Analogue Imprecision in MLPs - Implications and Learning Improvements

Peter J. Edwards

Thesis submitted for the degree of
Doctor of Philosophy
University of Edinburgh
1994
Abstract

Analogue hardware implementations of Multi-Layer Perceptrons (MLP) have a limited precision that has a detrimental affect on the result of synaptic multiplication. At the same time however the accuracy of the circuits can be very high with good design. This thesis investigates the consequences of the imprecision on the performance of the MLP, examining whether it is accuracy or precision that is of importance in neural computation.

The results of this thesis demonstrate that far from having a detrimental effect, the imprecision or synaptic weight noise enhances the performance of the solution. In particular the fault tolerance and generalisation ability are improved. In addition, under certain conditions, the learning trajectory of the training network is also improved. Through a mathematical analysis and subsequent verification experiments the enhancements are reported. Simulation experiments examine the underlying mechanisms and probe the limitations of the technique as an enhancement scheme.

For a variety of problems, precision is shown to be significantly less important than accuracy. In fact imprecision can have beneficial effects on learning performance.
I declare that this thesis has been completed by myself and that, except where indicated to the contrary, the research documented is entirely my own.

Peter J. Edwards
Acknowledgements

During the course of the time I have spent in Edinburgh I have had the support and help of many people. I am now glad to have the opportunity to thank them all, although there are far too many to name individually. Firstly I am indebted to Alan Murray under whose supervision I have carried out all the work contained here, and those who spent so long in proof-reading this thesis. Also I would like to thank all those who have "lived" (some more permanently than others) in Alrick 2.2 who have contributed to the friendly atmosphere that our group has. The people with whom I have shared this three year period outside work also deserve my thanks for their continued distraction and support. You can work out for yourselves in which category you lie! Finally thanks also to my family who have put up with me for far longer than anyone else.
# Table of Contents

## 1. Introduction

1.1 Aims and Objectives ........................................... 3
1.2 Multi-Layer Perceptrons ...................................... 4
  1.2.1 Introduction ........................................... 4
  1.2.2 Training Considerations ................................. 6
  1.2.3 Summary ................................................ 7
1.3 Stochastic Systems ............................................ 7
  1.3.1 Introduction ........................................... 7
  1.3.2 Noise in Neural Implementations .......................... 10
  1.3.3 Imprecision vs. Inaccuracy ............................... 11
  1.3.4 Digital Inaccuracy ..................................... 12
  1.3.5 Analogue Imprecision ................................... 14
  1.3.6 Summary ................................................ 15
1.4 Thesis Outline ................................................ 16
1.5 Chapter Summary ............................................. 18

## 2. Neural Network Performance Metrics .......................... 19

2.1 Introduction ................................................ 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Fault Tolerance</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Introduction</td>
<td>20</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Issues in Fault Tolerance</td>
<td>21</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Achieving Operational Fault Tolerance</td>
<td>23</td>
</tr>
<tr>
<td>2.2.4</td>
<td>“In-Training” Mechanisms</td>
<td>24</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Post-Training Strategies</td>
<td>25</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Re-training</td>
<td>26</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Summary</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>Generalisation</td>
<td>28</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Introduction</td>
<td>28</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Pre-training parameter adjustment</td>
<td>30</td>
</tr>
<tr>
<td>2.3.3</td>
<td>“In-training” complexity constraints</td>
<td>32</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Post-training destructive algorithms</td>
<td>34</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Summary</td>
<td>36</td>
</tr>
<tr>
<td>2.4</td>
<td>Learning Trajectory and Speed</td>
<td>37</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Introduction</td>
<td>37</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Improvement methods</td>
<td>39</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Local Adaptation Rates</td>
<td>40</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Second Order Methods</td>
<td>41</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Stochastic Learning</td>
<td>42</td>
</tr>
<tr>
<td>2.4.6</td>
<td>Summary</td>
<td>43</td>
</tr>
<tr>
<td>2.5</td>
<td>Chapter Summary</td>
<td>44</td>
</tr>
</tbody>
</table>
# Table of Contents

3. Noise in Neural Implementations .............................................. 45

3.1 Introduction ........................................................................... 45

3.2 Implementation Errors ......................................................... 46

3.2.1 Introduction ........................................................................ 46

3.2.2 In-The-Loop Implementations ........................................... 47

3.2.3 Learning "On-chip" ........................................................... 52

3.2.4 Summary ............................................................................ 54

3.3 An Implementation Error Model ............................................ 55

3.3.1 Introduction ........................................................................ 55

3.3.2 The Model .......................................................................... 57

3.3.3 Summary ............................................................................ 58

3.4 The Mathematical Model ...................................................... 59

3.4.1 Introduction ........................................................................ 59

3.4.2 A Cost Function With Injected Noise ................................. 60

3.4.3 Predictions .......................................................................... 62

3.4.4 Summary ............................................................................ 73

3.5 Chapter Summary .................................................................... 73

4. Simulation Requirements and Environment ............................ 75

4.1 Introduction ........................................................................... 75

4.2 Simulation Requirements ...................................................... 77

4.2.1 Introduction ........................................................................ 77

4.2.2 Problem Database ............................................................. 81

4.2.3 Summary ............................................................................ 86
# Table of Contents

4.3 Simulation Environment ........................................ 87  
4.3.1 Introduction ............................................. 87  
4.3.2 Noise Generation ............................................ 88  
4.3.3 Learning Algorithm ......................................... 89  
4.3.4 Stopping Criteria ........................................... 91  
4.3.5 Error Measures ................................................ 93  
4.3.6 Testing Environment ......................................... 94  
4.3.7 Summary ..................................................... 94  
4.4 Chapter Summary ................................................ 94  

5. Fault Tolerance – Performance Implications ............................ 96  
5.1 Introduction ..................................................... 96  
5.2 Test Environment ................................................ 97  
5.2.1 Introduction .................................................. 97  
5.2.2 The Fault Model .............................................. 98  
5.2.3 Random Damage ............................................. 98  
5.2.4 Weight Agitation ............................................. 99  
5.2.5 Analysis of the Weight Distribution ......................... 100  
5.2.6 Summary ..................................................... 100  
5.3 Results .......................................................... 100  
5.3.1 Introduction .................................................. 101  
5.3.2 Verification of the Basic Prediction ......................... 101  
5.3.3 Enhancement Mechanisms .................................... 102  
5.3.4 The Limitations of the Improvement ......................... 106
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.3 Analysis of the Enhancement Mechanisms</td>
<td>138</td>
</tr>
<tr>
<td>7.2.4 Assessing the Limitations of the Technique</td>
<td>139</td>
</tr>
<tr>
<td>7.2.5 Summary</td>
<td>140</td>
</tr>
<tr>
<td>7.3 Results</td>
<td>141</td>
</tr>
<tr>
<td>7.3.1 Introduction</td>
<td>141</td>
</tr>
<tr>
<td>7.3.2 Learning Speed</td>
<td>142</td>
</tr>
<tr>
<td>7.3.3 Enhancement Mechanisms</td>
<td>145</td>
</tr>
<tr>
<td>7.3.4 The Limitations of the Technique</td>
<td>146</td>
</tr>
<tr>
<td>7.3.5 Additive Noise</td>
<td>149</td>
</tr>
<tr>
<td>7.3.6 Summary</td>
<td>152</td>
</tr>
<tr>
<td>7.4 Chapter Summary</td>
<td>152</td>
</tr>
<tr>
<td>8. Conclusions</td>
<td>154</td>
</tr>
<tr>
<td>8.1 Introduction</td>
<td>154</td>
</tr>
<tr>
<td>8.2 Implications for Analogue Hardware</td>
<td>155</td>
</tr>
<tr>
<td>8.2.1 Introduction</td>
<td>155</td>
</tr>
<tr>
<td>8.2.2 An Assessment of the Hardware Model</td>
<td>156</td>
</tr>
<tr>
<td>8.2.3 Precision vs. Accuracy</td>
<td>158</td>
</tr>
<tr>
<td>8.2.4 Summary</td>
<td>159</td>
</tr>
<tr>
<td>8.3 Synaptic Noise as an Enhancement Scheme</td>
<td>159</td>
</tr>
<tr>
<td>8.3.1 Introduction</td>
<td>159</td>
</tr>
<tr>
<td>8.3.2 A Summary of the Software Implementation Enhancements</td>
<td>160</td>
</tr>
<tr>
<td>8.3.3 A Hardware Implementation</td>
<td>163</td>
</tr>
<tr>
<td>8.3.4 Summary</td>
<td>165</td>
</tr>
<tr>
<td>8.4 General Conclusions</td>
<td>165</td>
</tr>
</tbody>
</table>
List of Figures

1-1 The multi-layered perceptron labelled with the mathematical notation used throughout this thesis. ........................................ 5
1-2 Idealised view of a single neuron. ........................................ 9
1-3 Signal \( f(x) \) is approximated by two other signals, \( f_1(x) \) and \( f_2(x) \). ..................................................... 12
3-1 Conceptualised probability density function of weight values and levels of allowed quantisation voltage values. .......................... 49
3-2 Model incorporating noise on the weights in the synaptic multiplication. ................................................................. 58
3-3 Details of the mathematical derivation between (3.9) and (3.11). . 61
3-4 Details of the mathematical derivation between (3.11) and the two extra cost function terms, (3.12) and (3.13). ..................... 62
3-5 The derivation of the network output with respect to the hidden-output weights. ............................................................. 65
3-6 The ubiquitous sigmoid squashing function and its derivative. .......... 65
3-7 The derivation of the network output with respect to the input-hidden weights. ............................................................... 66
3-8 Conceptualised exemplar output-weight dependence graph, with first and second partial derivatives. ................................ 69
3-9 Details of the mathematical derivation of the noise mediated perceptron update equation. ................................................. 71
List of Figures

4-1 The character encoder task. ........................................ 81
4-2 An idealised view of the eye/not-eye classifier problem. .......... 82
4-3 An idealised view of the L-shaped room in the localisation problem. 84
4-4 A locus plot of x(t) against y(t) for the Henon map problem. .... 86
4-5 Graph showing a comparison between virtual targets and back-
propagation on the character encoder problem. ........................ 90
4-6 An idealised view of a validation curve and the windowing system
used for stopping learning in the localisation problem. ............. 93

5-1 Graph showing the reduction in classification error with increasing
levels of multiplicative synaptic weight noise, injected during
training, at different levels of random damage. i.e. fault tolerance
performance enhancement on the character encoder problem. .... 102
5-2 Graph showing the variation in standard deviation of the error
caused by the agitation of individual weights across the network,
for networks trained on the character encoder problem. ............. 104
5-3 Final weight value distributions following training with multiplica-
tive synaptic noise for the character encoder problem. ............. 105
5-4 Graph showing the variation in standard deviation of the error
cau sed by weight agitation across the network, for networks trained
on the localisation problem. ........................................... 107
5-5 Graph showing the classification error with increasing levels of ad-
ditive synaptic weight noise injected during training at different
levels of random damage. These results were obtained using a set
of weights trained from an initial weight range of 0.1. .............. 108
5-6 Graph showing the classification error with increasing levels of ad-
ditive synaptic weight noise injected during training at different
levels of random damage. These results were obtained using a set
of weights trained from an initial weight range of 0.01. ............. 110
5-7 Final weight value distributions following training with additive synaptic noise for the character encoder problem. The initial weight range was 0.1.................. 112

6-1 Graph showing the sigmoid curve with varying “temperature”, $\theta$. 119

6-2 Graph showing the reduction in classification error due to “input corruption” with increasing levels of multiplicative synaptic weight noise, injected during training for the character encoder problem. 122

6-3 Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected noise during training for the character encoder problem. .................. 125

6-4 Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected noise during training for the eye/not-eye classifier. .................. 125

6-5 Graph showing the reduction in classification error $\epsilon_i$ as the synaptic noise level is increased during training. The temperature parameter is varied in the sigmoid function for networks trained on the character encoder. .................. 126

6-6 Graph showing the change in the training and generalisation classification error at increasing levels of synaptic noise, for the localisation problem. .................. 128

6-7 Graph showing the average value of the hidden layer derivative against noise for the localisation problem. .................. 128

6-8 Graph showing the change in the training and generalisation classification error at increasing numbers of hidden layer units for the localisation problem. .................. 130

6-9 Graph showing the reduction in classification error due to “input corruption” with increasing levels of additive synaptic weight noise, injected during training for the character encoder problem. .. 132
6-10 Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected additive noise during training for the character encoder problem. .................................. 133

7-1 Graph showing a typical pair of learning curves for the character encoder problem trained with the virtual targets algorithm. .................................. 143

7-2 Probability density function for learning times (in terms of epochs) for networks with different random starting points for the character encoder problem at 10% multiplicative noise. .................................. 143

7-3 Graph showing the variation in mean learning time verses the multiplicative synaptic noise level used during the training phase for the character encoder problem. .................................. 144

7-4 The second derivative \( x \) error term trajectory for injected synaptic noise levels 0% and 7% .................................. 145

7-5 Graph showing the mean square error learning curves for three levels of synaptic noise, 0%, 2% and 4% for the Henon map problem. .................................. 148

7-6 Graph showing the average mean square error for networks trained on the Henon map problem for 10000 epochs using increasing levels of synaptic noise. .................................. 149

7-7 Graph showing the trajectory of the average value of the hidden layer output derivative for the three learning trajectories plotted above. .................................. 150

7-8 Graph showing the variation in mean learning time verses the additive synaptic noise level used during the learning phase for the character encoder problem. .................................. 151

B-1 A noise distribution with a mean of \( \lambda \). .................................. 186

B-2 Graph showing the variation in fault tolerance performance for varying \( \lambda \) with multiplicative noise. .................................. 190
B-3 Graph showing the variation in fault tolerance performance for varying $\lambda$ with additive noise. . . . . . . . . . . . . . . . . . . . . . 190

B-4 Graph showing the variation in tolerance to input corruption (or generalisation ability) for varying $\lambda$ with multiplicative noise. . 191

B-5 Graph showing the variation in tolerance to input corruption (or generalisation ability) for varying $\lambda$ with additive noise. . . . 191

B-6 Graph showing the variation in mean learning time for varying $\lambda$ with multiplicative noise. . . . . . . . . . . . . . . . . . . . . . 193

B-7 Graph showing the variation in mean learning time for varying $\lambda$ with additive noise. . . . . . . . . . . . . . . . . . . . . . 193
Chapter 1

Introduction

She asked me many questions and propounded many problems - the most of which were idle tales... Some I answered, and some I said were foolish. Many wear the Robe, but few keep the Way.

