \sin(t_i) - \cos(t_i)))^2$
where $t_i = i/5$

(c) $x_0 = (25, 5, -5, -1)$

(d) $F = 85822.2\ldots$ if $m = 20$

12. *Gulf research and development function (11)*

(a) $n = 3, \quad n \leq m \leq 100$; we used $m = 99$

(b) $f_i(x) = \exp\left[-\frac{|y_i\, m\, i\, x_2|^{x_3}}{x_1}\right] - t_i$
where $t_i = i/100$
and $y_i = 25 + (-50 \ln(t_i))^{2/3}$

(c) $x_0 = (5, 2.5, 0.15)$

(d) $F = 0$ at $(50, 25, 1.5)$

13. *Trigonometric function (26)*

(a) n variable, $\quad m = n$; we used $n = 20$

(b) $f_i(x) = n - \sum_{j=1}^{n} \cos x_j + i(1 - \cos x_i) - \sin x_i$

(c) $x_0 = (1/n, \ldots, 1/n)$

(d) $F = 0$

14. *Extended Rosenbrock function (21)*

(a) n variable but even, $\quad m = n$; we used $n = 14$

(b) $f_{2i-1}(x) = 10(x_{2i} - x_{i-1}^2)$
$f_{2i}(x) = 1 - x_{2i-1}$

(c) $x_0 = (\xi_j)$ where $\xi_{2j-1} = -1.2,\ \xi_{2j} = 1$

(d) $F = 0$ at $(1, \ldots, 1)$

15. *Extended Powell singular function (22)*

(a) n variable but a multiple of 4, $m = n$; we used $n = 16$

(b) $f_{4i-3}(x) = x_{4i-3} + 10x_{4i-2}$

$f_{4i-2}(x) = 5^{1/2}(x_{4i-1} - x_{4i})$

$f_{4i-1}(x) = (x_{4i-2} - 2x_{4i-1})^2$

$f_{4i}(x) = 10^{1/2}(x_{4i-3} - x_{4i})^2$

(c) $x_0 = (\xi_j)$

where $\xi_{4j-3} = 3$, $\xi_{4j-2} = -1$, $\xi_{4j-1} = 0$, $\xi_{4j} = 1$

(d) $F = 0$ at the origin

16. *Beale function (5)*

(a) $n = 2$, $m = 3$

(b) $f_i(x) = y_i - x_1(1 - x_2^i)$,

where $y_1 = 1.5$, $y_2 = 2.25$, $y_3 = 2.625$

(c) $x_0 = (1, 1)$

(d) $F = 0$ at $(3, 0.5)$

17. *Wood function (14)*

(a) $n = 4, m = 6$

(b) $f_1(x) = 10(x_2 - x_1^2)$

$f_2(x) = 1 - x_1$

$f_3(x) = (90)^{1/2}(x_4 - x_3^2)$

$f_4(x) = 1 - x_3$

$f_5(x) = (10)^{1/2}(x_2 + x_4 - 2)$

$f_6(x) = (10)^{-1/2}(x_2 - x_4)$

(c) $x_0 = (-3, -1, -3, -1)$

(d) $F = 0$ at $(1, 1, 1, 1)$

18. *Chebyquad function (35)*

    (a) n variable, $m \geq n$; we used $n = m = 8$

    (b) $f_i(x) = \frac{1}{n}\sum_{j=1}^{n} T_i(x_j) - \int_0^1 T_i(x)dx$

    where T_i is the ith Chebyshev polynomial shifted to the interval $[0,1]$

    and hence,

    $\int_0^1 T_i(x)dx = 0$ for i odd,

    $\int_0^1 T_i(x)dx = \frac{-1}{(i^2-1)}$ for i even

    (c) $x_0 = \xi_j$ where $\xi_j = j/(n+1)$

    (d) $F = 0$ for $m = n$, $1 \leq n \leq 7$, and $n = 9$

    $F = 3.51687\ldots 10^{-3}$ for $m = n = 8$

    $F = 6.50395\ldots 10^{-3}$ for $m = n = 10$

19. *Strictly Convex (1)*

    (a) n variable

    (b) $F(x) = \sum_{i=1}^{n}(e^{x_i} - x_i)$

    (c) $x_0 = (\frac{1}{n}, \ldots, \frac{i}{n}, \ldots, 1)$

    (d) $x^* = (0, \ldots, 0)$

20. *Strictly Convex (2)*

    (a) n variable

    (b) $F(x) = \sum_{i=1}^{n} \frac{i}{10}(e^{x_i} - x_i)$

    (c) $x_0 = (1, 1, \ldots, 1)$

    (d) $x^* = (0, 0, \ldots, 0)$

21. *One-dimensional 4th order Polynomial*

    (a) $n = 1$

(b) $F(x) = x^4 + x^2$

(c) $x_0 = 10$

(d) $x^* = 0$

22. *Multidimensional 2nd order Polynomial*

(a) $n = 50$

(b) $F(x) = \sum_{i=1}^{n} ix_i^2 + \sum_{i=1}^{n-1} x_i x_{i+1}$

(c) $x_0 = (1, 1, \cdots, 1)$

(d) $x^* = (0, 0, \cdots, 0)$

23. *One-dimensional Quadratic*

(a) $n = 1$

(b) $F(x) = 100 + x^2$

(c) $x_0 = 100$

(d) $x^* = 0$

The last function (Nr. 23) was, mainly, selected because, due its simplicity, facilitates better interpretability of the corresponding results and their comparisons. So, whenever all the compared versions of the tested algorithms produced identical results, this function is not present in the corresponding tables.

2.4.1 Deterministic Tests

In the tables presented in the following chapters for the deterministic optimisation cases we report the **No. of iterations/No. of function evaluations/No. of gradient evaluations**. The maximum number of function evaluations is set to 9999, unless otherwise stated in the corresponding tables.

2.4.2 Stochastic Tests

In these test we assume that we are not able to obtain the exact value of the function. Hence, we use

$$f(x) = F(x) + \varepsilon$$

as an estimate of the function value at x, with

$$\varepsilon = a\varepsilon'$$

being the stochastic noise, with a a constant we shall call the "noise scale factor" in the following, and ε' a standard normal random variable (that is, $\varepsilon' \sim N(0,1)$). Consequently, we use a number of different gradient estimators in our numerical experiments, the specific form of which we will define in the subsequent sections. It is clear that the noise can be dominant in the gradient estimation near a solution where $\nabla F = 0$. In fact, it is not difficult to observe that finding a solution to these noisy optimization problems is anything but a trivial task. In the following it is assumed that $a = 0.1$, unless if defined explicitly otherwise.

We run every experiment 10 times for every algorithm and each one of the problems, using different random seeds in the noise generator, and in the tables included in the following chapters referring to the stochastic case we report **the number of successful runs**. We have tried also three independent batches of ten runs and very similar outcomes have been observed. The maximum number of function evaluations is set to 9999.

In most of the results tables we report also, in parentheses, the Mean of the number of iterations needed to satisfy the stoping criteria, and the Mean Absolute Deviation (MAD) from the Mean, over the 10 independent runs.

Note that without any specific modification for the reduction of the number of function evaluations, an SSC-based algorithm will require two function evaluations at each iteration. However, it will need only one gradient estimation at each step, which is well known to be the most expensive part of the computational load.

Noisy versions of the problems defined above have been used in a variety of studies of optimisation methods (Anderson and Ferris, 2001; Barton and Ivey, 1996; Humphrey and Wilson, 1998; Elster and Neumaier, 1995; Elster and Neumaier, 1997; Humphrey and Wilson, 1998; Neddermeijer et al., 2000). However, many of them are using only a small subset of the problems. For instance, in (Anderson and Ferris, 2001) and (Neddermeijer et al., 2000) only two of the above functions are included, while in (Humphrey and Wilson, 1998) only five of them are tested. Furthermore, noises are introduced in a variety of different ways and a wide range of different stopping criteria are used. For example in (Elster and Neumaier, 1995; Elster and Neumaier, 1997) the noisy function is defined as $f(x) = F(x)(1 + e)$, where F is the exact underlying mathematical model and e is the noise. In this case the effects of noise are diminished as we approach to the minimum, unlike the present study where the noise is additive and has the same effects in the whole range of the function values. In (Barton and Ivey, 1996), on the other hand, a truncated Gaussian distribution is used, which also imposes different conditions than the unbounded Gaussian used in our experiments. Additionally, an example of the use of an alternative stopping rule is given in (Humphrey and Wilson, 1998) where convergence is checked in terms of the percentage of improvement of the current distance from the solution with respect to the distance corresponding to the beginning of the iterations. Unfortunately, this diversity in parameters and criteria used in the above works renders comparisons of the results reported there as well as in the present study impossible.

2.5 Convergence criteria

2.5.1 Deterministic Case

For the deterministic version of the benchmarking problems presented above we used the following stopping rule:

$$|\nabla f(x_k)| \leq 10^{-6}. \tag{2.8}$$

2.5.2 Stochastic Case

When the noisy version of the benchmarking problems is employed we use the following stopping rules:

$$|\hat{x}_k - x^*| < eps, \tag{2.9}$$

if x^* is known, or otherwise

$$|(\hat{f}_k) - F(x^*)| < eps^2, \tag{2.10}$$

where $eps = 0.01$, x^* is the minimizer and

$$\hat{x}_k = \sum_{i=0}^{19} x_{k-i}/20,$$

$$\hat{f}_k = \sum_{i=0}^{19} f(x_{k-i})/20.$$

Note that these convergence criteria are rather hard to meet when strong noises are present. More so, when finite difference or other similar approximation to the gradients are used which tend to amplify the effects of the noise in the objective function.

2.6 Benchmarking Algorithms

In order to establish a better understanding of the behaviour of the algorithms proposed in this study in comparison to known algorithms used in the field of unconstrained optimisation we employed the following two as benchmarks:

The first is the classical Stochastic Approximation algorithm (henceforth denoted as SA) defined as:

Given x_0

$$x_{k+1} = x_k - t_k g_k, \; k = 0, 1, 2, ...,$$

where g_k is, as usual, an estimator of the gradient ∇F at x_k. It is probably

the most popular algorithm for stochastic optimization problems. It is, also, well-known that, although it is globally convergent in general, it is too slow to achieve high numerical accuracy. One of the main difficulties in practice is to select the appropriate sequences $\{t_k\}$ for a particular problem (see Section 1.2.6 for an introduction and (Kushner and Clark, 1978; Kushner and Yin, 1997) for a comprehensive discussion). The sequence of step sizes $\{t_k\}$ (*gain sequence*), we used in the majority of the cases were:

$$t_k = 1/k, \text{ or } t_k = 1/k^{0.5}.$$

Whenever a constant step size was used in our experiments it is explicitly mentioned in the corresponding tables of results. The choice of this algorithm as a benchmark for our comparisons, despite its drawbacks, was based on two reasons. First, due to its popularity, it is included in a large number of comparative studies, so it provides a common base for performance assessments. Second, as it will be shown in the next chapter, it constitutes the Supervisor algorithm (SR) of the basic SSC-based algorithm examined in this study. Hence, relevant comparisons can reveal the changes in the qualitative characteristics of the algorithms caused by the switching mechanism incorporated in the schemes designed according to the SSC framework.

The second benchmark is an algorithm introduced in (Raydan, 1997) under the name "Global Barzilai and Borwein Algorithm" (GBB), because it uses a step size originally proposed by (Barzilai and Borwein, 1988). It can be defined as follows (where the superscript T denotes vector transposition):
Given x_0, α_0, integer $M \geq 0$, $\gamma \in (0,1)$, $\delta > 0$, $0 < \sigma_1 < \sigma_2 < 1$,$0 < \epsilon < 1$, set $k = 0$ and:

Step 1: If $\| g_k \| = 0$ stop

Step 2: If $\alpha_k \leq \epsilon$ or $\alpha_k \geq 1/\epsilon$ then set $\alpha_k = \delta$

Step 3: Set $\lambda = 1/\alpha_k$

Step 4: (nonmonotone line search)

If $f(x_k - \lambda_{g_k}) \leq \max_{0 \leq j \leq \min(k,M)}(f_{k-j}) - \gamma \lambda g_k^T g_k$

then set $\lambda_k = \lambda$,$x_{k+1} = x_k - \lambda_k g_k$, and go to Step 6

Step 5: Choose $\sigma \in [\sigma_1, \sigma_2]$, set $\lambda = \sigma\lambda$, and go to Step 4

Step 6: Set $\alpha_{k+1} = -(g_k^T y_k)/(\lambda_k g_k^T g_k)$, $k = k + 1$, and go to Step 1.

Remarks:

(1) The object of Step 2 is to avoid uphill directions and to keep the sequence $\{\lambda_k\}$ uniformly bounded. In fact, for all k

$$0 < \min(\epsilon, 1/\delta) \leq \lambda_k \leq \max(1/\epsilon, 1/\delta).$$

(2) The algorithm cannot cycle indefinitely between steps 4 and 5.

The parameter values suggested in (Raydan, 1997)—and adopted in this study as well—are: $\gamma = 10^{-4}$, $\epsilon = 10^{-10}$, $\sigma_1 = 0.1$, $\sigma_2 = 0.5$, $a_0 = 1$, and $M = 10$. In step 5 σ is chosen by a quadratic interpolation procedure and δ in step 2 is chosen as follows:

$$\delta = \begin{cases} 1 & \text{if } ||g_k||_2 > 1, \\ ||g_k||_2^{-1} & \text{if } 10^{-5} \leq ||g_k||_2 \leq 1, \\ 10^5 & \text{if } ||g_k||_2 < 10^{-5}. \end{cases}$$

As it can be seen GBB incorporates a non-monotone line search in order to ensure global convergence for general functions. In fact relevant theoretical results are reported in (Raydan, 1997) under rather mild assumptions. It was, also, reported there that in the noise-free unconstrained optimisation setting the resulting algorithm, GBB, is rather fast—comparable to fast implementations of the Conjugate Gradient Algorithm (CG) in many cases. Furthermore, it was found to be faster than the CG for some large scale convex optimization problems. The reported speed was one of the reasons for the choice of GBB as benchmark in our experimental investigations. The second reason was to examine the performance

of an algorithm using a *non-monotone* line search in comparison to the supervising principle of the SSC framework. And finally, the third reason was that the basic SSC algorithm—which will be defined formally in the next chapter—uses also the step size introduced in (Barzilai and Borwein, 1988) in its Search Engine (SE) constituent algorithm.

Chapter 3

A Basic SSC Algorithm – SABB

We start by introducing a basic algorithm developed within the Supervisor-Searcher Cooperation framework.

3.1 Formulation

First, we have to select an appropriate supervisor algorithm. The Stochastic Approximation (SA) algorithm is a suitable candidate, since it is simple and robust.
Let $\{t_k\}$ $(k = 0, 1, 2, ...)$ be such that
i) $t_k > 0$;
ii) $\sum_{k=0}^{\infty} t_k = +\infty$;
These conditions are assumed throughout this thesis, unless otherwise stated.
In the following we shall take $sr_k = t_k g_k$, where $g_k = \nabla f(x_k)$, an estimate of $\nabla F(x_k)$. Therefore the supervisor (SR) is the following Stochastic Approximation algorithm:
Given x_0

$$x_{k+1} = x_k - t_k g_k, \ k = 0, 1, 2, ...$$

It is well known that the SA method is globally convergent and has been widely used for stochastic optimization problems. However, in general, it is too slow to achieve high numerical accuracy, and it is difficult to select the appropriate

sequences $\{t_k\}$ for a particular problem. More details can be found in (Benveniste et al., 1990; Kushner and Clark, 1978; Kushner and Yin, 1997; Ljung, 1986). The search engine (SE) is based on the following gradient algorithm:
Given x_0

$$x_{k+1} = x_k - r_k g_k, \ k = 0, 1, 2, ...,$$

where $r_0 = 1$ and for $k \geq 1$

$$r_k = |x_k - x_{k-1}|^2 / ((x_k - x_{k-1})^T (\nabla f(x_k) - \nabla f(x_{k-1})).$$

We are now in the position to define the SSC gradient algorithm:
Let $x_0 \in R^n$ and $r_0 = 1$ be given. Let $T_k \geq 0$ be given for $k = 0, 1, 2, ...$. Assume that $f \geq 0$. Then define the following gradient optimization algorithm (SSC-SABB):

$$x_{k+1} = x_k - t_k g_k \ \text{ if } T_k f(x_k - t_k g_k) \leq f(x_k - r_k g_k), \ \ (k = 1, 2, ...),$$

otherwise

$$x_{k+1} = x_k - r_k g_k.$$

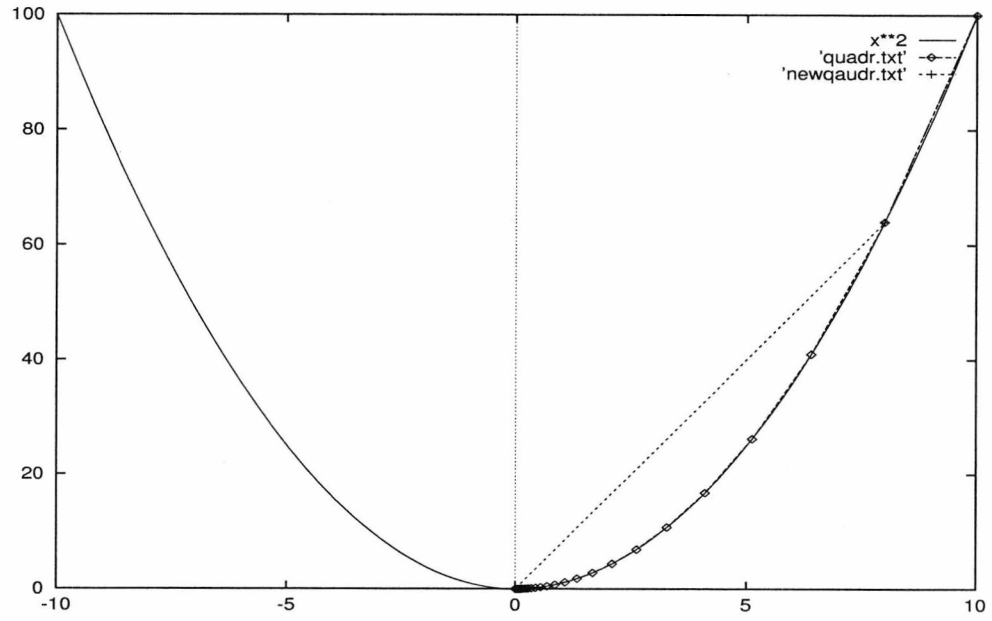
The resulting algorithm is referred to as the SSC-SABB algorithm. We refer to the step length r_k as the BB stepsize as it was initially proposed by Barzilai and Borwein in (Barzilai and Borwein, 1988) for deterministic problems. The gradient algorithm which uses the BB stepsize is further studied in (Raydan, 1993). In computations, $\{|r_k|\}$ is normally forced to be bounded. This algorithm has been shown to be locally convergent and R-superlinear for the two dimensional convex quadratic functions, and much faster than the steepest descent method. It was also shown to be locally convergent and R-linear for general convex quadratic functions in (Raydan, 1993). In (Raydan, 1997), a non-monotone line search is added to the BB algorithm in order to make it globally convergent. It was reported there that when $\epsilon = 0$, the resulting algorithm, GBB, is rather fast—comparable to the CG in many cases. In fact, it is faster than the CG for some large scale

convex optimization problems. This motivates us to use the BB algorithm as the SE. Some initial tests with the SSC-SABB have been carried out in (Liu et al., 1999b) and (Liu and Dai, 2001) mainly for deterministic but also a number of stochastic problems.

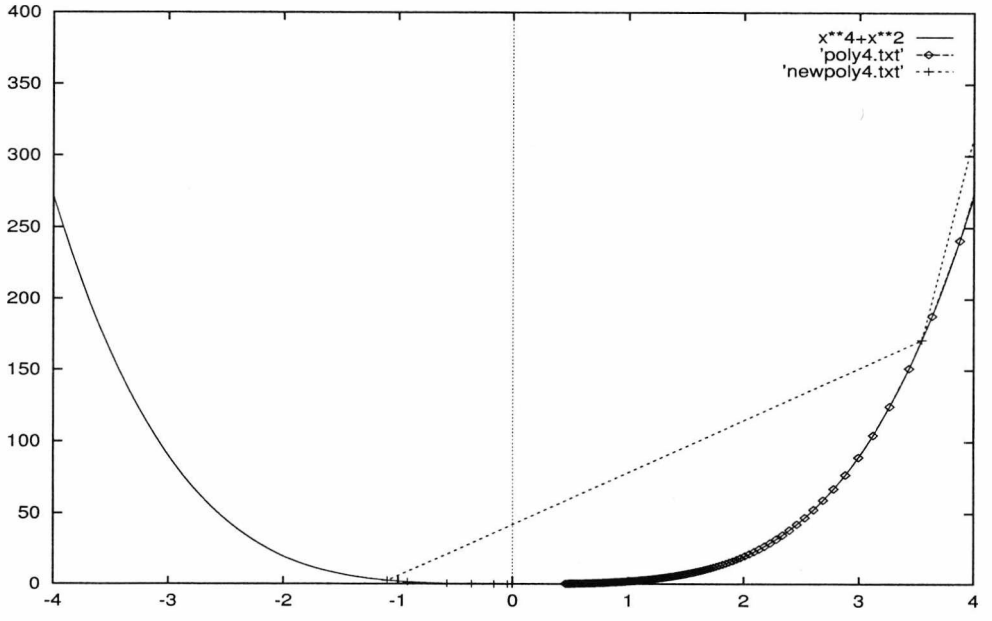
It is important to note that there is no line search in the SSC-SABB algorithm. The SSC-SABB switches between the two algorithms. All the switching is decided by an extra evaluation of the objective function value. This certainly should have less computational demand than an exact line search procedure, and should be, at least, comparable to an inexact line search procedure. Also since this extra evaluation can be easily done in parallel, it should not necessarily cause extra loss of speed in real computations.

It will be seen from the next section that to a large extent, the supervisor guarantees the global convergence of SSC-SABB algorithm and the search engine decides the local convergence rate, at least when $\epsilon = 0$. As far as the supervisor is concerned, the gain is a possible increase of the speed and efficiency, and as far as the search engine is concerned, it may be extra robustness and global convergence. The value of the "preference" switching parameter T_k, as mentioned previously, determines the monotonicity or not of the algorithm in the deterministic case. If the Search Engine is decreasing, as is the case for SA after k is large enough, values of T_k less than or equal to 1 imply monotonically decreasing function values for the SABB algorithm as well. Under the presence of stochastic noises, however, the SABB is always non-monotone and values of T_k greater than 1 give only increased preference to the steps proposed by the Search Engine (SE), which may cause the algorithm to overcome poor local minima.

Figure 1 gives two illustrative examples of the behaviour of the SABB algorithm in comparison to that of the classical Stochastic Approximation (SA) in the deterministic case. In Figure 1(a), where a one-dimensional quadratic function is plotted, it can be easily seen that the SABB (dashed line with crosses) chooses the Search Engine steps arrives at the minimum much faster than the SA (dashed line with diamonds). The same observation holds also for Figure 1(b), where the



(a) Quadratic



(b) Polynomial

Figure 1: Illustration of the paths followed by the SSCSABB (lines with crosses) and the SA (lines with diamonds) algorithms on a quadratic (a) and a polynomial (b) functions.

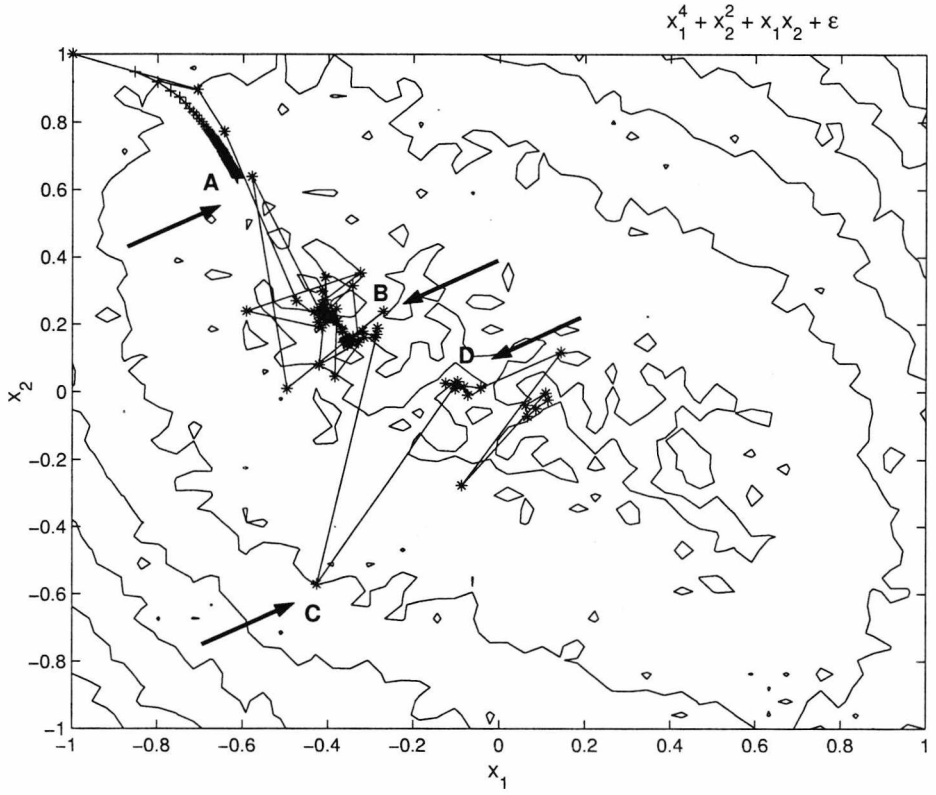


Figure 2: Illustration of the paths followed by the SSCSABB (line with stars) and the SA (line with crosses) algorithms on a polynomial function with noise. Observe the SA's premature convergence away from the minimum (0,0). Observe also the effect of non-monotonicity of the SSC algorithm allowing it to jump to a higher isoline (from point B to C) and finally reach the minimum's neighbourhood (point D).

case of a one-dimensional 4th order polynomial is presented. However, in this case the SABB, although it initially overshoots the minimum, recovers fast while the SA algorithm converges prematurely and far away from the solution, due to the flatness of the function (small gradient value) in the neighbourhood of the minimum.

Figure 2 demonstrates the comparative behaviour of the same algorithms SABB (line with stars) and SA (line with crosses) in the case of a two-dimensional polynomial function (defined on the top of the figure) under the presence noise drawn from the Standard Normal distribution. There are two important points to be observed here. First, the SA's premature convergence away from the minimum (0,0), caused again by the flatness of the function at the ravine across the

one diagonal of the plot. Second, and most interestingly, the effect of the non-monotonicity of the SSC algorithm ($T = 2.0$ was used in this case) which allows it to jump to a higher isoline (from point B to C) and thus, finally, to reach the minimum's neighbourhood (point D).

If f is not non-negative, then the above definition may be modified as

$$x_{k+1} = x_k - t_k g_k \text{ if } T_k^{sign(f(x_k - t_k g_k))} f(x_k - t_k g_k) \leq f(x_k - r_k g_k),$$

otherwise

$$x_{k+1} = x_k - r_k g_k.$$

However it was found that it is more efficient to add a positive constant to the objective function to make it non-negative. We can easily define also a *multi-step version* of the SSC-SABB algorithm, which for completeness of exposition is included in Appendix C along with some theoretic results about its behaviour presented originally in (Liu and Sirlantzis, 2001a; Liu and Sirlantzis, 2001b). However, in this study we shall concentrate on the basic version of SABB and a particular type of its variants and therefore we shall not go into further details about it.

3.2 Theoretical Results

In this section we examine convergence and speed of the SSC-SABB algorithm. We will assume that $\epsilon = 0$ in this section. However the analysis carried out here should pave the way for fuller theoretical investigations on the algorithm. Due to the special selection of SR in SSC-SABB, the resulting algorithm is always globally convergent for a wide selections of $\{T_k\}$. In SSC-SABB, $\{T_k\}$ adjusts the balance of the robustness and efficiency of the SSC-SABB algorithm.

Theorem 3.1 *(Theorem 4.1 in (Liu and Dai, 2001)) Let f be twice continuously differentiable and bounded below. Assume that ∇f is Lipschitz with a global Lipschitz constant. Let $\{x_k\}$ be generated by the SSC-SABB algorithm. Assume that*

$T_k \leq 1$ *for* $k = 0, 1, 2, ...,$. *Then there is an* $\epsilon(f) > 0$ *such that* $\{\sum_0^k t_k |\nabla f(x_k)|^2\}$ *is convergent as* $k \to \infty$ *for any sequence* $\{t_k\}$ *such that there is an* $N > 0$ *satisfying that* $t_k \leq \epsilon(f)$ *after* $k \geq N$.

It can be seen that this theorem is, in fact, a special case of Theorem 2.1, which, as we already noted, apply to quite a general class of engines. The proof of this theorem follows the same line of arguments of Theorem 2.1 and is included in Appendix B.

The case $T_k > 1$ is actually important, because in many cases, the larger T_k is, the faster the algorithm may be. There may be many different ways to cure this no-convergence problem. For instance, one may let T_k to tend to 1 as $k \to \infty$. Another simpler way to ensure global convergence is to let a finite numbers of $T_k = 1$, and then let the rest of $T_k = T > 1$. Global convergence can still be established, as it will be shown in the following theorem. We also examine the convergence rate of the SSC algorithm. In the following theorem, it is assumed that $t_k \to 0$ though it is not difficult to see that it holds for the case where t_k is very small after k is large enough, as in theorem 3.1.

Theorem 3.2 *Let* $f > 0$ *be three times continuously differentiable function and strictly convex. Let* x^* *be the minimizer of* f. *Let* $\{x_k^N\}$ *be defined by the SSC-SABB algorithm with the following choice of* $\{T_k\}$. *Let* $N > 0$ *be a fixed integer and* $T > 1$ *a fixed real number. Let* $T_k = 1$ *for* $k = 0, 1, 2, ..., N$ *and* $T_k = T > 1$ *for* $k > N$. *Then there is a* $N(x_0, f) > 0$ *such that* $\{x_k^N\}$ *is convergent, whenever* $N > N(x_0, f)$. *Furthermore, the SSC-SABB algorithm is as fast the BB algorithm locally, at least R-linearly convergent, whenever* $N > N(x_0, f)$.

The proof of the above theorem is rather involved, as it has, first, to be proved that the BB algorithm is locally R-linearly convergent. Although its local convergence has been proved for the quadratic objective function in (Raydan, 1993), R-linear convergence (or even simply convergence) of the BB algorithm for general convex objective functions still needs many extra tedious estimates to prove. On

the other hand, the principle of the proof is quite simple. (Details of the proof from (Liu and Dai, 2001) are included in Appendix B.

In practical computations T_k is often fixed to a constant $T > 1$. The convergence rate of the SSC-SABB algorithm is in general much better than that of SA algorithm, due to the faster search engine used in the algorithm - the BB algorithm is sometimes R-superlinear. This is indeed confirmed in our numerical tests. The condition $f > 0$ may be met by adding a large positive number C to the original objective function.

The above analysis confirms the expectation that is that the global convergence is largely decided by the supervisor SR while efficiency of the algorithm depends much on the search engine SE. In the next section we carry out some numerical tests for the SSC-SABB algorithm.

3.3 Deterministic Experiments

We present numerical experiments for deterministic test problems. The purpose of these tests is to see whether or not this SSC algorithm is efficient in the noise-free case. We use 22 test problems in this experiment.

We compare SSC-SABB with SA, and GBB. The latter was chosen because it was reported to be rather efficient and because it uses the BB step size as well. For GBB, we adopt all the recommended restrictions and procedures given in (Raydan, 1997), which we have already described along with a formal definition of the algorithm in Section 2.6. For SSC-SABB, we have no restriction for r_k. We take $T_k = 5$ in all the tests. We have used three different sequences of $\{t_k\}$ in our experiments:

$$t_k = \min(1.5/k, 0.01), t_k = \min(1.5/\sqrt{k}, 0.01), t_k = 0.01.$$

Table 3.1: Comparisons between GBB and SSC-SABB with different types of step lengths in the Supervisor (SR).

Problem No.	GBB	$\min(1.5/k, 0.01)$	$\min(1.5/\sqrt{k}, 0.01)$	0.01
1	221/272/222	143/287/144	143/287/144	143/287/144
2	1073/1458/1074	45/91/46	45/91/46	45/91/46
3	4/6/5	5/11/6	5/11/6	5/11/6
4	>9999	>9999	>9999	>9999
5	266/352/267	14/29/15	14/29/15	14/29/15
6	12/18/13	19/39/20	19/39/20	19/39/20
7	>9999	>9999	>9999	>9999
8	145/150/146	>9999	>9999	>9999
9	14/17/15	20/41/21	20/41/21	20/41/21
10	> 9999	> 9999	>9999	>9999
11	50/61/51	128/257/129	128/257/129	128/257/129
12	> 9999	1/3/2	1/3/2	1/3/2
13	82/87/83	92/185/93	92/185/93	92/185/93
14	75/107/76	415/831/416	289/579/290	289/579/290
15	> 9999	402/805/403	320/641/321	320/641/321
16	44/51/45	48/97/49	48/97/49	48/97/49
17	914/1238/915	668/1337/669	1573/3147/1574	1573/3147/1574
18	56/67/57	68/137/69	68/137/69	68/137/69
19	7/8/8	7/15/8	7/15/8	7/15/8
20	105/119/106	104/209/105	104/209/105	104/209/105
21	14/18/15	17/35/18	17/35/18	17/35/18
22	113/128/114	101/203/102	101/203/102	101/203/102
23	1/3/2	2/5/3	2/5/3	2/5/3

In the following tables we report the No. of iterations/No. of function evaluations/No. of gradient evaluations, with the stopping rule

$$|\nabla f(x_k)| \leq 10^{-6}.$$

The maximum number of function evaluations is set to 9999. It is found that SA fails for most of the test problems so that it is not included here.

From an examination of Table 3.1, it follows that on average SSC-SABB is as efficient as GBB in terms of the number of function and gradient evaluations.

It was found that SSC-SABB is faster than GBB in terms of CPU time - SSC solves the commonly solved problems in 0.42 (s) while GBB uses 0.78 (s) on a Sun UltraSparc 1 station. (The GNU g77 compiler was used without any optimization flags). This may be caused by the fact that in two cases, SSC finds different local minima thus consuming less CPU time. However, we should at least be able to state that the overall performances of these two algorithms are similar.

It can also be seen that the performance of the SSC is not very sensitive to the selection of $\{t_k\}$. It was found that taking a smaller value of T will somewhat slow down the algorithm, but will not change the number of solved problems. From our experience, T=5 seems to be the best choice for the deterministic case.

In the deterministic case, it is not difficult to speed up SSC-SABB. One can use, for example, the following rule: if $|g_k| \leq 0.01$, then only the BB step is used. Furthermore we can use line search in SE. The resulting algorithm will be referred to as SAGBB; that is, we use SA with $t_k = \min(1.5/\sqrt{k}, 0.01)$ as supervisor and the GBB algorithm as search engine. The test results with these improvements are shown in the Table 3.2.

It can be seen that the speed of SSC-SABB is further increased. Particularly SAGBB solves almost all the test problems. In fact, it only needs an extra few tens of iterations to solve the only unsolved test problem 7.

3.4 Stochastic Experiments

We compare SSC-SABB and some of its variants with SA and GBB. The latter was chosen because it was reported to be rather efficient and because it uses the BB step size as well. For GBB, we adopt all the recommended restrictions given in Section 2.6. In SSC-SABB, we adopt for r_k the restrictions and procedures recommended in (Raydan, 1997). We take $T_k = 1$ in all the tests unless when otherwise stated. We have used a number of different sequences for $\{t_k\}$, but in the following it can be assumed $t_k = c/(k+1)$ in every case where no other sequence is defined explicitly. In contrast to the deterministic case, the value of c has to be

Table 3.2: Comparisons between SAGBB and SSC-SABB with different types of step lengths in the Supervisor (SR) and modified switching rule.

Problem No.	$\min(1.5/k, 0.01)$	$\min(1.5/\sqrt{k}, 0.01)$	0.01	SAGBB
1	112/209/113	112/209/113	112/209/113	128/278/129
2	45/74/46	45/74/46	45/74/46	76/123/77
3	5/7/6	5/7/6	5/7/6	5/7/6
4	>9999	>9999	>9999	4/16/5
5	14/25/15	14/25/15	14/25/15	20/49/21
6	19/38/20	19/38/20	19/38/20	19/38/20
7	>9999	>9999	>9999	>9999
8	32/51/33	32/51/33	32/51/33	31/53/32
9	20/36/21	20/36/21	20/36/21	18/37/19
10	>9999	>9999	>9999	174/608/175
11	128/248/129	128/248/129	128/248/129	130/250/131
12	1/3/2	1/3/2	1/3/2	1/3/2
13	192/249/193	192/249/193	192/249/193	535/648/536
14	416/828/417	266/529/267	266/529/267	61/133/62
15	261/412/262	214/356/215	214/356/215	546/823/547
16	36/63/37	36/63/37	36/63/37	33/67/34
17	564/1092/565	1537/2957/1538	1537/2957/1538	659/1357/660
18	68/117/69	68/117/69	68/117/69	75/140/76
19	7/13/8	7/13/8	7/13/8	7/13/8
20	104/151/105	104/151/105	104/151/105	83/132/84
21	17/33/18	17/33/18	17/33/18	17/35/18
22	101/154/102	101/154/102	101/154/102	117/178/118
23	2/5/3	2/5/3	2/5/3	2/5/3

adjusted according to the problem: for problems 3, 12, 17 and 18, $c = 10^{-3}$; for problems 4, 6, 10 and 11 $c = 10^{-6}$; for problems 19–23 $c = 1.0$; while for the rest of the test problems $c = 0.1$. Although SSC-SABB is not very sensitive to the selection of $\{t_k\}$, the starting value of t_0 is important to ensure fast convergence, due to the stochastic nature of the test problems. We emphasize again that these stochastic problems are non-trivial and the above adjustments are commonly used in engineering computations in order to solve realistic problems. Furthermore, we sometimes deliberately select $\{t_k\}, \{c_k\}$ to just break the convergence conditions required in Theorems 3.1 and 3.2, in order to test robustness of these conditions.

For example, we sometimes take $t_k = 1/k^s$ with the critical value $s = 0.5$ to just break the condition (II) in Section 3.1.

3.4.1 Tests with Gradient Estimators of the Perturbation type

In the experiments presented in this section we use the following type of gradient estimators:

$$g_k = \nabla F(x_k) + \varepsilon \tag{3.1}$$

where $\varepsilon = a\varepsilon'$, with $\varepsilon' \sim N(0, 1.0)$.

Tables 3.3, 3.4 and 3.5 present the number of successful runs (out of ten) obtained with different values for a (called henceforth noise scale factor) both for the function value and for the gradient estimators. The tables contain results produced using three different T_k values; namely, we used $T_k \equiv T > 0$ in every case with $T = 5.0$, $T = 1.0$, or $T = 0.1$.

It is easy to observe, as, in fact, predicted by our convergence analysis, that reasonable levels of noises (for example $N(0, 1.0)$) do not significantly influence the performance of the algorithm in terms of the number of problems solved. Note that here we used noise scale factors a equal to 0.1 and 0.01. However, there is an indication that the effect which the level of the noise will have to the SSC-based algorithm relates to how sensitive is to noise the parent algorithm which is given preference through the value of T_k (in this case $T_k \equiv T$).

In Table 3.6 we explore the effects of the choice of the sequences t_k (i.e. the stepsizes of the supervisor SA algorithm) on the final performance of the SSC algorithm. We use $T_k \equiv 1$ and again $\varepsilon = 0.1\varepsilon'$ with $\varepsilon' \sim N(0, 1.0)$ for both the function value and the gradient estimators.

Although, from Table 3.6 it is easy to realise that the SSC-SABB algorithm is not particularly sensitive to the choice of the sequence $\{t_k\}$, there is an indication that sequences which decrease at a rate lower than $1/k$ tend to cause some improvement in the number of the solved problems.

Table 3.3: Effect of noise level on SABB; function noise scale factor = 0.1, gradient noise scale factor = 0.1, $t_k = c/(k+1)$

Problem No.	SA	SSC-SABB T=1.0	SSC-SABB max.iter.=59999	SSC-SABB T=0.1	SSC-SABB T=5.0	GBB
1	0	10	10	0	1	0
2	0	1	10	10	9	0
3	10	10	10	10	10	0
4	0	0	0	0	0	0
5	0	10	10	10	9	1
6	0	10	10	9	9	8
7	0	9	10	10	10	1
8	0	2	9	9	8	1
9	0	10	10	10	10	0
10	0	0	0	0	0	0
11	0	10	10	0	0	1
12	0	0	0	0	0	0
13	0	5	8	10	7	0
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	0	10	10	10	10	1
17	0	0	6	1	1	0
18	10	10	10	10	10	0
19	0	0	10	0	0	0
20	0	0	0	0	0	0
21	0	10	10	10	10	10
22	0	0	10	0	0	10

3.4.2 The Finite Difference Approximation to Gradients

As we have mentioned above we want to minimise, over the R^n-valued parameter x, the function:

$$f(x) = F(x) + \varepsilon \tag{3.2}$$

where $F(x)$ is continuously differentiable function and ε is an error vector. Let x_k denote the kth estimate of the minimum. Let $c_k \to 0$ be a finite difference interval and let e_i be the standard unit vector across the ith coordinate. Then, by the Kiefer-Wolfowitz version of the standard stochastic approximation algorithm the ith component of the kth finite difference approximation to the function's gradient

Table 3.4: Effect of noise level on SABB; function noise scale factor = 0.01, gradient noise scale factor = 0.01, $t_k = c/(k+1)$

Problem No.	SA	SSC-SABB T=1.0	SSC-SABB T=0.1	SSC-SABB T=5.0	GBB
1	0	10	0	4	5
2	0	0	0	0	0
3	10	6	10	10	0
4	0	0	0	0	0
5	0	10	10	10	3
6	0	10	0	8	10
7	0	10	0	0	5
8	0	9	10	10	2
9	0	7	10	10	2
10	0	0	0	0	1
11	0	10	0	10	5
12	0	0	0	0	0
13	10	9	10	10	0
14	0	0	0	0	0
15	0	0	0	0	0
16	0	10	5	6	1
17	0	3	0	0	0
18	0	10	10	10	5
19	0	10	10	10	0
20	0	0	0	0	0
21	0	10	10	10	10
22	0	0	0	0	10

will be:

$$\hat{g}_{k,i} = \frac{f(x_k + c_k e_i) - f(x_k - c_k e_i)}{2c_k} \tag{3.3}$$

then we can update x_k by:

$$x_{k+1} = x_k - \epsilon_k \hat{g}_k. \tag{3.4}$$

Then define

$$\psi_{k,i} = [f(x_k + c_k e_i) - F(x_k + c_k e_i)] - [f(x_k - c_k e_i) - F(x_k - c_k e_i)] \tag{3.5}$$

Table 3.5: Effect of noise level on SABB; function noise scale factor = 0.01, gradient noise scale factor = 0.1, $t_k = c/(k+1)$

Problem No.	SA	SSC-SABB T=1.0	SSC-SABB T=0.1	SSC-SABB T=5.0	GBB
1	0	10	0	1	2
2	0	0	0	0	0
3	10	10	10	10	1
4	0	0	0	0	0
5	0	10	10	10	2
6	0	10	4	3	9
7	0	9	0	0	7
8	0	10	10	10	3
9	0	10	10	10	4
10	0	0	0	0	0
11	0	10	0	0	5
12	0	0	0	0	0
13	0	10	10	10	0
14	0	0	0	0	0
15	0	0	0	0	0
16	0	10	0	0	4
17	0	0	0	0	0
18	0	10	10	10	8
19	0	0	0	0	0
20	0	0	0	0	0
21	0	10	10	10	10
22	0	0	0	0	10

and write

$$\frac{f(x_k + c_k e_i) - f(x_k - c_k e_i)}{2c_k} = f_{x_k^i}(x_k) - \beta_{k,i} \tag{3.6}$$

where $-\beta_{k,i}$ is the bias introduced by estimating $f(x_k)$ via central differences. If we set $\psi_k = (\psi_{k,1}, \ldots, \psi_{k,r})$ and $\beta_k = (\beta_{k,1}, \ldots, \beta_{k,r})$ then the algorithm can be rewritten as

$$x_{k+1} = x_k - \epsilon_k \nabla F(x_k) + \epsilon_k \frac{\psi_k}{2c_k} + \epsilon_k \beta_k. \tag{3.7}$$

From Eq. 3.7 we can see that in the beginning of the simulation, when x_k is most likely away from the solution x^*, one can afford a higher bias so that the variance of the effective noise $(\frac{\psi_k}{2c_k})$ will be reduced. However, as the simulation

Table 3.6: Effect of t_k on SSC-SABB; function noise scale factor = 0.1, gradient noise scale factor = 0.1.

Problem No.	SSC-SABB $t_k = c/k+1$	SSC-SABB $t_k = c/\sqrt{(k+1)}$	SSC-SABB $t_k = c$	SSC-SABB $t_k = \min(c/(k+1), 0.001)$	SSC-SABB $t_k = \max(c/\sqrt{(k+1)}, 0.001)$
1	10 (593/209)	10 (603/145)	10 (395/127)	10 (494/203)	10 (603/145)
2	2 (4626/336)	10 (3350/400)	8 (4196/394)	1 (4688/0)	10 (3350/400)
3	10 (1023/605)	9 (1488/808)	3 (493/116)	8 (605/331)	4 (1539/1407)
4	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
5	9 (1470/985)	6 (1273/994)	8 (1778/1281)	7 (2734/1381)	7 (1967/1176)
6	10 (53/35)	9 (45/14)	10 (28/0)	10 (71/48)	10 (158/235)
7	8 (1116/776)	5 (401/280)	3 (1852/1994)	8 (746/522)	7 (228/51)
8	0 (–/–)	3 (4200/405)	0 (–/–)	0 (–/–)	0 (–/–)
9	10 (1062/646)	3 (3762/186)	9 (1762/1314)	10 (1849/827)	5 (1965/1301)
10	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
11	10 (1055/1019)	5 (3108/1088)	10 (612/296)	9 (622/371)	10 (487/258)
12	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
13	0 (–/–)	1 (4227/0)	2 (3393/1156)	1 (2247/0)	3 (1970/1545)
14	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
16	10 (451/288)	10 (258/132)	10 (320/149)	10 (371/214)	10 (256/99)
17	0 (–/–)	7 (3210/596)	10 (3064/922)	0 (–/–)	10 (3262/735)
18	10 (866/672)	9 (2083/1243)	1 (1752/0)	10 (1069/826)	5 (1978/1054)
19	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
21	10 (34/5)	10 (35/4)	10 (39/7)	10 (41/7)	10 (37/5)
22	10 (94/40)	10 (88/33)	10 (841/274)	10 (1215/164)	10 (129/38)
23	10 (28/7)	10 (30/11)	10 (39/16)	10 (44/20)	10 (23/2)

evolves to approach the solution one will desire to minimize bias in the expense of larger noise. So, we want to c_k to decrease at least up to a small constant value c. There is however an alternative path to follow. Since the magnitude of the effective noise contribution in (3.7) is $\epsilon_k \frac{\psi_k}{2c_k}$, it is controlled by the rate ϵ_k decreases. That is, if ϵ_k decreases by a rate faster than of equal to c_k (i.e. $\epsilon_k = 1/(k+1)$ and $c_k = 1/\sqrt{k+1}$) then one can expect that the noise variability will have a diminishing effect as we approach the solution. However, in practical situations, roundoff errors can have a significant effect when the finite differences interval c_k is near to 0, due to the subtraction of very similar function values at each step. So, it might be beneficial not to allow c_k to get smaller than a specific value, although, it should be noted that the appropriate value depends on the problem in hand and it is not easily identified (Kushner and Yin, 1997).

In this section we shall use central finite differences of the values of f as gradient estimators in our computations. Let x_k denote the kth estimate of the minimum. Let $c_k \to 0$ be a finite difference interval and let e_i be the standard unit vector across the ith coordinate. Then, define a two-point central finite differences estimator as:

$$\hat{g}_{k,i} = \frac{f(x_k + c_k e_i) - f(x_k - c_k e_i)}{2c_k}. \tag{3.8}$$

where c_k satisfies the standard conditions stated in Theorem 2.4 on page 52. It is known (see (Kushner and Yin, 1997)) that, normally, the selection of suitable $\{c_k\}$ will depend on individual problems.

In Table 3.7 we report results produced by the SA, SSC-SABB and GBB algorithms using the two-point central finite differences estimators defined above with the following intervals:

for CFDSA1, SSC-CFDSABB1, and CFDGBB1 $c_k = c/\sqrt{k+1}$,

for CFDSA2, SSC-CFDSABB2, and CFDGBB2 $c_{k,i} = c(|x_{k,i}| + 1)$,

where $c_{k,i}$ is the interval used to estimate the ith component of the gradient $\hat{g}_{k,i}$ at the kth iteration. In every case c was equal to 0.01.

A more accurate approximation to the gradients albeit more expensive in terms

Table 3.7: Two-Point Central Finite Difference estimator of the gradient; function noise scale factor = 0.1, $t_k = c/(k+1)$.

Problem No.	CFDSA1	CFDSA2	SSC-CFDSABB1	SSC-CFDSABB2	CFDGBB1	CFDGBB2
1	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	1 (1001/0)	0 (–/–)
2	0 (–/–)	0 (–/–)	2 (3254/1247)	1 (1291/0)	0 (–/–)	0 (–/–)
3	10 (356/194)	10 (530/352)	10 (1403/1152)	6 (1363/935)	0 (–/–)	0 (–/–)
4	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
5	0 (–/–)	0 (–/–)	9 (1249/796)	10 (584/396)	1 (364/0)	2 (216/57)
6	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
7	0 (–/–)	0 (–/–)	0 (–/–)	3 (2040/580)	0 (–/–)	2 (124/11)
8	0 (–/–)	0 (–/–)	3 (2051/306)	10 (423/325)	1 (598/0)	0 (–/–)
9	0 (–/–)	0 (–/–)	10 (2176/892)	9 (1230/785)	1 (138/0)	3 (66/2)
10	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
11	0 (–/–)	0 (–/–)	0 (–/–)	10 (716/647)	2 (153/0)	1 (92/0)
12	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
13	0 (–/–)	0 (–/–)	0 (–/–)	10 (764/295)	0 (–/–)	0 (–/–)
14	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
16	0 (–/–)	0 (–/–)	6 (1126/907)	0 (–/–)	4 (290/53)	3 (190/95)
17	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
18	1 (21/0)	1 (675/0)	8 (742/697)	9 (1214/1160)	0 (–/–)	0 (–/–)
19	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
21	0 (–/–)	0 (–/–)	10 (176/106)	10 (74/39)	10 (125/52)	9 (79/32)
22	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	10 (77/33)	10 (63/22)

of function evaluations required at each step can be established by using a four-point estimator defined as:

$$\hat{g}'_{k,i} = \frac{-f(x_k + 2c_k e_i) + 8f(x_k + c_k e_i) - 8f(x_k + c_k e_i) + f(x_k - 2c_k e_i)}{12c_k} \tag{3.9}$$

Table 3.8 presents results produced again by the SA and SSC-SABB algorithms but using the four-point gradient estimators defined in Eq. 3.9, and the following finite difference intervals:
for 4CFDSA, SSC-4CFDSABB2, and 4CFDGBB2 $c_k = 0.01$, and
for SSC-4CFDSABB1, and 4CFDGBB1 $c_k = 0.1$.
It is not difficult to realise from the comparison of Tables 3.7 and 3.8 that the four-point approximation improves the performance of the algorithm.

3.4.3 The Random Directions Approximation to Gradients

The central finite differences algorithms described in the previous section provide reasonably accurate estimates of the gradient but they are increasingly expensive in terms of computational load. For a n-dimensional problem the two-point estimator requires $2n$ function evaluations for each gradient estimator and the four-point one requires $4n$. One enticing alternative, known as Random Directions approximation, is to choose randomly one direction only to update by central finite differences at each iteration. In particular, the random directions algorithm we adopt in this work, introduced by Spall (Spall, 1992), showed advantages over the classical method in such multidimensional problems. This method, called also Simultaneous Perturbation approximation to the gradient, chooses the directions at random on the vertices of a n-dimensional unit hypercube with the origin at the centre. Given a sequence d_k of n-dimensional random directions vectors, the gradient estimator is: (see also (Spall, 1998; Spall, 1999) for the details)

$$\tilde{g}_k = d_k \frac{f(x_k + c_k d_k) - f(x_k - c_k d_k)}{2c_k} \tag{3.10}$$

Table 3.8: Four-Point Central Finite Difference estimator of the gradient; function noise scale factor = 0.1, $t_k = c/(k+1)$.

Problem No.	4CFDSA	SSC-4CFDSABB1	SSC-4CFDSABB2	4CFDGBB1	4CFDGBB2
1	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	1 (561/0)
2	0 (–/–)	0 (–/–)	1 (829/0)	0 (–/–)	0 (–/–)
3	10 (231/111)	8 (1007/734)	7 (2080/1384)	0 (–/–)	0 (–/–)
4	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
5	0 (–/–)	10 (1050/611)	9 (765/502)	1 (89/0)	0 (–/–)
6	0 (–/–)	6 (182/251)	0 (–/–)	0 (–/–)	0 (–/–)
7	0 (–/–)	9 (2635/797)	3 (1535/653)	1 (68/0)	1 (99/0)
8	0 (–/–)	5 (707/905)	8 (383/251)	3 (57/13)	2 (223/37)
9	10 (1197/1713)	10 (1513/918)	10 (609/386)	1 (26/0)	2 (101/7)
10	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
11	0 (–/–)	10 (697/675)	8 (1230/572)	0 (–/–)	0 (–/–)
12	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
13	10 (280/207)	2 (208/98)	10 (945/238)	0 (–/–)	0 (–/–)
14	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
16	0 (–/–)	2 (1630/1323)	2 (3459/112)	1 (56/0)	2 (233/22)
17	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
18	0 (–/–)	0 (–/–)	9 (708/586)	0 (–/–)	0 (–/–)
19	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
21	0 (–/–)	10 (59/29)	10 (65/31)	10 (39/8)	10 (57/17)
22	10 (55/30)	10 (39/18)	10 (175/111)	10 (25/2)	10 (91/33)
23	10 (42/18)	10 (38/11)	10 (61/22)	10 (39/8)	10 (88/31)

where c_k is again a sequence of finite difference intervals such that $c_k > 0$, and either $c_k \to c$, with $c \geq 0$ or $c_k \equiv c$, with $c > 0$. The algorithm is then defined as follows:

$$x_{k+1} = x_k - \epsilon_k \tilde{g_k} \tag{3.11}$$

where $\tilde{g_k}$ is the Simultaneous Perturbation (SP) of the gradient $g_k \equiv \nabla F(x_k)$. Then defining the finite differences noise at the kth iteration, ϕ_k, as:

$$\psi_k = [f(x_k + c_k d_k) - F(x_k + c_k d_k)] - [f(x_k - c_k d_k) - F(x_k - c_k d_k)] \tag{3.12}$$

Eq. 3.11 can then be expanded as follows:

$$x_{k+1} = x_k - \epsilon_k d_k d_k^T \nabla F(x_k) + \epsilon_k \beta_k + \epsilon_k \frac{d_k \psi_k}{2c_k} \tag{3.13}$$

where β_k is the bias due to the symmetric finite difference estimator of the gradient of $\nabla F(x_k)$, in the direction d_k. Eq. 3.13 can now be rewritten as:

$$x_{k+1} = x_k - \epsilon_k [\nabla F(x_k) - \beta_k] + \epsilon_k \frac{d_k \psi_k}{2c_k} + \epsilon_k \tilde{\psi_k}, \tag{3.14}$$

where

$$\tilde{\psi_k} = [I - d_k d_k^T] \nabla F(x_k) \tag{3.15}$$

can be defined as the random direction noise. From Eq. 3.15 it is apparent the advantage of the Spall's Simultaneous Perturbation method as in this case $\tilde{\phi_k} = 0$.

In Table 3.9 we present results produced using the above random directions estimator of the gradient for the SA, SABB and GBB algorithms and $c_k = c/\sqrt{k+1}$ with c defined as follows:

for RDSA1, SSC-RDSABB1, and RDGBB1, $c = 0.01$,

for RDSA2, SSC-RDSABB2, and RDGBB2, $c = 0.1$.

Table 3.10 reports results produced again by the SA, SABB and GBB algorithms using a four-point random directions estimator of the gradient, analogous

Table 3.9: Two-Point Random Directions estimator of the gradient; function noise scale factor = 0.1, $t_k = c/(k+1)$.

Problem No.	RDSA1	RDSA2	SSC-RDSABB1	SSC-RDSABB2	RDGBB1	RDGBB2
1	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	1 (176/0)	0 (–/–)
2	0 (–/–)	0 (–/–)	1 (2726/0)	1 (2726/0)	0 (–/–)	0 (–/–)
3	10 (226/183)	10 (366/203)	9 (1813/1026)	9 (1813/1026)	0 (–/–)	0 (–/–)
4	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
5	0 (–/–)	0 (–/–)	2 (1576/450)	2 (1576/450)	0 (–/–)	0 (–/–)
6	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
7	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	2 (254/60)
8	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	1 (679/0)	1 (93/0)
9	0 (–/–)	0 (–/–)	7 (1471/750)	7 (1471/750)	2 (81/31)	0 (–/–)
10	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
11	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	3 (476/107)	1 (477/0)
12	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
13	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	1 (109/0)	1 (42/0)
14	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
16	0 (–/–)	0 (–/–)	5 (2335/1455)	5 (2335/1455)	2 (228/91)	3 (116/35)
17	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
18	2 (1030/235)	1 (105/0)	9 (634/555)	9 (634/555)	0 (–/–)	0 (–/–)
19	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
21	0 (–/–)	0 (–/–)	10 (229/84)	10 (229/84)	10 (162/75)	10 (33/15)
22	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)	8 (107/52)	10 (49/8)

Table 3.10: Four-Point Random Directions estimators of the gradient; function noise scale factor = 0.1.

Problem No.	SSC-4RDSABB1	SSC-4RDSABB2	SSC-4RDSABB3
1	1 (586/0)	0 (-/-)	0 (-/-)
2	0 (-/-)	1 (1790/0)	1 (4866/0)
3	9 (1069/746)	3 (1938/784)	6 (2231/1404)
4	0 (-/-)	0 (-/-)	0 (-/-)
5	5 (1247/754)	5 (1557/978)	5 (1344/701)
6	0 (-/-)	0 (-/-)	0 (-/-)
7	0 (-/-)	0 (-/-)	0 (-/-)
8	0 (-/-)	0 (-/-)	0 (-/-)
9	10 (1017/540)	7 (656/538)	9 (881/647)
10	0 (-/-)	0 (-/-)	0 (-/-)
11	0 (-/-)	0 (-/-)	0 (-/-)
12	0 (-/-)	0 (-/-)	0 (-/-)
13	8 (1969/553)	8 (2289/848)	3 (1928/121)
14	0 (-/-)	0 (-/-)	0 (-/-)
15	0 (-/-)	0 (-/-)	0 (-/-)
16	0 (-/-)	0 (-/-)	1 (641/0)
17	0 (-/-)	0 (-/-)	0 (-/-)
18	0 (-/-)	10 (573/405)	9 (855/638)
19	0 (-/-)	0 (-/-)	0 (-/-)
20	0 (-/-)	0 (-/-)	0 (-/-)
21	10 (51/16)	10 (107/64)	10 (63/22)
22	10 (65/23)	10 (127/76)	10 (138/51)
23	10 (55/33)	10 (65/17)	10 (113/85)

to the four-point finite difference estimator described in the previous section, defined as:

$$\tilde{g}'_k = \frac{-f(x_k + 2c_k d_k) + 8f(x_k + c_k d_k) - 8f(x_k + c_k d_k) + f(x_k - 2c_k d_k)}{12c_k} \quad (3.16)$$

In particular, the sequences c_k used, were: SSC-4RDSABB1:$c_k = 0.1$,

SSC-4RDSABB2:$c_k = 0.01$,

SSC-4RDSABB3:$c_k = 0.01/(k+1)^{0.1}$.

3.4.4 Robust Algorithms via Averaging of the Estimators

In many cases the accuracy achieved by any of the gradient estimators presented above is not satisfactory to address really hard stochastic optimisation problems, like some of the benchmarking test problems we are using. A classical proposal which emanates naturally from the suggestions in robust statistics is to incorporate an averaging process into the estimation to diminish the effects of the noises to the estimators. One can either average over the estimators of the gradient $\hat{g}_k$ at the kth iterate x_k, defining:

$$\breve{g}_k = \sum_{i=1}^{l} \hat{g}_k^i \tag{3.17}$$

where $\hat{g}_k^i$ is the ith estimator of the gradient at x_k. Alternatively, we can use averaging over the function value evaluations at the kth step:

$$\breve{f}_k = \sum_{i=1}^{l} \hat{f}_k^i \tag{3.18}$$

where $\hat{f}_k^i$ is the ith estimator of the function value at x_k. Or, finally, we can use both averaging process at the same time.

The following tables present results of our experiments using the above type of averaging procedures for the two-point and four point central finite difference and random direction estimators we introduced in the previous section, with $l = 4$. For all the algorithms presented here the prefix (GA) denotes the use of Gradient Averaging, (FA) Function Averaging, and (FGA) Function and Gradient Averaging. In particular, Table 3.11 present results produced by the SA, SABB, and GBB algorithms.

Table 3.12 includes results from corresponding averaged two-point random directions gradient estimators as follows:
SSC-FARDSABB: SSC Function value Averaging Random Directions SABB,
SSC-FGARDSABB: SSC Function value and Gradient estimator Averaging Random Directions SABB.

Table 3.11: Averaging of the Two-Point Central Finite Difference and Random Directions estimators of the gradient; function noise scale factor = 0.1, $t_k = c/(k+1)$.

Problem No.	GACFDSA	SSC-GACFDSABB	SSC-GARDSABB	GARDGBB
1	0 (-/-)	0 (-/-)	1 (773/0)	0 (-/-)
2	0 (-/-)	1 (3253/0)	1 (2918/0)	0 (-/-)
3	10 (339/189)	10 (2451/1179)	9 (2453/1227)	0 (-/-)
4	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
5	0 (-/-)	10 (1081/702)	9 (1325/691)	2 (209/96)
6	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
7	0 (-/-)	0 (-/-)	0 (-/-)	1 (378/0)
8	0 (-/-)	9 (2278/965)	4 (1884/1278)	1 (475/0)
9	6 (3462/2827)	10 (757/506)	10 (1368/683)	0 (-/-)
10	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
11	0 (-/-)	0 (-/-)	0 (-/-)	1 (295/0)
12	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
13	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
14	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
15	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
16	0 (-/-)	4 (2272/818)	6 (2049/1443)	1 (94/0)
17	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
18	10 (835/792)	8 (566/232)	9 (728/680)	1 (92/0)
19	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
20	0 (-/-)	0 (-/-)	0 (-/-)	0 (-/-)
21	10 (436/301)	10 (92/57)	10 (144/64)	9 (95/42)
22	0 (-/-)	0 (-/-)	0 (-/-)	9 (94/55)

And finally, Table 3.13 reports on the use of averaged four-point random directions gradient estimators with various sequences of finite difference approximation intervals c_k as follows:

SSC-GA4CFDSABB: $c_k = 0.1$,

SSC-GA4RDSABB1: $c_k = 0.01$,

SSC-GA4RDSABB2: $c_k = 0.01/(k+1)^{0.1}$.

Our findings, using these averaging processes, did not show significant improvements in the performance of our algorithms with respect to either the number of iterations required to solution, or the number of the problems solved. We believe this is because we used only 4 values for the averages ($n = 4$) in every case, which

Table 3.12: Function and Gradient Averaging of the Random Directions estimators of the gradient; function noise scale factor = 0.1, $t_k = c/(k+1)$.

Problem No.	SSC-FARDSABB	SSC-FGARDSABB
1	0 (–/–)	2 (2185/1175)
2	0 (–/–)	0 (–/–)
3	10 (861/379)	9 (1363/808)
4	0 (–/–)	0 (–/–)
5	3 (382/250)	4 (1609/1223)
6	0 (–/–)	0 (–/–)
7	0 (–/–)	0 (–/–)
8	0 (–/–)	4 (1400/584)
9	10 (926/521)	10 (997/465)
10	0 (–/–)	0 (–/–)
11	0 (–/–)	0 (–/–)
12	0 (–/–)	0 (–/–)
13	0 (–/–)	0 (–/–)
14	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)
16	3 (1441/482)	3 (1279/646)
17	0 (–/–)	0 (–/–)
18	9 (195/135)	10 (208/137)
19	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)
21	10 (103/63)	10 (69/41)
22	0 (–/–)	0 (–/–)

might not be adequate. However, increasing the number of estimator values used will cause a multiplicative increase in the number of function evaluations needed along with the associated computational cost, especially in the cases where a classical central finite differences estimating procedure is used. Thus, it seems that an adaptive averaging procedure will be needed.

Table 3.13: Averaging of the Four-Point Central Finite Difference and Random Directions estimators of the gradient; function noise scale factor = 0.1.

Problem No.	SSC-GA4CFDSABB	SSC-GA4RDSABB1	SSC-GA4RDSABB2
1	0 (-/-)	0 (-/-)	1 (370/0)
2	0 (-/-)	0 (-/-)	0 (-/-)
3	4 (2543/1929)	9 (1768/1170)	6 (2805/1264)
4	0 (-/-)	0 (-/-)	0 (-/-)
5	9 (841/543)	7 (1300/650)	6 (1363/965)
6	0 (-/-)	0 (-/-)	0 (-/-)
7	4 (2466/972)	0 (-/-)	0 (-/-)
8	9 (376/321)	2 (524/93)	2 (1565/785)
9	10 (1065/704)	10 (1194/910)	9 (670/270)
10	0 (-/-)	0 (-/-)	0 (-/-)
11	10 (824/418)	0 (-/-)	0 (-/-)
12	0 (-/-)	0 (-/-)	0 (-/-)
13	9 (816/698)	10 (692/215)	10 (927/197)
14	0 (-/-)	0 (-/-)	0 (-/-)
15	0 (-/-)	0 (-/-)	0 (-/-)
16	1 (4638/0)	0 (-/-)	0 (-/-)
17	0 (-/-)	0 (-/-)	0 (-/-)
18	9 (937/723)	10 (609/373)	10 (668/495)
19	0 (-/-)	0 (-/-)	0 (-/-)
20	0 (-/-)	0 (-/-)	0 (-/-)
21	10 (71/33)	10 (81/50)	10 (133/95)
22	10 (93/46)	10 (114/60)	10 (86/29)

3.5 Some Additional Tests

3.5.1 Speed

In order to gain some insight on the comparative speed of the algorithms on a well understood test problem, we analyse the speed of the tested algorithms for problem Nr. 22 (the 2nd order multidimensional polynomial) with noise. Table 3.14 reports averages over 10 runs for each one of the algorithms shown in the first column.

It can be seen that SSC-SABB is much faster than SA in this classical test problem. We have also tested a similar problem (2nd order polynomial) where stochastic noises are present in the coefficients of the parameters x_k (multiplicative

Table 3.14: Speed comparisons

Algorithms	No. of Func. Eval.	No. of Grad. Eval.	CPU - seconds
SA ($t_k = 0.1/(k+1)$)	> 9999	> 9999	-
SA ($t_k = 0.1/\sqrt{k+1}$)	> 9999	> 9999	-
SA ($t_k = 0.001$)	2207	2207	128
GBB (100-Restarts)	Line search failure	Line search failure	-
SSC-SABB (n=50)	183	366	4
SSC-SABB (n=100)	230	460	15

noise), and similar results were observed, though GBB and SA show improved performance since in that particular problem, the effect of the noises is diminishing when the approximation x_k is approaching the real solution $x^* = (0, ..., 0)$.

3.5.2 Robustness

In Tables 3.15 and 3.16 we present as series of experiments aiming to test the robustness of the SSC type of algorithms with respect to the distance of the starting point x_0 in each case from the solution $x^* = 0$. Additionally, we wished to compare their performance with that of the classical SA algorithm under the same conditions. The problem used is the one-dimensional quadratic with added noise (stochastic version of problem Nr. 23 in Section 2.4). The algorithms compared are shown in the second column with the corresponding different switching parameters T_k used for the SSC-SABB in the third column.

No generic quantitative claims can be made for general functions based on the outcomes of these experiments. Nevertheless, we believe it is reasonable to argue that they provide adequate demonstration of the qualitative differences, in terms of robustness with respect to the starting points, of the SSC algorithms as compared to the classical Stochastic Approximation (SA) algorithm. It is easily observed that as the starting point of the iterations is moving away from the solution $x^* = 0$ only the SSC algorithm with $T = 1.0$ retains its robustness and efficiency achieving not only 10 out of 10 successful runs but also the fastest runs

Table 3.15: Sensitivity to initial conditions and parameter values of the standard Stochastic Approximation (SA) and the SSC-SABB for the noisy univariate quadratic function on different accuracy levels (successful runs out of 10, and average number of iterations required in parantheses; $t_k = 1/k$ for both algorithms).

Starting Point	*Algorithm*	*Parameters*	*Accuracy*		
			0.01	0.001	0.0001
$x_0 = 1.0$	SSC	$T = 0.1$	8 (74)	7 (114)	1 (158)
		$T = 1.0$	10 (28)	10 (45)	10 (132)
		$T = 5.5$	9 (87)	5 (115)	1 (276)
		$T = 10.0$	10 (71)	5 (101)	1 (264)
	SA	-	10 (42)	9 (91)	3 (149)
$x_0 = 41.0$	SSC	$T = 0.1$	7 (40)	7 (124)	3 (139)
		$T = 1.0$	10 (31)	10 (38)	10 (93)
		$T = 5.5$	10 (38)	8 (158)	0 (-)
		$T = 10.0$	10 (67)	5 (88)	1 (56)
	SA	-	10 (47)	8 (131)	3 (109)
$x_0 = 81.0$	SSC	$T = 0.1$	9 (104)	4 (126)	1 (43)
		$T = 1.0$	10 (42)	10 (42)	10 (74)
		$T = 5.5$	10 (53)	4 (87)	0 (-)
		$T = 10.0$	10 (55)	6 (88)	0 (-)
	SA	-	10 (42)	8 (83)	4 (142)

in every case in terms of the average number of function evaluations required (presented in parentheses).

3.5.3 Consistency

Finally, we include results from two experiments we performed to explore the adequacy of the number of replications we use to assess the average behaviour of the proposed algorithms. To this end we report in Tables 3.17 and 3.18 the outcome of a consistency test involving the comparison of the results obtained from three independent repetitions of the ten simulation runs setting, which form the basis of the experimental set up in this study. The version of the SABB algorithm select for these experiments is the on using Random Directions (Simultaneous Perturbation) estimators of the gradients with Averaging in the function and gradient

Table 3.16: Sensitivity to initial conditions and parameter values of the standard Stochastic Approximation (SA) and the SSC-SABB for the noisy univariate quadratic function on different accuracy levels (successful runs out of 10, and average number of iterations required in parentheses; $t_k = 1/\sqrt{k}$ for both algorithms).

Starting Point	*Algorithm*	*Parameters*	*Accuracy*		
			0.01	0.001	0.0001
$x_0 = 1.0$	SSC	$T = 0.1$	10 (38)	10 (112)	1 (94)
		$T = 1.0$	10 (32)	9 (82)	9 (187)
		$T = 5.5$	10 (43)	9 (160)	1 (186)
		$T = 10.0$	10 (40)	9 (127)	1 (110)
	SA	-	10 (57)	8 (70)	2 (107)
$x_0 = 41.0$	SSC	$T = 0.1$	10 (35)	7 (196)	1 (42)
		$T = 1.0$	10 (28)	10 (66)	7 (143)
		$T = 5.5$	10 (43)	5 (140)	1 (263)
		$T = 10.0$	10 (44)	7 (116)	0 (-)
	SA	-	10 (47)	9 (132)	1 (176)
$x_0 = 81.0$	SSC	$T = 0.1$	10 (54)	8 (114)	2 (91)
		$T = 1.0$	10 (42)	10 (69)	9 (160)
		$T = 5.5$	10 (53)	8 (144)	0 (-)
		$T = 10.0$	10 (55)	4 (176)	0 (-)
	SA	-	10 (42)	10 (120)	2 (156)

values. This version was chosen due to the type gradient estimators used which imply increased effects of the noises, as it is already seen in the results presented previously. So, if discrepancies are to be observed among the independent sets of runs, this is the case in which is more likely to appear. The noise scale factor is the standard used in the majority of our experiments $a = 0.1$ and the noise distribution is again Gaussian.

Pairwise correlations between the columns of both tables, as measured by the Pearson r coefficient, indicate strong consistency for the three independently obtained sets of results in each case. The calculated correlation coefficient range from 0.96 to 0.99.

Table 3.17: Function and Gradient Averaging Random Direction SABB: consistency test; noise scale factor $a = 0.1$

Problem No.	SSC-FGARDSABB		
	Set 1	Set 2	Set 3
1	2 (2185/1175)	2 (2116/1806)	0 (-/-)
2	0 (-/-)	0 (-/-)	0 (-/-)
3	9 (1363/808)	9 (2094/1028)	10 (1611/832)
4	0 (-/-)	0 (-/-)	0 (-/-)
5	4 (1609/1223)	5 (632/343)	6 (1561/1301)
6	0 (-/-)	0 (-/-)	0 (-/-)
7	0 (-/-)	0 (-/-)	0 (-/-)
8	4 (1400/584)	0 (-/-)	1 (2772/0)
9	10 (997/465)	10 (663/453)	10 (999/566)
10	0 (-/-)	0 (-/-)	0 (-/-)
11	0 (-/-)	0 (-/-)	0 (-/-)
12	0 (-/-)	0 (-/-)	0 (-/-)
13	0 (-/-)	0 (-/-)	0 (-/-)
14	0 (-/-)	0 (-/-)	0 (-/-)
15	0 (-/-)	0 (-/-)	0 (-/-)
16	3 (1279/646)	6 (1150/603)	3 (1667/1421)
17	0 (-/-)	0 (-/-)	0 (-/-)
18	10 (208/137)	10 (110/70)	10 (192/130)
19	0 (-/-)	0 (-/-)	0 (-/-)
20	0 (-/-)	0 (-/-)	0 (-/-)
21	10 (69/41)	10 (76/48)	10 (45/19)
22	0 (-/-)	0 (-/-)	0 (-/-)

3.6 Concluding Remarks

This chapter aimed to examine the properties of algorithms constructed using the Supervisor Searcher Co-operation framework in a unconstrained optimisation setting. First, a basic algorithm was introduced. We subsequently, presented a theoretical analysis of the convergence properties and the speed of this SSC algorithm, which shows that under quite mild conditions the algorithm will converge to a local optimum at least as fast as the search engine (SE) "parent" algorithm, in the noise-free case. Subsequently, we tested our algorithm on an extended set of non-trivial benchmarking problems. These experimental results show that our

Table 3.18: Function and Gradient Averaging Random Direction SABB: consistency test; noise scale factor $a = 0.01$

Problem No.	SSC-FGARDSABB		
	Set 1	Set 2	Set 3
1	0 (-/-)	0 (-/-)	0 (-/-)
2	0 (-/-)	0 (-/-)	0 (-/-)
3	10 (697/597)	9 (527/346)	10 (594/355)
4	0 (-/-)	0 (-/-)	0 (-/-)
5	7 (492/562)	4 (762/705)	6 (1235/1240)
6	0 (-/-)	0 (-/-)	0 (-/-)
7	0 (-/-)	0 (-/-)	0 (-/-)
8	2 (222/98)	1 (884/0)	2 (367/117)
9	10 (185/188)	10 (197/164)	10 (576/429)
10	0 (-/-)	0 (-/-)	0 (-/-)
11	0 (-/-)	0 (-/-)	0 (-/-)
12	0 (-/-)	0 (-/-)	0 (-/-)
13	2 (1986/1471)	1 (2697/0)	2 (2239/259)
14	0 (-/-)	0 (-/-)	0 (-/-)
15	0 (-/-)	0 (-/-)	0 (-/-)
16	0 (-/-)	1 (4070/0)	1 (3070/0)
17	0 (-/-)	0 (-/-)	0 (-/-)
18	0 (-/-)	0 (-/-)	0 (-/-)
19	0 (-/-)	0 (-/-)	0 (-/-)
20	0 (-/-)	0 (-/-)	0 (-/-)
21	10 (57/41)	10 (43/14)	10 (40/17)
22	0 (-/-)	0 (-/-)	0 (-/-)

algorithm is significantly more efficient and robust, in terms of the numbers of problems solved and function and gradient evaluations required, than the classical SA algorithm in all cases examined, and at least as fast as an algorithm (GBB) claimed to be faster than the well-known Conjugate Gradient (CG) algorithm in the majority of the problems tested. However, as already pointed out, the main interest in practice lies with situations where the function values involve some kind of noise which usually affects the gradient estimates as well. Hence, initially, to simulate such cases we performed a series of experiments employing the same algorithms an the stochastic version of our benchmarking problems while using

a gradient estimator of the perturbation type. The empirical evidence suggest that in this case also the SSC-based algorithm is more robust and faster than the SA and GBB for different levels of noises. It is also indicated that in the presence of stochastic noise a value of 1 for the switching parameter T_k is the more appropriate. Moreover it was demonstrated that the SABB algorithm was not particularly sensitive to the choice of the type of step sizes t_k for its Supervisor (SR) algorithm. Additional comparative tests using gradient estimators of the finite differences type indicate that although there was a decrease in performance (as expected due to stronger noise in the gradient estimator) the algorithm proposed here retained its robustness characteristics, compared to the SA and GBB algorithms, in accordance to the corresponding theoretical analysis we presented.