Rudyard Kipling, *Kim*

To believe the claims of the popular press the uninitiated might be convinced that the “new” concept of artificial neural networks, “modelling the human brain”, could solve a significant number of the so far unsolved problems presented to the modern day technologist. Any truth behind this rhetoric certainly cannot be found in the majority of popular science articles and indeed cannot be assumed at all. However, the current literature of the neural network field indicates that the abstract notion of *parallel distributed processing* with *neuron-like elements* has been, and is increasingly being applied successfully to real world problems. Although the question as to whether many of these applications could have been solved by the well established and proven methods of signal processing, control theory and image processing still remains unanswered, the success does add credibility to the techniques. Therefore, theory is emerging and applications are being presented that are leading engineers to the task of implementing the neural techniques in hardware - a process perhaps motivated by the hidden agendas of marketing strategists, but one which implies “usefulness” in the concepts. Leaving more philosophical discussion aside, this non-trivial implementation task
is bringing to light more issues that need to be addressed and, because of this, studies as to the effects of the implementation processes on the algorithms are being carried out. This thesis presents one such study.

For a notional concept to progress from being an academic study to becoming a practical tool for industrial applications can take many years of developmental work. Much of this time can be taken up in progressing from the concept level to one of implementation, whether in hardware or software. In the neural network field, the current literature contains many such software studies, exploring the issues of implementing neural network concepts. These algorithmic implementations have reached a stage where they can be used to solve real world problems (see for example [79][60][7]) and so it is appropriate for the more complicated step to a hardware implementation to take place. In fact, hardware implementations have already been produced and used with varying degrees of success (see for example [36][69]). Also the question as to whether hardware is necessary has been discussed in depth, and the pros and cons raised [51][20]. Here it is assumed that hardware implementations are necessary, or at least could be beneficial to some problems. Therefore, studies on the effects of the implementation must be carried out to consider the implications on the algorithms. Motivated by these issues, this thesis presents a study of the effects of analogue imprecision on such an implementation.

From the superfluity of existing neural architectures and algorithms this study considers the most commonly used - the Multi-Layer Perceptron (MLP) trained with an error back-propagation technique. By constructing a model of analogue hardware and incorporating this into a software implementation, an investigation of the implications of the hardware model is carried out. In fact far from having the detrimental effects that were expected, the results show that the analogue imprecision (or synaptic weight noise) introduces performance enhancements into the learning and the final solution. Therefore this thesis considers analogue imprecision in MLPs and explores its associated implications and learning improvements.
Having thus presented a brief background to the ideas presented in this thesis, the following sections consider in more depth the aims and objectives of the work and the details of the concepts considered.

1.1 Aims and Objectives

The previous section presented a brief discussion on the motivation for carrying out a project of this kind. In effect, it explained the need for a clear understanding of the implications of implementing neural algorithms in hardware on the algorithms themselves. This section addresses the aims and objectives of the project and how these developed over its course.

Essentially at its outset the grandiose aim of the project was to try to smooth the transition from using software implementations of neural algorithms to building custom hardware ones. In the digital hardware case this change can be very subtle as what is software after all but a means of implementing some algorithm on a general purpose hardware processor? However, for useful and realistic custom implementations to be designed, compromises from full floating-point accuracy must be made. In the analogue hardware case that is of interest here, the transition is much more significant and appears hazardous. It is because of these apparent dangers that studies such as this are necessary. This project is restricted to only a small, but important area of this subject by considering analogue imprecision and its effect on the neural algorithm's performance. In addition, by showing that the implications of incorporating the hardware model into a neural algorithm were not detrimental to the algorithmic performance, the work's aim developed to become an exploration of the issues of exploiting the improvements seen as an enhancement scheme.

To achieve these aims a few intermediary goals were set allowing the work to progress in stages. These goals were as follows :-
• Construct a general model of analogue hardware from the plethora of implementation issues.

• Carry out a mathematical analysis to investigate the effects.

• Implement the neural algorithms with the model included.

• Assess the implications of the model through simulation experiments in the areas of fault tolerance, generalisation ability and learning trajectory and speed.

• Investigate the underlying mechanisms and the limitations of the technique as an enhancement scheme.

• Identify the implications for analogue hardware designers and for the technique as an enhancement scheme.

Having presented the aims and objectives of the project and also having noted the developments that came out of them, a more comprehensive discussion of the subject area can now be embarked upon. The following section considers the multi-layer perceptron and mechanisms by which it can be trained.

1.2 Multi-Layer Perceptrons

This section presents a brief introduction to the Multi-Layer Perceptron (MLP). Descriptions of the full history and analysis of the MLP have been produced many times before and so this introduction will contain only a summary. For a more complete introduction see [61][5][27].

1.2.1 Introduction

The MLP is made up of elements, or perceptrons, that are based on the McCulloch and Pitts model of a simple human neuron [47]. When connected in
Chapter 1. Introduction

Outputs: \( o_j = f \left( \sum o_i \omega_{ji} \right) \)

Inputs

\( o_1 \)
\( o_2 \)
\( \vdots \)
\( o_I \)

Weights

\( \omega_{ji} \)
\( \omega_{kj} \)

Targets

\( t_1 \)
\( \vdots \)
\( t_K \)

Layer: 1 J K

Figure 1-1: The multi-layered perceptron labelled with the mathematical notation used throughout this thesis.

a network with more than one layer the subsequent structure is called a MLP and forms a powerful means of performing non-linear mapping functions. Fig. 1-1 depicts a multi-layered perceptron with a mathematical description using the notation used throughout this thesis. In 1969 Minsky and Papert reported that single layered networks of perceptrons were only capable of solving linearly separable problems, significantly limiting their usefulness. However, when connected in multiple layers\(^1\) the requirement of linear separability is overcome [45]. The perceptron units are made up of multiple inputs linked via weighted connections (synapses or weights) to a summing and non-linear squashing function (see Fig. 1-1). The result of this squashing function (usually the sigmoid function) is then passed to the next layer as an input. Using a network of weighted connections

\(^1\)In fact only two are necessary to solve any problem provided there are sufficient numbers of units [33].
in this way, non-linear mappings can be made between pairs of input and output vectors.

Using this ability trained MLPs (MLPs with the correct weight configurations) can be used to solve many problems in the fields of pattern recognition, system identification and time series prediction. The strength of the MLP, indeed why it can be used in such a variety of applications, is that the underlying rules of the function do not need to be known prior to the training of the network. During the training process, assuming a representative training data set, the network extracts the mapping function in its internal representation, thus obtaining a solution. It is important to note that in some applications it is necessary to know how a problem has been solved and the black box nature of the MLP can be a problem rather than a strength. In defence, it could be said that some understanding of the solution can be gained by examining the internal representation, however, a completely satisfactory method of doing this is yet to be established. Despite this problem, the MLP remains a powerful tool for solving a wide range of application problems.

1.2.2 Training Considerations

The process of producing a correct weight configuration for this MLP structure was a task that slowed the field down significantly after Minsky and Papert published their book in 1969 [45]. Single-layered perceptrons could be trained using the perceptron delta rule [76], based on the hebbian learning technique [25], while there was no corresponding technique for the MLP. Finally however, Rumelhart et al published their generalised delta rule or error back-propagation (BP) method that allowed MLPs to be trained, assuming a continuously differentiable squashing function is used. This gradient descent technique revolutionised the field and along with many variants is still widely used as a training method. The basic scheme involves iteratively presenting training example patterns, calculating an error from a set of target and actual network output comparisons, and making weight adjustments using the gradient of the error with that particular
weight. This process is repeated until an acceptable solution is reached and the
problem is solved. For the exact details of the algorithm see [62].

1.2.3 Summary

This section has given a very concise description of the MLP, giving a brief history,
explaining how it is used and showing how the internal configuration is obtained.
This particular type of artificial neural network is used throughout the course
of this project, although the training scheme is varied at some stages. These
variations are detailed at the appropriate points. The following section describes
stochastic systems with a view to understanding how a hardware model can be
considered and compared to studies of other non-deterministic learning methods.

1.3 Stochastic Systems

This section introduces the vast field of stochastic systems. In particular it looks
at "noise" (defined here as random numerical imprecision) in neural networks,
including biological networks and electronic hardware implementations. Also it
contains a brief investigation of purely stochastic search algorithms as a compar-
ison with neural methods. The section gives a background to the topic of noise
in neural computation, showing how noise occurs naturally and how it is mod-
elled. The comparison is then drawn with hardware implementations discussing
the implications for algorithmic performance and how noise is combated and used
in neural applications.

1.3.1 Introduction

A stochastic system is one that is non-deterministic in nature. In other words,
at any stage random events can occur which imply that the future state of the
system is unpredictable. In fact, implementations of many algorithms will necessarily include stochastic effects due to the quantisation of the pure continuous mathematical case. Depending on the degree of discretisation and the problem, the solution, at any one time, may deviate from the predicted result. Consider, for example, a gradient descent algorithm. In the continuous mathematical case it is predicted that the algorithm will travel down a gradient until a minimum is reached and then will stop. However, when implemented, the continuous becomes discrete and fixed steps are taken down the gradient. The algorithm is only truly monotonic if the steps are infinitesimally small, as otherwise the fixed step can "overshoot" the minimum. This unpredictability, or non-determinism due to a stochastic element, is often put to good use in optimisation algorithms, where a set of fixed rules (heuristics) will be unable to solve a particular problem. Kirkpatrick et al, [41] in their classic simulated annealing paper state,

Heuristics are rather problem-specific : there is no guarantee that a heuristic procedure for finding near-optimal solutions for one NP-complete problem will be effective for another.

Simulated annealing (SA) is a stochastic search algorithm in which controlled departures from gradient descent are allowed. These departures are controlled by a global "temperature" term which is reduced as learning progresses. The probability of increases in the overall error are reduced by "cooling" as the algorithm adapts to a solution. Stochasticity is therefore often very useful and algorithms such as SA are widely used (see [34] and included references), although prohibitively long learning times can arise.

Biological systems also show evidence of stochasticity although standard artificial neural network models do not acknowledge this. The original neural model on which much of the current artificial neural network research is based, was devised by McCulloch and Pitts [47]. Their model is essentially a logic element, far removed from the complex structures that are reported in brain physiology studies [67]. The simplicity of the model has, however, proved useful. The McCulloch and Pitts neuron responds to active inputs that are modified by synapses of
Chapter 1. Introduction

which there are two kinds, excitatory and inhibitory. Over some time period the neuron integrates these active signals and if the resulting “activation” is greater than some threshold then the neuron becomes active. Otherwise the binary output remains inactive. Fig 1–2 shows an idealised view of a basic neuron, where

![Idealised view of a single neuron](image)

**Figure 1–2:** Idealised view of a single neuron.

the dendrites are the connecting inputs, synapses excite or inhibit the incoming signal and the axon is the output from the cell body where thresholding takes place. Taylor[72] deemed this basic model to be inadequate and so produced his own with more features. In particular he introduced a stochastic element into the processing, thus producing spontaneous behaviour. This stochastic term was based on the spontaneous emission known to occur in the brain due to variations in the permittivity of the synaptic membranes in the synaptic cleft[14]. Taylor notes that if the number of dendrites connected to each cell body were to reach a certain level then the probability of spontaneous emission of the neuron would be high. The value he quoted was 10,000 dendrites, a value that is common in many mammals and one that is often surpassed. The value was estimated using a set of parameters for the synaptic emission, all of which were estimates. Nevertheless the result suggests that there is a high level of synaptic noise in the brain. Taylor
states, rather vaguely, that this spontaneity could give the networks a greater adaptive power, widening the search area in the initial stages of training.

Noise has been used in neural networks to produce useful enhancements to learning. Sequin and Clay [65] randomly destroy units to improve fault tolerance, while Hanson[23] has introduced noise into the weights. He uses an annealing process in his stochastic version of the delta rule, helping the network to avoid local minima and thus finding global solutions more rapidly. This work will be referred to and discussed in more detail in chapter 2. Here, however, it is apparent that stochastic elements occur in biological neural networks and also that stochasticity has a use in artificial neural systems and other search algorithms. Whether the types of noise used during the term of this project, and those seen in other research work, map at all to the effects seen in the brain, is highly debatable, uncomfortably uncertain and well outside the scope of this thesis.

The following sections consider the errors that are introduced in hardware implementations of neural networks. By examining the literature for work done in investigating the effects of digital inaccuracy and analogue imprecision, a background is given to the work carried out in this thesis.