Chapter 4

Modifications and Extensions

4.1 Modifications

In this section we will introduce three alternative ways to exploit the SSC framework to design variants of the algorithm defined in the previous chapter in order to satisfy specific requirements that may be of interest in real world applications.

4.1.1 Truncated Algorithms

In simulation experiments when the noise comes from an unbounded distribution, for example a Gaussian, an excessive sample value of a noise term is not unlikely and it might drive the algorithm iterate far from the solution. In real world applications, also, where we have got no control over the noisy observations similar effects may be observed. To anticipate for these undesirable events a procedure analogous to those suggested by *robust statistics* might be sought. The one we adopted here is to define a series of bounded real-valued functions

$$\psi_i(\cdot),\ i = 1, \ldots, n,$$

so that

$$\psi(g_k) = (\psi_1(g_{k,1}), \ldots, \psi_d(g_{k,n})),$$

KENT TEMPLEMAN LIBRARY UNIVERSITY

where $g_{k,i}$ is, as previously, the ith component of an estimate of $\nabla f(x_k)$, and

$$\psi_i(u) = \text{sign}(u) \min(|u|, K_i),$$

with $K_i > 0$ arbitrary constants. In effect we apply a truncation procedure on the noisy gradient estimates to avoid instability of the resulting iterates.

Experimental Results

Our aim is to test the basic SSC-SABB algorithm described in the previous chapter against an algorithm which incorporates a robust procedure. If the original algorithm allows extreme noise values to influence the stability of the iterates, introducing the truncating procedure should produce significant performance improvements in terms of the numbers of test problems solved. In Table 4.1 we report results of the SSC-SABB and the following truncated versions:
SSC-TRSABB1: with $K_i \equiv K = 10^3$,
SSC-TRSABB2: with $K_i \equiv K = 10^4$,
SSC-TRSABB3: with $K_i \equiv K = 10^5$.

A careful examination of Table 4.1 reveals that there is no improvement gained with the use of the truncation procedure in the SSC-SABB algorithm. We believe that this is a strong indication that the function value monitoring mechanism incorporated into SSC-based algorithms provides adequate control for possible extreme noise values, so an additional controlling procedure (like gradient truncation) has no further gains to offer.

4.1.2 Continuously Switching Algorithm

In some applications computational load is of paramount importance so a basic SSC algorithm which involves two function evaluations at each iteration might not present such an attractive choice. However, the SSC framework *provides the flexibility* to modify the basic algorithm to accommodate such needs. In

LIBRARY UNIVERSITY

Table 4.1: SSC-Truncated SABB vs the original SABB; function noise scale factor = 0.1, gradient noise scale factor = 0.1, $t_k = c/(k+1)$, $T = 1.0$.

Problem No.	SSC-TRSABB1	SSC-TRSABB2	SSC-TRSABB3	SSC-SABB
1	10 (321/96)	10 (434/195)	10 (428/179)	10
2	2 (4420/476)	4 (4458/470)	1 (4941/0)	1
3	10 (879/588)	10 (833/287)	10 (1019/493)	10
4	0 (-/-)	0 (-/-)	0 (-/-)	0
5	10 (1480/977)	10 (1754/1102)	10 (2552/779)	10
6	0 (-/-)	0 (-/-)	10 (88/86)	10
7	6 (1091/514)	9 (807/930)	7 (1471/996)	9
8	4 (2116/453)	3 (1978/1326)	4 (2583/558)	2
9	10 (2389/1026)	10 (1159/706)	10 (1262/545)	10
10	0 (-/-)	0 (-/-)	0 (-/-)	0
11	0 (-/-)	0 (-/-)	0 (-/-)	10
12	0 (-/-)	0 (-/-)	0 (-/-)	0
13	4 (2444/977)	2 (3162/832)	3 (3466/1305)	5
14	0 (-/-)	0 (-/-)	0 (-/-)	0
15	0 (-/-)	0 (-/-)	0 (-/-)	0
16	10 (264/84)	10 (416/230)	10 (383/188)	10
17	5 (1539/1016)	1 (4518/0)	0 (-/-)	0
18	10 (1061/678)	9 (931/552)	10 (948/711)	10
19	0 (-/-)	0 (-/-)	0 (-/-)	0
20	0 (-/-)	0 (-/-)	0 (-/-)	0
21	10 (32/3)	10 (24/3)	10 (27/6)	10
22	10 (51/14)	10 (40/14)	10 (78/9)	0

Tables 4.2 and 4.3 we present results obtained from two easily implemented modifications which show promising performance. A number of highly successful but distinctly different modifications of the basic algorithm to those presented here, which address the same issues will be introduced in 4.1.4.

Here we first define a simple SSC algorithm which alternates periodically between its two constituent "parent" algorithms. That is:
Given x_0, define for $(k = 1, \ldots, n)$ the following algorithm:

$$x_{k+1} = x_k - se_k, \quad \text{if } x_k = x_{k-1} - sr_{k-1},$$

otherwise

$$x_{k+1} = x_k - sr_k.$$

Experimental Results

For the results reported in Table 4.2 we used the SA algorithm as a supervisor (SR) and the BB algorithm as the search engine (SE). It can be easily observed that in contrast to the original SABB this algorithm does not involve monitoring the function values produced by either the supervisor step sr_k, or the search engine step se_k, so we call it a Non-Monitored SABB algorithm (NMSABB). It is reasonable to consider that in this modification the supervisor stepsize sequence is of increased importance as it provides the only means of supervision to the resulting SSC algorithm, since the function values are not monitored. Therefore, we present results for a number of stepsizes which decrease at a variety of speeds. One should expect, of course, a natural trade-off between the lower computational load imposed by this algorithm and its performance. This can be, indeed, verified by the results included in the table.

4.1.3 Selectively Monitored Algorithm

Finally, we describe an algorithm which is based on the previous modification but balances the requirement for lower computational load with the losses in performance observed in the non-monitored version. It achieves that by *selectively* monitor the function value when there is an indication for abnormally big steps in the iterates. To this end, the search engine stepsize is compared to that of the supervisor, and when a pre-specified criterion of extremity is satisfied, the function values corresponding to the stepsizes proposed by the SR and the SE are both calculated and compared in the same way as in the original SABB. Otherwise, the algorithm is allowed to continuously accept the SE stepsize, without any additional function value check taking place. Formally, this algorithm, called Selectively Monitored SABB (SMSABB), can be defined as follows:

Table 4.2: SSC-Non-Monitored SABB vs the original SSC-SABB; function noise scale factor = 0.1, gradient noise scale factor = 0.1.

Problem No.	SSC-NMSABB $t_k = c/k+1$	SSC-NMSABB $t_k = c/\sqrt{(k+1)}$	SSC-NMSABB $t_k = c/(k+1)^{0.1}$	SSC-NMSABB $t_k = \max(1/\sqrt{(k+1)}, 0.001)$
1	10 (295/225)	10 (477/77)	0 (–/–)	10 (541/137)
2	0 (–/–)	4 (4179/2465)	9 (3012/555)	2 (4760/3366)
3	0 (–/–)	2 (44/22)	0 (–/–)	1 (57/0)
4	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
5	2 (1195/1089)	10 (587/464)	10 (209/139)	10 (754/533)
6	0 (–/–)	0 (–/–)	8 (578/105)	9 (704/332)
7	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
8	10 (327/281)	10 (333/300)	10 (652/223)	10 (350/206)
9	0 (–/–)	6 (162/107)	7 (219/162)	9 (318/211)
10	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
11	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
12	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
13	10 (423/290)	10 (418/365)	10 (452/222)	10 (411/305)
14	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
15	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
16	0 (–/–)	0 (–/–)	6 (148/61)	1 (266/0)
17	0 (–/–)	0 (–/–)	0 (–/–)	1 (467/0)
18	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
19	4 (9135/763)	0 (–/–)	0 (–/–)	0 (–/–)
20	0 (–/–)	0 (–/–)	0 (–/–)	0 (–/–)
21	10 (88/9)	10 (396/9)	10 (3905/5)	10 (390/6)
22	10 (96/51)	10 (171/36)	0 (–/–)	10 (175/28)
23	10 (25/5)	10 (27/7)	10 (27/5)	10 (26/7)

Given x_0, t_0,$(k = 1, 2, \ldots)$:

$$\text{if } r_k \leq bt_k :$$

$$x_{k+1} = x_k - r_k g_k,$$

$$\text{otherwise :}$$

$$x_{k+1} = x_k - t_k g_k \text{ if } T_k f(x_k - t_k g_k) \leq f(x_k - r_k g_k) \text{ or,}$$

$$x_{k+1} = x_k - r_k g_k \text{ otherwise,}$$

where b is a constant, and t_k and r_k are the SA and BB stepsizes respectively, as defined previously.

Experimental Results

Table 4.3 presents results obtained using a number of sequences t_k which decrease at different rates, and for two different values of b as follows:
SSC-SMSABB1: with $b = 10^2$,
SSC-SMSABB2: with $b = 10^3$.
In this particular table, instead of reporting only the Mean value of the number of **iterations** required to solve each problem and the Mean Absolute Deviation (MAD) from this Mean, we also show the Mean value of the number of **function evaluations** needed in every case and include the corresponding MAD. The gains obtained in comparison to the original SSC-SABB, due to this type of modification in the basic algorithm, can be easily observed using the above described statistics we report in parentheses.

4.1.4 Inexact Variants

Algorithms designed according to the SSC framework, typically, require, as already mentioned, additional evaluations of the objective function. Although the computational expense is not high in small or medium sized problems, it might cause some concern for very large scale problems. Motivated by these thoughts

Table 4.3: The original SSC-SABB vs the SSC-Selectively Monitored SABB; function noise scale factor = 0.1, gradient noise scale factor =0.1.

Problem No.	SSC-SABB $t_k = \max(\frac{c}{\sqrt{(k+1)}}, 0.001)$	SSC-SMSABB1 $t_k = c/(k+1)$	SSC-SMSABB1 $t_k = 0.001$	SSC-SMSABB1 $t_k = \max(\frac{c}{\sqrt{(k+1)}}, 0.001)$	SSC-SMSABB2 $t_k = \max(\frac{c}{\sqrt{(k+1)}}, 0.001)$
1	10 (603/1205/291)	10 (503/547/188)	10 (490/532/200)	10 (226/229/88)	10 (217/217/80)
2	10 (3350/6700/800)	9 (6699/8138/1423	10 (4679/5141/1022)	10 (2113/2192/631)	10 (1389/1390/337)
3	4 (1539/3078/2814)	10 (388/524/305)	10 (221/274/198)	10 (269/321/221)	10 (368/373/297)
4	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
5	7 (1967/3933/2353)	10 (310/379/234)	10 (366/447/419)	10 (362/372/222)	10 (251/251/98)
6	10 (158/316/471)	10 (32/52/5)	10 (31/31/0)	10 (31/31/0)	10 (457/457/756)
7	7 (228/455/102)	10 (426/433/183)	10 (357/367/129)	10 (434/434/173)	10 (493/493/247)
8	0 (-/-/-)	10 (284/300/197)	10 (260/278/182)	10 (297/305/203)	10 (581/583/276)
9	5 (1965/3930/2603)	10 (366/451/410)	10 (450/526/381)	10 (394/399/197)	10 (569/570/300)
10	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
11	10 (487/973/516)	10 (307/503/196)	10 (336/352/199)	10 (313/324/210)	10 (1429/1429/1267)
12	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
13	3 (1970/3939/3090)	10 (189/191/117)	10 (177/181/78)	10 (379/379/259)	10 (270/270/172)
14	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
15	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
16	10 (256/512/198)	10 (298/352/164)	10 (239/281/78)	8 (193/193/96)	8 (149/149/37)
17	10 (3262/6523/1470)	0 (-/-/-)	10 (3266/3444/656)	10 (3477/3657/682)	0 (-/-/-)
18	5 (1978/3956/2108)	10 (154/188/126)	10 (222/225/106)	10 (271/275/129)	8 (286/286/209)
19	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
20	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)	0 (-/-/-)
21	10 (37/73/11)	10 (35/36/2)	10 (43/57/10)	10 (37/37/4)	10 (38/38/4)
22	10 (129/257/77)	10 (289/289/138)	10 (243/244/129)	10 (251/251/152)	10 (176/176/73)
23	10 (23/46/5)	10 (24/24/4)	10 (31/40/13)	10 (30/30/10)	10 (24/24/2)

we introduced a cheaper, in terms of computational load, alternative. In this modification, instead of using an exact function evaluation in the first part of the algorithm (the switching decision part), we employ an approximation of the objective function value at the proposed point. In the following tables we present results for the SSC-SABB using three types of such approximations. The algorithm can be rewritten as follows:

$$x_{k+1} = x_k - t_k g_k \text{ if } T_k \hat{f}(x_k - t_k g_k) \leq \hat{f}(x_k - Z_k g_k), \quad (k = 0, 1, 2, ...),$$

otherwise:

$$x_{k+1} = x_k - Z_k g_k,$$

where, for the linear approximation:

$$\hat{f}(x_k - \eta_k g_k) = |f(x_k) - \eta_k g_k^T g_k|,$$

for the quadratic approximation:

$$\hat{f}(x_k - \eta_k g_k) = |f(x_k) - \eta_k g_k^T g_k + \frac{1}{2}\eta_k^2 H'_k g_k|,$$

and for the cubic approximation:

$$\hat{f}(x_k - \eta_k g_k) = |f(x_k) - \eta_k g_k^T g_k + \frac{1}{2}\eta_k^2 H'_k g_k - \frac{1}{6}\eta_k^3 H''_k g_k|.$$

Where the superscript T denotes vector transposition and $H'_k = (h'_{ij,k})$, with

$$h'_{ij,k} = \begin{cases} 0 & i \neq j \\ h'_{i,k} & i = j \end{cases},$$

where $h'_{i,k} = \Delta g_{i,k} / \Delta x_{i,k}$ and $\Delta g_{i,k}$ and $\Delta x_{i,k}$ are the ith components of Δg_k and Δx_k respectively;

and $H''_k = (h''_{ij,k})$, with

$$h''_{ij,k} = \begin{cases} 0 & i \neq j \\ h''_{i,k} & i = j \end{cases},$$

where

$$h''_{i,k} = [(\Delta g_{i,k}/\Delta x_{i,k}) - (\Delta g_{i,k-1}/\Delta x_{i,k-1})]/\Delta x_{i,k}.$$

Subsequently, in the decision step of the SSC algorithm we take first $\eta_k = t_k$ so that $\hat{f}(x_k - t_k g_k)$ is an approximation of $f(x_k - t_k g_k)$, and then $\eta_k = Z_k$, with Z_k taking any one of the forms defined in Sections 3.1 and 4.2.1, so that $\hat{f}(x_k - Z_k g_k)$ is an approximation of the function value $f(x_k - Z_k g_k)$. Note that in the above algorithms we only need one evaluation of the function per iteration—which we will use to obtain the approximation of the function value in the next iteration—while in the exact version of the algorithms we need two evaluations to use in the decision phase.

Deterministic Experiments

We will first examine the deterministic case. The results presented in Table 4.4 below have been obtained using $Z_k = r_k I$, with I the unit matrix, i.e. the BB step length, $T_k = 5.0$, $t_k = \min(1/k, 0.01)$, and maximum number of function evaluations set to 9999. The stopping rule was kept the same as in the exact case to facilitate comparisons. The last column of the table contains results from a run with the maximum number of function evaluations increased to 14999 to check whether this would help the algorithm to solve more problems.

As it can be seen in the above table a considerable number of problems are solved, in comparison to the number solved by the exact version (see Table 3.1). It is not difficult to conclude that the approximations we used (i.e. linear, quadratic, and cubic) proved to be, in general, acceptable local approximations to the exact function values for a significant number of our test problems. Furthermore, the savings in computational load are easy to realise observing that the number of function evaluations in Table 4.4 is equal to the number of iterations, in contrast

Table 4.4: Results for the deterministic test problems from the Inexact Variants of SSC-SABB; reported are the No. of function evaluations/No. of gradient evaluations/No. of iterations.

Prob. Nr.	Linear	Quadratic	Cubic	Cubic ($k_{max} = 14999$)
1	132/133/133	90/91/91	172/173/173	172/173/173
2	527/528/528	539/540/540	93/94/94	93/94/94
3	5/6/6	5/6/6	5/6/6	5/6/6
4	FL	FL	FL	FL
5	51/52/52	51/52/52	66/67/67	66/67/67
6	19/20/20	19/20/20	19/20/20	19/20/20
7	FL	FL	FL	FL
8	34/35/35	34/35/35	30/31/31	30/31/31/
9	31/32/32	31/32/32	39/40/40	39/40/40
10	FL	FL	FL	FL
11	163/164/164	163/164/164	163/164/164	163/64/164
12	1/2/2	1/2/2	1/2/2	1/2/2
13	243/244/244	191/192/192	1005/1006/1006	1005/1006/1006
14	697/698/698	899/900/900	FL	FL
15	276/277/277	458/459/459	539/540/540	539/540/540
16	FL	FL	FL	FL
17	1053/1054/1054	1059/1060/1060	964/965/965	964/965/965
18	89/90/90	89/90/90	240/241/241	240/241/241
19	9/9/9	8/9/9	8/9/9	8/99
20	288/289/289	FL	203/204/204	203/204/204
21	18/19/19	18/19/19	18/19/19	18/19/19
22	117/118/118	117/118/118	110/111/111	110/111/111

to Table 3.1, where the former is double the latter.

However, there is a number of problems solved by the exact implementation which are not addressed successfully when the function approximation is employed. So, this inexact implementation offers an interesting and cheap alternative, in terms of the computational effort required, albeit at the expense of the range of problems that can be successfully addressed by it. Thus, it seems that when some prior knowledge about the "good" nature of the problem in hand is available, and computational resources are at a premium, an inexact SSC type algorithm it is worth a try.

Stochastic Experiments

In this section, we report results for the same type of inexact versions of the SSC-SABB algorithm but for the set of the 22 stochastic test problems. We used the same noise level and stopping rule as in the exact versions. The maximum number of function evaluations was also set to 9999, $T = 1.0$, and $t_k = C/k + 1$. Table 4.5 reports the number of successes for each algorithm out of 10 runs for each one of the problems. The column labelled 'Linear2' includes results from a modified version of the linear approximation in which $T = 2.0$ was used in the decision phase when $\sqrt{|\Delta x_k|} < 10^{-2}$. As it can be seen, from a comparison with the first column (labelled 'Linear'), the number of successful runs was not significantly affected by this modification.

It is interesting to observe once again that the linear approximation presents more successes than the quadratic and cubic ones. We believe that this is caused mainly by the fact that the latter contain approximations of the values of the second and third order derivatives at the current point. The error in these approximations is likely to be amplified as a result of the presence of the strong stochastic noises. It is important to note that, for example, problem 22 which is a rather large but "well behaved" one, is solved by the inexact implementation in half the time required (2 secs CPU time) by the exact one (4 secs CPU time). It is useful to remind here that these inexact versions of the algorithm are much cheaper in computational load, especially for complex high-dimensional functions. They require half the number of function evaluations at each iteration, while the additional load for the calculation of the approximation can be considered negligible. These inexact variants seem to be much more significant for the stochastic case. For example, the "Linear" version achieves over 9 successful runs, out of a total of 10, in 15 problems while the corresponding exact version of SABB achieves the same for only 12 problems. It is not difficult to realise that in the stochastic case even the exact versions use approximations of the function values of the underlying mathematical model, due to the presence of noise. So,

Table 4.5: Results for the stochastic test problems from the Inexact Variants of SSC-SABB; reported are the No. of function evaluations/No. of gradient evaluations/No. of iterations.

Problem Nr.	Linear	Linear2	Quadratic	Cubic
1	10	10	1	0
2	10	9	0	0
3	10	10	10	10
4	0	0	0	0
5	10	10	10	10
6	9	9	10	6
7	10	10	10	10
8	10	10	10	10
9	10	10	10	10
10	0	0	0	0
11	10	10	10	0
12	0	0	0	0
13	10	10	10	10
14	0	0	0	0
15	0	0	0	0
16	10	10	7	0
17	10	10	0	0
18	10	10	10	10
19	0	0	5	0
20	0	0	0	0
21	10	10	10	10
22	10	10	10	10

using one of the inexact variants in this case might not have an adverse effect on the performance of the algorithm as in the deterministic case. Hence, taking into account the significant number of solved problems they achieve, in comparison to the corresponding exact version (see, for instance, Table 3.3), they seem to present an alternative worth considering. This is exactly what it is suggested by the results presented here.

4.2 Extensions

4.2.1 Alternative Stepsizes – SAQPROP

Quickpropagation (QPROP) was proposed as a heuristic (and indeed, a risky one, according to its creator, see (Fahlman, 1988)) neural networks training algorithm. However, it has been widely used in neural networks training since; see (Chichocki and Unbehauen, 1993) and (Veitch and Holmes, 1991). It was reported to be surprisingly efficient. In fact, it is much faster than the classical Backpropagation of Errors (BPR) algorithm (described in Section 5.2), which is essentially a variant of SA using a constant step size. White (White, 1989) provides an extensive discussion on the relation of the SA algorithm and the original Backpropagation in the neural networks training context. However, convergence—global or local—has never been proved for the QPROP algorithm. Thus, the Quickpropagation algorithm alone, being heuristic, cannot be applied with full confidence. As a matter of fact, it can be seen in our numerical results that QPROP is unlikely to converge for general objective functions, though it works exceptionally well for a particular class of objective functions. This may explain its success in neural networks training.

We start by selecting a supervisor algorithm. Clearly the Stochastic Approximation (SA) algorithm is a suitable candidate, since it is simple and robust.
Let $\{t_k\}$ $(k = 0, 1, 2, ...)$ be such that
i) $t_k > 0$;
ii) $\sum_{k=0}^{\infty} t_k = +\infty$;
These conditions are assumed throughout the paper. It should be noted that we do not assume that $t_k \to 0$ as $k \to \infty$.
In the following we shall take $sr_k = t_k g_k$, where $g_k = \nabla f(x_k)$; that is, an estimate of $\nabla F(x_k)$. Therefore the supervisor (SR) is the following Stochastic Approximation algorithm:
Given x_0

$$x_{k+1} = x_k - t_k g_k, \ k = 0, 1, 2, \ldots .$$

It is well known that the SA method is globally convergent and has been widely used for stochastic optimization problems. However, in general, it is too slow to achieve high numerical accuracy, and it is difficult to select the appropriate sequences $\{t_k\}$ for a particular problem. The readers are referred to (Benveniste et al., 1990), (Kushner and Clark, 1978), (Kushner and Yin, 1997), and (Ljung, 1986) for more details.

The search engine can be defined as the following simple gradient algorithm:
Given x_0

$$x_{k+1} = x_k - B_k g_k, \ k = 0, 1, 2, \ldots ,$$

where $B_0 = 0.1 * I$, and for $k \geq 1$ $B_k = (b_{ij,k})$, with

$$b_{ij,k} = \begin{cases} 0 & i \neq j \\ b_{i,k} & i = j \end{cases} ,$$

where

$$b_{i,k} = (x_{i,k} - x_{i,k-1})/(g_{i,k} - g_{i,k-1}),$$

where I is the unit matrix and $x_{i,k}$ and $g_{i,k}$ are the ith components of x_k and $\nabla f(x_k)$ respectively.
We will refer to this step length B_k as the QPROP1 step size, due to the original algorithm where it comes from, called the Quickpropagation (QPROP) algorithm (see (Fahlman, 1988)). An alternative to the above simple version is the following modification (which, in fact, is closer to the originally proposed Quickpropagation):
Given x_0

$$x_{k+1} = x_k - B'_k g_k, \ k = 0, 1, 2, \ldots ,$$

where B'_k is also a diagonal matrix and $B'_0 = B_0$, and for $k \geq 1$

$$b'_{i,k} = \text{sign}(b_{i,k}) \min(b|\frac{\Delta x_{i,k}}{g_{i,k}}|, |b_{i,k}|),$$

where $\Delta x_{i,k} = x_{i,k} - x_{i,k-1}$ and b is a constant with most commonly suggested

values $1.5 \leq b \leq 3.5$. In (Fahlman, 1988) it was reported that a value of $b \approx 1.75$ seems to give the best results for a class of benchmark problems within the neural networks training context. This step length B'_k will be referred to as the QPROP2 step size.

An interesting observation at this point is that both the BB and QPROP stepsizes can be derived from the following iterative procedure:

$$x_{k+1} = x_k + Z'_k g_k,$$

where $Z'_k = r_k I$, or $Z'_k = B_k$, then we can solve for

$$Z'_k = \arg\min |\Delta x_k - Z'_k \Delta g_k|^2,$$

where $\Delta x_k = x_k - x_{k-1}$ and $\Delta g_k = g_k - g_{k-1}$. The choice of this representation is motivated by the fact that it provides a two-point approximation to the secant equation underlying quasi-Newton methods. For more details the readers are referred to (Barzilai and Borwein, 1988) and the references therein.

We are now in the position to define the corresponding SSC gradient algorithms:
Let $x_0 \in R^n$ and $r_0 = 1$, $B_0 = B'_0 = 0.1 * I$ be given. Let $T_k \geq 0$ be given for $k = 0, 1, 2, \ldots$. Assume that $f \geq 0$. We define the following algorithms constructed via the SSC principle:

$$x_{k+1} = x_k - t_k g_k \text{ if } T_k f(x_k - t_k g_k) \leq f(x_k - Z_k g_k), \quad (k = 0, 1, 2, \ldots),$$

otherwise

$$x_{k+1} = x_k - Z_k g_k.$$

Then for $Z_k = r_k I$ we have the SSC-SABB algorithm,
for $Z_k = B_k$ we have the SSC-SAQPROP1 algorithm, and
for $Z_k = B'_k$ we have the SSC-SAQPROP2 algorithm.

Note that for the convergence and speed properties of the SSC-SAQPROP algorithm one can apply directly the results presented in Theorems 2.1 and 2.2 for the noise-free and the deterministic error cases.

If f is not non-negative, then the first part of the above definitions may be modified as

$$x_{k+1} = x_k - t_k g_k \text{ if } T_k^{sign(f(x_k - t_k g_k))} f(x_k - t_k g_k) \leq f(x_k - Z_k g_k).$$

However, our practical experience indicates that it is more efficient to add a positive constant to the objective function to make it non-negative.

In the following sections we present numerical experiments for the deterministic version of our test problems. The purpose of these tests is to examine whether the introduction of the SSC monitoring mechanism will reduce the instability inherent in the search engine (SE) algorithm, and therefore, the resulting SSC algorithm will be in practice, more efficient in the noise-free case, than its constituent "parent" algorithms.

Experimental Results

We now proceed to present results obtained using the basic algorithm (SAQPROP) and its variant along with the original Quickpropagation (QPROP) algorithm for the same set of deterministic general test problems. We chose here to set $T = 1.0$ as the QPROP search engine seemed to be rather unstable–in (Fahlman, 1988) it was reported that in the neural network training case the algorithm is likely to diverge to infinity if the step lengths are not appropriately restricted. Having this in mind we used for the original QPROP a restriction parameter $b = 1.75$ (see Section 4.2.1), and a tighter supervision for SAQPROP1, while for SAQPROP2 we used $b = 3.5$ and tested it also for $T = 5.0$.

It is easy to observe from Table 4.6 that the original QPROP algorithm fails in many problems, as it was expected. It also appears to need a significantly greater number of iterations than the SABB (c.f. Table 3.1) to converge in some

Table 4.6: Comparison between QPROP and the two versions of SSC-SAQPROP for the deterministic problems; reported are the No. of function evaluations/No. of gradient evaluations/No. of iterations.

Problem Nr.	QPROP	SAQPROP1	SAQPROP2 $T = 1.0$	SAQPROP2 $T = 5.0$
1	9531	639/1279/640	918/1837/919	FL
2	5301	FL	FL	4955/9911/4956
3	25	20/41/21	27/55/28	24/49/25
4	FL	FL	FL	FL
5	340	FL	FL	68/137/69
6	56	19/39/20	35/71/36	35/71/36
7	FL	FL	FL	FL
8	FL	FL	FL	24/29/25
9	195	30/61/31	107/215/108	178/357/179
10	41	15/31/16	26/53/27	46/93/47
11	699	493/987/494	625/1251/626	696/1393/697
12	2	1/3/2	1/3/2	1/3/2
13	525	FL	FL	894/1789/895
14	FL	FL	1499/2999/1500	FL
15	FL	FL	FL	FL
16	FL	FL	9/19/10	9/19/10
17	FL	FL	FL	FL
18	1451	177/355/178	272/545/273	1037/2075/1038
19	8	6/13/7	7/15/8	7/15/8
20	14	13/27/14	21/43/22	19/39/20
21	35	15/31/16	23/47/24	25/51/26
22	101	109/219/110	81/163/82	116/233/117

cases like problems 1 and 2. The invocation of the supervisor in the SSC based algorithms produces only marginal improvement either in the number of problems solved or in speed. This will be further discussed in the next section. It seems that for these algorithms the nature of the problem plays a significant role also. An interesting case is presented by problems 1 and 2 again. While problem 1 is solved quite fast, in comparison to QPROP, by SAQPROP1 and SAQPROP2 with $T = 1.0$, it is not solved by SAQPROP2 with $T = 5.0$. In contrast, the latter solves problem 2 which is not solved by any of the former ones. This seems to indicate that a balance between the restrictions to the step length and the degree

of supervision is essential for the success of the SSC type algorithms employing QPROP as a search engine. The rather volatile characteristics of this search engine should be appropriately contained to obtain a useful algorithm. However, it is very important to note here that problem 10, which is not solved by the SABB algorithm, is easily and quickly solved by all algorithms incorporating QPROP. This observation seems to confirm our initial hypothesis that the QPROP is well suited for specific kinds of problems; a fact supported, also, by the continuous and successful use of the algorithm in the neural networks training context.

One of the reasons that neither SAQPROP1 nor SAQPROP2 obtained significant improvements in comparison to the original QPROP algorithm, in the set of the our test problems, seems to be that, by switching to SA, the SAQPROP destroys the information about previous iterations which is built in Δx_k and Δg_k. This "memory", however, should play an important role in the success of the original QPROP for the class of optimization problems corresponding to neural networks training. A possible remedy for this case could be a multi-step or "the multi-stage" SSC algorithm, which will attempt to compensate for this loss of "memory" by allowing the search engine (SE) to run for a fixed number of iterations before a switching is attempted. Such an extension has been introduced in (Liu and Sirlantzis, 2001a; Liu and Sirlantzis, 2001b) and its behaviour was studied theoretically. For completeness of our investigation of the SSC-type of algorithms we include the relevant material in Appendix C.

4.2.2 Combination of more than 2 algorithms

It can been seen from the previous numerical results that SABB is much better than QPROP or SAQPROP for general problems. However, a closer examination of the numerical results reveals that either QPROP or SAQPROP works extremely well on some particular problems, such as problem Nr. 10, on which SABB fails even to converge. This observation led us to consider an extension of the two basic algorithms introduced previously, namely the SABB and the SAQPROP. Our aim was to combine the desirable attributes of QPROP and BB in a new,

possibly improved, SSC-type algorithm, by extending the co-operation principle to accommodate two search engines.

Let $x_0 \in R^n$ and $r_0 = 1$, $\beta_0 = 0.1$ be given. Let $T_k \geq 0$ be given for $k = 0, 1, 2, ...$ Assume that $f \geq 0$. Then define the following gradient optimization algorithm (SSC-SABBQPROP):

$$x_{k+1} = x_k - t_k g_k$$

$$\text{if } f(x_k - t_k g_k) \leq \min(T_k f(x_k - Z_k g_k), f(x_k - r_k g_k)), \quad (k = 0, 1, 2, ...),$$

otherwise

$$\text{if } f(x_k - r_k g_k) \leq T_k f(x_k - Z_k g_k)$$

$$x_{k+1} = x_k - r_k g_k,$$

otherwise

$$x_{k+1} = x_k - Z_k g_k,$$

where r_k is the BB step size, and Z_k can be either B_k or B'_k of the two QPROP step sizes presented in Section 4.2.1. The motivation is to make the BB and QPROP algorithms work together. We expect that this extended SSC algorithm shall be able to solve more test problems, and example problem 10, and this is indeed confirmed by our tests.

Although this extended algorithm is stated only for a particular pair of SR and SE, the general principle on which it is based is applicable to more general cases. Below we examine the convergence properties of such extensions.

We will assume that $\epsilon = 0$ in this section. However, the analysis carried out here should pave the way for a more rigorous theoretical investigation of this type of algorithms in the presence of stronger noises as well.

Theorem 4.1 *Let f be twice continuously differentiable and bounded below. Assume that ∇f is Lipschitz with a global Lipschitz constant. Let x_k be generated by the SSC-SABBQPROP algorithm. Then, assuming $t_k \to 0$, $\{\sum_0^k t_k |\nabla f(x_k)|^2\}$ is*

convergent as $k \to \infty$ *provided* $\prod_0^\infty \max(1, 1/T_k) < \infty$.

Proof: Following the definition of the algorithm we have:
either

$$f(x_{k+1}) = f(x_k - t_k g_k),$$

or

$$\text{if } \min(T_k f(x_k - Z_k g_k), f(x_k - r_k g_k)) < f(x_k - t_k g_k),$$

then

$$f(x_{k+1}) \neq f(x_k - t_k g_k).$$

In the latter case, if $f(x_k - r_k g_k)$ is the smallest between $(T_k f(x_k - Z_k g_k), f(x_k - r_k g_k))$, then

$$f(x_{k+1}) = f(x_k - r_k g_k) < f(x_k - t_k g_k).$$

Otherwise

$$f(x_{k+1}) = f(x_k - Z_k g_k) < f(x_k - t_k g_k)/T_k.$$

In summary, for both cases,

$$f(x_{k+1}) \leq \max(f(x_k - t_k g_k), f(x_k - t_k g_k)/T_k) = \max(1, 1/T_k) f(x_k - t_k g_k).$$

Then we have

$$f(x_{k+1}) \leq \max(1, 1/T_k) f(x_k - t_k g_k).$$

From this we can easily apply exactly the same line of arguments used in the proof of Theorem 2.1 to show that the sequence $\{\sum_0^k t_k |\nabla f(x_k)|^2\}$ is convergent as $k \to \infty$ provided $\prod_0^\infty \max(1, 1/T_k) < \infty$.

Note that convergence results can be established for the cases of either deterministic or stochastic bounded noises following the same line of arguments presented in Remark 2.2.

As a general observation for both the basic and extended versions we note that $\{T_k\}$ seems to adjust the balance of the robustness and efficiency of the SSC

type algorithms, by influencing the levels of co-operation among the participating "parent" algorithms. It also offers a mechanism to *assign preference* to a particular "parent" algorithm. This, in turn, can prove an efficient way of incorporating prior information about a particular class of problems, by allowing more frequent use of the "parent" algorithm which seems to possess the most suitable properties to tackle this class of problems.

Experimental Results

Next, we present results of our numerical experiments on the same set of the 22 deterministic optimization problems obtained by the first of the extended algorithms, namely the SSC-SABBQPROP, while results for the multi-stage extension will be presented in the second paper of this series for the neural networks training problem. As already mentioned, the purpose of this extension was twofold. On the one hand we aimed to enhance the efficiency and robustness of the SAQPROP algorithm for a wider range of problems, as it seemed to fail in a larger number of problems than SABB and to require a greater number of function evaluations for some of the problems it solved. On the other hand, we wished to infuse SABB with the ability of the QPROP search engine to solve quickly particular types of problems, like problem 10, because one of the main purposes of this extension was to develop efficient algorithms for the special problem posed by the neural networks training. The following results will be better appreciated if considered in conjunction with those presented in Tables 3.3-3.6 and 4.6.

In order to compensate for the unstable nature of the QPROP search engine we took a tighter view on the degree of supervision and used $T = 2.0$ in all experiments. However, due to the simultaneous presence of the SA and the BB, we decided to allow the steplength restriction constant of the QPROP engine to be quite large, i.e. $b = 20.0$. Table 4.7 reports the No. of iterations/No. of function evaluations/No. of gradient evaluations and in order to make the comparisons easier, we retain also the stopping rule, that is $|\nabla f(x_k)| \leq 10^{-6}$. The three columns of the table contain results for three different sequences t_k, as indicated

in the corresponding labels.

The first important observation from the table is the vast increase in the number of solved problems in comparison to the SAQPROP algorithm. One can notice easily that the number of iterations required to solve the problems has also been improved dramatically. The run with $t_k = 0.01$ seems to show better performance overall, although it requires an excessive number of iterations to solve problem 12. This should be caused by the fact that it converges to a different local minimum.

The second important outcome is that, in every trial, problem 10—which SABB was not able to solve, nor GBB for that matter—was easily solved in a minimal number of steps.

As a general, final observation one should acknowledge the fact that SABBQPROP presents more successes than SABB and SAQPROP alone, while the speed of the extended algorithm seems to be also of a magnitude comparable to that of the basic algorithms. Although the number of function evaluations needed for this extension is larger at each iteration due to the addition of one more component search engine, we do think that this should not increase the computational load dramatically, at least for moderate sized problems. For problems of rather large scale we could easily use one of the inexact variants presented above, appropriately modified for two search engines.

4.3 Concluding Remarks

The motivation for the studies included in this chapter stems from our interest to develop efficient algorithms that can be applied to a variety of *real world applications* involving noisy optimization. To this end we, first, proposed new algorithms developed using the Supervisor Searcher Co-operation (SSC) framework, which constitute either modifications of the basic algorithm defined in Section 3.1, or extensions of it based on the SSC principle. In order to verify that they are robust and efficient enough we tested the resulting algorithms using the whole set

Table 4.7: Results from the three-algorithm extension SSC-SABBQPROP, with different step length (t_k) types for the Supervisor, on the deterministic problems; reported are the No. of function evaluations/No. of gradient evaluations/No. of iterations.

Prob. Nr.	SSC-SABBQPROP $t_k = \min(1.5/\sqrt{k}, 0.01)$	SSC-SABBQPROP $t_k = 0.01$	SSC-SABBQPROP $t_k = 0.1$
1	153/460/154	146/439/147	255/766/256
2	927/2782/928	45/136/46	792/2377/793
3	12/37/13	5/16/6	4/13/5/
4	FL	FL	FL
5	2144/6433/2145	737/2212/738	32/97/33
6	19/58/20	19/58/20	19/58/20
7	FL	FL	FL
8	FL	FL	FL
9	22/67/23	20/61/21	15/46/16
10	18/55/19	18/55/19	22/67/23
11	73/220/74	63/190/64	66/199/67
12	1/4/2	2177/6532/2178	1/4/2
13	116/349/117	284/853/285	96/289/97
14	186/559/187	153/460/154	174/523/175
15	588/1765/589	642/1927/643	513/1540/514
16	FL	228/685/229	33/100/34
17	200/601/201	1564/4693/1565	1223/3670/1224
18	54/163/55	54/163/55	199/5598/200
19	7/22/8	7/22/8	7/22/8
20	94/283/95	104/313/105	80/241/81
21	13/40/14	13/40/14	18/55/19
22	97/292/98	101/304/102	88/265/89
23	2/7/3	2/7/3	2/7/3

of non-trivial benchmarking problems described in Section 2.4, either in their deterministic or stochastic form. The three modifications and three inexact variants of the basic SSC-SABB algorithm we designed form an example illustrative of the flexibility of the SSC framework. The aim was to address particular requirements often arising in real world applications, such as, for instance, reduced computational load. The corresponding results, especially in the stochastic case, proved to be surprisingly satisfactory.

We demonstrated, also, that the SSC constitutes a powerful and flexible framework able to address a wide range of issues arising in the design of efficient and robust optimization algorithms. In its basic form it offers an appealing way to combine two arbitrary algorithms—one of which could exhibit unstable behaviour—in a new more efficient one. To illustrate this point we presented an algorithm resulting from the combination of the classical Stochastic Approximation algorithm and an algorithm praised for its speed in the neural networks training context. Our numerical results indicated that in the case of general functions (represented by the range of our benchmarking problems) the resulting algorithm showed marginally improved performance in comparison to the original one. Motivated by such considerations we designed another possible extension developed again within the SSC framework. It allowed for the combination of more than two "parent" algorithms. Subsequently, we established a generalization of our previous theoretical results about the convergence properties and the speed of the extension involving the combination of three algorithms. Our additional numerical experiments showed that this extension addressed the benchmarking problems with significant success.

Chapter 5

Neural Networks Learning

In this chapter we introduce concepts and discuss issues which will be necessary in order to develop, navigate and explain our experimental investigations concerning the application of the SSC type algorithms in the Artificial Neural Networks (ANNs) training which are presented in the next chapter.

5.1 A Theoretical Analysis of ANN Learning

The neural networks training problem is often paralleled to that of non-parametric non-linear regression while in other cases to that of the construction of discriminators in classification (see for example (Bishop, 1995) and (Ripley, 1993)). In *supervised* learning (see Section 1.3.8), which is of our particular interest here, it is assumed that we have a set of data pairs $D = \{(x_i, y_i)\}, \quad i = 1, ..., N\}$, also called training set, where $\{y_i\}$ are the *desired* outputs of the network. If we consider the neural network as a directed graph then the input and output vectors are represented as *nodes* organised hierarchically in the so called input and output *layers*. Between them it may exist one or more layers of nodes called *hidden* layers and hidden nodes respectively. The nodes have assigned some *activation function* used to calculate the *activation level* or output of the node with respect to its input. Thus the nodes are the main processing units of the neural network. In this hierarchical structure a particular combination of the outputs of the nodes of

each layer is propagated to become the input of each node of the next layer. Then a vector of adjustable parameters $w = [w_1, ..., w_M]^T$, called *weights*, represent the strength of the connections of this propagation procedure. These are the main parameters of the model which are adjusted during the *training* (or *learning*) phase. The aim is to estimate the parameter values of a particular function defined by the neural network *architecture*, which will produce optimal performance with respect to a predefined measure. It is, in fact, an unconstrained problem of optimization of the performance function. However, it could be a rather complex one mainly due to the uncertainty associated with the data on which the definition of the performance function itself is based. In the following we adopt the theoretical analysis of learning presented in (White, 1989), which is, also, suggestive of the relation between neural networks and statistics.

Assume now, that the observations of a phenomenon we are interested in, (x_i, y_i), are realizations of the random variables X_i, Y_i. They are jointly distributed according to the (unknown) probability law $\nu(X, Y)$. It is a well known fact that ν can be decomposed into a probability law $\mu(X)$which describes the behaviour of X, and the conditional probability law , which completely summarises the relation between Y and X. In this context the natural object of interest for a study of this relationship is the discovery of γ. However, in neural networks training the goal is often for the network to perform acceptable well in using X to predict or explain Y. In such cases the primary aspect of γ which becomes the focus of training is the conditional expectation of Y given X, denoted $E(Y|X)$,which is a function $g(X)$ of X, that is $g(X) = E(Y|X)$. Then each realization y_i can be represented as

$$y_i = g(x_i) + \epsilon_i .$$

where ϵ_i are random errors for which $E(\epsilon|X) = 0$. That is, ϵ is zero "on average" given any realizations of X, but it is non zero with positive probability for any particular realization x_i . The degenerate case where $\epsilon = 0$ for all realizations of X (as it happens in two-way classification problems or the so-called parity problem,

where Y is allowed to take only one of two possible values for each X), is a special case in the above context. This fact actually reveals how the neural network training can be considered as an optimization problem with either stochastic of deterministic noises, as we will show in detail bellow.

A neural network can be represented with a function f (usually non-linear) which, given a parameter vector w, associates each *input* vector x_i with an output vector $o_i = f(x_i, w)$. Note that w is not considered here as a random vector (such an approach would transfer us to the realm of the Bayesian approach to neural networks, which is out of the scope of the present discussion). We can now define a function π which expresses in an appropriate sense how well the neural network represents the relation between x_i and y_i using the outputs o_i, so that $\pi(o_i, y_i) = \pi(f(x_i, w), y_i)$. Given the set of observations D and the corresponding values of π a natural measure of performance is:

$$\lambda(w) = E(\pi(f(X, w), Y)) = \int \pi(f(x, w), y)\nu(dx, dy)$$

or

$$\lambda'(w) = E(\pi(f(X, w), Y)|X) = \int \pi(f(x, w), y)\gamma(dy|x)$$

where the integrals are Lebesgue integrals. The first formula expresses λ as the average performance over the population from which D has been drawn, and the second expresses the average performance given a particular value of x. The aim of training is consequently to estimate the parameter vector w^* which optimizes λ. If λ is an error function, this becomes a minimization problem while in the case of a likelihood function it is a maximisation one. We can then in general state the problem as:

$$\min_{w \in W} \lambda(w)$$

where W is the set of allowable parameter values, usually a subset of R^n, within

which hopefully exists the solution (assuming maximisation of a performance function):

$$w^* = \arg\max \lambda(w)$$

for the above problem, so that

$$\lambda(w) \leq \lambda(w^*) \ \ \forall w \in W.$$

Typically, $\pi(\cdot)$ depends on the available pairs of data.

There is an important and revealing observation to be made here: The function π is a function of the random variable Y, hence a random quantity itself, and as neither of the probability laws ν and γ are known, we can only *estimate* its expected value for particular values of w. Following this observation the problem becomes one of estimating the parameter value w^* for which the expected value of a random quantity, known only from observations at levels $w_k \ \ k = 1, ..., N$, attains its extremum (either minimum or maximum value). This is exactly the problem of stochastic optimization investigated initially by Robbins & Monro in (Robbins and Monro, 1951) and Kiefer & Wolfowitz in (Kiefer and Wolfowitz, 1952). A number of prominent researchers investigating the relations between neural networks training and statistics have revealed and discussed this correspondence (see for example (Kushner and Yin, 1997) and (White, 1989)).

As an example regarding the discussion above, let us consider the performance measure to be the squared error, i.e. $\pi(o, y) = ||o-y||^2$. This is the most frequently used measure in training of Multilayer Perceptrons (MLPs), on which the present study is focused. Then $\lambda(w)$ can be rewritten as follows:

$$\begin{aligned} \lambda(w) &= E([Y - f(X, w)]^2) \\ &= E([Y - E(Y|X)]^2) + E([E(Y|X) - f(X, w)]^2) \end{aligned}$$

The first part of the second formula in the right hand side is just the variance of Y while the second part describes how well "on average" $f(X, w)$ approximates

the regression function $g(X) = E(Y|X)$ (often called the bias of f. Recall that expectations are taken with respect to the unknown probability law ν from which a particular sample D is drawn. So the quantities in the second formula are estimators of the real population parameters. This is the primary source of stochastic noise in $\lambda(w)$. Observe that if $f(X, w) = E(Y|X)$, that is f is an unbiased estimator of $E(Y|X)$ then the variance of Y is the only source of noise in λ. In the special case, mentioned early in this section, where $\epsilon = 0$ for all x_i then $E([Y - E(Y|X)]^2)$ becomes zero so that the bias is the only source of noise in λ. If additionally to that the data available is an exhaustive set of the possible cases, i.e. it is the whole population, the noise in λ is deterministic and is, in fact, the square of the error of approximating g with f. If the network architecture is flexible and rich enough to contain g then f can become an unbiased estimator of g and in this special case the training problem becomes a deterministic optimization problem. The above mentioned parity problem and its variations is an example of this special case.

In the context of π being a random quantity, and the fact that it is based on the incomplete information available to us, we can only obtain an estimator $\hat{\lambda}(w)$ of its expected value $\lambda(w)$. In the light of the available data, a reasonable thing to do is to estimate λ with the corresponding sample mean, that is:

$$\hat{\lambda} = N^{-1} \sum_{i=1}^{N} \pi(f(x_i, w), y_i).$$

Hence $\hat{\lambda}(w)$ is also a random quantity. The characteristics of $\hat{\lambda}$ depend, in general, on how representative the available data is with respect to the probability law ν, and how flexible is the set of allowable network output functions $f(X, w)$. The MLPs training problem can then be approximated by:

$$\max_{w \in W} \hat{\lambda}(w).$$

Thus one may use (an approximator of) the maximizer of $\hat{\lambda}$ as the estimator of the real maximizer w^* of λ. This is obviously a stochastic optimization problem,

though for a fixed set of observations, $\hat{\lambda}$ may appear to be a deterministic function of w. The quality of the solution depends on how close the estimator $\hat{w}^*$ is to the real solution w^*, NOT necessarily on how close the estimator is to the real maximizer of $\hat{\lambda}$ for any fixed set of observations. This provides an explanation to the fact that it is common in neural networks literature to suggest "early stopping" of the training process, in an attempt to avoid the poor quality of $\hat{w}^*$ if this, for some reason, happens to be far away from the real solution w^*. Similar situations can be found in many application areas like in the Maximum Likelihood estimation problem, where what matters is not only the speed of the algorithms used on deterministic optimization problems, but also (and, in fact, more importantly) the overall robustness of these algorithms. For instance, it seems that the Newton's algorithm often fails to produce better estimates in the MLPs training problem than the slower Stochastic Approximation algorithm

5.1.1 Backpropagation

The most well known and widely used method for training multilayer feedforward neural networks is the *Error Backpropagation* (EBP) or *Generalized Delta Rule* which was introduced by Werbos (Werbos, 1974) but only found widespread use after it was independently re-invented by Rumelhardt et al. (Rumelhardt et al., 1986). Our special interest in this learning algorithm is because it is one of the more suitable and general rules that can be applied to Multilayer Perceptrons, which are the most widely used neural networks for regression and classification purposes. Its power resides in its ability to efficiently handle the learning procedure in *ordered–systems*, such as the multilayer feedforward networks, where the arrangement of the nodes admits a hierarchical structure. In these, to produce and output the flow of information is allowed only in one direction, from input to output layers. This is usually called a *forward pass.* Then the error associated with each one of the nodes is calculated by applying the chain rule of differentiation and *backpropagating* the corresponding error in the opposite direction, that is from the output to the input layers. This is called a *backward pass.* Finally,

the weight values are updated according to the calculated errors. A full cycle of training, often called *epoch* comprises a forward and a backward pass through all the available training data pairs and the corresponding weight updates. As we shall discuss in more detail bellow there are two ways to perform the weight updates. The first, called *online* training corresponds to one update after a single pairs is propagated through the network, while the second, called *batch* training corresponds to a single update after all the available data has been processed. The revolutionary (but with hindsight obvious) idea in the Backpropagation method, is the chain rule of differentiation as commented by its inventor:

> At the core of backpropagation is a method for calculating derivatives exactly and efficiently in any large system made up of elementary subsystems or calculations which are represented by known, differentiable functions (Werbos, 1990, p. 1550).

According to (Reed and Marks, 1999) the term 'Backpropagation' refers usually to two different things. First, it describes, as we have already mentioned, a particular method to calculate the derivatives of the training error with respect to current weight values. Second, it defines a training algorithm (sometimes equivalent to gradient descent, as we shall see below), which is using the calculated derivatives to minimise the error function.

In our previous set up, we may define:
y_k^t = the desired output of pattern pair t corresponding to the kth output unit,
s_k^t = the network output of the kth output unit,
w_{ij} = the weight of the link from the ith unit (being either input or hidden) to the jth unit(being either hidden or output). Notice here that we do not refer to any particular kind of connection weights, to allow for general multilayer networks.
Then for a network with one hidden layer we can define:
$\sigma_i, i = 1, \ldots, p.$ is the ith input unit activation state,
$\bar{s}_j, j = 1, \ldots, q.$ is the jth hidden unit activation state,
And:

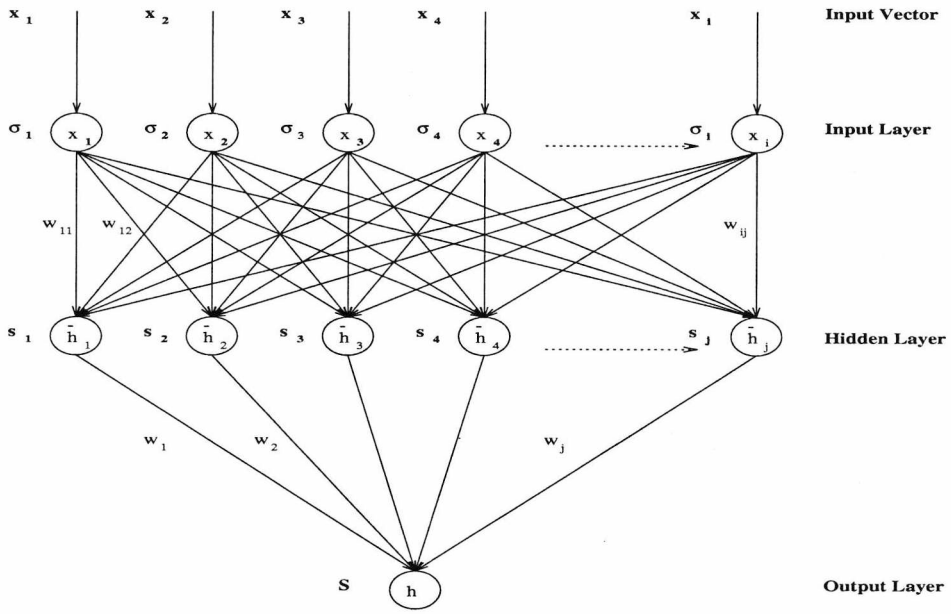


Figure 3: Schematic representation of a Feedforward Neural Network with one hidden layer.

$\sigma_i = \tilde{f}(x_i)$,

$\bar{s}_j = \bar{f}(\bar{h}_j)$ where $\bar{h}_j = \sum_i \bar{w}_{ij}\sigma_i$, and $\bar{w}_{ij}$ is the connection weight between the jth hidden and the ith input units,

$s_k = f(h_k)$ where $h_k = \sum_j w_j \bar{s}_j$, and w_{jk} is the connection weight between the jth hidden unit and the kth output unit.

A commonly used set of activation functions are the following:

$\tilde{f}(z) = z$ (identity input unit activation function),

$\bar{f}(z) = 1/(1 + e^{-z})$ or $\tanh(z)$ (sigmoid hidden unit activation function),

$f(z) = z$ (identity output unit activation function).

See Figure 3 for a graphical representation of the structure of an MLP with one node S in the output layer.

First, to consider, Backpropagation as a method for calculating the error derivatives we define the per-pattern t error function as follows:

$$E^t = \sum E_k^t \tag{5.1}$$

where as above k indexes the output nodes. Then the total or epoch error is:

$$E = \sum_t E^t, \quad \text{or} \quad E = \frac{1}{N} \sum_t E^t \tag{5.2}$$

where for $t = 1, \ldots, N$, the whole set of examples forms the *epoch*. Then for any weight w the error derivative is:

$$\frac{\partial E}{\partial w} = \sum_t \frac{\partial E^t}{\partial w} \tag{5.3}$$

And:

$$\frac{\partial E^t}{\partial w} = \sum_k \frac{\partial E^t}{\partial a_k} \frac{\partial a_k}{\partial w} \tag{5.4}$$

where a_k denotes the input to node k. Then, in our previous setting of a single hidden layer network, for $\bar{w}_{ij}$ and w_{jk} being the input-to-hidden and hidden-to-output weights respectively the error derivatives can be calculated as follows:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial \bar{s}_j} \frac{\partial \bar{s}_j}{\partial w_{jk}} \tag{5.5}$$

and:

$$\frac{\partial E}{\partial \bar{w}_{ij}} = \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial \bar{s}_j} \frac{\partial \bar{s}_j}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial \bar{w}_{ij}} \tag{5.6}$$

In these formulas the index t has been dropped for simplicity. After defining a suitable error function simple manipulations of the above formulas give the exact expressions used to update the weights in every case. The sigmoid functions mentioned above are usually chosen due to the simplicity of their first derivatives which will finally be used in these exact expressions.

The advantage of backpropagation applied in an *ordered system*, such as feed-forward networks, is that it allows us to determine the change δw of every weight w independently and hence possibly in parallel, by simply applying the differentiation chain rule to the appropriate error function. Its disadvantage is that it requires the latter function to be continuously differentiable.

Now, considering the Backpropagation as a training algorithm (denoted henceforth BPR) which aims to minimize E by applying adjustments to the weights, the corresponding learning rule can be defined as follows:

$$w^{new} = w^{old} + \delta w^{old} \tag{5.7}$$

Note that 'old' and 'new' here denotes an update in general either in the on-line or the batch mode of the algorithm. In order to find a minimum (ideally global) on the surface defined by the corresponding error function, a gradient descent path is followed by defining:

$$\delta w = -\eta \frac{\partial E}{\partial w} \tag{5.8}$$

where η is a suitably chosen small positive value (usually less than 1) named the *learning rate* in the neural networks jargon. The method works as the gradient always points in the direction of strongest change of the function, hence, the negative gradient follows a descent direction on the error surface .

The most commonly used, and a natural choice in regression problems, for the error measure would be the squared error:

$$E = E^t = \frac{1}{2} \sum_k (y_k^t - s_k^t)^2 \tag{5.9}$$

for which we discuss in more detail along with other possible error functions in Section 5.3.1.

The version of the algorithm called *pattern mode* (it assumes updates after each pattern (example pair) has been presented to the network) is also characterised as *stochastic updating*, because it assumes a random search of the error surface via a randomization of the order of presentation of the patterns. So the weight updates do not fall into a pre-specified cyclic pattern. This is often useful and produces better results when the order of the patterns is irrelevant for the application, like in the classification problems. Alternatively in *batch or epoch updating* the error is computed or averaged over the whole training set, before applying the updates

the error function.

In the latter case it can be considered that the gradient is calculated exactly and the weight changes are proportional to the gradient of the error function, so this mode approximates gradient descent for small step sizes (learning rates). In general, the batch mode approaches pure gradient descent when the learning rate becomes infinitesimal (Reed and Marks, 1999, p. 57), while the pattern or *on-line mode* is related to stochastic gradient descent or to the Stochastic Approximation algorithm (Robbins and Monro, 1951). This analogy has been noticed and investigated both in the optimisation field (Kushner and Yin, 1997) and in the neural networks community (White, 1989). Bishop (Bishop, 1995, p. 264) notes that on-line training, also called *sequential learning*, "... is reminiscent of the Robbins-Monro procedure", and "The analogy becomes precise and we are assured of convergence, if the learning rate parameter is made to decrease at each step of the algorithm ...". According to relevant theoretical results appropriate requirements for the rate of decrease are satisfied, for example, by choosing $\epsilon_k \propto 1/k$, where k is the iteration number. However, Bishop also notes that **in practice** the guarantee of convergence is often sacrificed in exchange to faster convergence. In general, using a constant learning rate, is usually reported to be more efficient.

5.1.2 Improvements and Extensions

Stochastic Algorithms

From an analytical point of view the on-line variant of Backpropagation is no longer a simple approximation to gradient descent. The gradients based on the single patterns can be viewed as noisy estimates of the true gradient. So, if the gradient is strong at a point the sum of these estimates has a positive projection on the gradient causing the error to decrease after most of the weight updates. However, there can still be a number of single-pattern estimates with either negative projections, or large orthogonal deviations from the true gradient, forcing the error to increase after some of the updates (Reed and Marks, 1999).

One of the disadvantages of the stochastic nature of single-pattern learning is that, especially when a relatively large learning rate is used to accelerate training, it tends to result in weights jittering around the error surface and occasionally moving uphill. The magnitude of the jitter is proportional to that of the learning rate used and if significantly large can obscure any useful information about the true (deterministic) gradient. Additionally, the weight vector will never settle to a stable value even when a 'good' minimum has been reached. Since the randomness arises by the constantly changing random order of presentation of the patterns, cyclic fixed orders can cause convergence to limit cycles. The addition of a *momentum* term, which is equivalent to an exponentially weighted moving average of recently presented patterns and will be discussed further below, is often added to Eq. 5.8 in order to smooth the gradient estimates and dampen the oscillations (Ripley, 1993). Although this can be considered as an alternative method to epoch (batch) updating, which usually follows a smooth path on the error surface, in some implementations of Backpropagation are used simultaneously.

On the plus side, however, the stochastic behaviour in the on-line gradient estimation process allows the algorithm to escape from shallow local minima and hence to have higher probability of finding better (even optimal) solutions (minima). When the pure gradient descent (bach mode), on the other hand, arrives at a local minimum (even a poor one) it simply gets stuck. This, in general, is not an exclusive characteristic of the latter, but can be considered typical behaviour of algorithms with the property that each step is guaranteed not to increase the objective function. In the following we shall discuss some of these algorithms, mostly drawn from the numerical optimisation field, which are usually called *exact* or *deterministic*, and when at a local minimum tend to remain indefinitely there (Bishop, 1995, pp. 264, 272–292).

The difference between the two training modes (on-line and batch) becomes minimal when the learning rate is very small. In general it should be noted that in the optimisation community gradient descent is not highly regarded due to its slow rates of convergence, especially in the cases where the Hessian of the function

is ill-conditioned, i.e. when the gradient changes much faster in some directions than others such as in the case of the so-called 'ravines' of the error surface (Reed and Marks, 1999, p. 155).

Heuristic Algorithms

A large number of modifications and extensions to the basic Backpropagation algorithm have been proposed from the early years of its appearance to present date. They mainly attempt to address the issues of low speed, oscillations of the weights, and convergence to local minima, although in some cases there are also concerns about theoretical guarantees for global convergence. They can be categorised, broadly, in two classes: (i) those based on heuristics and, (ii) those adapted directly from the field of numerical optimisation. In the following we will briefly introduce the most important of these training procedures and attempt to draw some conclusions about their comparative performance. The aim of this discussion is to identify an algorithm adequately diverse and more efficient than the basic BPR in order to enhance the basis of comparisons with the SSC-based training algorithm proposed in this study.

Most of the modern texts about neural networks being either introductory (Fausett, 1994), or look into ANNs from the perspective of a particular application area, such as (Bishop, 1995; Ripley, 1993) for pattern recognition, (Refenes, 1995) for financial forecasting, or, finally, provide practical guides for the use of the techniques, such as the recent "Neural Smithing" (Reed and Marks, 1999), include some sections on modifications of the basic BPR which due to theoretical, but mainly, to practical reasons seem to have survived the test of time. Most of these texts agree on the above categorisation of these algorithms into 'heuristic' and 'deterministic' or 'exact'. An alternative classification would, naturally, be into 'first' and 'second' order methods depending on the order of information they use about the error function gradient. Our exposition shall follow here the former while commenting about the latter as we proceed.

The first and most commonly used extension to the basic BPR has already

been mentioned previously to be the addition of a *momentum* term to the weight update formula which aims to restrict oscillations and improve speed. The weights w_k update formula becomes:

$$w_{k+1} = w_k + \delta w_k + h\delta w_{k-1},$$

where δw_{k-1} is the momentum term and h a gain parameter, usually less than 1. This has been reported to generally achieve its goals but it introduces an additional (to the learning rate) parameter to be tuned. Subsequent modifications that belong in both of the above mentioned classes have basically proposed techniques to adapt either or both of these parameters dynamically during the evolution of training, often taking into account the topology of the error function surface. Among them first to be mentioned is the Delta–bar–Delta (Jacobs, 1988) rule which proposes individual changes to the stepsizes for each weight according to sign changes between the current partial derivative and an exponential average of the partial derivatives of past iterations. The Resilient Backpropagation (known as RPROP) (Riedmiller, 1994) monitors the changes in the sign of the gradients only to propose an increase in the learning rates when no change is observed and a decrease according to an exponential schedule in the opposite case. Quickprop (Fahlman, 1988) (QPROP) introduces independent stepsizes for each direction of the gradient (every weight) as a result of an approximate secant solution to a local quadratic approximation of the error surface.

In empirical studies, Delta–bar–Delta was among the fastest method for classification problems but slow in other cases (Alpsan et al., 1995). According to some reports it showed greater sensitivity to its parameter values than RPROP and QPROP (Reed and Marks, 1999, p. 139). The latter seem to be the fastest and most reliable heuristic algorithms, reported to be faster in some cases even to second–order gradient methods such as the conjugate gradient or Levenberg–Marquardt (see below) (Reed and Marks, 1999, pp. 144, 147, 152). For other heuristic adaptive methods such as the 'Bold Driver' (Vogl et al., 1988), the Silva

and Almeida (Silva and Almeida, 1990) modification to Delta–bar–Delta rule and the 'SuperSAB' (Tollenaere, 1990), the results reported in the literature are inconsistent and rather problem dependent (Alpsan et al., 1995).

Deterministic Methods

On the other hand, in the 'deterministic' category one could start with adaptations of the Newton's method for neural networks (Bello, 1992; Buntine and Weigend, 1994). This suggests learning rates based on the (exact) calculation of the Hessian matrix of the second partial derivatives of the error function $H = \nabla^2 E$ as a result of a quadratic local approximation. The weights $\{w_k\}$ update formula, in this case, is :

$$w_{k+1} = w_k - H^{-1} g_k,$$

where $g_k = \nabla E$ defines the search direction.

However, exact calculation of the Hessian can be expensive in large scale problems and Newton's method requires H to be positive definite, so approximations are often used. Among them, the Gauss–Newton and Levenberg–Marquardt techniques seem to give good results in comparison to the basic Backpropagation in a neural network training setting for only moderately sized problems (Hagan and Menhaj, 1994). Another class is the quasi–Newton, often also called *variable metric* methods, among which major variations are the Davidon–Fletcher–Powell (DFP) and the Broyden–Fletcher–Goldforb–Shanno (BFGS) methods (see, for instance, (Reed and Marks, 1999) for definitions regarding ANN training). These build approximations of the Hessian matrix iteratively using only first–order gradient information. According to (Alpsan et al., 1994) in a neural network setting they both converged to good local minima only half of the runs tried, they were slower than standard Backpropagation in classification problems and exhibited poor generalisation. Finally, methods that use conjugate gradients (CG) to define the search directions such as the Fletcher–Rieves (FR) or the Polak–Ribiere (PR) variants, are usually praised for their speed, which in some cases can be greater

than that of second–order methods. They either use line search procedures to determine the learning rate or not as it is the case with the Scaled Conjugate Gradient algorithms (Moller, 1993). A large number of comparative studies with the Backpropagation has been published. It seems that the outcomes of these comparisons are problem dependent. In (Lee and Lippmann, 1990), for example, CG algorithms appear to work effectively in simple problems while they quickly converge to poor local minima in more difficult problems.

5.1.3 Comparative Studies

A large number of other modifications have, and still are, being put forward in the literature (indicatively see (Lee, 1997; Cho and Chow, 1999; Ghorbani and Bayat, 2000; Amari et al., 2000)). They correspond to a diverse range of solutions and starting points, but they are usually tested in specific application areas, hence no general deductions can be made about their properties.

During the past years a significant amount of reports have been accumulated about comparisons of the main modifications presented previously either with BPR and its variations, or among themselves, or both (Magoulas et al., 1999; Sperduti and Starita, 1993; Jacobs, 1988; Battiti, 1989; Robitaille et al., 1996; Kinsella, 1992; Vrahatis et al., 2000a; Schiffmann et al., 1993). It should be noted, as an initial word of caution, that assessments in these works are based on such a wide and diverse set of measures that renders them usually incomparable. In general, the results presented in these and other studies, are at best controversial. An initial observation could be that when the authors consider learning in neural networks as a deterministic problem and present results only on the minimisation of the error on the training set, the deterministic methods appear to be favorable. Similar claims are made, also, when the examples used do not involve noise, or use exhaustive samples (as is the case, for example, in the XOR problem, or its extension, the n-parity problem), and/or the problems are of small to moderate size. Methods that involve line searches tend to show poor performance, as it is expected, in cases with noisy data.

Finally, general remarks in (Bishop, 1995, p. 264), (Reed and Marks, 1999, pp. 152,156,168,182), and (Alpsan et al., 1995) tend to agree in the following two points. First, for classification problems, where tolerance in the final error achieved is higher, simpler methods, mainly based on heuristics, tend to be the favorable choice. However, in the case of 'function approximations' (regression) deterministic methods seem to perform better. Second, fast ("greedy") methods, such as the majority of the deterministic algorithms we discussed, show tendency either to overshoot the minima, or, when combined with a monotone requirement for error reduction at every iteration, to get trapped at poor local minima.

As a result of our previous review we decided to choose the Quickprop (QPROP) algorithm as the one to form, along with the basic BPR, the group against which the SSC training algorithm proposed here will be tested. The main properties of QPROP supporting this choice are:
(*i*) its simplicity (Fahlman, 1988),
(*ii*) it uses second–order information,
(*iii*) it uses different learning rates for every weight in the network so it belongs to the more sophisticated end of the spectrum. In contrast, the SSC algorithm we propose here uses a global learning rate for all the weights (as we shall see in the next section), and finally,
(*iv*) whenever QPROP was included in comparative studies, either with the Back-propagation algorithm and its variants, or the deterministic type of algorithms, it was reported either to outperform them or at least to exhibit equal speed and generalisation ability (Fahlman, 1988; Schiffmann et al., 1993; Hasenjager and Ritter, 1999).

Although it has been reported to present some sensitivity to the values used to bound the learning rates produced by its update rule (Vrahatis et al., 2000b; Hasenjager and Ritter, 1999), and more elaborate schemes have been proposed to remedy the situation (Vrahatis et al., 2000b), it seems that it is generally accepted that Quickprop performs favourably if its constructor suggestions are

followed (Reed and Marks, 1999, pp. 147,152). Thus, we use the basic form of the algorithm which we shall introduce formally in the next section.

5.2 Definition of the Algorithms

We are now in the position to formally define the three algorithms selected to be used in the experiments presented in the next Chapter. Namely we introduce the standard Backpropagation, the Quickpropagation (QPROP), and the SABB as adapted for use in neural network training. The formalism we adopt helps to identify the similarities with the algorithms used in the first part of this thesis. We also use the standard notation for the neural network parameters with w_k denoting the weight vector at iteration k. Note that all the algorithms we used update the weights in *batch* mode, so the values of weights at iteration k correspond to those resulting after an update was performed at the end of the kth *epoch*.

BPR

The first algorithm is the classical standard Backpropagation training algorithm (henceforth denoted BPR to identify it from the Error Backpropagation (EBP) method for derivative calculation discussed previously) defined as follows:
Given an initial weight vector w_0,

$$w_{k+1} = w_k - Zg_k, \; k = 0, 1, 2, ...,$$

where $Z = \eta I$ is a diagonal matrix and I denotes the unit matrix. Also, g_k is, as usual, an estimator of the gradient $\nabla E(D, w_k)$ at w_k, with $E(D, w_k)$ the network error function as defined by Eq. 5.2. This notation is used to emphasise the dependence of the error function to the training set D. The same notation for the error function is used for the definitions of the remaining algorithms as well. As already mentioned previously, η is typically a small positive constant, usually less than 1.

QPROP

The Quickpropagation can be defined as the following gradient-based algorithm: Given an initial weight vector w_0,

$$w_{k+1} = w_k - B_k g_k, \; k = 0, 1, 2, \ldots ,$$

where

$$B_k = \begin{cases} B_0 & \text{if } k = 0, \text{or} \Delta w_{i,k} = 0 \\ B'_k & \text{otherwise} \end{cases} ,$$

where $B_0 = lI$, I is the unit matrix, l is a small positive constant, usually less than 1, indicated as "alternative stepsize" in the tables presented in the next chapter, and $w_{i,k}$ and $g_{i,k}$ are the ith components of w_k and $\nabla E(D, w_k)$ (as defined previously) respectively. For $k \geq 1$ $B'_k = (b'_{ij,k})$, with

$$b'_{ij,k} = \begin{cases} 0 & i \neq j \\ b'_{i,k} & i = j \end{cases} ,$$

where

$$b'_{i,k} = \text{sign}(b_{i,k}) \min(b|\frac{\Delta w_{i,k}}{g_{i,k}}|, |b_{i,k}|),$$

and

$$b_{i,k} = (w_{i,k} - w_{i,k-1})/(g_{i,k} - g_{i,k-1}),$$

with $\Delta w_{i,k} = w_{i,k} - w_{i,k-1}$, and b a constant with most commonly suggested values $1.5 \leq b \leq 3.5$. We followed the advise given in (Fahlman, 1988) and used a value of $b \approx 1.75$, which was reported to give the best results for a class of benchmark problems within the neural networks training context.

SABB

The original SSC-based SABB algorithm defined in Section 3.1 can be adapted to the neural network training formalism in the following way. First, we have to

redefine the Supervisor algorithm (SR), which is the classical Stochastic Approximation (SA) algorithm:
Given an initial weight vector w_0,

$$w_{k+1} = w_k - t_k g_k, \ k = 0, 1, 2, ...$$

where $g_k = \nabla E(D, w_k)$, an estimate of the gradient of the network error function $E(D, w_k)$. In the following we assume, as usual, the following assumptions to hold for the gain sequences $\{t_k\}$:
i) $t_k > 0$;
ii) $\sum_{k=0}^{\infty} t_k = +\infty$;

Then the search engine (SE) can be formulated based on the following gradient algorithm:
Given w_0

$$w_{k+1} = w_k - r_k g_k, \ k = 0, 1, 2, ...,$$

where $r_0 = 1$ and for $k \geq 1$

$$r_k = |w_k - w_{k-1}|^2 / ((w_k - w_{k-1})^T (\nabla E(D, w_k) - \nabla E(D, w_{k-1})).$$

where the step length r_k is the BB stepsize, as usual, proposed by Barzilai and Borwein in (Barzilai and Borwein, 1988) (see also Section 3.1). We are now in the position to define the SSC-based algorithm in the neural network training context as follows:
Let $T_k > 0$ be given for $k = 0, 1, 2, ...$. Then define the following algorithm (SSC-SABB):

$$w_{k+1} = w_k - t_k g_k \text{ if } T_k E(D, w_k - t_k g_k) \leq E(D, w_k - r_k g_k), \ (k = 1, 2, ...),$$

otherwise

$$w_{k+1} = w_k - r_k g_k.$$

In our computations, $\{|r_k|\}$ is normally forced to be bounded to avoid instabilities.

5.3 Performance Measures and Comparisons

5.3.1 Error Functions

The role of the objective or error function used in the learning process is, as already noted, to define the difference between good and bad performances with respect to the targets and thus to guide the search for a solution. In this sense it has a fundamental effect on the outcome so it is important to choose the function accurately reflecting the experiments design goals. Furthermore, it was already pointed out that whether the error is evaluated over the training or the test set or on both has implications for the type of performance we are interested to assess, be it memorisation or generalisation.

The most commonly used error function has already been described previously while introducing the Backpropagation algorithm. This is the sum–of–squares which, given the notation introduced previously, can be written for the batch training mode as:

$$E = \sum_t \sum_k (y_k^t - s_k^t)^2$$

where, as usual, t indexes the patterns and k the output nodes. Assuming linear activation functions in the output nodes $g(a_k) = a_k$, where a_k is the input to the kth output node, the error derivative with respect to a_k is:

$$\frac{\partial E}{\partial a_k} = \sum_t (y_k^t - s_k^t)$$

This function is obtained by the maximum likelihood principle under the assumption that the targets are generated from a smooth deterministic function

with added Gaussian noise (Bishop, 1995). Although this is a reasonable starting point for regression problems, it is clearly inappropriate for the case of classification where the targets are binary variables and the activation of the output nodes is often interpreted as probabilities.