1.3.2 Noise in Neural Implementations

As soon as an algorithm is implemented in hardware or software then errors are introduced into the computation, giving discrepancies between the mathematical expectations and those obtained in reality. In software implementations the discrepancies are small because of the high resolution of floating point numbers used in the calculation. In hardware implementations, however, the errors can become a significant factor in the algorithmic performance.

Implementations vary in the extent to which hardware is incorporated. At one end of the scale there is the software implementation where hardware is remote from the implementation details. At the other end of the scale is on-chip learning, where the hardware constraints are very much a reality. In between there are
many varieties of hardware/software hybrids. One common architecture is “in-the-loop” hardware, where the hardware is concerned with only the calculation of the forward pass of the MLP algorithm. The outputs from the hardware are then taken into a host processor, a PC for instance, where the learning takes place and new weights are “downloaded” to the hardware. The work in this thesis concentrates on this system.

Another implementation consideration is the choice between analogue and digital designs. The implications of this choice are not as straightforward as it would at first seem. The following sections look at the issues that arise, the required accuracy and precision and ways of combating the implications of implementation errors.

### 1.3.3 Imprecision vs. Inaccuracy

One of the confusing issues arising in assessing the viability of implementing neural algorithms, is the necessity of accurate and precise computation. Kirk in his thesis [40] argues the point well, stating that precision and accuracy are different things. He defines precision as “the degree of agreement of repeated measurements of a quantity” and accuracy as “the degree of conformity to some recognised standard value”. In Fig. 1-3 the signal $f(x)$ is approximated by $f_1(x)$ and $f_2(x)$. $f_1(x)$ is a precise approximation, although not accurate. $f_2(x)$ is an accurate approximation, but not precise. A precise quantity should always, therefore, be qualified by some degree of accuracy.

In digital computation precision is not a problem using Kirk’s definition. However, accuracy varies and can be enhanced by increasing the number of bits. Different problems will require varying degrees of accuracy and, therefore, different numbers of bits. In analogue computation the definitions are more complicated. In general circuits are designed within certain constraints to meet a specification. Therefore, depending on that specification, the design precision varies, but essentially is much lower than the binary nature of the digital case. This is due to variations in component characteristics, temperature or even poor
circuit design. More specifically for two "identical" circuits, made from a common design with the same components and having the same inputs, the outputs will not be precisely the same. Accuracy, on the other hand, is not limited by bit resolution as in the digital case, but the output of devices can apparently take on an "infinite" number of values within some range. In reality an extremely high accuracy can be produced with good analogue circuit design. The question therefore arises as to what is important for neural computation. The following two sections review the current literature to find an answer to this question, much of which is confused by incorrect terminology. This thesis seeks to eradicate this confusion.

1.3.4 Digital Inaccuracy

This section examines accuracy requirements in digital implementations of neural algorithms. Various researchers have investigated the implications of limited resolution (number of bits) in the forward pass through a network and the learning phase for algorithms such as BP. From the literature it can be seen that the required accuracy is problem specific.
Holt and Hwang [32] analyse errors introduced into the weights, inputs and the non-linear function due to rounding, truncation and the use of discrete variables. They note that 8 bits is sufficient for a forward pass, while at least 14-16 bits are required for BP, dependent on the problem. Classification problems such as the XOR problem require as few as 12-13 bits while other more demanding problems, such as some found in the field of time series prediction (TSP), require more bits. These results are generally corroborated by the literature [2][29].

A number of researchers have developed techniques for reducing the required number of bits. Hoehfield and Fahlmann [28] note that the cause for failure is that weight updates are quantised to zero below some threshold. Hence the authors, using the cascade correlation algorithm [18], implement a probabilistic rounding method where a weight change may occur in the weights even if the actual change is less than the smallest quantisation step. Vincent and Myers [74] use a similar technique they christen weight dithering. Hoehfield and Fahlmann also use a variable gain in the neurons so that larger weights can be used with lower bit resolution. This is also common in the literature [74][2].

Xie and Jabri [78] in their report take a different approach to the problem of digital accuracy. They employ a custom-designed algorithm to overcome the problem of small weight changes being quantised to zero. Their algorithm, a combination of a modified weight perturbation (WP) [35] algorithm and a random search method, is optimised for digital hardware implementation. The algorithm gives impressive results for low levels of weight or calculation accuracy.

Hollis et al [29], in their much cited paper, look at a hybrid architecture of analogue neurons, for the forward pass calculation, and a digital implementation of BP. They examine the problems of weight resolution in the learning, weight truncation at the input of the analogue device and restricted accuracy at the output. Their results confirm those obtained by other researchers in the field, showing that even with the low precision achieved by the analogue multipliers, learning is possible.

The work reviewed above demonstrates the accuracy requirements for digital
implementation of neural algorithms. Various techniques have been employed to reduce the needed bit resolution, although the improved results are all problem specific.

1.3.5 Analogue Imprecision

This section looks at the implications of an analogue implementation of neural algorithms. As was discussed earlier, the errors introduced by analogue implementations are qualitatively different from those in the digital case. Therefore, the techniques used to calculate bit resolution requirements are no longer valid. However, the accuracy of values passed to the analogue device is a limiting factor (see Hollis et al[29]). The papers reviewed in this section look at the precision limitations of analogue systems.

Frye et al[19], consider the statistical variation in components, noting that exact values do not exist and so presumed values have to be used in calculations. The authors state that in their experiments pre-calculation of the synaptic weights is ineffective for their TSP problem, performing no better than a random set. They carry out learning experiments on random weights in the presence of the variations (in-the-loop) and find that the weights are trained to compensate for the noise and the effects are negligible. Simulations are carried out using multiplicative noise (noise of value proportional to the weight magnitude) in the weights in the forward pass and using ideal floating point accuracy for the update calculation. The authors note no loss in performance at levels of up to 30% noise and conclude that even large non-uniformities do not cause problems as long as training is carried out on the hardware itself, in-the-loop.

Tarassenko and Tombs[71] also examine precision requirements for learning in their search for an algorithm to implement on-chip-learning. They select the WP algorithm and carry out experiments using an extremely hard classification
problem\textsuperscript{2} to test its suitability. The tests they use are specific to the WP algorithm and, by limiting the minimum update step, the weight range and the minimum perturbation, using values measured from a real analogue chip, they conclude that on-chip learning should be possible with the WP algorithm.

The work discussed in this section sets the scene for an investigation of the implications of analogue imprecision on the implementation of neural algorithms. The limited work available in the literature suggests that learning should be possible even for significant device variations and imperfections. However the body of work is indeed limited and the studies are far from conclusive, as in the digital case.

1.3.6 Summary

This section has introduced the topic of stochastic systems looking at a wide variety of cases where noise is evident. The occurrence of noise in biological neural networks is considered and neural models discussed. After this background to stochastic systems, the section concentrated on noise in hardware implementations of neural networks, investigating the implications of the noise and the accuracy and precision requirements, as seen by the current literature. The section also shows that noise is sometimes used to aid learning in neural algorithms.

The question of whether it is accuracy or precision that is important in artificial neural networks has still not been answered. From the previous discussion it has been seen that for digital implementations, accuracy requirements are the limiting factor. Specifically, the implementation needs to be able to adjust the weights by a sufficiently small step for the solution to be reached for a given problem. This requirement has been shown by the experimental analysis of a

\textsuperscript{2}This problem is the task of locating the nearest corner in a room from a robot making a $360^\circ$ scan. The problem is also used in the work of this project and is described in chapter 4.
number of researchers, reviewed in section 1.3.4. In the analogue case the precision requirements are unclear. It has been shown that even hard problems can be solved in the presence of noise with BP [19], although the reasons for this are still to be addressed. This question and its complex answer will be returned to in later chapters of this thesis.

1.4 Thesis Outline

Now that the subject area for the project has been introduced, this section gives a synopsis of the entire thesis. For a more detailed breakdown each chapter includes a summary which outlines the main themes of that chapter.

Chapter 1 is the introduction chapter, introducing the main themes of the thesis and the motivation behind the work. It starts with a brief discussion on artificial neural networks and defines how this project relates to the rest of the field. The following section moves on from the motivation of the project to examine the specific aims and objectives of the work. Having given a background to the work, a more specific introduction to some of the central themes of the project is presented. This introduction includes the MLP and the wide topic of stochastic systems.

Chapter 2 continues the literature review started in chapter 1 by considering the three MLP performance metrics - fault tolerance, generalisation ability and learning trajectory and speed. In each of these areas an explanation of the concepts is presented and the issues discussed. Then, by considering the current literature, methods of enhancing the network performance are reviewed.

From this wide background of information and ideas, chapter 3 specifically considers the issue of hardware implementations. The aim of this chapter is to produce a model of a hardware implementation which can be used to analyse the effects of such an implementation on the algorithm itself. The model is then
analysed mathematically and predictions are made as to the effects on the three performance metrics discussed in chapter 2.

Chapter 4 aims to produce a simulation environment that will allow a full testing of the predictions made in chapter 3. It does this by considering the types of test problems that will be needed to verify the predictions, allow an understanding of the mechanisms and define how the performance improvements predicted are limited. Problems fulfilling the criteria are then presented, and the simulation environment needed for each discussed in detail.

The following three chapters contain all the results of the simulations carried out during the project. These simulations, their aims and specific details of how they were carried out, are reported and results are presented and discussed. The results verify the predictions made in chapter 3 and through the ensuing discussion, an understanding of the underlying mechanisms is gained and the limitations of the enhancements assessed.

Chapter 8 then concludes the project by considering the implications of the work carried out. In particular it discusses how the aims and objectives have been achieved and what significance the results have for the field. The implications are discussed for their significance to designers of analogue hardware and also as an enhancement scheme.

The work reported in this thesis includes all the research carried out during the course of this project. Apart from chapter 3 where the mathematical analysis was carried out jointly with Alan Murray, the work is entirely my own. In addition, much but not all of this work has been published prior to the completion of this full thesis in a variety of conference proceedings, journals and as book chapters. These papers are listed in appendix C.
1.5 Chapter Summary

This chapter has presented an introduction to the work of this project. By considering the neural network field it has given a motivation for the work and a background to the concepts presented. More specifically, a brief introduction to MLPs has been given and a more comprehensive discussion held on stochastic systems. This discussion has given an introduction to how the synaptic weight noise technique developed throughout this thesis relates to other research published in the current literature. Following this presentation of background work and the central themes of the thesis, an outline of the thesis as a whole has been given.
Chapter 2

Neural Network Performance Metrics

*We know by experience it selfe, that it is a maruelous paine, to finde oute but a short waie, by long wandering.*

Roger Ascham, *The Scholemaster, 1570*

2.1 Introduction

This chapter reviews the current literature for the work carried out in the three neural network performance metrics - fault tolerance, generalisation ability and learning trajectory and speed. In chapter 1 the incorporation of the hardware model into a neural algorithm was noted as giving improvements in all three of these performance areas. Therefore, to allow a discussion of these enhancements to be carried out in the context of the current literature, a review of the metrics must be undertaken. From the literature a selection of representative work is considered, investigating the issues involved when assessing the performance and the techniques used to give performance enhancement. In this way the enhancements seen in later chapters can be examined in the light of the current research and conclusions drawn as to the implications of this work for the field. Therefore this chapter gives a background to the concepts raised in later chapters and the performance improvements that are reported.
2.2 Fault Tolerance

This section addresses the issue of fault tolerance as a performance metric and explains how fault tolerance relates to the central work of this thesis. It starts by defining what is meant by fault tolerance, illustrating its importance and answering the question of whether it is inherent in neural networks. This introduction will be followed by a discussion of the issues that arise when considering how fault tolerance can be achieved in a network and how networks can be tested. Finally the fault tolerance improvements seen when training with synaptic weight noise will be placed in the context of other enhancement methods occurring in the literature.

2.2.1 Introduction

The literature contains a number of definitions of fault tolerance, each with a different emphasis. Here a fault tolerant system is defined as one that is able to withstand the breakdown of a number of its component parts without catastrophic failure occurring. In a neural network the component parts are the neuron nodes and the linking interconnections. In hardware implementations of networks it is important that some degree of fault tolerance is achieved so that occurring faults do not immediately compromise operation of the overall system. This is especially important in fault critical applications. It is, therefore, desirable for the network to be able to withstand the failure of a number of its units (connections or nodes) before total demise of the system occurs. Also, as the number of faulty units increases, the network should exhibit a graceful degradation in performance, rather than a catastrophic failure. In analogue implementations it is also necessary for the network to be robust against small errors in the connections and node operation, as well as complete failure of these units.
In the early investigations of neural networks a degree of fault tolerance was assumed because of the massively parallel nature of the architecture and the analogy drawn with biological systems that are obviously tolerant to faults, see [5] for example and also the numerous references in [9] and [11]. More recently, however, a number of researchers have worked explicitly in the area of fault tolerance and have gone about, in the words of Segee, [64], “debunking the myth of inherent fault tolerance”. Bolt, [9], in his comprehensive Ph.D thesis, challenges the claim of inherent fault tolerance and concludes that although networks have the potential for being fault tolerant, present-day, general learning algorithms (such as BP) do not generate the correct weight configurations for it to be a reality. Phatak, [57], also gives an impressive introduction and review of the issues and problems involved in achieving fault tolerance in MLP neural networks. Fault tolerance is, therefore, something that cannot be assumed, but is a problem that must be addressed alongside the other factors under consideration when training a network.

### 2.2.2 Issues in Fault Tolerance

This section investigates some of the issues raised when dealing with the subject of fault tolerance. First of all the different modes of operation of a neural network are defined and then the concept of storing information in networks, and how this affects the fault tolerance performance, is discussed. Finally the models used for faulting the networks are examined along with methods for simulating these faults. These issues are discussed in length in the literature, [64][9][57], but the following brief summary is necessary to place the arguments and techniques used in later chapters in context.