For the two–class problems we usually have a single target assuming the values 0 or 1 and a single corresponding output node. The probability model used for the target in such cases is a particular case of the binomial family called the Bernouli distribution. The error function corresponding to this model is the so–called "cross entropy" function (Hopfield, 1987; Solla et al., 1988):

$$E = \sum_t y^t \ln s^t (1 - y^t) \ln(1 - s^t)$$

In (Bishop, 1995, pp. 82–84) it is shown that for the activation of an output node to be interpreted as a probability it is appropriate to use the logistic activation function, $g(a) = [1 + \exp(-a)]^{-1}$, which is often associated also with the *logistic discrimination* in statistics. Then the derivative of the error function can be written, after some algebraic manipulations, as:

$$\frac{\partial E}{\partial a} = \sum_t (y^t - s^t)$$

Following the same line of reasoning in the multiclass case, where the targets can belong to 1-of-k classes and the network has k output nodes, an error function analogue to the "cross entropy", based on the assumption of binomially distributed target values, is the following:

$$E = \sum_t \sum_k y_k^t \ln \left(\frac{s_k^t}{y_k^t} \right)$$

In this case, the appropriate activation function to use, so that the output values sum up to unity, is a generalisation of the logistic, often called the "softmax"

function (Bridle, 1990) defined as follows:

$$g(a_k) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$

with $k' \neq k$, which can be rewritten as:

$$g(a_k) = [1 + \exp(-A_k)]^{-1}$$

with

$$A_k = a_k - \ln\{\sum_{k' \neq k} \exp(a_{k'})\}$$

Again after some algebraic manipulations the error derivative turns up to be :

$$\frac{\partial E}{\partial a_k} = \sum_t (y_k^t - t_k^t)$$

It is easy to see that considering all three cases we examined there is a natural pairing between the error functions and the corresponding activation functions. These functions have a number of desirable properties that justify their popularity. First, they are smooth and easily differentiable. Second, they are well–understood and they allow valuable theoretical study by simplifying analysis considerably. Although, as pointed out in (Michie et al., 1994, p. 86), "it might seem more natural to use a percentage misclassification error measure in classification problems" it is advantageous to use the above functions during the learning process and utilise the recognition rate as an additional measure of evaluation. This exactly the strategy we adopted in the experiments we shall present in the next chapter.

As consequence of the above discussion, in this study we adopted an experimental strategy according to which we run a number of simulations for every benchmarking problem we use and measure the errors both on training and test sets at different stages of the simulation. We then use these estimates to construct statistical comparisons of the mean performance of the algorithms we test. Subsequently, we assess the convergence properties of the algorithms by comparing

the number of runs they converged and the epochs required to achieve a pre-specified level of error in the **test** set, thus evaluating their generalisation ability. Finally, in the classification examples we also report the misclassification rate on the unknown data.

5.3.2 Assessment Methods

A 'naive' estimator of generalisation would be the level of error reached by the network over the training set. But, this measure is well known in statistics that is downward biased measuring the minimum risk, that is the expectation of the error over the whole ensemble of possible cases, so it is overoptimistic with respect to the generalisation ability.

A natural and more effective way of assessing performance in cases *out–of–sample*, has been proposed by Hecht-Nielsen (Hecht-Nielsen, 1990), as far as training (learning phase) is concerned. It is suggested to use two sets of examples. The first will be the training set and the other the validation set of examples. These two sets are disjoint and the main idea consists of measuring the value of the error function over the validation set, which contains unknown cases, at intervals during the training. As long as the error over the validation set decreases, training is continued, while, when it begins to increase, although it might continue decreasing over the training set, learning is stopped. The fact that this is the case, is illustrated with examples in (Geman et al., 1992). The authors show that the total error in the whole ensemble of possible cases will follow this path (decrease-increase) through a decomposition of it to its bias and variance components. The bias component is minimized as a result of continuing the adjustment to weights in order to fit the given data, while the variance is increased, due to an increasing sensitivity of the resulting network, introduced by the more close fit to certain cases represented in the training set. This is the case when, trying to fit noisy data, the network begins to learn the noise as well as the underlying signal. The best generalisation performance is then accomplished when a compromise between variance and bias is achieved, thus minimizing the total error.

5.3.3 Comparisons Methodology

Following the preceding discussion it is not difficult to deduce that, unlike the simple comparison of optimization algorithms of different benchmarking functions, effective comparisons of neural networks training algorithms bear significant complexity due to the diversity of the additional parameters involved. In such cases, a well-established field to draw ideas from is that of designs of experiments, being the area of statistical inference which studies in a structured way the effects of different conditions (often called 'treatments') to the outcomes of experiments, denoted as 'responses'. A well-known result in this field is that given two treatments and their corresponding responses Y_1 and Y_2 we can analyse their differences $(Y_1 - Y_2)$ as independent samples. Denoting the variance, covariance and correlation by 'var', 'cov', and 'corr' respectively, we can write:

$$\text{var}(Y_1 - Y_2) = \text{var}(Y_1) + \text{var}(Y_2) - 2\text{cov}(Y_1, Y_2)$$

Then it is obvious if $\text{corr}(Y_1, Y_2) > 0$ we will obtain a more accurate estimate of the mean difference, since it will result in lower variance. The formula also demonstrates that the estimation procedure will be more effective if the treatments affect the mean responses but not their variance.

In order to transfer the comparison of neural networks simulation experiments to this framework, we should first consider the characteristics of the networks we would like to compare as different treatments to the problem of learning the relations underlying the data. According to our previous discussion, the parameters that affect the outcome (responses) of the experiments, that is the observed error (and its bias/variance components), are the network architecture, quality and quantity of the available data, and the training algorithms used. Therefore, if our focus is the comparative assessment of the training algorithms, as it is the case in this study, we should limit the influence of the remaining parameters to the resulting responses. To this end, we keep invariant across all the repetitions

of our experiment both the network architectures and the training/test set partitions for the same benchmarking problem. Furthermore, since the starting point, namely the initial weight vector, can affect the quality of the solution found by a training algorithm we use common random seeds for all the methods compared, changing them only over the successive runs of the same simulation experiment. This corresponds to the variance reduction method known as *common random variates* in the field of stochastic simulation (see (Ripley, 1993) for additional details). Finally, it is notable that by following the above methodology we further achieve to restrict the effects to the variability of the observed responses (being either the training or the generalisation error) only to those induced by the compared training algorithms themselves, causing our analysis of their differences to be more effective.

Chapter 6

Empirical Comparisons

In this chapter we present a series of empirical investigations aiming to assess the the SSC algorithm we propose for training feedforward neural networks. The properties we are interested in are efficiency and robustness in training under various types of noises, but mainly and most importantly the *generalisation* ability of the resulting trained network, since the latter constitutes the ultimate goal of learning. Furthermore, we compare our proposal with the two algorithms chosen as benchmarks, namely the standard batch Backpropagation (BPR) and the Quickprop (QPROP).

6.1 Experimental Design

For every problem we examine and for all the algorithms we repeat the training process ten times, initialising the weights to random values uniformly distributed in the interval $[-0.01, +0.01]$. This is a common practice in the neural network community. The results obtained from these repetitions (also called 'runs') provide the basis to perform valid statistical analyses of the performance characteristics of the training algorithms we are interested in. The quantities included in the following tables correspond to mean values calculated over the whole set of runs and, where appropriate, are annotated by the corresponding standard deviations. It should be noted here that all versions of the algorithms start from the same

initial weight vector for the same run. For example, for the first of the ten runs all versions of BPR, QPR, and SSC-SABB start training using the same vector of randomly assigned weights. However, this vector differs from the one used in the second training run. This is done, following the discussion in Section 5.3.3, in order to reduce the variance of the resulting performance measure estimator and hence the variance of the estimators of their differences. As a consequence of the same argument, we also decided to use the same partition of the available data to disjoint training and test sets for all the algorithms during the same run. This way we restrict the parameters influencing the differences observed in performance only to the effects due to the use of different training methods.

The experiments we conduct aim to evaluate the performance, first, with respect to a range of of values for the adjustable parameters of the SSC-SABB algorithm, namely the preference switch T_k and the supervisor algorithm's stepsize t_k. The objective is to explore the level of sensitivity to these parameters under various conditions (different tasks), and provide possible guidelines for effective values. At the same time, we try different values for the control parameters of the benchmark algorithms, in order to facilitate a comparative view of their sensitivity as well. For BPR we use two different learning rates η, while for the QPROP two alternative stepsizes l. The full set of different versions we have used for each algorithm along with the corresponding parameter values are included in Table 6.1.

In order to observe the effect these parameters have in the *training speed* and *accuracy* of the algorithms, we report the error levels obtained at the *middle* and at the *end* of the simulation runs. Although in some of our examples we use the cross–entropy as error function (see Section 5.3.1) during training, due to the nature of the problems (classification), as we shall discuss below, in the relevant tables, for uniformity of exposition we present the corresponding Mean Squared Error (MSE) in all cases.

As a final remark, it should be noted that we chose not to report on the CPU time required for our simulation runs as this seems to be controversial issue among

Table 6.1: Parameters of algorithms tested; η: learning rate of BPR; l: alternative stepsize for QPROP; t_k: supervisor's stepsize; T: switching parameter for SABB.

Algorithms	*Ver.*	*Parameters*	
BPR	1	$\eta = 0.001$	
	2	$\eta = 0.1$	
QPROP	1	$l = 0.001$	
	2	$l = 0.1$	
SSCSABB	1	$t_k = 0.001/k$,	$T = 1$
	2	$t_k = 0.1/k$,	$T = 1$
	3	$t_k = 10.0/k$,	$T = 1$
	4	$t_k = 0.1/\sqrt{k}$,	$T = 1$
	5	$t_k = 0.1$,	$T = 1$
	6	$t_k = 0.1/k$,	$T = 5$
	7	$t_k = 0.1/\sqrt{k}$,	$T = 5$
	8	$t_k = 0.1$,	$T = 5$
	9	$t_k = 0.1/k$,	$T = 2.5$
	10	$t_k = 0.1$,	$T = 2.5$

the research community. We believe that most probably CPU time is an unsafe measure which is likely to cause misleading observations. In fact, it has been argued (Demuth and Beale, 1999) that observed time of CPU usage in neural network learning algorithm implementations may vary according to whether or not one of them can exploit particular structures of the computational engine or the compiler used for the development of the corresponding programme. Such cases result in floating point operations savings, and consequently CPU time reductions, even if the algorithms do not have any differences in the number of function or gradient evaluations. It has been also demonstrated, in the same work, that even the number floating point operations does not have a linear relation to the time of CPU usage observed, due to the same fact. Following this line of reasoning we base all our results on the notion of *epoch*, as a measure of time in the experiments we report. An epoch consists of full forward and backward pass through all the training patterns, including, of course, the corresponding error function and gradient evaluations. Furthermore, many authors (Kinsella, 1992; Magoulas et al.,

1999; Reed and Marks, 1999; Riedmiller, 1994) argue that one gradient evaluation amounts to 2 to 3 times the computational expense needed for a function evaluation. This provides additional support to our decision since all three algorithms we use require the same number of gradient operations per epoch.

To form a broad basis for our evaluation experiments we employ four examples for each of the categories of applications identified in the previous chapter to be most frequently associated with neural networks, namely regression and classification. The problems that belong to each of these categories, can be further divided into two groups either of which covers a specialised area of application.

The group of the first two examples in the regression category represent conventional regression problems with Normally distributed noises in both cases, but with the presence of outlier values in the second one. The second group of the same category, however, deals with a rather more complex area, that of time series observations produced by dynamical systems which exhibit chaotic behaviour. In the first example of this group, the added noise component has again Gaussian distribution but it is incorporated in the evolution of the system dynamics. In the second case the noise is the result of the integration process of a continuous time system. Therefore, under the name of regression we effectively consider a wide variety of disturbances, namely *symmetrically* and *asymmetrically* distributed *additive* noises, *dynamic* noise, and finally, *discretisation* noise. This is, in fact, a range of noise types representative of those most frequently met in real world applications.

On the other hand, both problems of the first group in the classification category represent the well-known problem of Parity, a two class problem in which all possible cases can be enumerated (details will be presented below). They were included here to illustrate the difference between *memorisation* and *generalisation* even when no noise is present in the data. In the first example, the sample available for training includes all possible cases so the issue of generalisation does not arise. However, in the second task, half of the cases are kept as a test set so generalisation is relevant since the networks are required to deduce the input-output

mapping from the incomplete information of the training set. Although noise is not incorporated in the input-target pairs, the uncertainty induced by restricting the information available gives rise to the issue of generalisation which can be measured by the performance in the test set.

The last two classification tasks are well-known real world problems which involve different types of noises in the measurements. In the first, the objective is to identify mines from rocks based on sonar signals corrupted by measurement noise. The second task is a digit pattern recognition problem in which the input patterns consist of Fourier coefficients extracted as features of the raw binary digit images. The uncertainty, and hence the noise component in the data, in this case, is introduced by the feature extraction process and the corresponding loss of information.

In the following, we first discuss in detail each one of the above tasks, the procedures used to generate the corresponding data sets, some of their characteristics, as well as the architectures and the training parameters of the neural networks employed to tackle them. Additionally, we shall comment, based on exploratory inspection of the corresponding *learning curves*, on the differences observed between the behaviour of the algorithm proposed here and that of the two algorithms used as benchmarks. Finally, we shall proceed in presenting the statistical results obtained according to the experimental schedule described previously and further discuss a series of statistical analyses based on them. As a final note, it is worth reminding here that the main interest of our investigations is focussed on the generalisation ability of the learning process as opposed to the memorisation of the training sample.

6.2 Regression Problems

Before proceeding in discussing the individual details for each one of the regression tasks, it is useful to note, as a general remark, that all the networks employed

Table 6.2: Parameters used in the simulation runs for the **Regression** problems; i-h-o: number of input,hidden, output nodes

Problems	*Architecture* *i-h-o*	*Max Nr.* of Epochs	*Convergence Criterion (MSE)*
Regression	1-8-1	10000	0.12
Regression with Outliers	1-15-1	10000	0.1
Logistic Map	1-30-1	10000	0.05
Mackey–Glass	1-50-1	10000	0.02

this class of problems had identity activation functions for their input and output nodes and symmetric sigmoids (hyperbolic tangent) for their hidden nodes. Furthermore, they were all trained to minimise a Mean Squared Error function, according to suggestions arising from the discussion in the previous chapter. Table 6.2 presents the network architectures employed for each one of the regression problems we examined as well as other parameter values used in the training evaluation process.

6.2.1 Simple Regression

We first consider a simple regression example. Our data consists of realisations of two real-valued variables X and Y of which the former is taking values x_i from a standard Normal distribution while the values y_i for the latter are sampled from a Normal distribution with mean:

$$\mu_{y_i} = 0.3 + 0.4x_i + 0.5\sin(2.7x_i) \tag{6.1}$$

For both cases the standard deviation is 0.01. In fact this represents a sinusoidal wave with a trend and an added noise component. We generated 200 values from this model. The networks' task is to estimate the conditional expectation function of Y given X. Furthermore, we wish to use our estimator to predict y_i when x_i is known. To this end we divide the data set in two sets of pairs of the

form $\{(x_1, y_1), \ldots, (x_N, y_N)\}$. The first set of 100 pairs is used to train the neural networks (i.e. to approximate the regression function), and the remaining 100 pairs are used to evaluate the predictive power (or generalisation ability) of the trained neural networks.

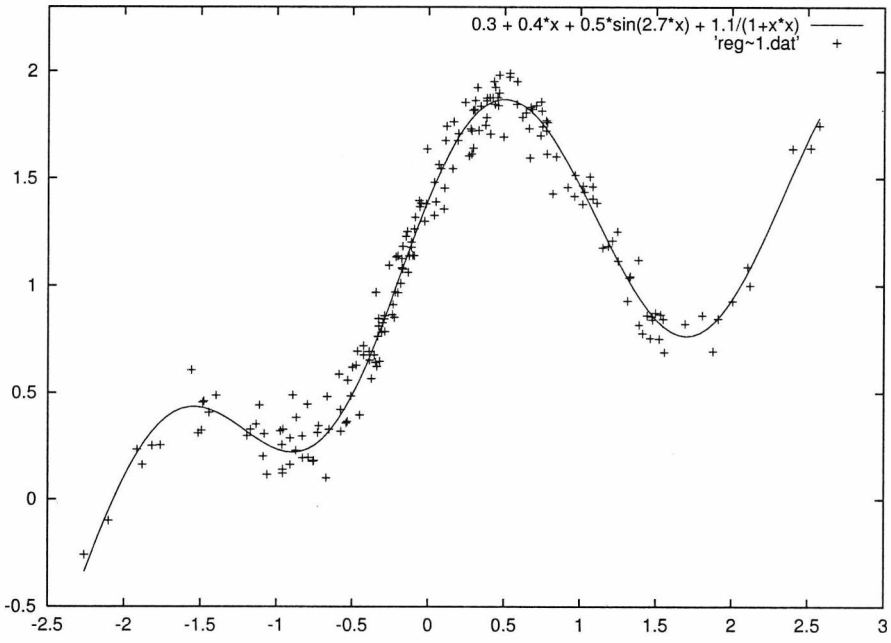
Fig. 4(a) presents a plot of x_i against y_i for the data set used for training (crosses). The actual regression function is plotted as a line. It is not difficult to observe the noise in the available data, as well as its symmetric distribution around the regression function. The networks trained by the three algorithms (BPR,QPROP,SSCSABB) had 1 input node, 8 hidden nodes and 1 output node. Training was continued in this case for 10000 epochs. Fig. 4(b) gives a typical example of the learning curves (number of epochs against the Mean Squared Error) obtained on the test set by the networks trained with the most successful version of each of the three algorithms. It is easy to observe that in this simple example they all exhibit similar generalisation behaviour with the BPR only slightly slower to achieve the same error level.

6.2.2 Regression with Outliers

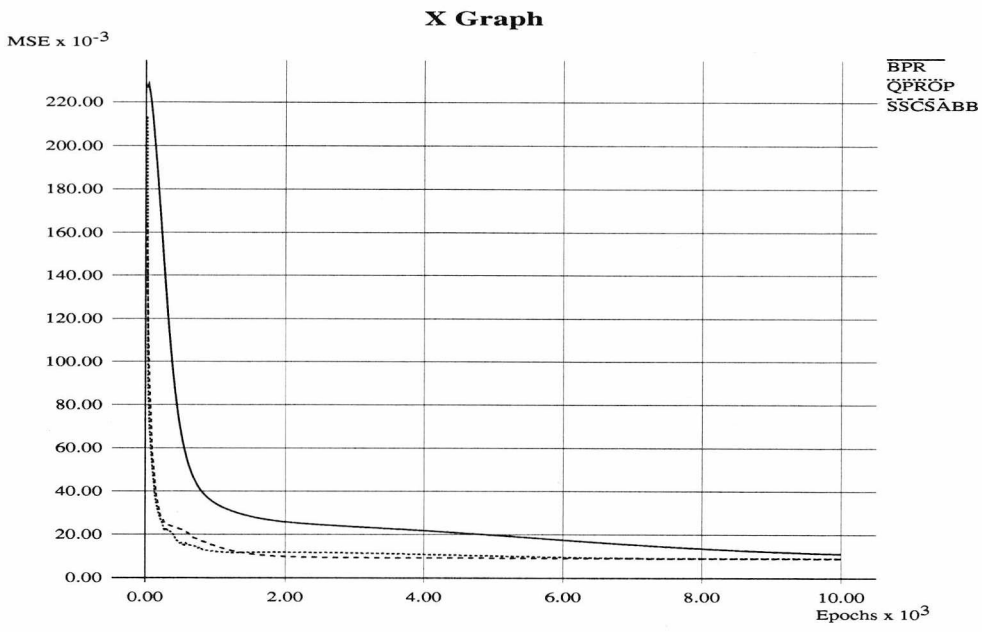
This example is based on the same setup as the simple regression problem described above. The input variables x_i are sampled from a standard Normal distribution and the targets y_i are coming from a distribution with mean:

$$\mu_{y_i} = 0.3 + 0.4x_i + 0.5\sin(2.7x_i) \tag{6.2}$$

Although in most of the cases the distribution about this mean was Normal with standard deviation 0.1, in some cases, occurring with probability 0.05, an "outlier" appears for which the corresponding standard deviation is 1.0. The effects of this process can be seen in Fig. 5(a) where again the training pairs (x_i, y_i) are plotted as crosses along the line representing the regression function. However, in this case some of the crosses are located at considerably longer distances from the line, indicating much wider dispersion.

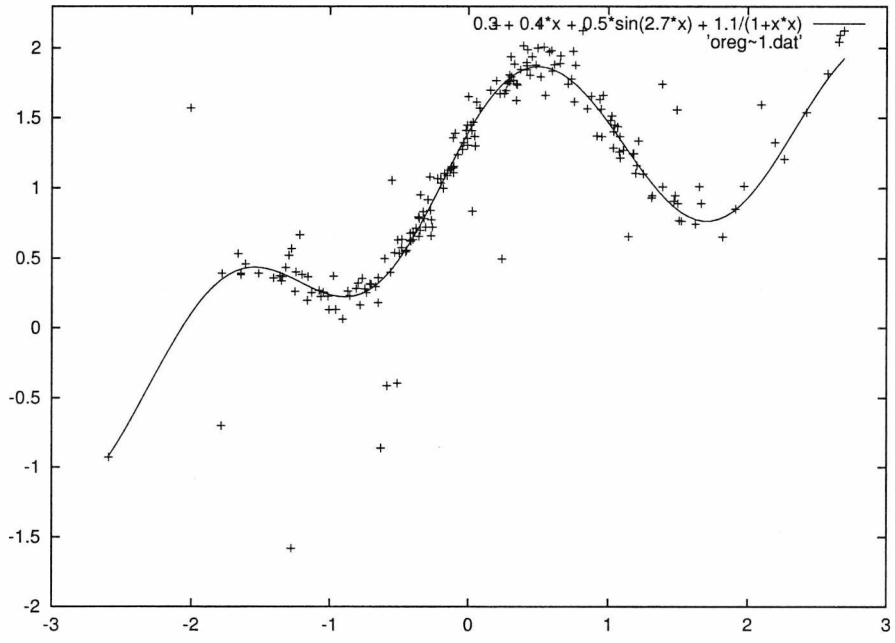


(a) Scatterplot of the training data

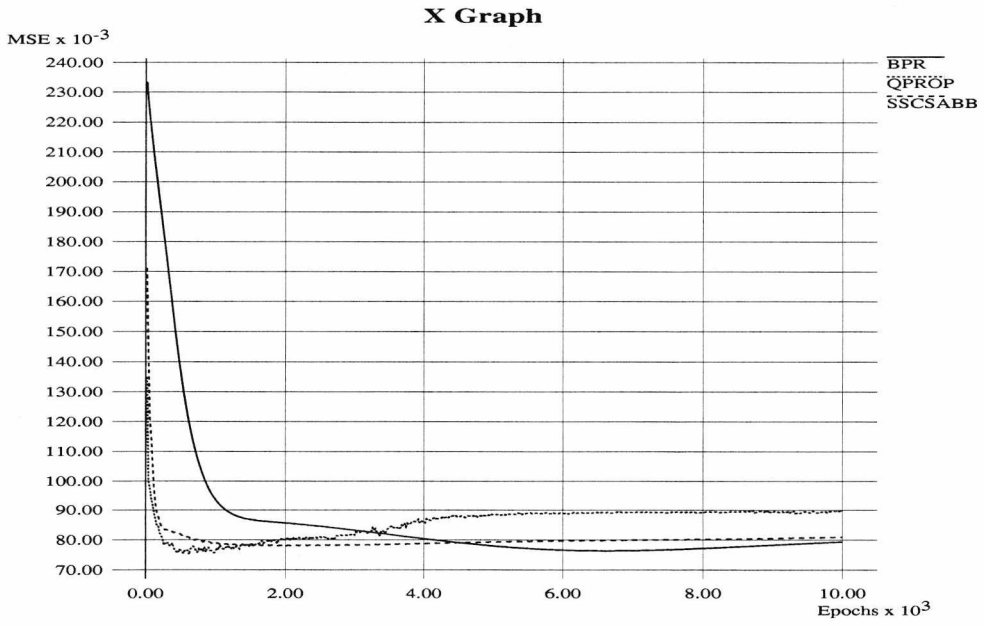


(b) Learning curves on the test set

Figure 4: Simple Regression with noise



(a) Scatterplot of the training data



(b) Learning curves on the test set

Figure 5: Simple Regression with noise and outliers

This is a much harder problem than the one presented previously especially for neural network training procedures based on the minimisation of a squared error function. In such cases the outliers tend to be given excessive weight leading to estimators biased towards them. The standard BPR is not, normally, expected to show this behaviour, since the constant learning rate would not allow it to penetrate very deeply in narrow wells of the error surface. For greedy algorithms with adaptive stepsizes to avoid learning the characteristics of the outlying values (and hence generalise poorly) the remedy often suggested is "early stopping" of the training. The results of this phenomenon are easily observed in Fig. 5(b) presenting typical examples of the learning curves corresponding to the three algorithms. While initially their behaviour is similar to that in the regression example without outliers, as training is continued the generalisation error of QPROP is starting to increase until it saturates at a higher level than the error of the other two algorithms. It is worthwhile noting that SSCSABB, along with BPR, does not exhibit this type of behaviour. We postulate that this is a consequence of the presence of SA as a Supervisor algorithm. Again, in this case, the training phase lasted 10000 epochs and the networks used had an architecture of 1 input, 15 hidden, and 1 output nodes.

6.2.3 Logistic Map

This is an one-dimensional dynamic system expressed by the simple formula:

$$x_t = \alpha x_{t-1}(1 - x_{t-1}) \tag{6.3}$$

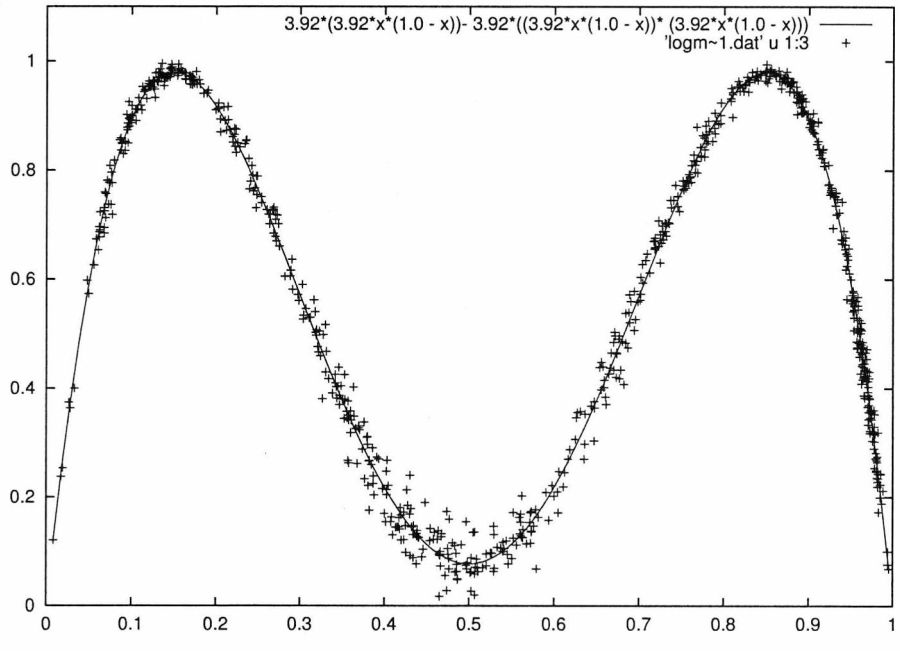
It is well known that this system can exhibit considerable chaotic behaviour for values of its parameter (α) around 4 (Müller and Reinhardt, 1990, p. 62–63). In our study we add small amounts of noise to the model, aiming to create a *stochastic* system, that has a chaotic skeleton, in order to obtain a more interesting and realistic behaviour for our tests. The actual form of the model we used to

generate the series is:

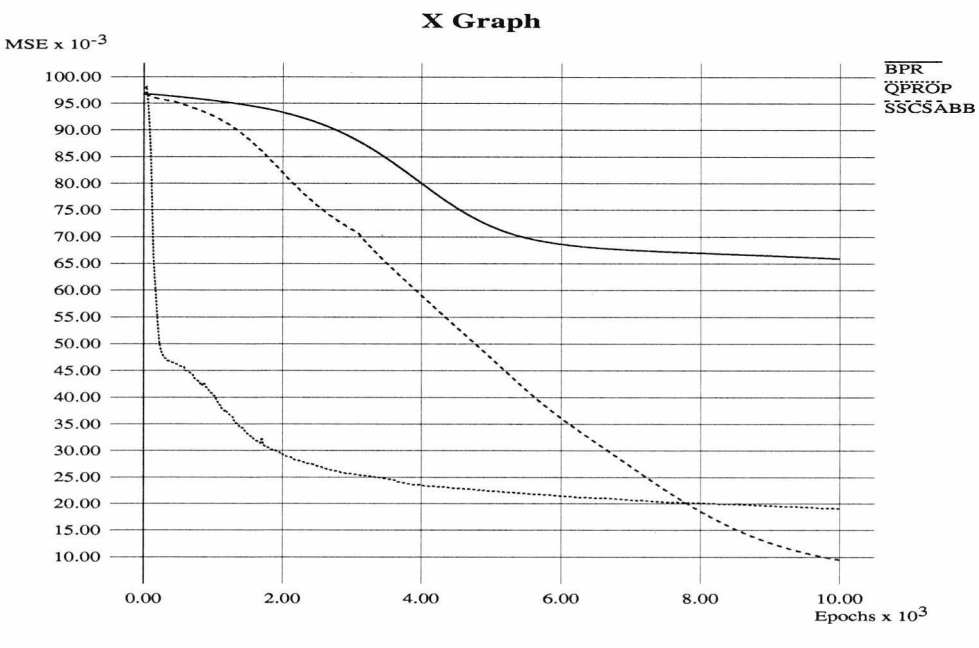
$$x_t = 3.92x_{t-1}(1 - x_{t-1}) + \epsilon_t, \; t \geq 1 \tag{6.4}$$

where $t = 0, 1, \ldots$ is a discrete time indicator and $\epsilon_t \rightarrow N(0, 0.01)$. The error term has bounded support which is necessary for the stationarity of the series to hold, as suggested in (Yao and Tong, 1994). Expression (6.4) represents a stochastic version of the logistic map (6.3) in the chaotic regime ($\alpha = 3.92$). A sample of 600 values was generated for $x_0 = 0.1$. The level of noise in this set is on average approximately 18%. The task is, again, for the networks to approximate the mapping underlying the data given a sample of input-target pairs of the form $(x(t), x(t + \tau))$ where $\tau = 1, 2, \ldots$ usually indicates how far in time we wish the network to learn to predict. A widely known property of chaotic systems is their sensitive dependence to initial conditions. This means that an infinitesimal change in the initial values results in exponentially growing deviations of the values in the future. As a consequence, the ability to predict the behaviour of a chaotic system dramatically diminishes as the distance from the current point in time (t) increases. In our case, the effects of this phenomenon are further amplified by the presence of the random disturbances. For $\tau = 1$ the mapping is a hyperbola and it was shown in (Lapedes and Farber, 1987b) that multilayered neural networks can produce good approximations relatively easy. In the present study, however, in order to test our training algorithm in really demanding task we adopt a time interval of 2. In this case the noise free mapping (often called *skeleton*) is formed from a superposition of two hyperbolas on a parabola. From the 600 values generated we construct disjoint training and test sets consisting of 250 pairs of the form $(x(t), x(t + 2))$.

A graphical representation of the training set is shown in Fig. 6(a) where input-target pairs are indicated as crosses. In the figure x_t is plotted against x_{t+2}. The dynamic skeleton of the system, represented as a line in the plot, is disturbed by the added noise, and moreover the variation of the disturbances is attenuated



(a) Scatterplot of the training data



(b) Learning curves on the test set

Figure 6: The Logistic Map

through time at a rate depending on the values of initial conditions (x_t). This phenomenon is amplified as the time interval between successive data points is increased (for example, a plot of x_t versus x_{t+3} would display significantly wider dispersion of the crosses away from the skeleton). The networks used in this case had 1 output, 30 hidden and 1 output nodes. Training was also continued for 10000 epochs like in the two previous examples. Fig. 6(b) shows typical learning curves on the test set for the three training algorithms we used. It can be easily observed that while QPROP reduces the error fast it saturates at a higher value than the one finally achieved by SSCSABB. We believe that this is the result of the state dependent rate of the noise amplification. As shown in Fig 6(a) noise is stronger at the lower part of the parabola than at the higher parts of the hyperbolas. This has similar effects with the outliers in the previous problem. So QPROP is affected in an analogous way.

6.2.4 Mackey-Glass Time Series

The so-called Mackey-Glass time series pose a highly demanding test to the predictive power of the neural networks and hence the corresponding learning algorithms. They series is generated by the following nonlinear delayed differential equation:

$$\frac{dx}{dt} = \frac{\alpha x(t-\tau)}{1+[x(t-\tau)]^{10}} - bx(t) \tag{6.5}$$

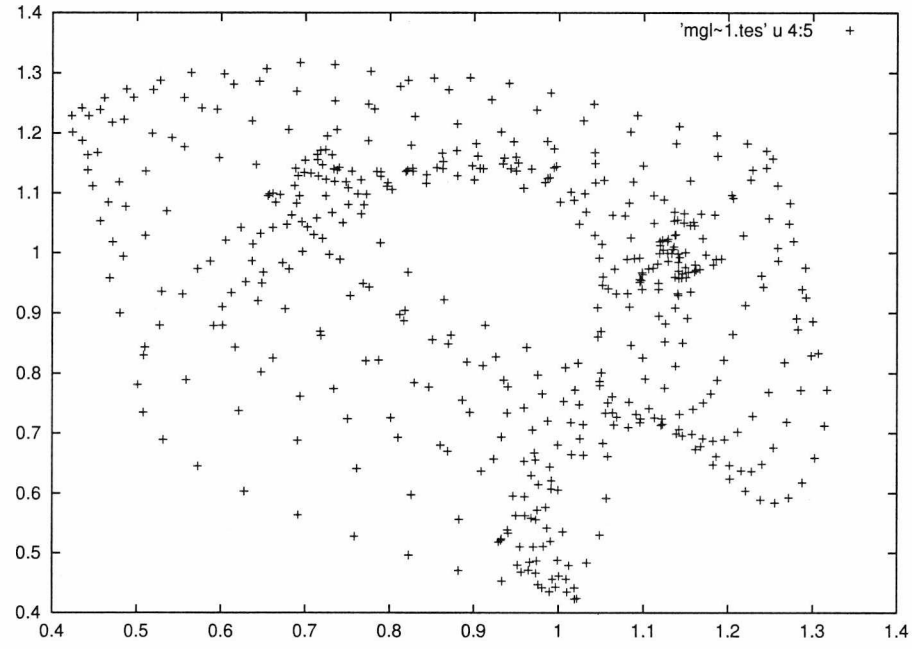
where t is the time index and τ is a time delay parameter. With $\alpha = \frac{1}{5}$ and $b = \frac{1}{10}$ it was first investigated in (Mackey and Glass, 1977). For $\tau = 17$ and $\tau = 30$ it was shown in (Lapedes and Farber, 1987b) that $x(t+n\Delta)$ with fixed step length $\Delta = 6$ exhibits chaotic behaviour. The data set we use here was obtained from the CMU Machine Learning Benchmark archive (Kantrowitz, 1993). It is generated for $\tau = 17$ and using a second order Runge-Kutta method as an integration process with stepsize of 0.1. As mentioned previously, the discretisation process infuses deterministic noise (truncation errors) in the resulting data. The task is

to use currently available points in the series to predict a future point, $t+P$. For values of P greater than the characteristic period of the underlying dynamical system, which is approximately 50, the problem bears considerable difficulty (see for example, (Lapedes and Farber, 1987b)). The most commonly used form of the problem, which we adopt here also, is to use four points ($x(t-18), x(t-12), x(t-6)$, and $x(t)$) to predict a point later in the series, usually $x(t+85)$. From the available training data generated for $t = 200$ to $t = 3200$ we used the first 485 points to form a training set consisting of 400 patterns.

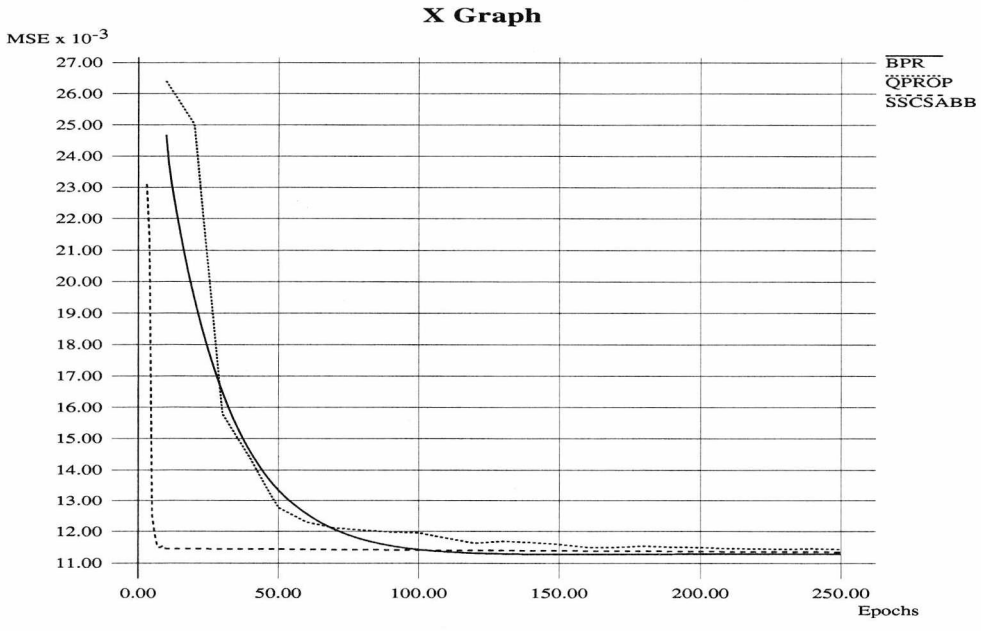
Fig. 7(a) presents a two-dimensional projection of our training data, where $x(t)$ is plotted against $x(t + 85)$. The complex nature of the underlying mapping can be easily realised. From the original test data corresponding to $t = 5000$ up to $t = 5500$, we formed similarly our test set of 400 patterns. Although many experiments have been performed with this series concerning neural network training (see for example (Lapedes and Farber, 1987b; Moody and Darken, 1988)), unfortunately, variations in the setup used render the corresponding results incomparable. The networks we trained consisted of 4 input, 50 hidden and 1 output units. The maximum number of training epochs was, again, set to 10000. Similarly to the previous problems, Fig. 7(b) illustrates the generalisation ability resulting from the three training algorithms by means of learning curves obtained on the test set. It is easily observed that BPR and QPROP exhibit very similar rate of error reduction while SSC-SABB is much faster. However, they all saturate at the same level of error. Neither of them show any indication of overtraining in this example.