A neural network operates in two distinct modes:

- operational "on-line" mode.
- learning mode.
Chapter 2. Neural Network Performance Metrics

When the network is operating in these two modes the occurrence of faults will have different effects. In operational mode, where the weights are fixed and are being used in a forward pass through the network, faults will have a direct result on the output performance. In learning mode the effect of faults will be more complicated because the resulting imperfect output is used to calculate weight updates. It therefore depends on the type of fault as to what the effects will be. The study of faults occurring “in-the-loop” (learning mode) is obviously closely related to the work of this thesis and has been considered in depth in the previous chapter in the introductory section on stochastic systems. The faults occurring in operational mode are the ones that will be discussed in this section.

The fault tolerance of a network is dependent on the degree of “constraint” in the network, where a highly constrained network has only just enough parameters to “store” the solution to a problem. In the limit, a truly minimal network, desirable for good generalisation performance [4], would show a degradation in performance if even a single unit fault occurred. Therefore some degree of redundancy is required for fault tolerance. However, the degree to which a fault causes degradation to the network performance can be minimised by considering the representation of information in the network. Considering a minimal network the classic example is that of the “Grandmother Cell”, see [3] and included related citations. This cell contains a complete description of Grandmother and is activated if she comes into view. This is a local representation. If this cell is destroyed then, disastrously, there remains no recollection of Grandmother. In a distributed representation, however, Grandmother’s description will be stored in a number of sub-cells, each representing a feature such as hair colour, eye colour, height ... etc. When she then comes into view all these individual cells will be activated for the recognition process. Faults occurring in one of these cells induces information loss, but will not cause the complete non-recollection of Grandmother. Individual faults occurring in this system have a much reduced significance for the network output. For good fault tolerance performance, therefore, a highly distributed representation is a positive factor, where individual outputs are not dependent on individual units.
When measuring the fault tolerance of a network one of the critical factors is the fault model that is used. This model relates the network to its implementation. In a hardware implementation the way in which the unit fails may cause different overall network results. The fault model should ideally cover the effects of these failures adequately and should also be computationally viable. Often, however, these two criteria compete and so some compromise must be reached.

Existing fault models can be classified into four different types:

- Stuck-at zero.
- Stuck-at $\pm \infty$ (realistically some maximum/minimum value, i.e. for weights $\pm W_{\text{max}}$).
- Going to some random value.
- Sign reversal.

These faults can occur singly or in large numbers and so, with even a small number of units in the network, the combinations are enormous. As soon as even a few units are faulted at the same time, to test all the possible combinations would be computationally inviable. To overcome this problem there are two solutions. The first would be to take the worst case for a single fault and use this fault in combination with the worst case for a second fault ... etc. This technique is known as sequential worst case [64]. The second would be to use a random sample of the possible combinations of faults a number of times to obtain an average fault tolerance performance value. For large networks this technique is the most reasonable compromise between full fault coverage and computational viability.

2.2.3 Achieving Operational Fault Tolerance

Investigations have been carried out in the literature concerning the on-line performance of various neural architectures and networks trained with different al-
golithms. Carter et al have performed studies on a number of these using the continuous sine-wave problem (predicting ahead in the curve from a sliding input window) and a worst case fault model. Their systematic investigations have drawn conclusions for network architectures such as Cerebellar Model Arithmetic Computer (CMAC) networks [12], radial basis functions (RBFs) [64] and MLPs. The latter was trained with various algorithms: BP [64], pruning algorithms [63], etc., all producing different results. A more formal study was carried out by Stevenson et al, [68], who have worked on finding out the sensitivity of networks to errors occurring in the weights. Initially they obtain an error probability for single perceptrons and then move on to multi-layer networks. The authors use a fault model of small errors occurring in the weights, and model the effects in simulation by perturbing weights and comparing the network output with a control experiment with no perturbation. As they use untrained random weights in all their experiments the effects of learning algorithms are not taken into account.

The following sections look at the important issue of how this "on-line" fault tolerance performance can be enhanced, considering in particular MLPs. The various methods seek to improve the operational mode fault tolerance. For simplicity the literature is classified into three sections:

- "In-training" mechanisms.
- Post-training strategies.
- Re-training.

### 2.2.4 "In-Training" Mechanisms

"In-training" methods for enhancing the fault tolerance, as the name suggests, introduce mechanisms into the learning process to give the resulting network a greater tolerance to faults. Of the techniques in this area, that of Sequin and Clay, [65], is the most often cited. They set random hidden layer node outputs to zero during learning at random intervals. This has the effect of imbu
network with an ability to withstand such faults after training. They investigate the tolerance improvements using single node faults and multiple faults, and note that with prolonged training the network can withstand even more failures than the number with which it was trained. This technique is verified by other researchers who add their modification such as setting weights to zero, rather than just unit outputs [8]. Judd and Munro also show that this technique causes a more optimal separation of the hidden layer representation in pattern space, thus giving enhanced fault tolerance performance [39].

Neti et al, [54], use a regularisation algorithm to apply a smoothness constraint to the network. This extra constraint is:

\[ \epsilon' (\omega') - \epsilon (\omega) \leq \epsilon \]  

(2.1)

i.e. \( \epsilon \) is the error to be minimised, \( \omega \) is the weight vector, \( \omega' \) is the weight vector with component \( \nu \) removed and \( \epsilon' \) is the subsequent error due to that fault. The authors use an optimisation method called the successive quadratic programming algorithm to solve this minimisation problem and show improved network fault tolerance.

These mechanisms, employed during the learning process, improve the fault tolerance performance of the resulting weight set. The improved fault tolerance seen in this thesis through the injection of synaptic weight noise also fits into this category where the noise is introduced into the weights during training in a similar way to the Sequin and Clay method.

### 2.2.5 Post-Training Strategies

The strategies reviewed in this section concern methods of improving fault tolerance after the initial trained weights have been produced for the problem. These weights are then manipulated to improve the robustness of the network. Bolt, [9], scales the weights after training to force the outputs of the hidden layer neurons into the extremes of the sigmoid curve. The effects of this technique are also evident when training with synaptic weight noise and cause the decision boundaries
to be much "firmer", or less susceptible to output errors due to small changes in the activation (see later chapters and [50]).

The other post-training methods all introduce added redundancy into the network, using architectural changes or employing supervisor algorithms to search for faults. Emmerson, [16], describes a method christened augmentation in which he scales and replicates units to distribute information more evenly on an expanded weight set. In this way, if a single weight is destroyed then it has less effect because of the reduced significance than if the original weight had been faulted. The networks however need to be much larger than the original "minimal" network for a corresponding improvement in fault tolerance. Phatak and Koren, [57], using a similar scheme replicate hidden nodes to add redundancy and adjust linking weights. The authors note that current learning algorithms develop non-uniformly salient weights thus reducing the impact of the replication. They plan future extensions to optimise the weight set for a distributed representation, before replication. Petsche and Dickinson, [56], use a redundant network of spare neurons alongside the original network which can be moved into operation if the active nodes fail. The authors employ a local representation learning scheme because of its simplicity, allowing only single nodes to become active in any layer for an individual pattern. Using this method it is simple for the supervisor algorithm to detect faults occurring and thus to activate the spare nodes.

These methods improve the fault tolerance performance of the network using post-training strategies. Also, they all incorporate the additional algorithms into the basic learning phase needed to adjust and replicate weight values.

2.2.6 Re-training

This final type of method for achieving fault tolerance uses re-training after the fault has occurred. Sequin and Clay, [65], investigate this method noting that if there is enough redundancy in the original specification then re-training the faulted network will produce a new solution quickly. If, however, there is no redundancy then replacement nodes can be added with randomised parameters
before re-training. They state that the re-training time is approximately the same for both, assuming that it is possible in the first case. Plaut, motivated by the study of brain damage patients, also indicates that performance recovery is very fast when units are removed from a network [59].

This work shows the use of a re-training scheme for achieving recovery from faults, needing a supervisor algorithm to dictate the necessity for re-training. Fast recovery is, however, possible.

2.2.7 Summary

In summary, fault tolerance performance in the implementations of neural networks is an important area for study. Many of the commonly held beliefs about the inherent nature of robustness in networks are completely unfounded - originating from biological analogies rather than scientific research. The “in-training” techniques for improving fault tolerance attempt to use the parallel nature of the architecture to achieve tolerance and are to a degree successful. Their success shows that it is the training algorithms that are deficient in supplying a robust weight set. More straightforward, but less interesting replication and voting schemes can achieve tolerance to faults (a common technique in digital systems). Paraphrasing Phatak, “nothing short of Triple Modular Redundancy and a polling scheme will give complete tolerance to a single fault” [57]. Despite this significant improvements can be made using the methods described above.

Using the background information summarised here, chapter 5 will report the improvements seen in fault tolerance performance when training with synaptic weight noise. Using a fault model the robustness will be quantified and also the information storage methods analysed.
2.3 Generalisation

This section discusses the issue of generalisation performance in neural networks. An overview and critique is given of the work carried out by researchers in this area. Initially generalisation is defined and its importance is illustrated. In addition, the concept of improved generalisation of a network is considered. This work is carried out to give a background to the generalisation enhancements reported in this thesis.

2.3.1 Introduction

The generalisation ability of a trained network is the ability to process (or classify) a series of input-output vectors that were not contained explicitly in the training data set. The network must therefore, during learning, extract the "underlying rules" of the training patterns. Perfect generalisation will have been achieved if a correct mapping is obtained for all the patterns in the problem distribution.

The necessity for a good generalisation ability is inherent in many real world problems where the complete problem distribution is not available at the time of training. It is in these problems that generalisation is required, although at the same time the network can only really be expected to generalise to patterns within the bounds of the training data. Generalisation can be expressed, therefore, as either interpolation or extrapolation. Interpolation within the bounds of the training data should be a solvable problem, while on the other hand, extrapolation outside these bounds is risky, as the "rules" of the data do not necessarily apply. Therefore, the network's processing or classification performance is dependent on the training pattern set. However, if a representative pattern set of the distribution can be obtained to train the network, then the network should be able to interpolate the other mappings within that distribution.

Often the patterns in the training set are a sparse representation of the problem distribution and, in addition, noise on these training patterns is also likely
to occur. As the majority of learning algorithms, such as BP, train a network by reducing some function of the difference between the actual output of the training set and a target output, the network can only encapsulate the information in the training set. If the \textbf{computational capacity} of the network is greater than the \textbf{required capacity} of the data set and the stopping criteria are such that they necessitate a low error, then "over-fitting" (learning the idiosyncrasies of the training data set) occurs. When this happens a poor performance of the trained network on the full problem distribution is seen (i.e. poor generalisation performance). The problem, therefore, is to achieve the best possible generalisation performance from a sparsely represented and noisy pattern set.

The neural network literature is rich in techniques that attempt to improve the generalisation performance of MLPs. To limit and simplify this analysis of these approaches, here they have been divided into three classes and significant work taken from each. The three classes are as follows:

1. \textbf{Pre-training parameter adjustment}. This class considers ideas for improving generalisation by changing parameters, such as: adaptation rates; numbers of hidden layers; numbers of hidden layer units and what stopping criteria to use.

2. \textbf{"In-training" complexity constraints}. This class contains algorithms that control the storage capacity adaptively to stop the training algorithm from over-fitting the training data.

3. \textbf{Post-training destructive algorithms} for removing any unnecessary units or weights and then re-training, to reduce the network capacity.

The following sections compare and contrast these three classes of generalisation improvement techniques.
2.3.2 Pre-training parameter adjustment

This section looks at the ways in which the generalisation performance of a neural network can be improved by influencing some of the decisions made before training begins. Some of the ideas that are presented below belong to the area of general neural network "folklore", but are nevertheless used widely as methods for improving the generalisation performance of a network and, therefore, must be included here.

The problem, as previously stated, is to learn a good, accurate solution to the problem in question despite having a sparsely represented and noisy training pattern set. The comparison of the network's computational capacity and the required capacity of the training pattern set is an important issue. A way of improving the generalisation, therefore, would either be to reduce the capacity of the network so that the underlying rules are extracted rather than idiosyncracies of individual patterns, or alternatively to increase the size of the training pattern set artificially.

One method for reducing how accurately the network can learn the training set (and hence its capacity) is to vary the rate of adaptation of the weights. This rate (with standard BP at least) is set at the start of training. Starting a series of training experiments, therefore, to analyse generalisation performance as it varies with adaptation rate, can produce enough statistics to optimise this learning parameter1.

One rather more obvious way of reducing the network's capacity is that of reducing the size of the hidden layer of a three layer network, or the hidden layers of a larger network. This whole issue is discussed in the seminal paper by Baum and Haussler, [4], who investigate the relationship between generalisation error ($\epsilon_g$), number of training patterns ($m$) and the total number of weights ($W$)

---

1Note that a similar approach can be used to optimise the momentum parameter for maximum generalisation performance.
in the network. In this paper the authors present the upper and lower bounds on the network architecture required for a particular problem. Lang et al, [42], use a similar approach in their comprehensive study of network architecture for a particular problem - that of spoken letter recognition. From their knowledge of the problem itself, they are able to make changes to the network architecture to improve the generalisation performance. The most interesting section of this work is in the discussion of the use of hidden layer receptive fields (weight sharing), where only a proportion of the inputs are applied to the hidden layer at any one time, thus reducing the free parameters in the network. They note an improvement in the generalisation performance on their limited training set. This technique is however, restricted by the need for a priori knowledge of the problem. Other techniques are also noted in this paper such as the use of a validation set (see later) and also other more complicated techniques that will be discussed in the following sections. The generalisation performance gains were, however, mostly made in this problem through pre-training architectural decisions.

Another method that limits how accurately a network can map the idiosyncrasies of the training pattern set, is in the use of a validation set. This set of patterns is used to keep a note during training of the generalisation error, $\epsilon_g$, as opposed to the training set error, $\epsilon_t$, as it is known that the minimum value of $\epsilon_g$ is not at the minimum of $\epsilon_t$ for a sparsely represented and noisy training pattern set. The use of a validation set although a valuable technique, reduces the number of patterns available for training purposes and also must be representative of the problem distribution for a good decision of when to stop training, to be made. Lang et al, [42], (mentioned above) use this method to determine an "optimal" length of training and then re-train with the validation set included in the training set. It is not clear, however, how good this "optimal length" of training is for the full training set.

The methods described above for improving generalisation all involve limiting the network computational capacity either by controlling the accuracy with which it can adapt to the information contained in the training set, or the capacity of
the network (in terms of parameters). The final technique for generalisation improvement considered in this section, is to increase the required capacity of the training data set artificially. One, and perhaps the most obvious way of doing this is by using noise in the input data of the network. The patterns are thus "smeared" using some previously chosen noise distribution and then used for training. Many papers have studied the strengths and weaknesses of this approach, such as what distribution of noise to use, Holmström and Koistinen [31] and many others including, Plaut et al [58], Matsuoka [44]. This approach is obviously also limited to the training patterns available. If the pattern under consideration is at the centre of its class in pattern space then "smearing" it will improve generalisation; however if the pattern is right on the edge of a class boundary then the noise may cause incorrect classifications to be made.

These methods of generalisation performance enhancement have all been concerned with decisions about network parameters, that have to be made prior to the training process taking place. For appropriate use to be made of these techniques an a priori knowledge of the problem is needed and often a large number of experiments to determine good parameter values.

2.3.3 "In-training" complexity constraints

This section considers a number of the ways that generalisation performance can be influenced during the training process itself. The following approaches all involve methods of controlling the computational capacity of the network and the evolution of the weights in the network. Again, as in the previous section the basis of many of these ideas have been around in the neural network community for many years and hence there are many permutations to the procedures. Below, however, are some of the more distinct and interesting approaches.

As in all methods for controlling generalisation performance the vital concern is to balance the computational capacity of the network against the required capacity of the training data set. The standard "in-training" method for improving generalisation ability is to add a complexity term $\epsilon_c$ to the cost function of the
learning algorithm. This term is sometimes known as a *penalty* term and at its simplest may just be some weight magnitude, thus introducing a *weight decay*. This type of cost function would be of the form:

\[ \epsilon_{Tot} = \epsilon_t + \lambda \epsilon_c = \epsilon_t + \lambda \sum_i (w_i^2) \]  

(2.2)

i.e. \( \epsilon_{Tot} \) is the total cost function error, \( \epsilon_t \) is the nominal error due to the training data set, \( \lambda \) is some parameter controlling the rate of weight decay and \( w_i \) is some particular synaptic weight. Intuitively this term has the effect of reducing the magnitude of the large weights that have small changes due to the nominal error. This type of complexity term has been used by many researchers including Lang *et al* [42] (see above), Hanson and Pratt [22], Plaut *et al* [58].

Weigend *et al*, [75], use a similar penalty term but with an extra parameter \( w_o \) which allows a number of the smaller weights to escape the effect of the weight decay. This complexity term is of the form:

\[ \epsilon_c = \lambda \sum_i \frac{w_i^2 / w_o^2}{1 + w_i^2 / w_o^2} \]  

(2.3)

One of the main arguments used against the simplest form of weight decay is that it favours two small weights instead of one large one. Weigend *et al* note that judicious use of the \( w_o \) parameter can avoid such an occurrence. Nowlan and Hinton, [55], in their excellent paper also report this weakness in pure weight decay and being motivated by the work of Lang *et al*, [42], take an approach known as *soft weight-sharing*. This algorithm forces the weight values into normal distributions. Weights individually come under the influence of a penalty term that tries to minimise the difference between the weights and some mean value. While some weights are reduced to zero by a narrow distribution centred on zero, broader distributions centred on other values take responsibility for larger weights. The authors note very good results comparing their work with pure weight decay and a cross validation scheme. That the complexity of this algorithm mitigates against easy implementation and fast learning is however, a criticism that could be levelled at this approach.
Chapter 2. Neural Network Performance Metrics

The final approach discussed in this section, Bishop [6], involves a scheme for minimising the curvature in the decision boundaries, i.e. small changes occurring in the outputs due to small changes in the inputs. The penalty term is of the form:

$$
\epsilon_c = \frac{1}{2P} \sum_P \sum_i \sum_k \left( \frac{\partial^2 o_{kp}}{\partial o_{ip}^2} \right)^2
$$

(2.4)

i.e. $P$ is the total number of patterns, $o_{kp}$ and $o_{ip}$ are the outputs from layers $k$ and $i$, for pattern $P$, respectively. The author discusses the limitations of standard weight decay techniques noting that the optimum value of $\lambda$ is dependent on the individual weight size. Ideally, therefore, each weight should have a corresponding $\lambda$. To overcome this problem the penalty term (2.4) regularises the network function rather than individual parameters.

The approaches described in this section improve the generalisation performance of a network by incorporating complexity terms into the learning cost function. The capacity of the network is effectively reduced by these extra terms. The weakness of these approaches is in the extra algorithmic complexity that occurs and hence a high probability of longer training times. However, the results shown on many of the papers reviewed have shown an improvement in the generalisation performance of the trained networks.

2.3.4 Post-training destructive algorithms

The final class of method for improving generalisation performance of a network concern the algorithms that take as input a fully or partially trained network. All the approaches in this section consider methods for reducing the network capacity selectively, removing network parameters that are not important to the problem solution. The whole philosophy of this kind of approach is based on the idea that a small network is less likely to over-fit the training data set. The major differences in the following reviewed work is in how the selection process for non-important parameters is made and also on the type of parameters destroyed.
Mozer and Smolensky [46] in their "skeletonisation" paper attempt to remove units (hidden or input layer neurons) from the network. They propose an algorithm that removes units that convey redundant information according to some relevance assessment. They note a weakness in generalisation enhancement methods that use penalty terms in the cost function, saying that with multiple error terms it is often impossible to avoid an increased number of local minima and that the algorithms often reach compromise solutions that partially satisfy each of the error terms. The authors note an obvious relevance assessment method:

\[ \rho_i \approx \epsilon_{\text{without unit } i} - \epsilon_{\text{with unit } i} \]  
\[ (2.5) \]

i.e. \( \rho_i \) is some measure of relevance of unit \( i \) and \( \epsilon \) is some error. This procedure requires a complete pass of the entire training pattern set and hence they use an approximate version using:

\[ \rho_i = -\frac{\partial \epsilon}{\partial \alpha} \]  
\[ (2.6) \]

i.e. \( \alpha \) is the attentional strength or measure of influence of a particular unit. This measure allows decisions to be made on which units are appropriate to remove.

Optimal Brain Damage (OBD), Le Cun et al [43], uses the hessian matrix (a matrix of the second derivatives of error with respect to the weights) to obtain some relevance or saliency assessment of individual weights in a network. The saliency \( s \) is given by:

\[ s_k = h_{kk} \frac{u_k^2}{2} \]  
\[ (2.7) \]

i.e. \( h_{kk} \) is a term in the diagonal hessian matrix and \( u_k \) is the parameter itself. The authors use an approximate diagonal version of the hessian matrix to reduce the algorithmic complexity. Parameters with a low saliency can, therefore, be removed from the network.

More recently Hassibi et al, [24], have produced the Optimal Brain Surgeon (OBS) algorithm, which uses the same procedure as OBD, but uses the full hessian matrix to calculate the weight saliencies. Also the authors note that whereas the two previous approaches need a certain amount of re-training after the pruning
has occurred, the OBS algorithm adjusts the remaining weights automatically to reduce the error.

All three of these approaches use some measure of relevance assessment of parameters in the network and then remove the unimportant ones to reduce the network computational capacity, thus improving the generalisation performance. It is interesting to note that the authors of the OBD paper [43] find only a small improvement in generalisation performance over a "state-of-the-art" network that has been previously highly constrained to the problem. Although there will be some enhancement of the solution, an increased overall training time from the additional computational requirements will be evident.

2.3.5 Summary

The previous three sections discuss three classes of methods improving the generalisation performance of networks when faced with training on an sparsely represented and noisy training data set. The methods address the issue of balancing the computational capacity of a network to the required capacity of a training data set. As in all neural network research, compromises must be made to achieve the desired performance for particular applications. Some of the more impressive results contained in the sections describing the second and third class of generalisation enhancement methods, require more processing during and after learning. The complexity of some of these algorithms also makes implementation difficult. Other methods in the first section are simpler, but their success is more problem specific. The choice of a method for generalisation improvements must, therefore, remain dependent on the problem and environment.

The generalisation ability enhancements studied in the work carried out in this project using synaptic weight noise fit into the second class of work described above. The enhancements seen will be discussed at length in chapter 6 using the literature and issues reviewed in this section as a background.
2.4 Learning Trajectory and Speed

This section gives a background to the effects of noise in the synaptic weights on the learning trajectory and the concomitant learning speed. An introduction is given to the issues involved when considering how weight changes move the current weight vector across the error surface. A search is also carried out of the literature for the general techniques used to adapt weights in an efficient manner. It is seen that there are many different "novel" algorithms that report improved learning times over the standard BP algorithm. Here, however, only a few can be reviewed and so only significant examples are considered.

2.4.1 Introduction

To help visualise a problem it is often useful to view it as a multi-dimensional error surface where changes in the network interconnection strengths cause movement over the surface. In reality, the more weighted connections there are in the network, the more dimensions there are to the error surface and the more impossible it is to visualise. However, considering the 3-D case is helpful and terms such as "gradient descent" and "up-hill" utilise this. Using this scheme we can consider learning algorithms as seeking to find a minimum in the error surface. Gradient descent algorithms in their pure form seek to move down slopes from the starting position until a minimum is found, where a move in any direction on the surface causes an increase in the error. Therefore the ability of the algorithm to find a global minimum is dependent on the error surface and the starting position. Often, however, problems occur when the weight vector gets "stuck" in a local minimum and is unable to continue moving across the error surface to a global one. Other algorithms, taking less deterministic approaches, cover more of the error surface by taking random steps while also trying to find a minimum.

For many applications of neural networks the speed at which a minimum is found may not be an important issue as it only needs to be done once. The
weights can then be used in the on-line network to perform its function. However, in problems where learning must be done in situ, continually adapting to a changing environment, speed is important. Also, for researchers to form conclusive comparisons of different algorithms on a wide variety of problems is extremely time consuming and can be impossible if the algorithms are slow. Therefore, it is important for fast, efficient algorithms to be designed and implemented.

Learning speed is not the only consideration. It is also important that algorithms must be scalable to different sizes of problem and easy to use. It is one thing to test an algorithm on a small artificial problem (e.g. the ubiquitous XOR), but it is quite another to try and solve a real world problem that can involve many thousands of input patterns. Also, the algorithm must be easy to use, if it is to be accepted widely. It must, therefore, not have many critical adjustable parameters that require specific knowledge of the problem for the correct combination to be found. In the words of Tollenaere [73], “the optimal region for a given parameter must be large and the number of parameters small”. This gives the “neural programmer” a robust algorithm to use on a wide variety of problems.

For the more deterministic algorithms, such as back-propagation, the rate at which convergence to a solution takes place depends on where the initialised set of weights position the network mapping on the error surface. From there steps must be taken towards a solution. Many algorithms use an adaptation rate parameter which controls how large a step is made down the current gradient. On flat sections large changes can be tolerated, but in a region with high curvature a large step may cause an over-shoot of the minimum. This discretisation of the gradient descent algorithm can cause uphill steps to be made, perhaps even causing the algorithm to become unstable. At very low adaptation rates steps can be made safely down gradients with high curvature, but learning is very slow over flat regions of the surface. Unfortunately for a general algorithm of this type no one learning rate parameter can be chosen, but must be selected dependent on the problem and error surface.
The speed of algorithms is often a source of contradiction in the literature. Researchers claim a set of results using an algorithm on a problem using a certain set of parameters. Other researchers then produce a new algorithm and test on a different problem. Bench-marks must, therefore, be used to quantify performances. Explicit note must also be kept on things such as stopping criteria, learning parameters ... etc., so that conclusive comparisons can be made.

This section has raised a number of issues that are important when assessing learning algorithms. The following sections consider the work in the current literature that addresses some of these issues and also various types of algorithm that seek to optimise a learning approach.

2.4.2 Improvement methods

Acknowledging the issues raised in the previous section, steps must be taken to improve the learned solution, i.e. the "trajectory", and the rate at which the result is achieved. Many researchers have undertaken this task considering some if not all of the issues, producing a variety of algorithms. Many techniques also exist in the neural network "folklore" for improving speed and evading local minima. One of the most common and simple ways of overcoming the local minima problem is simply to start a new network with a different random set of weights until a "good" solution is found. This technique although valid can cause discrepancies in the calculation of learning speed. Fahlman, [17], addresses these issues in his excellent paper, by incorporating restarts into the comparison measure that he uses to compare learning times. Many other researchers appear to simply ignore the networks that "fail" to reach an adequate solution, calculating speed using only good runs.