6.2.5 Statistical Analysis and Discussion

Observations based on the learning curves, such as the ones included in the previous discussion, are useful exploratory tools. However, they cannot form the basis for valid statistical analyses of the phenomena under consideration, especially in simulation experiments such as the training of neural networks. In these cases, the outcomes strongly depend on random quantities, in our cases being the random initialisation of the weight vectors. So the analysis should start from the



(a) Scatterplot of the training data



(b) Learning curves on the test set

Figure 7: The Mackey–Glass problem

Table 6.3: Mean Squared Error (MSE), in the **training set**, and its Standard Deviation (Std) in parentheses **at the middle** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Regression*	*Regression with Outliers*	*Logistic Map*	*Mackey-Glass*
BPR	1	0.14803	0.23926	0.09434	0.03160
		(0.00726)	(0.00577)	(0.00163)	(0.01770)
	2	0.02296	0.10318	0.08398	0.01197
		(0.00191)	(0.00563)	(0.00472)	(0.00006)
QPROP	1	0.01116	0.08185	0.01779	0.01173
		(0.00114)	(0.00734)	(0.01004)	(0.00029)
	2	0.01072	0.07999	0.01892	0.01168
		(0.00121)	(0.00724)	(0.00758)	(0.00031)
SSCSABB	1	0.01082	0.10134	0.05993	0.01195
		(0.00045)	(0.04712)	(0.01631)	(0.00007)
	2	0.01067	0.08731	0.06861	0.01196
		(0.00041)	(0.00045)	(0.00687)	(0.00005)
	3	0.09354	0.14214	0.08152	0.05043
		(0.04640)	(0.03790)	(0.02294)	(0.00182)
	4	0.01086	0.08742	0.06885	—
		(0.00028)	(0.00038)	(0.00599)	
	5	0.01043	0.08524	0.02588	0.01194
		(0.00019)	(0.00079)	(0.01411)	(0.00005)
	6	0.01057	0.08793	0.05262	—
		(0.00433)	(0.04802)	(0.04156)	
	7	0.01454	0.08618	0.05414	—
		(0.00887)	(0.01683)	(0.03227)	
	8	0.01045	0.06940	0.02960	—
		(0.00358)	(0.00470)	(0.02269)	
	9	0.03140	—	0.0842	—
		(0.03643)		(0.01315)	
	10	0.01380	—	0.0541	—
		(0.00438)		(0.02175)	

Table 6.4: Mean Squared Error (MSE), in the **training set**, and its Standard Deviation (Std) in parentheses **at the end** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Regression*	*Regression with Outliers*	*Logistic Map*	*Mackey-Glass*
BPR	1	0.13365	0.22981	0.09307	0.01995
		(0.00708)	(0.00572)	(0.00042)	(0.00571)
	2	0.01439	0.09081	0.07348	0.01194
		(0.00089)	(0.00320)	(0.00143)	(0.00005)
QPROP	1	0.01021	0.07191	0.01196	0.01147
		(0.00080)	(0.00402)	(0.00803)	(0.00037)
	2	0.00955	0.07424	0.01396	0.01143
		(0.00065)	(0.00652)	(0.00772)	(0.00042)
SSCSABB	1	0.01000	0.09748	0.02743	0.01191
		(0.00023)	(0.04851)	(0.01954)	(0.00006)
	2	0.01000	0.08461	0.03191	0.01192
		(0.00013)	(0.00090)	(0.01400)	(0.00005)
	3	0.08652	0.13817	0.07623	0.05030
		(0.05170)	(0.04226)	(0.03289)	(0.00216)
	4	0.01006	0.08475	0.03218	—
		(0.00013)	(0.00077)	(0.01422)	
	5	0.00984	0.08159	0.01641	0.01189
		(0.00011)	(0.00225)	(0.00825)	(0.00005)
	6	0.01115	0.08555	0.05261	—
		(0.00469)	(0.04938)	(0.04158)	
	7	0.01442	0.08519	0.04431	—
		(0.00887)	(0.01789)	(0.03259)	
	8	0.00865	0.06640	0.01369	—
		(0.00034)	(0.00407)	(0.01683)	
	9	0.02951	—	0.08420	—
		(0.03706)		(0.01315)	
	10	0.00877	—	0.03955	—
		(0.00367)		(0.02436)	

Table 6.5: Mean Squared Error (MSE), in the **test set**, and its Standard Deviation (Std) in parentheses **at the middle** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Regression*	*Regression with Outliers*	*Logistic Map*	*Mackey-Glass*
BPR	1	0.20181	0.22684	0.09854	0.03175
		(0.01073)	(0.01133)	(0.00202)	(0.01989)
	2	0.01900	0.07882	0.08425	0.01122
		(0.00181)	(0.00242)	(0.00684)	(0.00006)
QPROP	1	0.00982	0.08448	0.01606	0.01104
		(0.00070)	(0.00451)	(0.00849)	(0.00021)
	2	0.01017	0.08500	0.01762	0.01099
		(0.00088)	(0.00375)	(0.00672)	(0.00023)
SSCSABB	1	0.00924	0.09532	0.05447	0.01121
		(0.00025)	(0.04474)	(0.01477)	(0.00009)
	2	0.00918	0.08076	0.06240	0.01122
		(0.00024)	(0.00077)	(0.00684)	(0.00006)
	3	0.48780	18.03302	0.08390	0.05052
		(0.96568)	(45.32298)	(0.02617)	(0.00169)
	4	0.00927	0.08072	0.06259	—
		(0.00023)	(0.00075)	(0.00603)	
	5	0.00912	0.08192	0.02367	0.01119
		(0.00023)	(0.00076)	(0.01303)	(0.00005)
	6	0.01175	0.09715	0.05228	—
		(0.00261)	(0.03150)	(0.04329)	
	7	0.01445	0.08305	0.05188	—
		(0.00580)	(0.00237)	(0.03248)	
	8	0.01195	0.08889	0.02668	—
		(0.00152)	(0.00537)	(0.02051)	
	9	0.03455	—	0.08504	—
		(0.05009)		(0.01638)	
	10	0.01242	—	0.04883	—
		(0.00236)		(0.01988)	

Table 6.6: Mean Squared Error (MSE), in the **test set**, and its Standard Deviation (Std) in parentheses **at the end** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Regression*	*Regression with Outliers*	*Logistic Map*	*Mackey-Glass*
BPR	1	0.18128	0.21077	0.09686	0.01942
		(0.01156)	(0.01076)	(0.00056)	(0.00655)
	2	0.01094	0.07822	0.06747	0.01119
		(0.00070)	(0.00173)	(0.00227)	(0.00005)
QPROP	1	0.00953	0.09101	0.01084	0.01085
		(0.00055)	(0.00750)	(0.00731)	(0.00028)
	2	0.00994	0.08609	0.01275	0.01082
		(0.00086)	(0.00291)	(0.00694)	(0.00032)
SSCSABB	1	0.00897	0.09752	0.02496	0.01117
		(0.00017)	(0.04398)	(0.01780)	(0.00006)
	2	0.00895	0.08226	0.02887	0.01118
		(0.00012)	(0.00070)	(0.01287)	(0.00004)
	3	0.47620	15.92715	0.07909	0.05040
		(0.92341)	(38.89990)	(0.03496)	(0.00202)
	4	0.00897	0.08218	0.02911	—
		(0.00015)	(0.00067)	(0.01307)	
	5	0.00891	0.08400	0.01512	0.01115
		(0.00008)	(0.00129)	(0.00772)	(0.00004)
	6	0.01239	0.10355	0.05229	—
		(0.00292)	(0.03357)	(0.04328)	
	7	0.01509	0.08557	0.04172	—
		(0.00568)	(0.00827)	(0.03158)	
	8	0.01207	0.09020	0.01247	—
		(0.00160)	(0.00747)	(0.01522)	
	9	0.03326	—	0.08504	—
		(0.05046)		(0.01638)	
	10	0.01044	—	0.03556	—
		(0.00398)		(0.02277)	

Table 6.7: Best performing versions of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively.

Best Performances								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey--Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR	0.022	0.019	0.103	0.078	0.083	0.084	0.011	0.011
	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)
QPROP	0.010	0.009	0.079	0.084	0.017	0.016	0.011	0.010
	(2)	(1)	(2)	(1)	(1)	(1)	(2)	(2)
SSCSABB	0.010	0.009	0.069	0.080	0.025	0.023	0.011	0.011
	(5)	(5)	(8)	(4)	(5)	(5)	(5)	(5)

Table 6.8: Best performing versions of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively.

Best Performances								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey--Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR	0.014	0.010	0.090	0.078	0.073	0.067	0.011	0.011
	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)
QPROP	0.009	0.009	0.071	0.086	0.011	0.010	0.011	0.010
	(2)	(1)	(1)	(2)	(1)	(1)	(2)	(2)
SSCSABB	0.008	0.008	0.066	0.082	0.013	0.012	0.011	0.011
	(8)	(5)	(8)	(4)	(8)	(8)	(5)	(5)

Table 6.9: T–statistic values for the differences between the mean MSEs of the best performing version of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively (critical value $t_{\{1-0.05/2,18\}} = 2.109$).

T–Statistic								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey–Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR–QPROP	17.15	14.93	7.99	-3.50	18.87	19.78	2.97	3.03
BPR–SSCSABB	20.69	17.09	14.57	-2.37	12.35	13.02	1.43	1.12
SSCSABB–QPROP	0.74	2.97	3.88	2.61	-1.48	-1.55	-2.62	-2.68

Table 6.10: Hypothesis testing for differences between the average Mean Squared Errors of the best performing version of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively (0: accept null hypothesis H_0, 1: reject H_0, where the alternative hypothesis is that the first algorithm performs better than the second, and (*) denotes equality of performances).

Hypothesis Testing								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey–Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
QPROP–BPR	1	1	1	0	1	1	1	1
SSCSABB–BPR	1	1	1	0	1	1	0*	0*
SSCSABB–QPROP	0*	1	1	1	0*	0*	0	0

presentation of descriptive statistics over a number of repeated measurements. Consequently, we begin by presenting in Tables 6.3 and 6.4 the average of Mean Square Error (MSE) values achieved on the training set by all versions of the algorithms at the middle and at the end of the training period respectively. These tables include results for all regression problems so that exploratory comparison of the performances can be also performed. In addition, every value in the tables is annotated with the corresponding standard deviations as a measure of the variability around the average behaviour expressed by the mean values. Under an

Table 6.11: T–statistic values for the differences between the mean MSEs of the best performing version of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively (critical value $t_{\{1-0.05/2,18\}} = 2.109$).

T–Statistic								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey–Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR–QPROP	13.85	5.04	11.63	-7.36	23.86	23.40	3.86	3.60
BPR–SSCSABB	19.00	9.12	14.91	-6.75	11.19	11.30	2.10	1.58
SSCSABB–QPROP	3.92	3.55	3.05	4.15	-0.29	-0.30	-3.50	-3.28

alternative interpretation, these quantities could give an indication of the sensitivity of the training algorithms to different values for the starting points (initial weights). To complete the picture, similar tabulations of the results obtained in the test sets are shown in Tables 6.5 and 6.6 for the same points in time during learning. These bear particular significance for us since the main focus of our interests is performance assessment with respect to generalisation.

To start with the BPR, it is not difficult to conclude that there are differences between the two versions in both Tables 6.3 and 6.4. This observation holds for all the regression tasks and, furthermore, for the results of the testing sets as well (Tables 6.5 and 6.6). As expected, the version with the smaller learning rate (version 1) is slower in training resulting also in higher error rates in the test set for the same number of epochs. However, as pointed out previously, very large learning rates can also lead to poor performance as a result of oscillation. QPROP, however, does seem to be too sensitive, at least to the values of alternative stepsizes we tried, as it can be deduced by the very similar behaviour exhibited by its two versions in all cases. On the other hand, conclusions about SSC-SABB are not so straightforward since we have tried a greater number of different values for the adjustable parameters. The combinations of the range of values for the switching

Table 6.12: Hypothesis testing for differences between the average Mean Squared Errors of the best performing version of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively (0: accept null hypothesis H_0, 1: reject H_0, where the alternative hypothesis is that the first algorithm performs better than the second, and (*) denotes equality of performances).

Hypothesis Testing								
Algorithms Compared	*Regression*		*Regression with Outliers*		*Logistic Map*		*Mackey–Glass*	
	Train	Test	Train	Test	Train	Test	Train	Test
QPROP–BPR	1	1	1	0	1	1	1	1
SSCSABB–BPR	1	1	1	0	1	1	1	0*
SSCSABB–QPROP	1	1	1	1	0*	0*	0	0

Table 6.13: Number of successful runs (out of 10) in the **test sets**, with the corresponding mean number of epochs and their standard deviations in parentheses (rounded to the nearest integer).

Convergence statistics					
Algorithms	*Ver.*	*Regression*	*Regression with Outliers*	*Logistic Map*	*Mackey–Glass*
BPR	1	0	0	0	10 (1822/1321)
	2	10 (4567/831)	10 (817/159)	0	10 (22/10)
QPROP	1	10 (431/167)	10 (117/55)	10 (498/535)	10 (17/6)
	2	10 (430/136)	10 (92/32)	10 (312/120)	10 (20/9)
SSCSABB	1	10 (656/247)	9 (112/42)	9 (5812/2003)	10 (11/3)
	2	10 (641/228)	10 (109/41)	9 (7003/1326)	10 (6/2)
	3	1 (9060/0)	2 (4270/5939)	2 (4065/21)	0
	4	10 (1000/218)	10 (1194/229)	9 (7530/1334)	–
	5	10 (694/76)	10 (562/155)	10 (3722/1669)	10 (15/7)
	6	9 (127/25)	8 (185/126)	4 (202/25)	–
	7	7 (127/21)	7 (125/25)	5 (2088/3769)	–
	8	10 (1044/1526)	10 (783/1323)	10 (2802/2716)	–
	9	4 (125/5)	–	0	–
	10	10 (1860/1344)	–	7 (5097/2870)	–

(preference) parameter (T), with the different types and initial values for the step-sizes (t_k) of the supervising algorithm (SA), result in 10 different versions of the algorithm which are represented in the tables. However, this number of different versions is necessary in order to adequately explore the effective ranges of values for the parameters influencing the behaviour of SSC-SABB. It is worth reminding here that very purpose of the switching parameters is to provide additional preference to either the fast part of the SSC algorithm (search engine), by assuming values greater than 1, or to the conservative slower part (supervisor) when $T < 1$. Thus adjusting the balance between speed and robustness, hence the behaviour of the SSC algorithm according to the requirements of the task in hand. It should be pointed out that the existence of such an additional adjustable parameter is not necessarily a disadvantage since according to (Wolpert and Macready, 1996) no algorithm is uniformly better than any other for all possible tasks. Therefore, the flexibility to obtain a modified version of the algorithm by adjusting only one parameter provides the class of SSC algorithms with the ability to successfully overcome this limitation.

Although an initial exploration of the results concerning SSC-SABB in Tables 6.3 to 6.6 may appear to present an inconclusive picture, this is not the case after a more careful observation. First, we can say that version 2, which has values $t_k = 0.1/k$ and $T = 1$, consistently outperforms BPR both in the training and test sets. Second, the same version, which can be thought as having a standard choice of parameters, performs reasonably close to QPROP for the simple Regression and the Mackey–Glass problems. It can be observed that in these problems, as already discussed, the noises involved are quite easier to tackle being normally distributed in the first case and bounded approximation error in the second. However, in the other two problems, the Regression with Outliers and the Logistic Map, noises are much more difficult to address and in these cases different combinations of parameters (versions) perform comparably with the QPROP. In the first case, version 8, which gives preference to the fast search engine algorithm ($T = 5$), is better. It seems that, in this case, a more aggressive and flexible choice of learning

rates is needed to overcome the significant influence of the outliers in the process of the error minimisation, as we have already discussed. The same observation, however, does not hold for the corresponding results in the Tables 6.5 and 6.6 where the error in the test set (generalisation) is depicted. In these tables, version 4 appears to offer the best performance. This change in the favourable version between training and test sets demonstrates clearly the effects of overtraining as well as the importance of the switching parameter which allows to define versions of the algorithm that do not overtrain. The same is not true for the QPROP the generalisation error of which increases as training proceeds from the middle of the run (Table 6.5) to the end of it (Table 6.5). As a general observation from all the Tables 6.3 to 6.6, it seems that only in the case of the Logistic Map there is a significant decrease of MSE between the middle and the end of the corresponding training period, both in the training and the test set for all the algorithms. This provides an indication that this is the most difficult task to address. The results discussed previously provide useful exploratory information. It is common in the neural network research to produce similar tabulations with respect to different parameter values in order to select the most successful network for further analysis. In that sense, we now turn to a set of more comprehensive statistical analysis. The most successful versions of each algorithm is selected and the corresponding MSE is included in Tables 6.7 and 6.8 again for the training and test set at the middle and at the end of the training period. The version numbers are shown in parentheses. First, it is easy to realise that these tables provide additional support to the points raised in our previous discussion. As a general observation, we can state that SSC-SABB outperforms BPR in all cases and, with the exception of the Logistic Map at the middle of the simulation run, performs comparably with QPROP.

Now, in order to obtain a measure of the significance of the performance differences we observe between the best versions of the algorithms, we calculate the values of the T-statistic corresponding to the differences of the means of the MSEs over the 10 training runs for every problem. Tables 6.9 and 6.11 include the values

of the statistic corresponding to training and test sets for the middle and the end of the runs respectively. The first column of the tables shows the pairs of algorithms compared. For 18 degrees of freedom (10+10-2) and a significance level of 0.05 the critical value of the statistic is $t_{97,5,18} = 2.109$. Therefore values in the tables outside the critical region [-2.109, 2.109] indicate rejection of the null hypothesis of equality of means. In our case, this null hypothesis can be interpreted as equality in performances whatever the observed differences in the average MSE are. A noteworthy case from Table 6.9 is that of the comparison between SSC-SABB and QPROP for the Logistic Map. From the corresponding entries in Table 6.7 we observe a difference of the MSE in favour of QPROP. However, the corresponding t-statistics in Table 6.9 suggest that these differences both in the training and test set errors are superficial and do not hold statistical significance. So, we can conclude that, even in this case, despite the initial observations SSC-SABB and QPROP perform equivalently. In general, when equality of the means is rejected, additional tests are required to decide the direction of the relation. Tables 6.10 and 6.12 present the result of such tests for the alternative hypothesis (H_1) that the first algorithm from the pair shown in the first column outperforms the second. These tests are based on the values in Tables 6.9 and 6.11 respectively. A value of 0 in Tables 6.10 and 6.12 denotes acceptance of the null hypothesis that the first algorithm performs equally or worst than the second, with (*) indicating the case of equality. The opposite conclusion holds for a value of 1. Hence, through a quick overview of the two tables we can safely conclude that SSC-SABB performs better or at least equally with QPROP (indicated by 1s or 0*) with respect to both training efficiency and generalisation, for all problems but the Mackey–Glass series.

We conclude our analysis with a comparison of the number of runs each of succeeded in achieving a prespecified level of MSE (i.e. converged) in the test sets. The corresponding results are included in Table 6.13 along with the average number of the epochs required and their standard deviations in parentheses. The error levels used for each problem are included in Table 6.2. It is not difficult

Table 6.14: Parameters used in the simulation runs for the **Classification** problems; i-h-o: number of input,hidden, output nodes

Problems	*Architecture* *i-h-o*	*Max Nr.* of Epochs	*Convergence Criterion (MSE)*
Parity 8	8-16-1	90	0.0001
Parity 8 Generalisation	8-16-1	90	0.001
Sonar	60-24-2	2500	0.12
Digits	76-40-10	2500	0.25

to observe in Table 6.13 that for all problems, with the exception of the Logistic Map, at least one version of the SSC based algorithm is as robust (with respect to the number of times converged) and as fast as QPROP if not better.

6.3 Classification Problems

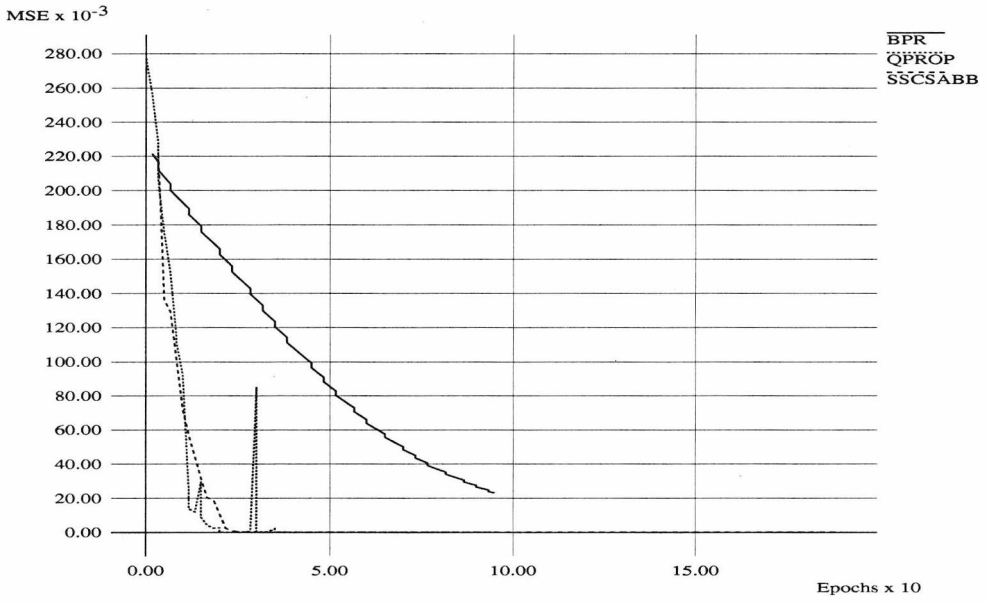
Again, as a general remark, it is useful to note here, that all the networks trained to address the classification tasks we examine in this study had identity activation functions for their input nodes and symmetrical sigmoid nonlinearities (hyperbolic tangent) for their hidden nodes. However, unlike the networks used in the regression cases, their output nodes were assigned logistic sigmoid activation functions, so that their outputs can be interpreted as class membership probabilities. The error function used for the two-class problems was the cross-entropy, while for the multi-class case its multivariate analogue (see Section 5.3.1). The network architectures corresponding to each of the problems along with the values for other parameters used during the performance assessment experiments are summarised in Table 6.14.

6.3.1 Parity

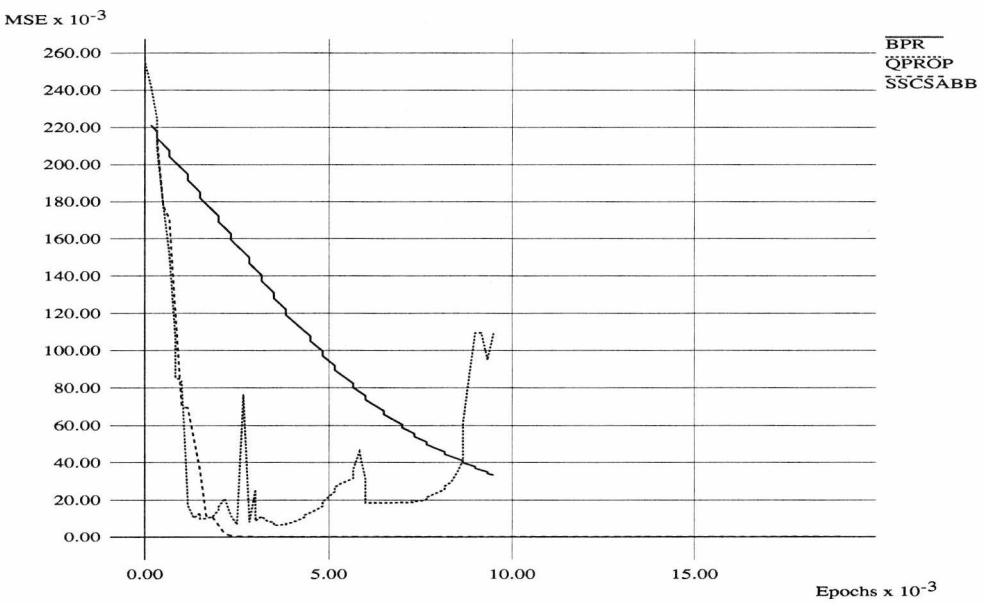
The parity problem is one of the best established benchmarks for neural network learning methods. It can be expressed as the sum, mod2, of n binary inputs. In

other words, the target value is 'true' (1) if and only if an odd number of inputs are 'true', otherwise it is 'false'. A special case of this task where there are only to binary input variables is the well-known 'exclusive or' (XOR) problem. Despite the simple underlying rule the parity class of problems is surprisingly hard to be addressed by neural networks trained by the backpropagation of error (Thornton, 1996). The particular significance of the parity task as an impossible problem for linear first-order perceptrons was pointed out by Minsky and Papert (Minsky and Papert, 1988). It belongs in the category of the so-called "statistically neutral" problems, for which the probability (i.e. observed frequency) to assign any particular input to a specific output class is always the chance value. For the two-class problem like the one we use here this value is 0.5. It has been argued that exactly this fact prevents learning methods based on extracting relationships from examples, such as the neural networks, from performing well in the parity problems (Ghorbani and Owrangh, 2001). In this study we use one of the harder versions which involves 8 input variables (characterised in the following as 'Parity 8'). The exhaustive set of cases includes $2^8 = 256$ patterns (input-target pairs). In (Fahlman, 1988) it is reported that QPROP performs favourably, in terms of speed, in the two-input equivalent 'XOR' problem even in comparison to a neural network trained by a "BFGS" based algorithm. We follow the advice given in the CMU Benchmark Learning Archive (Kantrowitz, 1993)that for n input parity problems the networks should have at least n hidden nodes. So, we train networks with 8 input, 16 hidden, and 1 output nodes, to a maximum of 90 epochs.

When training involves all possible cases the issue of generalisation is not relevant. Therefore, in order to test our algorithm in a much harder problem we constructed a second task from the same data set which involves performance in unseen data. This problems is denoted in the following as 'Parity Generalisation'. In this case the original data from the parity problem for 8 inputs was divided to two disjoint sets of equal size, one for training and one for testing. It has been reported recently in (Ghorbani and Owrangh, 2001) that networks trained with the standard Backpropagation algorithm exhibit poor generalisation



(a)



(b)

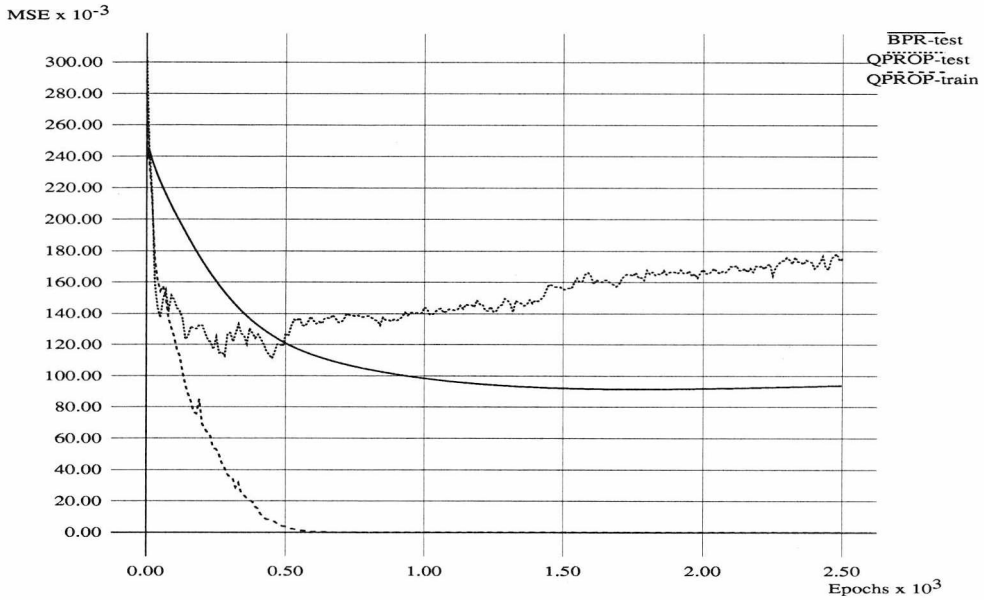
Figure 8: Parity 8 problem: Typical examples of learning curves of the QPROP and SSCSABB , in the training (a) and test(b) sets, compared to the corresponding curve for BPR in the test set.

ability even when minimal numbers of patterns are excluded from the training set. This can be verified from Fig. 8(b) where the corresponding learning curve (solid line) decreases much slower than those of the QPROP (dotted line) and the SSC-SABB (dashed line). Furthermore, BPR fails to achieve error levels similar to those achieved by the other two algorithms both in the training (Fig. 8(a)) and the test (Fig. 8(b)) sets. Another interesting observation from these graphs concerns the behaviour of QPROP. In the training set it achieves the same level of MSE as SSC-SABB and with the same speed, although it seems to be somewhat unstable (observe the spikes in the corresponding line). However, in the test set (Fig. 8(b)) it shows a completely different behaviour. After it reaches a minimum level the error increases significantly up to the end of the training period and the learning curves assumes the 'U' shape characteristic of the case of overtraining. In contrast, the generalisation error of the SSC-SABB decreases steadily as training continues and saturates at a very low level. Conclusions and possible reasons for this difference in behaviour between the two algorithms will be explored in the following section where statistical analyses of the results are discussed.

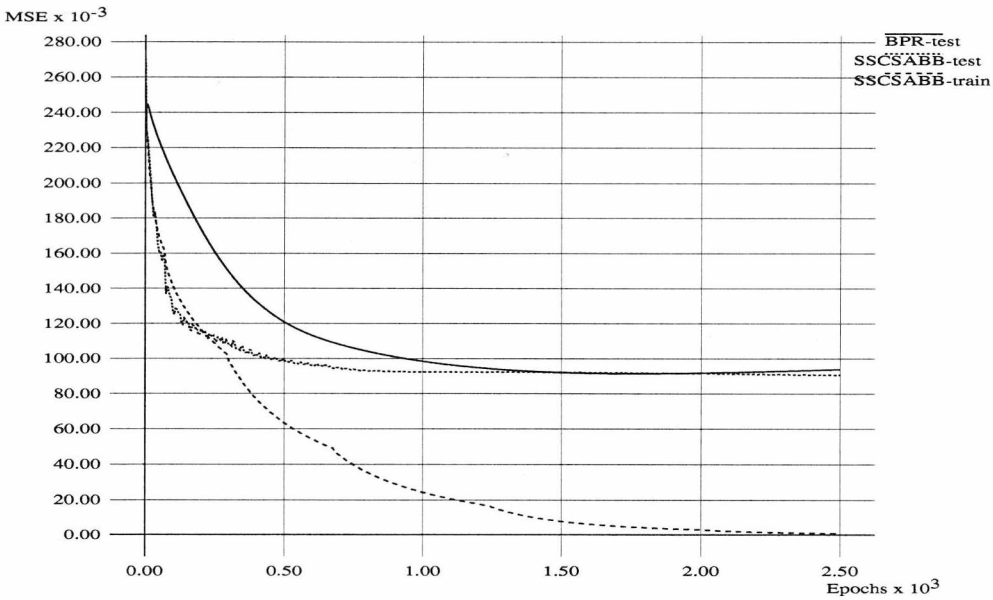
6.3.2 Sonar, Mines vs Rocks

This is a classical benchmarking data set for evaluating learning algorithms in the field of artificial neural networks. Since its first use in (Gorman and Sejnowski, 1988), it has been used extensively by researchers in the field (Ripley, 1993). However, due to the fact that there numerous ways to construct evaluating experiments with this data (see relevant comments in the CMU Learning Benchmark Archive (Kantrowitz, 1993)) comparisons are not straightforward. The task is to discriminate between sonar signals bounced off a metal cylinder (mine) from those obtained from a rock with roughly the same shape. The original data set consists of 208 cases each of which comprises a 60-dimensional vector of continuous real values serving as inputs and one enumerated variable, acting as target.

The data used in the present study was obtained from the CMU Learning Benchmark Archive (Kantrowitz, 1993) and is the one characterised as the 'aspect



(a)



(b)

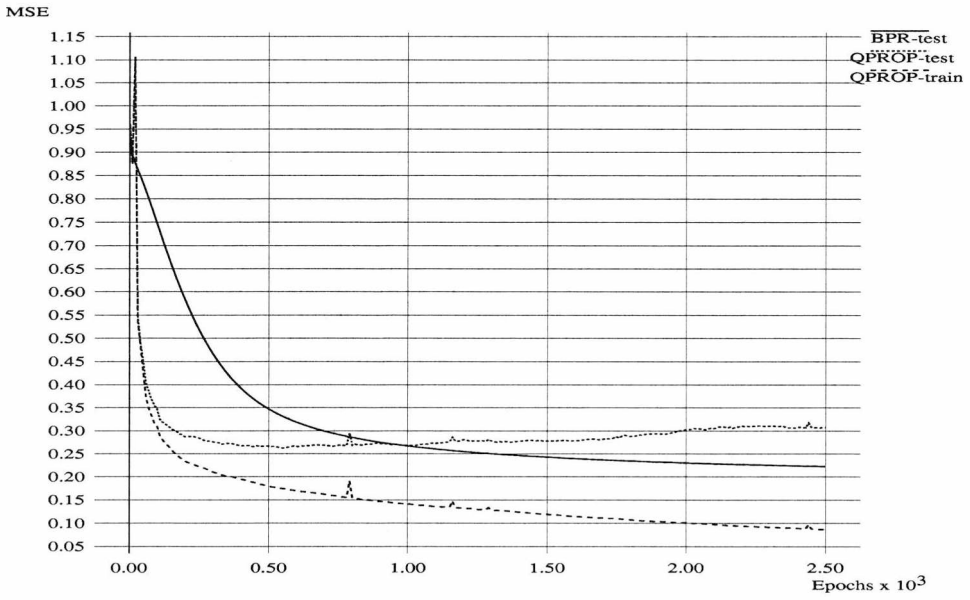
Figure 9: Sonar problem: Typical examples of learning curves of the QPROP and SSCSABB , in the training (a) and test (b) sets, compared to the corresponding curve for BPR in the test set.

angle dependent' set. Since the original sonar measurements did not include aspect angle information the contributors used clustering to create two disjoint sets in each of which a representative number of samples from all aspect angles would be included. Both sets contain 104 cases. We employ one of the set to train networks with 60 input, 24 hidden, and two output nodes. We allowed training to continue for 2500 epochs. Fig. 9 illustrates the training and generalisation behaviour of QPROP and SSC-SABB as compared to the generalisation obtained from the networks trained by BPR (solid line in both subfigures). It not difficult to observe that in this task also, QPROP tends to overtrain the networks and hence its predictive performance degrades. Once more SSC-SABB shows resilience to this phenomenon and its generalisation ability saturates at a level equal to that achieved by BPR, but at a much faster rate. In (Gorman and Sejnowski, 1988), a recognition rate of 89.2% in the test set is reported to have been achieved by a network with similar architecture to ours, and trained by a modified version of the BPR. As we shall see below, where we discuss similar type of results, our algorithm achieved comparable performance.

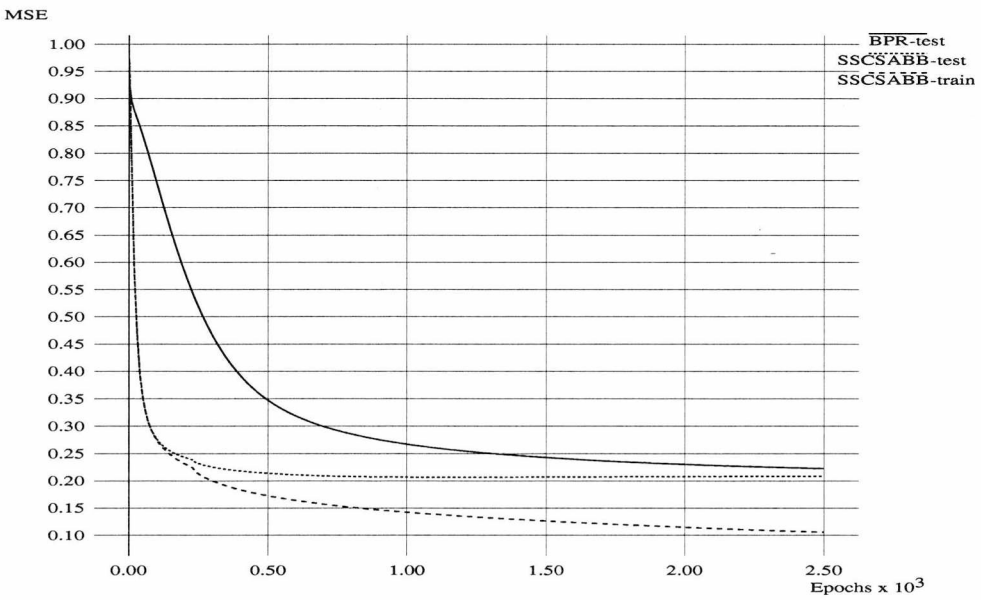
6.3.3 Handwritten Digit Recognition

In this example we employ a more complex and realistic task from the character recognition domain. The dataset consists of 76 Fourier coefficients extracted from each of 2000 binarised images of handwritten numerals (0–9), with an original resolution of 30 x 48 pixels, obtained from Dutch utility maps. More details about this dataset can be found in (Blake and Merz, 1998) under the name 'mfeat'. In the same form the set was used in (Duin and Tax, 2000). Samples are distributed equally over the classes (200 samples per class) and form a non-trivial 10-class pattern recognition problem. A training set was generated by randomly selecting 125 samples per class while the remaining 750 samples (75 per class) were used as a testing set.

All the networks trained for this task had 76 input, 40 hidden, and 10 output



(a)



(b)

Figure 10: Handwritten Digit recognition: Typical examples of learning curves of the QPROP (a) and SSCSABB (b), in the training and test sets, compared to the corresponding curve for BPR in the test set.

Table 6.15: Mean Squared Error (MSE), in the **training set**, and its Standard Deviation (Std) in parentheses **at the middle** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Parity 8*	*Parity 8 Generalisation*	*Sonar*	*Digits*
BPR	1	0.27365 (0.03482)	0.26945 (0.04023)	0.24063 (0.01465)	0.89120 (0.00832)
	2	0.11417 (0.01849)	0.10756 (0.03091)	0.05263 (0.00456)	0.25099 (0.00347)
QPROP	1	0.01264 (0.03873)	0.09267 (0.29303)	0.02494 (0.07879)	0.12445 (0.00581)
	2	0.00741 (0.01869)	0.00000 (0.00000)	0.00103 (0.00308)	0.12614 (0.00587)
SSCSABB	1	0.00000 (0.00000)	0.00000 (0.00000)	0.04645 (0.08325)	0.13415 (0.00501)
	2	0.00000 (0.00000)	0.00000 (0.00000)	0.02586 (0.05028)	0.13655 (0.00822)
	3	0.00004 (0.00013)	0.00000 (0.00000)	0.04738 (0.06194)	0.15305 (0.00989)
	5	0.00000 (0.00000)	0.00000 (0.00000)	0.00101 (0.00093)	0.12428 (0.00203)

nodes. Training was continued for 2500 epochs. Fig. 10 shows characteristic learning curves of the QPROP and the SSCSABB obtained on the training and test sets. Additionally, for comparison, it includes, in both subfigures, the curve corresponding to BPR for the test set. It is not difficult to observe in Fig. 10(a) that the generalisation error of QPROP (dotted line) after reaching a minimum value starts to increase as the training error (dashed line) decreases further. This illustrates clearly the results of memorisation of the training patterns (overtraining). Fig. 10(b) demonstrates that the same does not hold for the SSC-SABB, which exhibits a behaviour qualitatively more similar to that of BPR, and succeeds to avoid overtraining.

Table 6.16: Mean Squared Error (MSE), in the **training set**, and its Standard Deviation (Std) in parentheses **at the end** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)					
Algorithms	*Ver.*	*Parity 8*	*Parity 8 Generalisation*	*Sonar*	*Digits*
BPR	1	0.27054 (0.03421)	0.26563 (0.03860)	0.23105 (0.01319)	0.87256 (0.00957)
	2	0.03362 (0.00852)	0.02977 (0.01358)	0.01018 (0.00172)	0.21153 (0.00253)
QPROP	1	0.00003 (0.00009)	0.03961 (0.12527)	0.02520 (0.07969)	0.09165 (0.00935)
	2	0.00000 (0.00000)	0.00000 (0.00000)	0.00000 (0.00000)	0.09323 (0.00821)
SSCSABB	1	0.00000 (0.00000)	0.00000 (0.00000)	0.04099 (0.08609)	0.10679 (0.00421)
	2	0.00000 (0.00000)	0.00000 (0.00000)	0.01733 (0.05311)	0.11122 (0.00996)
	3	0.00000 (0.00000)	0.00000 (0.00000)	0.03888 (0.06222)	0.12658 (0.01038)
	5	0.00000 (0.00000)	0.00000 (0.00000)	0.00001 (0.00001)	0.09319 (0.00388)

Table 6.17: Mean Squared Error (MSE), in the **test set**, and its Standard Deviation (Std) in parentheses **at the middle** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)				
Algorithms	*Ver.*	*Parity 8 Generalisation*	*Sonar*	*Digits*
BPR	1	0.26891 (0.04299)	0.24775 (0.01923)	0.89036 (0.00934)
	2	0.11608 (0.02705)	0.09976 (0.00543)	0.25707 (0.00485)
QPROP	1	0.10067 (0.28455)	0.19131 (0.03971)	0.26142 (0.00746)
	2	0.02562 (0.06739)	0.19132 (0.04215)	0.26158 (0.01740)
SSCSABB	1	0.00000 (0.00000)	0.11915 (0.05052)	0.20792 (0.00374)
	2	0.00000 (0.00000)	0.10240 (0.02062)	0.20807 (0.00382)
	3	0.00002 (0.00004)	0.11889 (0.03316)	0.20985 (0.00514)
	5	0.00000 (0.00000)	0.10129 (0.00772)	0.20784 (0.00379)

Table 6.18: Mean Squared Error (MSE), in the **test set**, and its Standard Deviation (Std) in parentheses **at the end** of the simulation runs for the algorithms tested.

Mean Squared Error (Standard Deviation)				
Algorithms	*Ver.*	*Parity 8 Generalisation*	*Sonar*	*Digits*
BPR	1	0.26566 (0.04160)	0.23771 (0.01783)	0.87165 (0.01112)
	2	0.04350 (0.01195)	0.09499 (0.00458)	0.23049 (0.00443)
QPROP	1	0.04119 (0.13026)	0.19931 (0.03702)	0.28847 (0.01847)
	2	0.02570 (0.05097)	0.20605 (0.04203)	0.28235 (0.01734)
SSCSABB	1	0.00000 (0.00000)	0.12208 (0.04916)	0.20891 (0.00413)
	2	0.00000 (0.00000)	0.10544 (0.01977)	0.20850 (0.00330)
	3	0.00000 (0.00000)	0.11924 (0.03086)	0.20861 (0.00525)
	5	0.00000 (0.00000)	0.10337 (0.00959)	0.21080 (0.00483)

Table 6.19: Best performing version of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively.

Best performances								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR	0.114 (2)	-	0.107 (2)	0.116 (2)	0.052 (2)	0.099 (2)	0.250 (2)	0.257 (2)
QPROP	0.007 (2)	-	0.000 (2)	0.025 (2)	0.001 (2)	0.191 (1)	0.124 (1)	0.261 (1)
SSCSABB	0.000 (-)	-	0.000 (-)	0.000 (-)	0.001 (5)	0.101 (5)	0.124 (5)	0.207 (5)

Table 6.20: Best performing version of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively.

Best performances								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR	0.033 (2)	-	0.029 (2)	0.043 (2)	0.010 (2)	0.094 (2)	0.211 (2)	0.230 (2)
QPROP	0.000 (2)	-	0.000 (2)	0.025 (2)	0.000 (2)	0.199 (1)	0.091 (1)	0.282 (2)
SSCSABB	0.000 (-)	-	0.000 (-)	0.000 (-)	0.000 (5)	0.103 (5)	0.093 (5)	0.208 (2)

6.3.4 Statistical Analysis and Discussion

In the classification problems we perform the same type of analyses we presented previously. However, here the picture, formed by the corresponding results presented, appears to be much clearer in favour of the SSC-SABB algorithm. We start by reporting again the average MSEs over the ten training runs for each of the four benchmarking problems. Results in Tables 6.15 and 6.16 refer to the training set while these in Tables 6.17 and 6.18 to the test set. Note that for the conventional Parity problem (denoted as 'Parity 8' in the tables) where all possible cases are used for training, there are no results corresponding to the test set. In this task, as discussed, measurement of generalisation is not relevant. As mentioned previously, these tables provide the means to identify which of the versions of each algorithm are favourable as well as to monitor the evolution of the training error and the generalisation error (often called predictive error in statistics). It is not difficult to realise from these tables that, for the BPR, version 2 which has a learning rate of 0.1 (see Table 6.1) is the one exhibiting the best performance. In all problems tested the training error of BPR decreases considerably from the middle to the end of the training period. The same observation is, however, true only for the 'Parity 8–Generalisation' problem when the MSE in the test set is

Table 6.21: T–statistic values for the differences between the mean MSEs of the best performing version of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively (critical value $t_{\{1-0.05/2,18\}} = 2.109$).

T–Statistic								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR–QPROP	12.84	-	11.00	3.94	29.65	-7.22	59.15	-1.55
BPR–SSCSABB	19.53	-	11.00	13.57	35.06	-0.51	99.67	25.28
SSCSABB–QPROP	1.25	-	1.37	1.20	0.03	7.04	0.09	20.25

considered. It is important to note here that the latter (generalisation error) does not increase as training continues from the middle to the end of the run. The same, however, cannot be argued for QPROP as well. In all the problems included in Tables 6.17 and 6.18 the generalisation error of QPROP is, noticeably, higher at the end of the training period than in the middle. In fact, QPROP seems to present considerably different behaviour in the classification than that observed in the Regression ones. It can be seen from the corresponding entries in Tables 6.15 and 6.16 that for all the problems, except the Digits recognition, the performance appears to be sensitive to the choice of the alternative learning rate l, with the version having the larger value being favourable. The same observation holds only for 'Parity 8–Generalisation' task with respect to predictive error (Tables 6.17 and 6.18). We do not have any apparent explanation for this change in the behaviour of QPROP in the class of problems. Let us know consider SSCSABB. First, we employ a smaller number of value combinations (versions) in the classification problems. Second, we employ only version with switching parameter value $T = 1$. This is because, as we discussed in previous chapter, it has been argued in the literature that more conservative (even slower) algorithms, such as the BPR, can perform better than faster algorithms in classification tasks (Reed and Marks, 1999, pp. 152,156). The argument is based on the observation that

Table 6.22: Hypothesis testing for differences between the average Mean Squared Errors of the best performing version of the algorithms tested, **at the middle** of the simulation runs, in the **training and test sets** respectively (0: accept null hypothesis H_0, 1: reject H_0, where the alternative hypothesis is that the first algorithm performs better than the second, and (*) denotes equality of performances).

Hypothesis Testing								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
QPROP–BPR	1	-	1	1	1	0	1	0*
SSCSABB–BPR	1	-	1	1	1	0*	1	1
SSCSABB–QPROP	0*	-	0*	0*	0*	1	0*	1

in such tasks high accuracy in the approximation of the minimum in error is not required as long as the network outputs identify unambiguously the correct class. Furthermore, avoiding overtraining appears to be much more important in classification tasks so that algorithms incorporating some kind of mechanism to saturate before memorising the particular training patterns are preferable. The Stochastic Approximation algorithm (SA), which forms the supervising part of the SSC algorithm, can be considered to have such a mechanism.

Unlike the other two algorithms, all versions of SSC-SABB seem to tackle the Parity task very easily since they produce virtually zero MSE from the middle of the training period both on the training and the test sets. Diversification in the behaviour of the SSC-SABB versions is observed, however, in the other two tasks, the Sonar problem and the Digits recognition. The version with constant stepsize (version 5) seems to be favourable with respect to training error for all the problems. However, the same picture is not retained when the error in the test set is considered (Tables 6.17 and 6.18), where all versions exhibit similar performance. From the same tables, no considerable increase in the generalisation error can be observed, unlike the case of QPROP we discussed above. This provides support to our previous hypothesis that the supervisor algorithm along with the switching

Table 6.23: T–statistic values for the differences between the mean MSEs of the best performing version of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively (critical value $t_{\{1-0.05/2,18\}} = 2.109$).

T–Statistic								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
BPR–QPROP	12.47	-	6.93	1.08	18.68	-8.84	39.11	-9.17
BPR–SSCSABB	12.47	-	6.93	11.51	18.66	-2.49	80.71	12.59
SSCSABB–QPROP	1.00	-	1.00	1.59	-2.77	7.93	-0.48	13.23

mechanism form a successful device to tackle the problem of overtraining.

The results corresponding to the most successful versions for each algorithm from the previous tables are summarised in Tables 6.19 and 6.20 (rounded to three decimal points). Version number indicators are included in parentheses. The examination of the entries in the tables verify the superiority of the SSC-SABB for all the tasks with respect both of the training efficiency and generalisation. They also provide clear empirical evidence in support of the points raised in our previous discussion about the issue of overtraining.

We proceed by discussing a series of hypotheses testing analyses in order to define the statistical significance of the pairwise differences in performance observed among the versions of the algorithms selected to be included in Tables 6.19 and 6.20. To this end, we calculate and present in Tables 6.21 and 6.23 the t-statistic values corresponding to the differences of the means of MSEs over the 10 runs for the pairs of algorithms shown in the first column of the tables. Since the number of the runs is the same as in the Regression problems the critical region, corresponding to the t-statistic with 18 degrees of freedom (10+10-2) and at a significance level of 0.05, is again [-2.109, 2.109]. Values out of this interval suggest rejection of the null hypothesis (H_0) of equality for the means. This is actually the case for the generalisation error difference between SSC-SABB and

Table 6.24: Hypothesis testing for differences between the average Mean Squared Errors of the best performing version of the algorithms tested, **at the end** of the simulation runs, in the **training and test sets** respectively (0: accept null hypothesis H_0, 1: reject H_0, where the alternative hypothesis is that the first algorithm performs better than the second, and (*) denotes equality of performances).

Hypothesis Testing								
Algorithms Compared	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
	Train	Test	Train	Test	Train	Test	Train	Test
QPROP–BPR	1	-	1	0*	1	0	1	0
SSCSABB–BPR	1	-	1	1	1	0	1	1
SSCSABB–QPROP	0*	-	0*	0*	0	1	0*	1

Table 6.25: Number of successful runs (out of 10) in the **test sets**, with the corresponding mean number of epochs and their standard deviations in parentheses (rounded to the nearest integer).

Convergence statistics					
Algorithms	*Ver.*	*Parity 8*	*Parity 8 Generalisation*	*Sonar*	*Digits*
BPR	1	0	0	0	0
	2	10 (529/26)	1 (496/0)	10 (675/107)	10 (1510/151)
QPROP	1	10 (40/16)	10 (38/7)	3 (185/42)	7 (457/157)
	2	10 (33/14)	10 (69/82)	2 (230/14)	7 (310/108)
SSCSABB	1	10 (15/2)	10 (13/5)	8 (75/44)	10 (151/32)
	2	10 (17/2)	10 (12/3)	9 (136/59)	10 (141/22)
	3	10 (26/9)	10 (18/8)	7 (187/142)	10 (216/75)
	5	10 (17/2)	10 (14/5)	10 (55/22)	10 (114/17)

Table 6.26: Average classification error rates and the corresponding standard deviations, in the **test set**, **at the end** of simulation runs.

Classification Error Rates (%)									
Algorithms	*Ver.*	*Parity 8*		*Parity 8 Generalisation*		*Sonar*		*Digits*	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
BPR	1	51.80	3.03	46.64	4.38	40.87	4.79	76.34	1.88
	2	0.04	0.04	0.16	0.11	11.63	3.15	16.06	1.65
QPROP	1	0.00	0.00	4.45	0.44	22.11	4.01	18.88	1.75
	2	0.00	0.00	2.66	0.70	21.15	3.99	18.50	1.74
SSCSABB	1	0.00	0.00	0.00	0.00	17.21	3.49	14.92	1.59
	2	0.00	0.00	0.00	0.00	12.98	3.27	14.82	1.59
	3	0.00	0.00	0.00	0.00	15.77	3.50	14.80	1.59
	5	0.00	0.00	0.00	0.00	11.44	3.13	14.58	1.58

Table 6.27: Average classification error rates and the corresponding standard deviations, in the **test set**, after 10 epochs.

Classification Error Rates (%)					
Algorithms	*Ver.*	*Parity 8*		*Parity 8 Generalisation*	
		Mean	Std	Mean	Std
BPR	1	52.11	3.04	48.13	4.40
	2	33.13	2.92	30.00	3.74
QPROP	1	48.67	3.08	48.59	4.33
	2	20.78	2.14	23.05	3.23
SSCSABB	1	0.00	0.00	1.41	0.52
	2	0.00	0.00	0.08	0.08
	3	20.82	1.99	9.06	1.24
	5	0.00	0.00	0.00	0.00

QPROP for the Sonar and Digits problems both at the middle (Table 6.21) and at the end (Table 6.23) of the simulation runs. The corresponding entries in Tables 6.22 and 6.24 respectively, can help us define the direction of this difference. In these tables, values of 1 indicate acceptance of the alternative hypothesis (H_1), that the first algorithm of the pair shown in the first column has achieved statistically significant lower error than the second algorithm. Hence, it can be easily verified that the previous comparison both for the Sonar and Digits tasks turns in favour of SSC-SABB. In the rest of the cases, it is indicated that the SSC-SABB performs equally to QPROP. Additionally, SSC-SABB clearly outperforms BPR.

The next comparisons are performed with respect to robustness as expressed by the number of runs the learning algorithm achieved a prespecified level of generalisation error (MSE in the test set). The MSE values used as convergence criterion for every task are shown in Table 6.14. From Table 6.25 we can safely conclude that our training algorithm (SSC-SABB) outperforms the other two both in robustness and in speed.

However, the ultimate test for a classification system is the rate of misclassification it achieves when presented with unknown pattern. Consequently, Table 6.26 presents the average percentage of classification error rates for all the versions of the algorithms measured over the test sets at the end of the training period. It is worth pointing out that even the worst performing version of SCC-SABB outperforms the best version of either of the other algorithms. Finally, because in the Parity problems QPROP also presented satisfactory behaviour at the end of the training phase, we present in Table 6.27 the same type of results for the Parity problems but after only 10 epochs of training. It is easily observed that SSC-SABB clearly excels in both tasks with considerably significant magnitude for the performance differences.

6.4 Concluding Remarks

In summary, the algorithm proposed in this study (SSC-SABB) for application to Neural Network training produced in all cases and tasks tested performance superior to that of BPR at least by an order of magnitude. In the regression problems, SSC-SABB shows at least the same training behaviour as the QPROP and in many cases much better generalisation performance. However, in the classification problems excels in all cases both in training speed and generalisation due, we believe, its resistance to overtraining as a result of the supervision process imposed by the switching mechanism. In general, we can safely conclude that the neural networks trained with the SSC-SABB are not only excellent classifiers but also at least as competent function approximators (both for simple and complex (chaotic) functions) as those trained by algorithms (e.g. QPROP) reported to be among the most successful (Schiffman et al., 1994). This was supported in this study by a series of elaborate statistical analyses ranging from initial exploratory comparisons to statistical hypothesis testing for the observed differences in performances.

Chapter 7

Conclusions and Future Research

7.1 Conclusions

In this thesis we studied a novel class of algorithms for unconstrained optimisation with particular focus to the issues arising in the field of noisy optimisation. These algorithms were developed using an innovative framework for design of efficient and robust algorithms, namely the Supervisor and Searcher Co-operation (SSC) framework. This framework provides a systematic way to incorporate desirable characteristics of existing algorithms into a new scheme with improved qualitative properties. Thus, it constitutes an algorithmic *synthesis* tool which can be used to design algorithms which meet the complex requirements needed to successfully tackle noisy optimisation problems.

A detailed survey of the most popular classes of algorithms of currently used algorithms in the area of noisy unconstrained optimisation (Chapter 1) revealed that, on one hand, although gradient based algorithms compare favourably to those that do not use gradient information in terms of efficiency (speed), they suffer significant inconsistency of their performances under the presence of strong (possibly stochastic) noises in the gradient estimators and increased sensitivity to the selection of the appropriate stepsizes. Similar observations have been reported to hold also — and indeed verified in this work as well — for versions of these algorithms which utilise line search procedures of the inexact or non-monotone

type (see the results in Chapter 3 for the GBB algorithm which uses inexact line search). On the other hand, schemes that are based only on the function values, although proven to be, in general, more robust to the effects of noises, they need significantly greater number of function evaluations, especially as the accuracy required increases.

Therefore, based on these observations, a natural and logical conclusion is that an apparatus is needed to facilitate the synthesis of the diverse qualities of different algorithms (such as the one offered by the SSC framework), to successfully address the deficiencies of individual algorithms reported in the literature. The aim of the study reported in this thesis was to explore the properties of algorithms developed within the SSC framework, focusing in particular on their behaviour in practice under the presence of stochastic noises. In summary, the findings of the research suggest that the algorithms developed according to the Supervisor and Searcher Co-operation framework are demonstrably efficient in the deterministic optimisation case but their main advantage is that they are able to successfully address the difficulties arising in optimisation under the presence of strong (stochastic) noises. They are also amenable to modifications and extensions aiming to meet specific application requirements, while the resulting algorithms retain the desirable properties of the original algorithm on which they are based. Finally, in real world applications involving training neural networks for regression and classification problems, the SSC-based algorithm exhibited significantly better performance than the two algorithms used as benchmarks for comparisons, in the majority of the cases examined. Specifically, it was demonstrably faster with respect to reduction of the error in the training set but more importantly it showed increased ability to avoid overtraining and hence to generalise (perform successfully in unknown examples), which is the ultimate goal in neural network training.

When examining the results in detail, there are a number of notable findings that should be mentioned. To begin with, the common characteristic shared by the algorithms designed according to the SSC framework is that they all operate

in two phases. In the first phase, which can be characterised as the "switching" decision phase, the functions are evaluated at the points proposed by the different participant "parent" algorithms, the Supervisor and the Searcher, and a decision is made to switch or not from one to the other, according to a criterion which, depending on the value of a "preference" (switching) parameter T, favours the proposals of either the former or the latter.

There are two important points to note here. First, in the noise-free case the value of the switching parameter defines whether the algorithms would be monotone or not in terms of successive function values. For example, if the Supervisor is decreasing and $T \leq 1$, the SSC algorithm will be also decreasing in the noise-free case. In such cases, preference is placed on the Supervisor algorithm which is usually slower so, as mentioned in Chapter 3, the corresponding SSC algorithm will be slower. Values of T greater than 1 cause the algorithm to be non-monotone which may be beneficial in terms of performance on a number of problems, as indeed it is indicated by results in the same chapter.

Second, in the stochastic noises case, the SSC algorithms are always non-monotone, and the preference parameter T controls the level of non-monotonicity. In doing so, in effect, it controls the balance between efficiency, in terms of speed to convergence, and robustness, in terms of the range of problems in which the algorithms can consistently be successful. For example, increased preference on the steps proposed by the Search Engine "parent" algorithm combined with the occasional appearance of extreme values of the disturbances in the stochastic case, may lead the algorithm to a point far from the solution, from which a larger number of iterations may be needed in order to recover. On the other hand, higher levels of non-monotonicity allow the algorithm to easily overcome poor local optima of the noisy objective function. In fact, our results in Chapters 3, 4 and 6 indicate that in the majority of the noisy optimisation cases a value of 1 in the switching parameter T will provide a satisfactory trade-off between efficiency and robustness.

Based on the above discussion, it is not difficult to realise that a thorough study

of the properties of the SSC algorithms, and especially their behaviour in practice under a variety of conditions, e.g. different types of noises, is not a straightforward task. It involves their assessment with respect to a number of different criteria including efficiency, robustness, and amenability to modifications (flexibility), as well as practical applicability to realistic applications. To the best of our knowledge, no such work has been reported in the literature to date. Therefore, the present study constitutes an original and innovative contribution which enriches the body of scientific knowledge in the field of optimisation algorithms, as well as it provides useful insights for the practitioners in the area.

In order to evaluate the properties of algorithms developed using the SSC framework in a systematic way we started by defining a basic algorithm and evaluating its performance through a series of experiments involving a wide range of non-trivial deterministic and stochastic problems. Our corresponding results indicate that, in agreement to relevant theoretical analysis, the basic algorithm is more efficient in terms of speed, and robust in terms of the number of problems solved that the algorithm used as Supervisor, which was the classical Stochastic Approximation algorithm. It is also indicated that the SSC algorithm is at least as fast as its Search Engine algorithm in both the deterministic and stochastic cases. Additional tests, with respect to various levels of noises and a range of different types of gradient estimators, showed consistent behaviour and not significant sensitivity to the choice of controlling parameters for its constituent "parent" algorithms.

Subsequently, in order to test the flexibility of the SSC-based algorithm to adapt in particular requirements arising in realistic application areas, such as reduced computational cost, we introduced and tested a number of modifications of the basic algorithm. These included also inexact variants which utilise approximations to the objective function values during the switching decision phase, instead of exact function evaluations, as well as an SSC algorithm involving the combination of more than two "parent" algorithms. The corresponding results demonstrate that these algorithms were significantly successful in tackling the

issues designed to address.

Finally, in order to assess the applicability of the SSC algorithms in real world problems we proposed an adaptation to the basic algorithm examined in the present work for use in Multilayer Neural Network training. The corresponding evaluations were performed through a series of experiments on a number of regression and classification problems, designed to cover the complex issues associated with neural networks learning, such as overtraining and mainly generalisation ability. To this end, the results obtained were subjected to comprehensive statistical analyses involving also statistical hypothesis testing of the observed differences in performances. It is notable that this type of analysis with respect to learning algorithms is not often met in the relevant literature, so it can be considered an original methodological contribution of the present study in the area of the assessment of the properties of neural network training algorithms. The outcome of these investigations verified all the observations presented previously about the properties of the SSC algorithms. In addition, it can be safely argued that the neural networks trained with the SSC-based algorithms are not only excellent classifiers but also at least as competent function approximators as those trained by algorithms reported to be among the most successful in this field.

We consider this study to make an original contribution to the novel research area emerging in a number of science and engineering fields, that of combination or *synthesis* of methods. Examples of the new concepts developed and investigated in this area, which seems to flourish the last few years, can be found for instance in the field of pattern recognition, image analysis or biometrics, in the form of the design of Multiple Classifier Systems. In these, effective methods are sought to combine the outputs of a variety of individual classifiers (either at the decision or raw output (posterior probabilities of classes)) in order to obtain systems with improved performance. An extensive collection of current work in this area can be found in (Kittler and Roli, 2001). A second example of these ideas can be drawn from the combination of various types of neural networks into *ensembles* or *committees* (see, for instance, (Bishop, 1995, pp. 365–369)) and (Sharkey, 1999)

or alternatively in the same field the fusion of neural networks and fuzzy logic methodologies (Lin and Lee, 1996). In most of the above cases a common characteristic is that the participant component algorithms (be it a statistic classifier or a trained network) are combined after they have produced individually some kind of output. The additional innovative contribution of the material reported in this thesis, about algorithms developed within the Supervisor Searcher Co-operation framework, although focused here in the field of noisy optimisation, is that, in general, it investigates a method to perform a synthesis at the algorithmic (as opposed to output) level. So, it goes a step further in providing an understanding of possible ways to successfully integrate in a single scheme diverse desirable properties of existing methods.

7.2 Pointers to Future Research

It is a natural consequence of every scientific research, for every answer discovered, to open a number of new questions, thus identifying possible paths for future research to be pursued. This work although orientated mainly towards the empirical exploration of SSC-based algorithms in noisy optimisation problems, indicates paths for possibly fruitful further research in a number of areas both in the side of theoretical analysis and that of applications, to which we shall provide some pointers in the following.

On one hand, regarding the theoretical analysis of algorithms designed using the SSC framework, we have presented results establishing the corresponding asymptotic behaviour in the presence of Normally distributed independent noise variates either in the objective function values or in the gradient estimators. However, in practice, such i.i.d. disturbances are a rare occurrence. Usually, the error components exhibit some kind of dependence across the sequence of the iterations of the algorithms, and indeed the asymptotic behaviour, for instance, of classical Stochastic Approximation algorithms have been studied extensively in the literature under a variety of assumptions and models for this dependence. Therefore,

it would be an intriguing albeit highly challenging task to further the above mentioned results for the SSC algorithms to cover the asymptotic behaviour under such interdependent stochastic errors in the estimators used.

On the other hand, the application/development orientated side of possible lines for future research can be divided in two areas. In the first, one can consider the natural and direct extension of the work presented in this thesis, that is applying the algorithms already developed to new cases. These can involve noisy optimisation as their main subject or as a subproblem to be addressed in order to achieve their final goal. Examples of the former can be drawn, for instance, from the wide and increasingly important field of Simulation Optimisation, and of course the more classical area of Constrained Optimisation. For the latter, a perfect example is the neural network training task, as introduced in the present work. The ultimate goal in this case is not optimisation over the error function but generalisation ability of the network, which however requires optimisation of an error function at some stage. So, the application of the variants and inexact versions of the basic algorithm, introduced here, to neural network training to perform either regression or classification tasks forms an interesting and straightforward line in our research plans.

In the second path of future research leading towards the application and development side, again, the field of Neural Network training can be a prominent candidate. In particular, regarding the development of new types of algorithms within the SSC framework, the design of an on-line training version, for instance, of the basic algorithm (SABB) examined here, would be of particular interest. This is because, as explained in Chapter 5, the on-line version will use gradient estimators, analogous to stochastic gradients in the conventional optimisation setting, a case in which SSC algorithms have not yet been tested. Also, it is reported that, at least in the classical Backpropagation case, the on-line version is usually faster. It would be interesting to explore whether or not the same holds for the SSC algorithms as well. Finally, to return to the area of classical noisy optimisation a really interesting continuance of the present work appears to be the development

of practical algorithms with a varying sequence of switching parameters T_k along the iterations of the algorithm. Although most of the theoretical results presented here indicate that this can produce effective novel SSC algorithms, these ideas have not yet been tried in practice.

All the above form only a small sample of the open questions, arising from the work reported in this thesis, which lead to promising fruitful paths for further explorations. The adventure continues.

Appendix A

List of Publications

Parts of this thesis have been included in the following:

- Liu, W., Dai, Y., and Sirlantzis, K. (1999). A class of faster and robust algorithms for minimization problems with stronger noises. Work. Paper 35, Canterbury Business School, Univ. of Kent at Canterbury, Canterbury, UK.

- Liu, W. and Sirlantzis, K. (2001a). SSC Minimization Algorithms. In Floudas, C. and Pardalos, P., editors, *Encyclopedia of Optimization*, pages 253–257, Boston. Kluwer Academic Publishers.

- Liu, W. and Sirlantzis, K. (2001b). The Supervisor Searcher Co-operation Principle in practice: I - Novel Gradient Algorithms and their Extensions for Deterministic and Stochastic Minimization Problems. (submitted to EJOR).

- Sirlantzis, K. and Liu, W. (2001). The Supervisor-Searcher Co-operation framework: A class of powerful algorithmns for stochastic optimisation. In *Euro 2001-European Operational Research Conference*, page 218, Rotterdam, The Netherlands. Association of European Operational Research Societies.

- Sirlantzis, K., Feng, J., and Liu, W. (2001). Minimization Algorithms based on Supervisor and Searcher Co-operation: III - novel algorithms for

stochastic optimization. (submitted to JOTA).

Also material presented in this thesis has been included, as cordially acknowledged by the authors, in:

- Liu, W. and Dai, Y. (2001). Minimization algorithms based on Supervisor and Searcher Co-operation: I - faster and robust gradient algorithms for minimization problems with strong noise. *JOTA*, 111(2):359–379.
- Liu, W. and Dai, Y. (1999). Minimization algorithms based on Supervisor and Searcher Co-operation: II - a class of novel algorithms for minimization problems with strong noise. (submitted to JOTA).

Appendix B

Proofs of Theorems

Theorem 2.3

The results in this section are reproduced from (Sirlantzis et al., 2001b). The author wishes to acknowledge major contributions to this material by Prof. WB. Liu, and Dr. F. Feng.

In order to prove Theorem 2.3 the following intermediate results are necessary:

First, it is crucial for our proofs to observe that (see also the definition in Section 2.3.1)

$$\begin{aligned} F(x_{k+1}) &\leq F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))I_A + [TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\ &\quad -(\chi_{k+1} - T\zeta_{k+1})]I_{A^c} \\ &= F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + [(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\ &\quad -(\chi_{k+1} - T\zeta_{k+1})]I_{A^c} \\ &= F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + U_k \end{aligned} \tag{B.1}$$

where

$$A = \{T[F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + \zeta_{k+1}] \leq [F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) + \chi_{k+1}]\}$$

Denote $R_k = E(U_k|\mathcal{F}_k)$, and assume that $\sup_{x,y\in\mathbb{R}^d}(F(x) - F(y)) < \infty$. It is essential to estimate R_k in order to apply the martingale convergence theory.

Lemma B.1 *Let* $C = \min_{x \in \mathbb{R}^d} F(x)$ *and suppose that* $\chi_k \sim N(0, \gamma(k)), \zeta_k \sim N(0, \theta(k))$ *for* $\gamma(k) > 0, \theta(k) > 0$ *then for* $k \geq 1$

$$R_k \leq o(1) \frac{\sqrt{\gamma(k) + \theta(k)T^2}}{\sqrt{2\pi}} \exp\left(- \frac{[o(C) + (1-T)C]^2}{2(\gamma(k) + \theta(k)T^2)} \right)$$

provided that C *is large enough and* $T < 1$.

Proof: First of all we note that

$$\exp(-\frac{u^2}{2})(\frac{1}{u} - \frac{1}{u^3}) = \int_u^\infty \exp(-\frac{w^2}{2})(1 - \frac{3}{w^4})dw$$

for $u > 0$. Hence

$$\int_u^\infty \exp(-\frac{w^2}{2})dw \geq \exp(-\frac{u^2}{2})(\frac{1}{u} - \frac{1}{u^3}) \tag{B.2}$$

holds true for any $u > 0$.

Furthermore

$$\begin{aligned} R_k &= E([(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) - (\chi_{k+1} - T\zeta_{k+1})]I_{A^c}|\mathcal{F}_k) \\ &= E(E([(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) - (\chi_{k+1} - T\zeta_{k+1})]I_{A^c}|\mathcal{F}_k \\ &\quad \cup \{\xi_{k+1}, \eta_{k+1}\})|\mathcal{F}_k) \end{aligned}$$

where $\mathcal{F}_k \cup \{\xi_{k+1}, \eta_{k+1}\}$ is the sigma-algebra $\sigma(\mathcal{F}_k \cup \{\xi_{k+1}, \eta_{k+1}\})$. Define

$$\bar{R}_k = E([(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) - (\chi_{k+1} - T\zeta_{k+1})]I_{A^c}|\mathcal{F}_k \cup \{\xi_{k+1}, \eta_{k+1}\}),$$

we have

$$\begin{aligned} \bar{R}_k &= \int_{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]}^\infty \exp\left(- \frac{u^2}{2(\gamma(k) + \theta(k)T^2)} \right) \\ &\quad \cdot [(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + u] \frac{1}{\sqrt{2\pi(\gamma(k) + \theta(k)T^2)}} du \\ &= \frac{1}{\sqrt{2\pi}} \int_{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1})]/\sqrt{\gamma(k) + \theta(k)T^2}}^\infty \exp\left(- \frac{u^2}{2} \right) \\ &\quad \cdot [(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) + \sqrt{\gamma(k) + \theta(k)T^2}u] du \end{aligned}$$

Since

$$\begin{aligned}
& F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\
= \; & F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\
& +(1-T)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\
= \; & o(C) + (1-T)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1})) \\
> \; & 0
\end{aligned}$$

we obtain, using inequality (B.2)

$$\begin{aligned}
\bar{R}_k \leq \; & \frac{\sqrt{\gamma(k)+\theta(k)T^2}}{\sqrt{2\pi}}\Big[\frac{(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))}{F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))} \\
& -\frac{(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))}{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]^3} \\
& +1\Big]\exp\left(-\frac{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]^2}{2(\gamma(k)+\theta(k)T^2)}\right) \\
= \; & \frac{\sqrt{\gamma(k)+\theta(k)T^2}}{\sqrt{2\pi}}\Big[\frac{(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))}{o(C) + (1-T)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))} \\
& -\frac{(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))}{[o(C) + (1-T)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]^3} \\
& +1\Big]\exp\left(-\frac{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]^2}{2(\gamma(k)+\theta(k)T^2)}\right) \\
= \; & \frac{\sqrt{\gamma(k)+\theta(k)T^2}}{\sqrt{2\pi}}\Big[\frac{(T-1)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))}{(1-T)F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))} + o(1) \\
& +1\Big]\exp\left(-\frac{[F(x_k - \beta_k(\nabla F(x_k) + \eta_{k+1})) - TF(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))]^2}{2(\gamma(k)+\theta(k)T^2)}\right) \\
= \; & o(1)\frac{\sqrt{\gamma(k)+\theta(k)T^2}}{\sqrt{2\pi}}\exp\left(-\frac{[o(C) + (1-T)C]^2}{2(\gamma(k)+\theta(k)T^2)}\right)
\end{aligned}$$

which implies the conclusion of the lemma.

If χ_k, ζ_k $(k \geq 1)$ are bounded, then we have the following stronger estimates for R_k.

Lemma B.2 *Let $C = \min_{x \in \mathbb{R}^d} F(x)$ and suppose that χ_k, ζ_k $(k \geq 1)$ are bounded then*

$$R_k \leq 0$$

provided that C is large enough and $T < 1$.

Proof: Denote M_1 as the bound of $|\chi_k| + |\zeta_k|$. We have

$$R_k \leq [(T-1)C + M_1]E(I_{A^c}|\mathcal{F}_k)$$

Thus $R_k < 0$ if C is large enough. This completes the proof.

We can now proceed to the *Proof* of Theorem 2.3:

Proof: From Eq. (B.1), we see that

$$E(F(x_{k+1})|\mathcal{F}_k) \leq E(F(x_k - t_k(\nabla F(x_k) + \xi_{k+1}))|\mathcal{F}_k) + R_k.$$

Using Taylor formula and the fact that $F \in C^2(I\!R^d)$, the equation above turns out to be

$$\begin{aligned} E(F(x_{k+1})|\mathcal{F}_k) &\leq E(F(x_k)|\mathcal{F}_k) - t_k E((\nabla F(x_k))^T(\nabla F(x_k) + \xi_{k+1}))|\mathcal{F}_k) \\ &\quad + \frac{1}{2}t_k^2 E((\nabla F(x_k) + \xi_{k+1})^T H_k(\nabla F(x_k) + \xi_{k+1})|\mathcal{F}_k) + R_k \\ &= F(x_k) - t_k(\nabla F(x_k))^T \nabla F(x_k) \\ &\quad + \frac{1}{2}t_k^2(\nabla F(x_k))^T H_k \nabla F(x_k) + (\sum_{i=1}^{d} E\xi_{k+1,i}^2 h_{ii}^{(k)}) + R_k \end{aligned} \tag{B.3}$$

where $h_{ij}^{(k)} = \frac{\partial^2 F(x)}{\partial x_i \partial x_j}|_{x_k}$, $i, j = 1, \ldots, d$, $H_k \equiv [h_{ij}^{(k)}]$ the Hessian of F at x_k, and we have used the property that $\{\xi_k\}_{\{k\geq 1\}}$ is a martingale difference with respect to $\mathcal{F}_k$.

Denoting $M = \sup_{x_k}[\frac{1}{2}(\nabla F(x_k))^T H_k \nabla F(x_k) + (\sum_{i=1}^{d} E\xi_{k+1,i}^2 h_{ii}^{(k)})]$, then Eq. (B.3) becomes

$$E(F(x_{k+1})|\mathcal{F}_k) \leq F(x_k) - t_k(\nabla F(x_k))^T \nabla F(x_k) + t_k^2 M + R_k \tag{B.4}$$

Existence of $\lim_{k\to\infty} F(x_k)$ follows from Lemma 4.2 and Lemma 1.10 on page 9 in (Ljung et al., 1992) if χ_k, ζ_k are bounded.

When $\zeta_k \sim N(0, \gamma(k))$ and $\chi_k \sim N(0, \theta(k))$, by combining Lemma B.1 and

Lemma 1.10 on page 9 in (Ljung et al., 1992), we see that the limit also exists almost surely.

Now we turn to the second conclusion. Suppose that there is a set B with $P(B) > 0$ and

$$\liminf_{k\to\infty} dis(x_k, S) = \delta > 0$$

for $S \in B$. If χ_k, ζ_k are bounded, Lemma B.2 and Eq. (B.4) infer that

$$\begin{aligned} E(E(F(x_{k+1})|\mathcal{F}_k)I_B) &\leq E(F(x_0)I_B) - \textstyle\sum_k t_k E((\nabla F(x_k))^T \nabla F(x_k) I_B) + \sum_k t_k^2 M I_B \\ &\to -\infty. \end{aligned}$$

However F is a function bounded below and $E(E(F(x_{k+1})|\mathcal{F}_k)I_B) > -\infty$. Thus the second conclusion follows for the first case. When $\zeta_k \sim N(0, \gamma(k))$ and $\chi_k \sim N(0, \theta(k))$, the same conclusion can be similarly proved.