A method for improving speed that is commonly found in the literature is the variation of BP known as "stochastic" back-propagation (SBP). In this variation weight updates occur after the presentation of every pattern, rather than the more correct update from accumulated weight changes after the entire pattern set has been presented (batch mode). SBP has the effect of giving a moving
weight set as the patterns are presented and adapted to. On many problems this much faster method is an adequate learning method, although on problems where very accurate solutions are needed batch mode may be required.

Other techniques occurring in the literature are categorised below into one of three areas :-

- Local adaptation rates.
- Second order methods.
- Stochastic learning.

In the following sections algorithms used to improve learning performance over standard BP will be briefly discussed. For the complete algorithm description the referenced papers give the required details for implementation.

2.4.3 Local Adaptation Rates

This section considers the learning rate enhancement methods that use local adaptation rates. As each weight "sees" a different error surface with different gradients around its current position, various adaptation rates are applicable when calculating weight changes. Using this method, therefore, each weight has a corresponding adaptation rate that depends upon the local shape of the error surface.

Devos and Orban in their paper, [15], initialise individual adaptation rates for each weight and then vary it every epoch depending on the sign of the gradient, calculated using BP. If the sign remains the same for a weight, then the associated rate is increased exponentially. If, however, the sign changes (over-shooting a minimum) then the rate is re-initialised. Tollenaere improves on this scheme by introducing a slight variation on the algorithm. He removes any step that causes a change in the sign of the gradient and then decrements the rate, thus stopping any over-shoot. He notes rates of convergence much improved over standard BP
and also finds that the algorithm is robust to parameter changes and scales well to larger problems where BP struggles.

Jacobs, [37], gives a comprehensive review of the whole subject area carrying out a comparison of a variety of learning rate enhancement techniques. Of the ones relevant to this section he presents heuristics for adjusting local adaptation rates called the delta-bar-delta learning rule, that adjust the individual learning rates in a similar way to the above methods. The technique increments learning rates linearly so as not to let them grow too fast, while decrementing occurs in an exponential fashion, to keep them always positive and to allow rapid reduction. The results presented show very impressive rates of convergence for a variety of problems.

This section has discussed a number of techniques that have individual adaptation rates for each network weight. Although the computational and storage requirements are increased, significant reductions are reported for a number of problems.

2.4.4 Second Order Methods

This section considers methods that look at second order features in the error surface. By considering the curvature as well as the gradient more accurate steps can be taken towards a minimum. A pure second order method would compute a jump by calculating the hessian matrix and assuming that the surface is locally quadratic, take steps towards a minimum. This, however, is computationally very expensive for even small problems and impractical for real applications. Many researchers, therefore, use approximations of the second derivative to make "more optimal" jumps across the surface.

Perhaps the most common and simple second order method is the use of a momentum parameter in BP. This parameter is used to control how much of the previous weight update is added to the present update. On flat surfaces this history allows momentum to build up so that large steps can be made down shallow
slopes. Momentum has been shown by many researchers to improve convergence speed in particular problems, [17] [37] [73]. However, as Fahlman notes [17], an experienced operator and knowledge of the problem are required to make appropriate use of it. In his paper he considers a number of methods to use second order information, including momentum and the calculation of the full hessian matrix. After discounting these he uses a difference quotient to obtain second order information, noting that the assumptions involved are dubious, but expecting the iterative process to accommodate the inadequacies in his approximation. Using the information and some complicated update procedures he designs the quickprop (QP) algorithm. This is reported to be one of the fastest algorithms on a variety of problems [66].

Another second order method is the conjugate gradient learning technique. This has been studied by a significant number of researchers on a wide variety of optimisation problems, see Johansson [38] for a review. Similar to Fahlman’s method it uses an approximation of the second order term to calculate a minimum. A widely reported order of magnitude speed-up is noted when using this algorithm, [38] [66].

The algorithms discussed in this section look at curvature in the error surface to make more optimal steps in the weights towards a minimum. Significant speed-up is achieved compared with the techniques discussed earlier, although at the expense of increased computation and storage requirements.

2.4.5 Stochastic Learning

The third category, stochastic learning, uses random steps to widen the search area on the error surface. Instead of following the contours of the surrounding surface deterministically, the weight vector incorporates a random element. Using a combination of gradient descent and random jumps, therefore, a search can be made of the error surface for a minimum.
Using a technique based on Simulated Annealing (SA), Ackley et al., [1], produced their Boltzmann learning algorithm. As in SA the initial stages of learning use a high stochasticity and as learning progresses the random steps are reduced until a stable solution is found. Their results show the ability of the algorithm to train network architectures.

Hanson, [23], in his "stochastic version of the delta rule" uses noise in the synaptic connections to aid learning. He notes that SA techniques introduce noise into the network independently of the learning process, while his algorithm controls the noise more intelligently. The algorithm, instead of using accurate weights, adapts means and variances of weight distributions, for each weight, from which values are randomly chosen for the forward pass. Errors cause an increased variance and although there is a general overall decay in this term, if they consistently occur the variance will increase causing large random steps to be taken. The author notes that this enables the algorithm to escape from local minima and to find solutions significantly faster than the standard BP.

The two stochastic learning algorithms reviewed above use random steps to avoid local minima and to speed-up gradient descent based algorithms. Although in [23] learning time is reduced, this is at the expense of higher storage requirements and also increased computation. The SA based algorithm, [1], has been shown in other reviews [34] to be slow for optimisation problems of this type.

### 2.4.6 Summary

This section provides a background to the area of learning in neural network type architectures. The issues raised give an insight into the problems that are faced. The algorithms reviewed all seek to overcome these problems and are all partly successful. There is, however, a cost in terms of increased computational and storage requirements. The algorithm chosen to solve a problem must, therefore, be carefully chosen to suit the problem itself and the subsequent error surface. Some algorithms have proven to be robust to changes in problem and therefore will be most popular.
Chapter 2. Neural Network Performance Metrics

The work described by this thesis fits into the second and third categories of learning enhancement above, and is closely related to Hanson's work [23]. However, the computation and storage requirements are much reduced. Chapters 3 and 7 will discuss the effects of synaptic weight noise in the learning trajectory and speed, through a mathematical analysis and intensive simulation results on a variety of problems.

2.5 Chapter Summary

This chapter has discussed the three metrics by which neural network performance is judged. The issues raised and discussion of the work reported in the literature allows the work of this thesis to be placed in context. In particular, fault tolerance has been discussed in terms of how information is distributed in the network and generalisation ability in terms of network computational capacity and the required capacity of the problem. The issues involved in assessing these two metrics have therefore been considered, as well as the enhancement schemes found in the literature. These concepts will be important for later comparison with the enhancements seen when injecting synaptic weight noise into the learning. In addition, issues involved in analysing the learning trajectory and speed have been considered and the literature examined for enhancement techniques. In summary, this chapter has given a background for the work of this thesis by looking at the relevant work and ideas found in the current literature.
Chapter 3

Noise in Neural Implementations

My aim is true, my message is clear,

Its curtains for you Elizabeth my dear.

The Stone Roses

3.1 Introduction

This chapter describes a model for examining the effects of implementation errors on neural algorithms. By producing a model, the simulation and analysis of the implementation will be made possible, allowing conclusions to be drawn on the implications of errors on the algorithms. The model must be appropriate, but simple, allowing easy software implementation and detailed analysis.

Chapter 1 discussed the effects of analogue imprecision and digital inaccuracy. In the discussion the question was raised as to whether it was precision or accuracy that was of primary importance when considering a neural implementation. Although the answer to this question still remains unclear, it is evident from the current literature that significant and conclusive research has been carried out to examine the effects of digital bit resolution. Techniques have also been developed to minimise these accuracy requirements. The need for accuracy has, therefore, been quantified for a wide variety of problems. However, the precision
requirements for analogue neural algorithm implementation are presently unclear as only a limited amount of research has been carried out in this area. The model presented in this chapter aims to allow a study of the precision requirements in analogue hardware to be carried out.

To consider the important errors for forming an accurate model, this chapter looks at a wide variety of implementation errors from analogue and digital designs. From this spectrum of possibilities the area of interest is constrained and a model is developed for the work of this thesis. The model is then analysed mathematically. Using a simple expansion, predictions are made about the effects of the implementation on learning performance. In particular, the predictions address the effect of the imprecision on the three important neural network performance metrics - the fault tolerance and generalisation ability of the final solution, and the learning trajectory and speed.

3.2 Implementation Errors

This section discusses in detail errors that occur when implementing MLP-type neural algorithms in hardware. In particular, systems are considered where iterative learning is implemented, rather than where only operational mode networks are used with a pre-trained set of weights. In this section references to "implementations" refer to custom hardware rather than the software variety, unless explicitly stated.

3.2.1 Introduction

Errors occur in all implementations of algorithms whether software or hardware. In the software case the use of highly accurate and precise floating-point arithmetic with a large dynamic range invariably gives reliable algorithmic performance for most problems. In hardware, however, circuit and system limitations
put constraints on the possible capabilities of the implementation and, therefore, compromises must be made. The resulting errors generally fall into two categories:

- Analogue implementation imprecision.

- Digital implementation inaccuracy.

However, there are very few cases of purely analogue designs. Most hardware implementations are either digital, or at least have some digital circuitry in a hybrid system. The algorithms that are implemented in hardware also tend to be restricted to a forward pass through the network, rather than full learning algorithms. Usually in this type of implementation, if learning is required, then the hardware is placed “in-the-loop” of some software driven learning scheme. This enables the system to use floating-point accuracy to carry out the learning stages of the algorithm - updating weights which can then be down-loaded onto the hardware. For implementations where the forward pass and the learning are carried out in hardware, only a limited amount of work has been reported. Some researchers are however, moving towards such systems, [77], although many of the implications are still unexplored, especially in the purely analogue case. Here both types of implementation are considered.

### 3.2.2 In-The-Loop Implementations

In the forward pass of a MLP the basic operations are a multiply-and-add function and a non-linear “squashing function”. The hardware section of an in-the-loop learning implementation, therefore, will carry out these operations hopefully at high speed and as much as possible in parallel. In a standard three layer network the operations would be as follows:
1. Using input vector $\vec{a}_i$ and weight matrix $\vec{w}_{ji}$, the activation levels $\text{net}_j$ of $J$ neurons in the hidden layer are calculated:

$$\text{net}_j = w_{j0} + \sum_{i=1}^{I} a_i w_{ji}$$

(3.1)

i.e. $w_{j0}$ is the bias.

2. Calculate the outputs of the hidden layer neurons $\vec{o}_j$ using a non-linear function:

$$o_j = \mathcal{F}(\text{net}_j, \text{constant})$$

(3.2)

i.e. $\mathcal{F}()$ is usually the sigmoid function and the constant term is usually the "temperature".

3. Repeat the operation in (3.1) using $\vec{o}_j$ as the inputs and weight matrix $\vec{w}_{kj}$ to calculate the activations of the output neurons:

$$\text{net}_k = w_{k0} + \sum_{j=1}^{J} o_j w_{kj}$$

(3.3)

4. Again carry out the non-linear squashing function as in (3.2):

$$o_k = \mathcal{F}(\text{net}_k, \text{constant})$$

(3.4)

The algorithm is obviously repetitive and hence many hardware implementations only incorporate one layer of multiply-and-add, followed by a non-linear function. The second layer is then calculated using the same hardware, but with different controlling parameters.

To carry out this "in-the-loop" procedure there are a number of issues that must be considered. These are listed below:

1. Number representation - mapping the software floating-point values to hardware voltages.

2. Interfacing - quantisation effects at the hardware/software interface.
3. Dynamic range - making full use of the dynamic range of the hardware.

4. Input patterns - discrepancies between the real application and the patterns that reach the hardware.

These issues concern implementation efficiency and, therefore, the resulting errors can be minimised by careful design. However, there are a number of inherent hardware errors that will also be discussed below.

The first thing that must be considered in this type of implementation is how floating-point numbers in the controlling software are to be represented in the hardware. In the digital case there will be a limited number of bits available, although scaling techniques can be introduced [28]. In the analogue case there is ideally an infinite number of voltages that can be used within some range. Considering, for example, some weight distribution (see Fig. 3-1), the range of weights must be mapped onto the allowable voltages. Implementations vary on

![Figure 3-1: Conceptualised probability density function of weight values and levels of allowed quantisation voltage values.](image)

how negative numbers are represented, although clearly the position of zero is
always important. Scaling and shifting is, therefore, necessary when mapping weights onto the hardware. Also at the network output, there is a complex relationship between the output voltages and the floating-point target values in the controlling software. An exact correlation is impossible, making the use of analogue outputs a problem although, as long as the scaling factors used remain constant, the errors can be reduced through learning. As Frye et al, [19], noted in their paper, the precalculation of weights may be "a waste of time" for many real world applications and learning must be carried out with the hardware in-the-loop.

The second consideration is that of interfacing. Usually this is done digitally, at least for the weights as they are stored in memory on the hardware, although some analogue storage techniques are beginning to be developed (see [30] for example). Considering the general digital case, quantisation of the accurate floating-point values occurs, as they are downloaded onto the hardware. This process naturally causes errors to be introduced:

\[ w^* = w + \Delta \]  

i.e. \( w^* \) is the weight loaded onto the hardware, \( w \) the floating-point weight value and \( \Delta \) the occurring error. Also considering an analogue implementation, voltages at the network output are sampled and loaded into the software to be compared with target values for learning to take place. Here as with the weights, the sampled value will include some error. The quantisation allowable, therefore, will be a controlling factor in the design of the system and has been studied in [29].

The third issue is the dynamic range of the hardware. It is important that the downloaded values do not exceed the operating range of the hardware as "clipping" (saturation of a voltage to some maximum value) will occur. Scaling factors must, therefore, take the predicted final weight range into account, although some clipping may be acceptable as the majority of weights will be in the centre of the distribution (see Fig 3-1). In digital implementations, clipping or overflow will occur once the dynamic range has been exceeded, causing errors. In analogue networks, device characteristics tend to become non-linear at the
extremes of dynamic range and a form of clipping will also occur. Some work has been carried out to look at the effects of clipping, [13], and although conclusive experimentation was not carried out, the limited results showed no serious effects on the learning performance.

The final issue considered here, in this brief summary, is that of input distortion. Sometimes it is possible to connect inputs directly into the hardware by using hard-wired sensors, although sampling must take place and floating-point target values are also required by the learning software. Often it is necessary to digitally quantise the inputs, causing discrepancies between the problem "seen" by the hardware and the real one. The effect of this quantisation will be problem dependent, i.e. it is more serious if the application in question requires the use of fine detail to process the individual patterns.

Consideration of all these issues, therefore, must be undertaken before an efficient implementation can be achieved. Many of the errors occurring because of these implementation problems can be minimised by improving on hardware and software interfacing techniques and using quantisation schemes that are appropriate for the problem in question. However, for a general purpose piece of hardware capable of solving a variety of problems, errors are inevitable.

In spite of much optimising of the details of implementation, errors will inevitably occur due to the issues discussed above and also more fundamental hardware limitations. In digital designs the errors can be summarised as follows:

- Truncation errors due to dynamic range overflow.
- Quantisation due to limited bit accuracy.

These are widely discussed in the literature (see earlier) and the effects on algorithmic performance have been quantified for a range of problems. The accuracy requirements for digital hardware are, therefore, well known and techniques such as scaling, probabilistic rounding and gain control can be used to minimise them.
In analogue implementations the implications of the errors that occur, due to implementation constraints and those inherent to the hardware, are uncertain. Also the exact details of all the possible errors that will affect the learning performance are unclear, however, from the literature, [19][13][21], the main errors can be summarised as follows :-

- Noise in calculation, due to device variation.
- Non-linear multiplication characteristics.
- Offsets in references, such as the output zero point.

The direct effect of errors of these types is unclear, although Frye et al noted that learning is possible for even high levels of noise in their model of an analogue implementation [19]. The reasons behind this and the learning improvements shown experimentally by Murray [49] due to noise in the synaptic multiplication, remain unclear.

In summary, it is clear that the implementation of neural algorithms in hardware has complicated implications for the performance of the algorithm itself. Some of the errors that occur can be overcome by efficient use of hardware and software and a detailed a priori knowledge of the problem. Much research has been carried out, modelling the effects of digital inaccuracy in neural circuits. Limits have been noted for a wide variety of problems. Analogue imprecision is less well understood and hence there is scope for work to be carried out to look at the implications and effects of analogue implementation on neural algorithms. This theme forms a basis for the work carried out in this thesis.

### 3.2.3 Learning “On-chip”

This section briefly considers the complicated issue of implementing learning algorithms in hardware. While in the last section, using in-the-loop designs, the hardware was used to implement the neural feed-forward algorithm and learning
was carried out by controlling software, here the learning is considered as being carried out by custom hardware as well. The use of highly accurate floating-point arithmetic for learning, therefore, becomes impractical and compromises have to be made. Designing learning schemes "on-chip" raises more algorithmic error considerations. The hardware may be a single chip or may involve many discrete components. This issue, along with more general implications, will be discussed below.

The advantage of using an in-the-loop approach is that accurate weights can be stored and used to calculate accurate weight changes. It is widely quoted that more bit resolution (accuracy) is required in the learning process than in the forward pass (see chapter 1). Therefore, in digital designs more bits are required in the implementation increasing the size and complexity of the design. Essentially many of the complicated interfacing problems are removed by including learning in the implementation. Converting the number representation from one form to another becomes unnecessary, assuming an efficient implementation. However, more compromises are necessary and errors are inevitably involved. In a digital design the implications of this are relatively straightforward. Assumptions are made on the required accuracy for the particular problems that will be encountered by the implementation. This should make the transition from learning in-the-loop to learning on-chip a natural process.

Analogue implementations are much more complex. At present the implications of analogue imprecision in in-the-loop designs have had only limited attention in the literature. Therefore, making the step from using precise and accurate weights in the update calculation to weights that are subject to all the imprecision of analogue hardware, is one that needs much research. While it is necessary to proceed with design of analogue implementations of analogue hardware learning schemes, contemporaneous research must be carried out into the repercussions on the algorithms themselves. Obviously the hardware design is not necessarily completely analogue and techniques such as weight-refresh will aid the retention
of weight accuracy. However, the implications of an analogue implementation of neural algorithms must be investigated if efficient design is to be achieved.

There is considerable design experience for hardware implementation of neural learning algorithms, although there is still much work to be done in investigating the implications. There are a number of techniques that can be employed to carry out these designs, all of which will imply differing design considerations. The application of these designs to real world problems requires compromises. With a growing need for increased speed and smaller implementations, on-chip designs are required. Clearly, however, there is a price to be paid and more compromises, in the way of accuracy and precision, are required because of size and interconnect problems. Despite these problems, with improving fabrication and design techniques, on-chip neural hardware is becoming a reality.

3.2.4 Summary

This section has detailed some of the issues that face designers of hardware implementations of neural algorithms. It concentrates on feed-forward neural algorithms as these are the ones under consideration in this thesis. The errors due to implementation constraints and those inherent in the hardware are detailed. Issues involving implementation of full learning algorithms are also briefly discussed.

With these issues in mind, therefore, a model can now be constructed to investigate the effects of analogue hardware imprecision on neural algorithms.
3.3 An Implementation Error Model

Following the discussion in the previous section as to the important considerations when implementing a neural algorithm in hardware and software and where the likely errors will occur, this section looks at constructing a model of those errors. The model examines the hardware under consideration and represents the important errors and implementation defects that occur. In fact a number of models are used, investigating several possible errors. The resulting hardware description can then be used to predict the effects of implementation on the algorithmic performance.

3.3.1 Introduction

The aim in producing a hardware model is to be able to simulate adequately the effects of algorithm implementation on the algorithm itself. Once an accurate model has been produced the resulting simulations can be used to look at the compromises that have to be made when implementing the algorithm and assessing how they effect the performance. In some cases the results may show that the algorithm no longer performs to a reliable or even acceptable standard and so more accurate and precise implementations must be used. Hopefully, in other cases, the model may show that the hardware errors do not have a serious effect on the algorithmic performance and the implementation can proceed. The reliability of such simulations must depend on the accuracy of the initial model and so this is a critical stage in assessing implementations.

In the work contained in this thesis, the specific aim is to assess the errors introduced when implementing in-the-loop forward pass neural algorithms in analogue hardware. In particular, the issue of whether it is accuracy or precision that is of importance in iterative algorithms of this type, is of interest. Also, the aim of this work is to examine the effects of the analogue imprecision on the algorithms and learning performance metrics. The prediction of the effects on the final
solution and the learning itself will be analysed in more detail in later sections of this chapter, while here the construction of an accurate model is of primary importance.

The important features that need to be included in a full model of analogue hardware are complex and vary inevitably from implementation to implementation. However, it is possible to select a number of errors that occur and from these construct a model to look at the specific topic of interest:

1. Interfacing signal quantisation.
2. Analogue noise in the multiplication at the synapses.
3. Inaccuracy in the thresholding function.
4. Offsets in the neuron outputs.

Although these are all important issues, the second is the one of most interest in the work of this thesis as it concerns precision and accuracy. The analogue hardware model will be constructed around this implementation error. Of the other errors, the interfacing quantisation of the signals has been discussed by Hollis et al [29], who quantify the required digital resolution (number of bits) needed to retain sufficient accuracy for a number of problems. Also, the work of Hamilton et al [21], in their EPSILON chip, addresses implementation issues such as offsets in the neuron outputs, and they employ a scaling scheme called "auto-biasing" that essentially uses extra synapses to centre the neuron output on zero. There has, therefore, been some work carried out looking at implementation issues in analogue hardware, although few of them address the issue of precision constraints that is of interest here.

The important feature that is of interest, is that of noise in the multiplication at the synaptic connections and how this affects the learning in an iterative algorithm. The two variables that are used in the synaptic multiplication are the input and the weight. Noise on the inputs is something that has been studied
Chapter 3. Noise in Neural Implementations

by many researchers and has been shown to have the potential for positive effects on the generalisation performance. This was discussed in detail in the previous chapter. Here, therefore, noise in the weights is the important issue.

3.3.2 The Model

This section takes the implementation errors of interest and presents the resulting model. In particular, it looks at noise in the weights, assessing how the noise should be incorporated into the multiplication and looking at various noise distributions.

The basic model calculates the standard weight-input multiplication, but incorporates noise into the weights. i.e. the modified weight $w^*$ is some combination of the real weight $w$ and a noise component $\Delta$. The net output of the synaptic multiplication therefore, becomes some combination of input, weight and noise (see Fig. 3–2). The noise can be incorporated in different ways. Frye et al, [19], use a multiplicative model where :

$$w^* = w(1 + \Delta)$$

(3.6)

i.e. noise is added into the synapses in proportion to the size of the weight at that connection. A more obvious method would be to use purely additive noise :

$$w^* = w + \Delta$$

(3.7)

The way the noise should be modelled depends on how the individual implementation stores the weights. Here we consider both models and in later chapters discuss the differences.

The final issue that needs to be addressed, to construct the model, is the form of the noise. Here the noise is taken from a uniform distribution. As the computational overhead is significantly lower when calculating noise from a uniform distribution, this was chosen in preference over a perhaps intuitively realistic normal distribution. The majority of the work in this thesis, therefore, uses noise taken from a uniform distribution with a mean value of zero, although
A model has, therefore, been constructed to represent adequately the precision requirements of analogue implementations in neural algorithms. The errors of interest are those concerning noise in the weights at the synaptic connections.

3.3.3 Summary

This section has constructed a model of analogue hardware from a discussion of implementation issues. The model is designed to represent the errors of interest in analogue hardware implementations of in-the-loop neural algorithms, where iterative learning is carried out. These errors are those that cause imprecise calculations to occur, so that the model can be used to answer the question of whether precision is an important factor in neural algorithm implementation. By analysing the model, therefore, conclusions can also be drawn as to the effects of the noise on the learning performance metrics. These will be discussed in detail in the next section.
3.4 The Mathematical Model

In this section the restricted model of an analogue hardware implementation of a MLP neural algorithm is analysed mathematically. Using a simple expansion the model is examined and predictions are made as to the effects of the hardware imprecision on the algorithm itself.

3.4.1 Introduction

The analysis carried out here uses the learning cost function and considers the effects of noise in the weights of a forward pass on the outputs, to produce an expanded version of the function. It is important to note at this stage, that the complex terms involved are not actually used in the learning - they are artifacts of the analysis - the model itself is extremely simple and easy to implement for the simulation experiments.

The most common error term used as a cost function in MLP learning algorithms, is the mean square error between the actual output \( o_{kp} \) and the target output \( \bar{o}_{kp} \). Here the notation \( o_{kp} \) is the output of layer \( K \), in an \( I : J : K \) network, in this case for pattern \( p \) and is a function of all the network weights \( \{ w_{ab} \} \) (where \( w_{ab} \) is any weight in the network). Averaged across all the training vectors in pattern set \( P \), the standard cost function is:

\[
\epsilon_{\text{tot}} = \frac{1}{P} \times \sum_{p=1}^{P} \epsilon_p
\]

where,

\[
\epsilon_p = \frac{1}{2} \sum_{k=0}^{K-1} \epsilon_{kp}^2 = \frac{1}{2} \sum_{k=0}^{K-1} (o_{kp} \{w_{ab}\} - \bar{o}_{kp})^2
\]

i.e. \( \epsilon_{\text{tot}} \) is the total error, \( \epsilon_p \) the error for pattern \( p \) and \( \epsilon_{kp} \) the difference between the actual and target output of neuron \( k \).

Taking this simple error function and expanding the output \( o_{kp} \) to include terms that are introduced by the noise in the weights, the resulting cost function
can be analysed. The extra terms that are included clearly have implications for the error value being minimised, either causing a net increase or a net decrease. The former will be favoured by the learning process or at least stabilised, while the later will be penalised or at least de-stabilised. By analysing the extra terms introduced, therefore, predictions can be made as to the effect of the noise on the learning itself and hence the problem solution.

### 3.4.2 A Cost Function With Injected Noise

The mathematical analysis is summarised in the text below. The boxed figures Fig. 3–3 and Fig. 3–4, give more of the detail of the derivation, which is also reported in [52]. Also to avoid confusion, here only the multiplicative noise case is considered and the solution is generalised in later discussions.