Theorem 2.4

Remark B.1 *The proof of Theorem 2.4 is similar to the proof of Theorem 2.3 above.*

Theorem 3.1

The results in this section are reproduced from (Liu and Dai, 2001).

Proof: For ease of exposition, we assume that $f \geq 0$. It follows from the definition of the algorithm that for any $k \geq 0$

$$\begin{aligned} f(x_{k+1}) &\leq \min \ (f(x_k - t_k g_k), T_k f(x_k - t_k g_k)) \leq f(x_k - t_k g_k) \\ &= \quad f(x_k) - t_k g_k^T g_k + t_k^2 g_k^T H_k g_k / 2, \end{aligned}$$

where H_k is the Hessian matrix of f at a point θ_k in the line segment $[x_k, x_{k+1}]$. Therefore for $k \geq 1$

$$f(x_{k+1}) \leq f(x_0) - \sum_0^k t_k |g_k|^2 + \sum_0^k C t_k^2 |g_k|^2,$$

where C only depends on f. Then there is an $\epsilon > 0$ such that

$$t_k(1 - Ct_k) \geq c't_k, \text{ after } t_k \leq \epsilon,$$

where $c' > 0$ is a constant independent of k. Therefore if there is a $N > 0$ such that $t_k \leq \epsilon$ after $k \geq N$, then $\sum_0^k t_k|g_k|^2$ is convergent as $k \to \infty$ as f is bounded below.

Theorem 3.2

The results in this section are also reproduced from (Liu and Dai, 2001).

Proof: The proof of this theorem could be made rather technical, as first it should be proved that the BB algorithm is locally R-linearly convergent. Although its local convergence has been proved for the quadratic objective function in (Raydan, 1993), R-linear convergence (or even simply convergence) of the BB algorithm for general convex objective functions still needs many extra tedious estimates to prove. On the other hand, the principle of the proof is quite simple. Therefore here the details for these estimates are skipped.

First the R-linear convergence of the BB algorithm is shown if x_k is sufficient close to x^*, where x_k is generated by the BB formula. For any k, given $\hat{x}_k = x_k$ and $\hat{x}_{k+1} = x_{k+1}$, we define $\{\hat{x}_{k+j} : j = 0, 1, 2, \ldots\}$ to be the iterations generated by the BB algorithm for the quadratic function

$$\hat{f}(x) = f(x^*) + \frac{1}{2}(x - x^*)^T H(x - x^*),$$

where H is the Hessian matrix of f at x^*. It follows from (Raydan, 1993) that $\{\hat{x}_{k+j}\}$ converges to x^* as $j \to \infty$, if x_k is very near x^*.

Then for any $1 \leq l \leq m$, where m is some fixed integer, if

$$|\hat{x}_{k+j} - x^*| \geq c_1|\hat{x}_k - x^*|, \quad j = 1, \ldots, l,$$

where $c_1 > 0$ is constant, it can be proved by induction that there exists a positive

constant $c_2 > 0 = c_2(m, c_1, H)$, independent of k, such that

$$|x_{k+j} - \hat{x}_{k+j}| \leq c_2 |x_k - x^*|^2, \quad j = 1, \ldots, l. \tag{B.5}$$

Furthermore, it can be shown (see (Dai and Liao, 1999)) that there exist a constant $c_3 \in (0, 1)$ and an integer m which depends only on c_3 and H such that for any $k \geq 2$, there exists an integer $l \in [1, m]$ such that

$$|\hat{x}_{k+l} - x^*| \leq c_3 |\hat{x}_k - x^*|. \tag{B.6}$$

Now we denote $\delta = \frac{1-c_3}{2c_2}$ and let k_0 be so large that

$$|x_{k_0} - x^*| \leq \delta. \tag{B.7}$$

For this k_0, we let $\hat{x}_{k_0} = x_{k_0}$ and $\hat{x}_{k_0+1} = x_{k_0+1}$, and denote k_1 to be the least index for which

$$|\hat{x}_{k_1} - x^*| \leq c_3 |\hat{x}_{k_0} - x^*|. \tag{B.8}$$

It is obvious that $k_1 - k_0 \leq m$. Then by (B.5), (B.8), (B.7) and the choice of δ, we can show that

$$|x_{k_1} - x^*| \leq |\hat{x}_{k_1} - x^*| + |x_{k_1} - \hat{x}_{k_1}| \leq c_4 |x_{k_0} - x^*|,$$

where $c_4 = \frac{1+c_3}{2} < 1$. Repeating this procedure, we then can obtain an infinite subsequence $\{k_i : i = 1, 2, \ldots\}$ such that

$$k_{i+1} - k_i \leq m \tag{B.9}$$

and

$$|x_{k_{i+1}} - x^*| \leq c_4 |x_{k_i} - x^*|, \quad i = 1, 2, \ldots \tag{B.10}$$

In addition, note that there exists a constant $c_5 > 0$ such that the relation

$$|x_{k+1} - x^*| \leq c_5 |x_k - x^*| \tag{B.11}$$

holds for any large k. By (B.9)-(B.11), we then can prove that

$$|x_{k_0+j} - x^*| \leq M c_6^j |x_{k_0} - x^*|,$$

where $M = c_4^{-1} c_5^{m-1}$ and $c_6 = c_4^{\frac{1}{m}} < 1$. The above relation shows that the BB algorithm is R-linearly convergent.

In the following, it is shown that the SSC-SABB will only take the BB step sizes when it starts from a point very near the minimizer x^*, and then we prove that there is a subsequence of $\{x_k^N\}$ very close to x^* provided N is chosen large enough.

As $f(x^*) > 0$, there is a $r_0 > 0$ such that

$$Tf(x_k - t_k g_k) > f(y_k).$$

as long as $|x_k - x^*| < r_0$ and $|y_k - x^*| < r_0$, since $T > 1$ and $\{t_k\}$ is bounded.

Now let $\{y_k\}$ (k=0,1,2,...) be generated by the SSC-SABB with $y_0 = x_0$ and $T_k \equiv 1$. Then from Theorem 4.1 in (Liu and Dai, 2001) (see Theorem 3.1 in the present thesis), there will be a $k_0 > 0$, depending on x_0 and f, such that

$$|y_{k_0} - x^*| < \min(\bar{r}_0, \delta)/M,$$

where $\bar{r}_0 \leq r_0$ and δ is defined above, since f is strictly convex. Let the sequence $\{x_k^N\}$ be generated via the SSC-SABB from x_0 by letting $T = 1$ for $k = 1, 2, .., N$ and $T = T > 1$ for $k = N+1, ...,$. Let $N_0 = k_0$. Note that the first N_0 elements $\{x_0^N, x_1^N, ..., x_{N_0}^N\}$ of this sequence are identical with $\{y_0, y_1, ..., y_{N_0}\}$ provided $N \geq N_0$. Let x_{k_0+i} (i=1,2,..) be the sequence generated by the BB algorithm from $x_{k_0}^{N_0} = x_{N_0}^{N_0} = x_{N_0}^N$ $(N \geq N_0)$. Clearly $\bar{r}_0$ can be made so small that $|x_{k_0+1} - x^*| <$

$\min(r_0, \delta)/M$. Therefore

$$|x_{k_0+i} - x^*| \leq Mc_6^i |x_{k_0}^{N_0} - x^*| < \min(r_0, \delta), i = 1, 2, .., .$$

Therefore the SSC-SABB will only use the BB step sizes after $k_0 = N_0$ according to its switching rule. Therefore the sequence $\{x_k^N\}$ generated via the SSC-SABB by letting $T = 1$ for $k = 1, 2, .., N$ and $T = T > 1$ for $k = N + 1, ...,$ is as fast as the BB algorithm, as least R-linearly convergent for any $N \geq N_0$.

Appendix C

Multi-step Variant

We reproduce here material originally included in (Liu and Sirlantzis, 2001a; Liu and Sirlantzis, 2001b).

The multi-step SSC gradient algorithm can be defined formally as follows: Let $x_0 \in R^n$ be given. Let $T_k \geq 0$ be given for $k = 0, 1, 2, ...$, let $m > 0$. Assume that $f \geq 0$. Assume that we have x_k, then define

$$x_{k+m}^1 = x_k - t_k g_k,$$

$$x_{k+m}^2 = y_{k+m},$$

where y_{k+m} is defined by $y_k = x_k$,

$$y_{k+l+1} = y_{k+l} - Z_{k+l} \nabla f(y_{k+l}), l = 0, 1, ..., m-1,$$

where Z_k can be any of the $r_k I$, B_k, or B_k' (defined in Section 4.2.1). Then define

$$x_{k+1} = x_{k+m}^1, \text{ if } T_k f(x_{k+m}^1) \leq f(x_{k+m}^2),$$

otherwise

$$x_{k+1} = x_{k+m}^2.$$

Theorem C.1 *(Theorem 4.2 in (Liu and Sirlantzis, 2001b)) Let f be twice continuously differentiable and bounded below. Assume that ∇f is Lipschitz with a global Lipschitz constant. Let x_k be generated by the multi-stage SSC algorithm defined above. Then, assuming $t_k \to 0$, $\{\sum_0^k t_k |\nabla f(x_k)|^2\}$ is convergent as $k \to \infty$ provided $\prod_0^\infty \max(1, T_k) < \infty$.*

Proof: From the definition of the multi-stage SSC algorithm we have also: either

$$f(x_{k+1}) = f(x_k - t_k g_k),$$

or,

$$f(x_{k+1}) = f(x_{k+m}^2) \text{ if } T_k f(x_k - t_k g_k) > f(x_{k+m}^2).$$

Therefore, for both cases

$$f(x_{k+1}) \leq \max(f(x_k - t_k g_k), T_k f(x_k - t_k g_k)) = \max(1, T_k) f(x_k - t_k g_k).$$

It is trivial to see that from this point we can also apply the same line of arguments used in the proof of Theorem 2.1 to show that the sequence $\{\sum_0^k t_k |\nabla f(x_k)|^2\}$ is convergent as $k \to \infty$ provided $\prod_0^\infty \max(1, T_k) < \infty$.

Bibliography

Alpsan, D., Towsey, M., Ozdamar, O., Tsoi, A., and Ghista, D. (1994). Are modified back–propagation algorithms worth the effort? In *IEEE International Conference on Neural Networks (Orlando)*, volume 1, pages 567–571, New York. IEEE.

Alpsan, D., Towsey, M., Ozdamar, O., Tsoi, A., and Ghista, D. (1995). Efficacy of modified backpropagation and optimization methods on a real–world medical problem. *Neural Networks*, 8(6):945–962.

Amari, S., Park, H., and Fukumizu, K. (2000). Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12:1399–1409.

Anderson, E. and Ferris, M. (2001). A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal of Optimization*, 11(2):837–857.

Andradóttir, S. (1998). Simulation optimization. In Banks, J., editor, *Handbook of Simulation*, chapter 9. Wiley, New York.

Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16:1–3.

Avriel, M., editor (1976). *Nonlinear Programming, Analysis and Methods.* Prentice-Hall, Englewood Cliffs, NJ.

Azadivar, F. (1992). A tutorial on simulation optimization. In Swain, J., Goldsman, D., Crain, R., and Wilson, J., editors, *Proc. 1992 Winter Simulation Conf.*, pages 198–204, Piscataway, NJ. Institute of Electrical and Electronics Engineers.

Barton, R. and Ivey, J. (1996). Nelder-mead simplex modifications for simulation optimization. *Management Science*, 42(7):954–973.

Barzilai, J. and Borwein, M. (1988). Two point step size gradient methods. *IMA J. Numerical Analysis*, 8:141–148.

Battiti, R. (1989). Accelerated backpropagation learning: Two optimization methods. *Complex Systems*, 3:331–342.

Battiti, R. (1992). First- and second-order methods for learning: Between steepest decent and newton's method. *Neural Computation*, 4:141–166.

Baum, E. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1:151–160.

Bello, M. (1992). Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptrons. *IEEE Transactions on Neural Networks*, 3(6):864–875.

Benveniste, M., Metivier, M., and Priouret, P. (1990). *Adaptive algorithms and stochastic approximation.* Springer-Verlag, Berlin and New York.

Bertsekas, D. and Tsitsiklis, J. (2000). Gradient convergence in gradient methods with errors. *SIAM J. Optim.*, 10(3):627–642.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford.

Blake, C. L. and Merz, C. J. (1998). The UCI Repository of machine learning databases (Online). http://www.ics.uci.edu/~mlearn/MLRepository.html

(26/07/2002). University of California, Irvine, Dept. of Information and Computer Sciences.

Blum, J. (1954). Approximation methods which converge with probability one. *Ann. Math. Statistics*, 25:382–386.

Borkar, V. (1998). Asynchronous stochastic approximations. *SIAM J. Control Optim.*, 36:840–851.

Box, M. (1966). A comparison of several current optimization methods and the use of transformations in constrained problems. *Computer J.*, 9:67–77.

Brent, R. (1973). *Algorithms for Minimization without Derivatives.* Prentice-Hall Inc, Englewood Cliffs, NJ.

Bridle, J. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. and Hérault, J., editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer–Verlag, New York.

Buntine, W. and Weigend, A. (1994). Computing second derivatives in feedforward networks: a review. *IEEE Transactions on Neural Networks*, 5(3):480–488.

Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comp. Rend. Acad. Sci. Paris*, 25:536–538.

Chen, H. and Zhu, Y. (1996). *Stochastic Approximation.* Scientific and Technology Publishers, Shanghai.

Cheng, B. and Titterington, D. (1994). Neural networks: A review from a statistical perspective. *Statistical Science*, 9(1):2–54.

Chichocki, A. and Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing.* John Wiley and Sons Ltd, Stuttgart.

Chin, D. (1990). Comparative study of several multivariate stochastic approximation algorithms. In *Proc. Stat. Comput. Section ASA*, pages 223–228, Anaheim, CA, USA.

Chin, D. (1997). Comparative study of stochastic algorithms for system optimization based on gradient approximations. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 27(2):244–249.

Cho, S.-Y. and Chow, T. (1999). A fast heuristic global learning algorithm for multilayer neural networks. *Neural Processing Letters*, 9:177–187.

Dai, Y. and Liao, L. (1999). R-linear convergence of the Barzilai and Borwein gradient method. Report ICM-99-03, Institute of Computational Mathematics and Scientific Computing, Beijing, China.

Davidon, W. (1959). Variable metric methods for minimizing. Technical Report ANL-5990, Argonne National Laboratory, Argonne, IL.

Demuth, H. and Beale, M. (1999). Neural Network Toolbox for use with MATLAB.

Dennis, J. and Schnabel, R. (1983). *Numerical Methods for Unconstrained optimization and nonlinear equations.* Prentice-Hall, Englewood Cliffs, NJ.

Dennis, J. and Torczon, V. (1991). Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1:448–474.

Denton, J. and Hung, M. (1996). A comparison of nonlinear optimization methods for supervised learning in multi-layer feedforward neural networks. *European Journal of Operational Research*, 93:358–368.

Diggle, P. and Gratton, R. (1984). Monte Carlo methods of inference for implicit statistical models (with discussion). *J. Roy. Statist. Soc.*, B 46:193–227.

Duin, R. and Tax, D. (2000). Experiments with classifier combining rules. In Kittler, J. and Roli, F., editors, *First International Workshop on Multiple Classifier Systems*, pages 16–29. Springer.

Dupuis, P. and Simha, R. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In *IEEE Transactions on Automatic Control*, volume 275, pages 51–67, Dordrecht, The Netherlands. Kluwer Academic Publishers.

Dvoretzky, A. (1956). On stochastic approximation. In *Proc. Third Berkeley Symp. Math. Stat. and Prob.*, pages 39–55, Berkeley. Univ. of California Press.

Elsner, J. and Tsonis, A. (1992). Nonlinear prediction, chaos and noise. *Bull. Am. Met. Soc.*, 73:49–60.

Elster, C. and Neumaier, A. (1995). A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.*, 15:585–608.

Elster, C. and Neumaier, A. (1997). A trust region method for the optimization of noisy functions. *Computing*, 58:31–46.

Fahlman, S. (1988). An empirical study of learning speed in Back-Propagation neural networks. Technical Report CMU-CS-88-162, CMU, USA.

Fausett, L. (1994). *Fundamentals of Neural Networks: Architectures, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ.

Feng, J., Georgii, H., and Brown, D. (2000). Convergence to global minima for a class of diffusion processes. *Physica A*, 276:465–476.

Fletcher, R. (1987). *Practical Methods of Optimization.* John Wiley and Sons, Chichester, second edition.

Fletcher, R. and Powell, M. (1963). A rapidly convergent descent method for minimization. *Computer J.*, 6:163–168.

Fu, M. (1994). Optimization via simulation: A review. *Annals of Operations Research*, 53:199–247.

Fu, M. (2001a). Perturbation analysis. In Gass, S. and Harris, C., editors, *Encyclopedia of OR/MS*, pages 608–610, Boston. Kluwer Academic Publishers. Centennial Edition.

Fu, M. (2001b). Stochastic simulation. In Gass, S. and Harris, C., editors, *Encyclopedia of OR/MS*, pages 756–758, Boston. Kluwer Academic Publishers. Centennial Edition.

Garavaglia, S. (1991). An application of a counter-propagation neural network: Simulating the standard & poor's corporate bond rating system. In *Proceedings of the IEEE First International Conference on Artificial Intelligence Applications on Wall Street*, pages 278–287.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58.

Gershenfeld, N. and Weigend, A. (1993). The future of time series: Learning and understanding. In Weigend, A. and Gershenfeld, N., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pages 1–70. Addison-Wesley, Reading, MA.

Ghorbani, A. and Bayat, L. (2000). Accelerated backpropagation learning: Extended dynamic parallel tangent optimization algorithm. In Hamilton, H., editor, *Lecture Notes in Artificial Intelligence*, pages 524–532. Springer-Verlag, Berlin.

Ghorbani, A. and Owrangh, K. (2001). Stacked generalization in neural networks: Generalization on statistically neural problems. In *Proc. IEEE/INNS-IJCNN*, pages 1715–1720, Washington DC, USA.

Gill, P., Murray, W., and Wright, M. (1981). *Practical Optimization.* Academic Press, London.

Glad, T. and Goldstein, A. (1977). Optimization of functions whose values are subject to small errors. *BIT*, 17:160–169.

Glasserman, P. (1991a). *Gradient Estimation Via Perturbation Analysis.* Kluwer Academic, Dordrecht, The Netherlands.

Glasserman, P. (1991b). Structural conditions for perturbation analysis derivative estimation: finite time performance indices. *Oper. Res.*, 39:724–738.

Glynn, P. (1987). Likelihood ratio gradient estimation: An overview. In Thesen, A., Grant, H., and Kelton, W., editors, *Proc. 1987 Winter Simulation Conf.*, pages 366–375, Piscataway, NJ. Institute of Electrical and Electronics Engineers.

Glynn, P. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc.

Gorman, R. and Sejnowski, T. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89.

Grippo, L., Lampariello, F., and Lucidi, S. (1986). A nonmonotone line search technique for Newton's method. *SIAM J. Numer. Anal.*, 23(4):707–716.

Hagan, M. and Menhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993.

Hasenjager, M. and Ritter, H. (1999). Perceptron learning revisited: The sonar targets problem. *Neural Processing Letters*, 10:17–24.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume III, pages 11–14, San Diego, CA.

Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison-Wesley, Reading, MA.

Hedlund, P. and Gustavsson, A. (1992). Design and evaluation of modified simplex methods having enhanced convergence ability. *Analytica Chimica Acta*, 259:243–256.

Ho, Y. and Cao, X. (1991). *Perturbation Analysis of Discrete Event Dynamic Systems.* Kluwer Academic, Dordrecht, The Netherlands.

Ho, Y., Shi, L., Dai, L., and Gong, W. (1992). Optimizing discrete event systems via the gradient surface method. *Discr. Event Dyn. Syst.*, 2:99–120.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

Hooke, R. and Jeeves, T. (1961). Direct search solution of numerical and statistical problems. *J. Association for Computing Machinery*, 8:212–221.

Hopfield, J. (1987). Learning algorithms and probability distributions in feed–forward and feed–back networks. *Proceedings of the National Academy of Sciences*, 84:8429–8433.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Hu, M. (1964). Application of the adaline system to weather forecasting. Technical Report 6775-1, Stanford Electronic Laboratories, Stanford, CA.

Humphrey, D. and Wilson, J. (1998). A revised simplex search procedure for stochastic simulation response-surface optimization. In *Proceedings of the 1998 Winter Simulation Conference*, pages 751–760.

Hwang, J. and Moon, S. (1991). Temporal difference method for multi–step prediction: Application to power load forecasting. In *Proceedings of the First Forum on Applications of Neural Networks to Power Systems*, pages 41–45, Seattle, WA.

Jacobs, R. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307.

Jacobson, S. and Schruben, L. (1989). Techniques for simulation response optimization. *Operations Research Letters*, 8:1–9.

Jacobson, S. and Schruben, L. (1992). A review of techniques for simulation optimization. *Operations Research Letters*, 8:1–9.

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

Kantrowitz, M. (1993). The CMU Artificial Intelligence Repository (Online). http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ (26/07/2001). Carnegie Mellon University.

Karidis, J. and Turns, S. (1984). Efficient optimization of computationally expensive objective functions. Technical Report RC10698, IBM, Yorktown Heights, NY.

Khuri, A. and Cornell, J. (1987). *Response Surfaces: Designs and Analyses.* Marcel Dekker, New York.

Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Ann. Math. Statistics*, 23:462–466.

Kinsella, J. (1992). Comparison and evaluation of variants of the conjugate gradient method for efficient learning in feed-forward neural networks with backward error propagation. *Network*, 3:27–35.

Kirkpatrick, S., Gelatt, J., and Vecchi, M. (1982). Optimization by simulated annealing. Technical Report RC 9355, IBM.

Kittler, J. and Roli, F., editors (2001). *Multiple Classifier Systems*, volume 2096 of *LNCS*. Springer.

Kolmogorov, A. (1957). On the representation of continuous functions of many variables by superposition of continuous of one variable and addition. *Dokl. Akad. Nank. SSSR*, 114:953–956. (*American Mathematical Society Translation*, 28:55–59).

Kushner, H. and Clark, D. (1978). *Stochastic approximation methods for constrained and unconstrained systems.* Springer, New York.

Kushner, H. and Yin, G. (1997). *Stochastic approximation algorithms and applications.* Springer, New York.

Lachtermacher, G. and Fuller, J. (1995). Backpropagation in time–series forecasting. *Journal of Forecasting*, 14:381–393.

Lapedes, A. and Farber, R. (1987a). Nonlinear signal processing using neural networks. Technical Report LA-UR-87, Los Alamos National Laboratory, Los Alamos, NM.

Lapedes, A. and Farber, R. (1987b). Nonlinear signal processing using neural networks: Prediction and system modelling. Technical Report LA-UR-97-2662, Los Alamos National Laboratory.

L'Ecuyer, P. (1991). An overview of derivative estimation. In *Proceedings of the 1991 Winter Simulation Conference*, pages 207–217.

L'Ecuyer, P. and Yin, G. (1998). Budget-dependent convergence rate of stochastic approximation. *SIAM J. Optim.*, 8(1):217–247.

Lee, C. (1997). Training feedforward neural networks: an algorithm giving improved generalization. *Neural Networks*, 10(1):61–68.

Lee, K. and Park, J. (1992). Short–term load forecasting using an artificial neural network. *IEEE Transactions on Power Systems*, 7(1).

Lee, Y. and Lippmann, R. (1990). Practical characteristics of neural network and conventional pattern classifiers on artificial and speech problems. In

Advances in Neural Information Processing Systems, volume 1, pages 168–177, San Mateo. Morgan–Kaufmann.

Levin, E., Tishby, N., and Solla, S. (1990). A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574.

Lin, C.-T. and Lee, G. (1996). *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice Hall, New York.

Liu, W. and Dai, Y. (1999). Minimization algorithms based on Supervisor and Searcher Co-operation: I-faster and robust gradient algorithms for minimization problems with stronger noises. Work. Paper 34, Canterbury Business School, Univ. of Kent at Canterbury, Canterbury, UK.

Liu, W. and Dai, Y. (2001). Minimization algorithms based on Supervisor and Searcher Co-operation: I-faster and robust gradient algorithms for minimization problems with strong noise. *JOTA*, 111(2):359–379.

Liu, W., Dai, Y., and Sirlantzis, K. (1999a). A class of faster and robust algorithms for minimization problems with stronger noises. Work. Paper 35, Canterbury Business School, Univ. of Kent at Canterbury, Canterbury, UK.

Liu, W., Dai, Y., and Sirlantzis, K. (1999b). A class of faster and robust algorithms for minimization problems with stronger noises. (submitted to M.J.D. Powel & S. Scholtes (eds.), Proc. of the 19th IFIP TC7 Conf. on System Modelling and Optimization, Kluwer Academic).

Liu, W. and Sirlantzis, K. (2001a). SSC Minimization Algorithms. In Floudas, C. and Pardalos, P., editors, *Encyclopedia of Optimization*, pages 253–257, Boston. Kluwer Academic Publishers.

Liu, W. and Sirlantzis, K. (2001b). The Supervisor Searcher Co-operation Principle in practice: I - Novel Gradient Algorithms and their Extensions for Deterministic and Stochastic Minimization Problems. (submitted to EJOR).

Ljung, L. (1977). Analysis of recursive stochastic algorithms. *IEEE Trans. Automat. Control*, 22:551–575.

Ljung, L. (1986). *System Identification Theory for the User.* Prentice-Hall, Englewood Cliffs, NJ.

Ljung, L., Pflug, G., and Walk, H. (1992). *Stochastic Approxiamtion and Optimization of Random Systems.* Birkhäuser Verlag, Basel, Boston and Berlin.

Luenberger, D., editor (1973). *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, Reading, MA.

Luo, Z. and Tseng, P. (1994). Analysis of an approximate gradient projection method with applications to the backpropagation algorithm. *Optim. Methods Software*, 4:85–101.

Mackey, M. and Glass, L. (1977). Oscillations and chaos in physiological control systems. *Science*, 197:287.

Magoulas, G., Vrahatis, M., and Androulakis, G. (1999). Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Computation*, 11:1769–1796.

Mathews, J. (1992). *Numerical Methods for Mathematics, Science and Engineering.* Prentice-Hall, Englewood Cliffs, CA, 2nd edition.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, New York.

Minsky, M. L. and Papert, S. A. (1988). *Perceptrons. An Introduction to Computational Geometry.* The MIT Press, Cambridge, MA, expanded edition.

Moller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533.

Moody, J. and Darken, C. (1988). Learning with localized receptive fields. In *Connectionist Models: Proceedings of the 1988 Summer School*, pages 76–89, Yale Computer Science.

More, J., Garbow, B., and Hillstrom, K. (1981). Testing unconstrained optimization software. *ACM Trans. Math. Software*, 7:17–41.

Müller, B. and Reinhardt, J. (1990). *Neural Networks: An Introduction.* Springer-Verlag, Germany.

Müller, H. (1985). Kernel estimators of zeros and location and size of extrema of regression functions. *Scandinavian Journal of Statistics*, 12:221–232.

Müller, H. (1989). Adaptive nonparametric peak estimation. *The Annals of Statistics*, 17:1053–1069.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks.* Number 118 in Lecture Notes in Statistics. Springer, New York.

Neddermeijer, H., van Oortmarssen, G., Piersma, N., Dekker, R., and Habbema, J. (2000). Adaptive extensions of the nelder and mead simplex method for optimization of stochastic simulation models. Econometric Institute Report EI2000-22/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Nelder, J. and Mead, R. (1965). A simplex method for function minimization. *Computer J.*, 7:308–313.

Nocetal, J. (1992). Theory of algorithms for unconstrained optimization. In Iserles, A., editor, *Acta Numerica*, pages 199–242. Cambridge University Press, Cambridge.

Nowlan, S. and Hinton, G. (1992). Simplifying neural networks by soft weight–sharing. *Neur. Comp.*, 4:473–493.

Ortega, J. and Rheinboldt, W. (1970). *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York.

Pflug, G. (1996). *Optimization of Stochastic Models: The Interface Between Simulation and Optimization.* Kluwer Academic, Dordrecht, The Netherlands.

Plambeck, E., Fu, B.-R., Robinson, S., and Suri, R. (1996). Sample path optimization of convex stochastic performance functions. *Mathematical Programming*, 75:137–176.

Polak, E. (1971). *Computational Methods in Optimization.* Academic Press, New York.

Poljak, B. and Tsypkin, Y. (1973). Pseudogradient adaptation and training algorithms. *Automat. Remote Control*, 12:83–94.

Powell, M. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 17:155–162.

Powell, M. (1988). A review of algorithms for nonlinear equations and unconstrained optimization. In McKenna, J. and Temam, R., editors, *ICIAM 1987 Proceedings*, pages 220–232, Philadelphia. SIAM.

Powell, M. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis*, volume 275, pages 51–67, Dordrecht, The Netherlands. Kluwer Academic Publishers.

Powell, M. (April 1992). A direct search optimization method that models the objective and constraint functions by linear interpolation. Technical Report DAMTP 1992/NA5, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England.

Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. (1992). *Numerical Recipes, The Art of Scientific Computing.* Cambridge Univ. Press, New York, 2nd edition.

Rao, S., editor (1984). *Optimization, Theory and Applications.* Wiley Eastern Limited, New Delhi.

Raudys, S. and Jain, A. (1991). Small sample size problems in designing artificial neural networks. In Sethi, I. and Jain, A., editors, *Artificial Neural Networks and Statistical Pattern Recognition*, pages 33–50. North–Holland, Amsterdam.

Raydan, M. (1993). On the Barzilai and Borwein choise of steplength for the gradient method. *IMA J. Numerical Analysis*, 13:321–326.

Raydan, M. (1997). The Barzilai and Borwein gradient method for the large scale unconstraint minimization problem. *SIAM J. Optim.*, 7:26–33.

Reed, R. and Marks, R. (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* MIT Press, Cambridge, Massachusetts.

Refenes, A. (1991). Constructive learning and its application to currency exchange rate forecasting. In Turban, E. and Trippi, R., editors, *Neural Network Applications in Investment & Finance Services*, chapter 27. Probus Publishing, USA.

Refenes, A., Azema-Barac, M., Chen, L., and Karoussos, S. (1993). Currency exchange rate prediction and neural network design strategies. *Neural Computing & Applications*, 1:46–58.

Refenes, A. and Zaidi, A. (1992). Managing exchange rate prediction strategies with neural networks. In Lisboa, P. and Taylor, M., editors, *Neural Networks: Techniques & Applications*, chapter 24. Ellis Horwood, New York.

Refenes, A.-P. (1995). *Neural Networks in the Capital Markets.* John Wiley & Sons, Chichester.

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16:265–278.

Ripley, B. (1993). Statistical aspects of neural networks. In Brandorff-Nielsen, O., Jensen, J., and Kendall, W., editors, *Networks and Chaos - Statistical and Probabilistic Aspects*, volume 50 of *Monographs on Statistics and Applied Probability*, chapter 2, pages 40–123. Chapman and Hall, London.

Ripley, B. (1994). Neural networks and related methods for classification (with discussion). *Journal of Royal Statistical Society B*, 56:409–456.

Ripley, B. and Hjort, N. (1984). *Pattern Recognition and Neural Networks - A Statistical Approach.* Cambridge University Press, Cambridge.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statistics*, 22:400–407.

Robitaille, B., Marcos, B., Veillette, M., and Payre, G. (1996). Modified quasi-newton methods for training neural networks. *Computers and Chemical Engineering*, 20(9):1133–1140.

Rubinstein, R. and Shapiro, A. (1993). *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method.* Wiley, New York.

Rumelhardt, D., Hinton, G., and Williams, R. (1986). Learning representations by back–propagating errors. *Nature*, 323:533–536.

Ruppert, D., Reisch, R., Deriso, R., and Carroll, R. (1984). Optimization using stochastic approximation and Monte Carlo simulation (with application to the harvesting of atlantic menhaden). *Biometrics*, 40:535–545.

Safizadeh, M. (1990). Optimization in simulation: Current issues and the future outlook. *Naval Res. Logistics*, 37:807–825.

Schalkoff, R. J. (1992). *Pattern Recognition: Statistical, Structural and Neural Approaches.* Wiley, New York.

Schiffman, W., Joost, M., and Werner, R. (1994). Optimization of the back-propagation algorithm for training multilayer perceptrons. Technical report, Institute of Physics, University of Koblenz, Koblenz.

Schiffmann, W., Joost, M., and Werner, R. (1993). Comparison of optimized back-propagation algorithms. In *Proc. of the European Symposium on Artificial Neural Networks, ESSAN'93*, pages 97–104, Brussels.

Schoen, F. (1991). Stochastic techniques for global optimization: A survey of recent advances. *Journal Of Global Optimization*, 1:207–228.

Schöneburg, E. (1990). Stock price prediction using neural networks: A project report. *Neurocomputing*, 2:17–27.

Sharkey, A., editor (1999). *Multi-Net Systems.* Springer-Verlag, Berlin.

Silva, F. and Almeida, L. (1990). Accerelation techniques for the backpropagation algorithm. In Almeida, L. and Wellekens, C., editors, *Neural Networks, Proceedings EURASIP Workshop*, volume 412 of *Lecture Notes in Computer Science*, pages 110–119. Springer–Verlag, New York.

Sirlantzis, K. (1996). Interval Forecasting using Artificial Neural Networks with Application to Financial Time Series. Master's thesis, Institute of Mathematics and Statistics, University of Kent at Canterbury, Canterbury.

Sirlantzis, K., Fairhurst, M., and Hoque, M. S. (2001a). Genetic algorithms for multiple classifier system configuration: A case study in character recognition. In (Kittler and Roli, 2001), pages 99–108.

Sirlantzis, K., Feng, J., and Liu, W. (2001b). Minimization Algorithms based on Supervisor and Searcher Co-operation: III - novel algorithms for stochastic optimization. (submitted to JOTA).

Sirlantzis, K. and Liu, W. (2001). The Supervisor-Searcher Co-operation framework: A class of powerful algorithmns for stochastic optimisation. In *Euro 2001-European Operational Research Conference*, page 218, Rotterdam, The Netherlands. Association of European Operational Research Societies.

Smith, M. (1993). *Neural Networks for Statistical Modeling.* Van Nostrand Reinhold, New York.

Solla, S., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640.

Spall, J. (1992). Multivariate stochastic approximation using simultaneous perturbation gradient approximation. *IEEE Trans. Automat. Control*, AC-37:331–341.

Spall, J. (1998). Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823.

Spall, J. (1999). Stochastic optimization and the simultaneous perturbation method. In *Proceedings of the 1999 Winter Simulation Conference*, pages 101–109.

Sperduti, A. and Starita, A. (1993). Speed up learning and network optimization with extended back propagation. *Neural Networks*, 6:365–383.

Sprecher, D. (1965). On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355.

Swann, W. (1972). Direct search methods. In Murray, W., editor, *Numerical Methods for Unconstrained Optimization.* Academic Press, London.

Tang, Q., L'Ecuyer, P., and Chen, H. (1999). Asymptotic efficiency of perturbation-analysis based stochastic approximation with averaging. *SIAM J. Control Optim.*, 37:1822–1847.

Tang, Z., Almeide, C., and Fishwick, P. (1991). Time series forecasting using neural networks vs Box–Jenkins methodology. *Simulation*, 57(5):303–310.

Thodberg, H. (1996). A review of bayesian neural networks with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks*, 7:56–72.

Thornton, C. (1996). Parity: The problem that won't go away. In McCalla, G., editor, *Proceedings of AI-96, Toronto, Canada*, pages 362–374, Berlin. Springer-Verlag.

Tollenaere, T. (1990). SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3(5):561–573.

Tong, H. (1983). *Threshold Models in Non-Linear Time Series Analysis*, volume 21 of *Lecture Notes in Statistics*. Springer-Verlag, New York.

Tong, H. and Lim, K. (1980). Threshold autoregression, limit cycles and cyclical data. *Journal of Royal Statistical Society B*, 42:245–292.

Uryase'ev, S. (1992). A stochastic quasigradient algorithm with variable metric. *Ann. Oper. Resear.*, 39:251–267.

Utans, J. and Moody, J. (1991). Selecting neural network architectures via the prediction risk: Application to corporate bond rating prediction. *First International Conference on Artificial Intelligence: Applications on Wall Street*, pages 35–41.

Veitch, A. and Holmes, G. (1991). A modified quickprop algorithm. *Neural Computation*, 3:310–311.

Vogl, T., Mangis, J., Rigler, A., Zink, W., and Alkon, D. (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263.

Vrahatis, M., Androulakis, G., and Manoussakis, G. (1996). A new unconstrained optimization method for imprecise function and gradient values. *Journal of Mathematical Analysis and Applications*, 197:586–607.

Vrahatis, M., Androulakis, J., Lambrinos, J., and Magoulas, G. (2000a). A class of gradient unconstrained minimization algorithms with adaptive stepsize. *Journal of Computational and Applied Mathematics*, 114:367–386.

Vrahatis, M., Magoulas, G., and Plagianakos, V. (2000b). Globally convergent modification of the quickprop method. *Neural Processing Letters*, 12:159–169.

Weigend, A., Rumelhardt, D., and Huberman, B. (1990). Predicting the future: a connectionist approach. *International Journal of Neural Systems*, 1(3):193–209.

Weigend, A., Rumelhardt, D., and Huberman, B. (1991). Generalization by weight–elimination applied to currency exchange rate prediction. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 837–841.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA.

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, volume 78, pages 1550–1560, Los Alamos, CA.

White, H. (1988). Economic prediction using neural networks: The case of IBM daily stock returns. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 451–458, San Diego, CA.

White, H. (1989). Learning in artificial neural networks: a statistical perspective. *Neural Computation*, 1:425–464.

White, H. (1990). Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3:535–549.

White, H. (1992). *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Oxford.

Wolfowitz, J. (1952). On the stochastic approximation method of Robbins and Monro. *Ann. Math. Statistics*, 23:457–461.

Wolfowitz, J. (1958). Stochastic approximation. *Ann. Math. Statistics*, 27:1151–1156.

Wolpert, D. and Macready, W. (1996). No free lunch theorems for optimization. Technical Report N5Na/D3, IBM Almaden Research Center.

Yan, D. and Mukai, H. (1993). Optimization algorithms with probabilistic estimation. *J. Optim. Theory and Applic.*, 79:345–371.

Yao, Q. and Tong, H. (1994). Quantifying the influence of initial values on nonlinear prediction. *Journal of the Royal Statistical Society*, B 56(4):701–725.