To produce the new cost function with the injected noise terms included, weights $w_{ab}$ are augmented by a random noise source, such that $w_{ab} \rightarrow w_{ab} + \Delta w_{ab}$ (for the multiplicative noise case (3.6)), for the entire weight set $\{w_{ab}\}$ (input-hidden and hidden-output weights). Neuron bias terms are treated in precisely the same way. The cost function expansion is carried out by calculating a Taylor series of the output $o_{kp}$ to second order around the noise-free weight set $\{w_{nominal}\}$, as follows:

$$o_{kp} \rightarrow o_{kp} + \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) + \frac{1}{2} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) + O(\Delta^3) \quad (3.9)$$

Substituting this back into (3.8), dropping terms of order $\Delta^3$ (see Fig. 3–3) and noting that:

$$< \Delta_{ab} \Delta_{cd} >= \Delta^2, \quad < \Delta_{ab} >= 0 \quad (3.10)$$

(i.e. the noise sources are uncorrelated and assuming the case of zero mean noise sources), the cost function becomes:

$$< e_p > = < \epsilon_p(\{w_{nominal}\}) > + \frac{1}{2} \sum_{k=0}^{K-1} \sum_{ab} w_{ab}^2 \Delta^2 \left[ \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right)^2 + \epsilon_{kp} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab}^2} \right) \right] \quad (3.11)$$

i.e. the error on the pattern $p$ is given by the nominal error on the standard noise-free weight set (the normal mean square error), plus two extra terms. Averaging
(3.11) over the entire pattern set (see Fig. 3-4), the extra terms in the cost function become:

\[
\Delta^2 \sum_{ab} \left[ \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=0}^{K-1} w_{ab}^2 \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right)^2 \right]
\]

and

\[
\Delta^2 \sum_{ab} \left[ \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=0}^{K-1} \epsilon_{kp} w_{ab}^2 \left( \frac{\partial^2 o_{kp}}{\partial w_{ab}^2} \right) \right]
\]

Expanding on the text above, substituting (3.9) into (3.8), the error signal \( \epsilon_p \) is given by:

\[
\epsilon_p = \frac{1}{2} \sum_{k=0}^{K-1} \left[ \epsilon_{kp} + \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) + \frac{1}{2} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) - \partial_{kp} \right]^2
\]

\[
\epsilon_p = \frac{1}{2} \sum_{k=0}^{K-1} \left[ \epsilon_{kp} + \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) + \frac{1}{2} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) \right]^2
\]

Which, dropping any terms of order \( \Delta^3 \) and above, gives:

\[
\epsilon_p = \frac{1}{2} \sum_{k=0}^{K-1} \left[ \epsilon_{kp} + \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) + \frac{1}{2} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) \right]
\]

Or

\[
\epsilon_p = \epsilon_p(\{w_{\text{nominal}}\}) + \frac{1}{2} \sum_{k=0}^{K-1} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left[ \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) \left( \frac{\partial o_{kp}}{\partial w_{cd}} \right) \right] + \epsilon_{kp} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) + \frac{1}{2} \sum_{k=0}^{K-1} \left[ 2 \epsilon_{kp} \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) \right]
\]

Figure 3-3: Details of the mathematical derivation between (3.9) and (3.11).
Averaging (3.11) over all the \( P \) input patterns \( \{o_p\} \), gives a value for \( < \epsilon_{\text{tot}} > \) of:

\[
< \epsilon_{\text{tot}} > = \frac{1}{P} \sum_{p=1}^{P} < \epsilon_p > = < \epsilon_{\text{tot}}(\{w_{\text{nominal}}\}) > + \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=0}^{K-1} \Delta^2 \sum_{ab} w_{ab}^2 \left[ \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right)^2 + \epsilon_{kp} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab}^2} \right) \right] \quad (3.18)
\]

**Figure 3-4**: Details of the mathematical derivation between (3.11) and the two extra cost function terms, (3.12) and (3.13).

For the additive noise case the weights are augmented \( w_{ab} \rightarrow w_{ab} + \Delta \) and hence, following the same procedure as with the multiplicative noise case, the extra error terms are the same except without the \( w_{ab}^2 \) term in each.

The basis of the analysis, therefore, centres on the alteration of the cost function by the noise terms. The two terms that are introduced, (3.12) and (3.13), can adjust the error that is minimised during the learning process. The implications of these terms, how they change the cost function error, and the resultant effect on the learning trajectory and the final solution, will be discussed in detail in the following section and in later chapters.

### 3.4.3 Predictions

In this section the expanded cost function with its extra *penalty* terms are examined to predict their individual effect on the learning performance. In particular the noise effects are assessed using the standard performance measures:

- Fault Tolerance
- Generalisation Ability
- Learning Trajectory and Speed
By examining exactly how the two extra terms increase or decrease the error, predictions can be made to indicate the expected performance implications for the noise model on the neural algorithm in each of these three performance areas.

Fault Tolerance

Considering first the effect of the noise on the fault tolerance performance, the relevant term in the expanded cost function is (3.12), the first derivative term, which averaging over all patterns, output neurons and weights becomes:

$$A\Delta^2 w_{ab}^2 \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right)^2$$

where $A$ is some constant.

This term incorporates the noise ($\Delta^2$), the individual weight magnitude ($w_{ab}^2$) and the derivative of the output ($o_{kp}$) with respect to the weight, squared. The derivative defines the dependence of an individual output to an individual weight and hence, when the term is included in the cost function, it is minimised along with (3.8). By minimising the derivative, for significant weight magnitudes, individual weights become less important for a particular output and so a much more distributed representation of the problem is seen. From the argument raised in the chapter 2, if the problem solution is stored as information in a small proportion of the total number of weights then these will be susceptible to damage. If, however, the information is spread across the whole network with each weight contributing a small amount of information to the solution, then the network will be more tolerant to the removal or damage of one of its component parts. Crudely, any problem mapped into a network architecture requires an aggregate level of weight-output dependence. Where, in standard BP, usually very few weights are significant (i.e. the derivative is high for a few weights), here (3.19) will favour solutions where many weights have small derivatives. This will become manifest in an improved network fault tolerance as removing or damaging an individual weight will be less likely to have a catastrophic effect on the network as a whole. The noise term (3.12), therefore, has a positive effect in this case.
In chapter 5 fault tolerance will be measured by damaging the network according to some fault model and also by measuring the individual weight significance (or saliency in the language of [43]) directly, and the standard deviation across the network.

**Generalisation Ability**

The effect of noise on the second performance metric can also be analysed using (3.19). By looking at the internal representation directly, it should be possible to gauge the quality of the mapping. Consider the derivatives:

\[ \frac{\partial o_{kp}}{\partial w_{ab}} = \frac{\partial o_{kp}}{\partial w_{kj}} \quad \text{and} \quad \frac{\partial o_{kp}}{\partial w_{ji}} \quad (3.20) \]

i.e. the derivatives of the output of layer \( K \) with respect to the hidden-output weights and the input-hidden weights. The first is relatively straightforward, the term added being:

\[ B \Delta^2 w_{kj}^2 o_{kp}^2 o_{jp}^2 \quad (3.21) \]

where \( B \) is some constant.

This term (see Fig. 3-5 for the derivation) penalises solutions where the hidden node outputs are firing (i.e. \( o_{jp} \neq 0 \)) and the derivative of the output neurons is non-zero. In Fig. 3-6 the sigmoid function (the standard “squashing function”) and its derivative are shown, illustrating how the derivative is non-zero on the sloping section. Solutions will, therefore, be favoured where the hidden node states are firmly ON or OFF. For most classifier problems this will be a criterion imposed by the targets and so the term introduces no significant difference from the standard cost function.

Looking at the derivative with respect to the input-hidden weights, the term added to the cost function (see Fig. 3-7 for the full derivation) is:

\[ C \Delta^2 w_{ji}^2 o_{kp}^2 w_{kj}^2 o_{jp}^2 o_{jp}^2 \quad (3.24) \]

where \( C \) is some constant.
Expanding the output derivative with respect to the hidden-output weights:

$$\frac{\partial o_{kp}}{\partial w_{kj}} = \frac{\partial o_{kp}}{\partial \text{net}_{kp}} \frac{\partial \text{net}_{kp}}{\partial w_{kj}} \tag{3.22}$$

i.e. using the chain rule and noting that \(\text{net}_{kp}\) is the activation level for neuron \(k\) and pattern \(p\). Expanding out the activation term in the second partial derivative,

$$\frac{\partial o_{kp}}{\partial w_{kj}} = o'_{kp} \frac{\partial (\sum_j w_{kj} o_{jp})}{\partial w_{kj}} = o'_{kp} o_{jp} \tag{3.23}$$

i.e. the derivative is only non-zero when the weight in the summation is equal to the one with which it is being differentiated with respect to.

**Figure 3-5:** The derivation of the network output with respect to the hidden-output weights.

**Figure 3-6:** The ubiquitous sigmoid squashing function and its derivative.
Expanding the output derivative with respect to the input-hidden weights gives:

\[
\frac{\partial o_{kp}}{\partial w_{ji}} = \frac{\partial o_{kp}}{\partial \text{net}_{kp}} \frac{\partial \text{net}_{kp}}{\partial w_{ji}} \tag{3.25}
\]

i.e. using the chain rule. Also, expanding the activation term and applying the chain rule for a second time:

\[
\frac{\partial o_{kp}}{\partial w_{ji}} = o_{kp} \frac{\partial (\sum_j w_{kj} o_{jp})}{\partial o_{jp}} \frac{\partial o_{jp}}{\partial w_{ji}} \tag{3.26}
\]

Using a similar argument to that used in (3.23) and a third use of the chain rule:

\[
\frac{\partial o_{kp}}{\partial w_{ji}} = o_{kp} w_{kj} \frac{\partial o_{jp}}{\partial \text{net}_{jp}} \frac{\partial \text{net}_{jp}}{\partial w_{ji}} \tag{3.27}
\]

i.e. the derivative is only non-zero when the output in the summation is equal to the one with which it is being differentiated with respect to. Using the result of (3.22) and (3.23), therefore, the derivative is expanded to:

\[
\frac{\partial o_{kp}}{\partial w_{ji}} = o_{kp} w_{kj} o_{jp} o_{ip} \tag{3.28}
\]

**Figure 3–7:** The derivation of the network output with respect to the input-hidden weights.
Here again $o'_{kp}$ is introduced with the same minimal effect, as explained above, being noted. In addition, however, when the connecting weights and the input $o_{ip}$ are non-zero, a solution will be favoured where the hidden node states are firmly ON or OFF, thus minimising the derivative $o'_{jp}$. This hidden node state, where solutions having outputs on the slope of the sigmoid are de-stabilised, will have the effect of giving robust decision boundaries. With a robust decision boundary small variations in the activation will have less effect on the neuron output than if it was on the sloping section of the sigmoid curve. Bolt, [9], artificially changes weights to install this internal representation on his networks giving them an immunity to small changes in the weights, i.e. enhanced fault tolerance. Here it is also noted that the robust neurons will be inherently more stable against small variations in the input data. Also, constraining the hidden nodes to a more binary representation reduces the potential storage capacity and, following the argument outlined in chapter 2, this could be evidenced by an improved generalisation performance. Improved generalisation ability has been demonstrated experimentally using noise injected into the synapses in [49], on classification problems. On problems where analogue outputs are required, the noise will obviously be less helpful and there may be competition between reducing the error and installing robust internal states. Therefore, for networks performing applications where analogue outputs are required, the acceptable noise level may be significantly lower.

Chapter 6 quantifies the effects of noise on the weights on the generalisation performance metric. By measuring the hidden node output derivatives directly the effects of the noise can be seen. By considering two classifier problems, the predictions made above are verified and the limitations of the improvements expected explored. Again the noise on the weights is seen to be helpful for improving the final solution for many problems, although for some the effects may be more benign or even destructive.
Learning Trajectory and Speed

The final performance metric can be assessed using the second term added to the cost function (3.13), the second derivative term, which again averaging over all patterns, output neurons and weights becomes:

\[ D \Delta^2 \epsilon_{kp} w_{ab}^2 \left( \frac{\partial^2 \epsilon_{kp}}{\partial w_{ab}^2} \right) \]  

(3.29)

where \( D \) is some constant.

The analysis of (3.29) is far from simple as it contains a second derivative term and also the error \( \epsilon_{kp} \), both of which can be positive or negative. While the first derivative term (3.12) always increases the error (always being positive) this term can be positive or negative, thus increasing or decreasing the error. This implies that when \( \epsilon_{kp} \) is negative (\( o_{kp} \) too small in (3.8)), solutions with the second derivative also negative in (3.29), will be de-stabilised since the product of the two terms will increase the cost function error. Similarly, when \( \epsilon_{kp} \) is positive (\( o_{kp} \) too large), solutions where the second derivative is also positive in (3.29), will be de-stabilised.

To aid the understanding of this rather nebulous idea, consider Fig. 3-8 where a fictitious output-weight graph has been sketched, along with the first and second derivatives. From the previous argument a negative value for the error \( \epsilon_{kp} \) will be a solution where a positive second derivative is favoured. In Fig. 3-8 it is shown that for an increase in \( o_{kp} \) then a positive second derivative will occur (labelled (i) on the diagram). The noise-induced second derivative will, therefore, encourage \( o_{kp} \) to increase, thus reducing \( \epsilon_{kp} \). Conversely, if the error \( \epsilon_{kp} \) is positive then a negative second derivative will be favoured, a condition (labelled (ii) on the diagram) reducing \( o_{kp} \) and thus \( \epsilon_{kp} \).

This second derivative term is, therefore, a constructive term that can actually reduce the error via noise injection. The term (3.13) can be seen as a “sculpting of the error surface” in the early stages of training when \( \epsilon_{kp} \) is large. Using the second derivative curvature information a “look-ahead” property is introduced that can favour a weight change that, although it increases the raw error (3.8),
Figure 3-8: Conceptualised exemplar output-weight dependence graph, with first and second partial derivatives.
indicates a move to a more promising area via the noise injected terms. This
could lead to an enhanced learning trajectory and possibly the faster learning
already demonstrated experimentally, using noise in the weights, in [49]. This
behaviour will be prominent in the early stages of training, when the error is
substantial, and as $\epsilon_{kp}$ is reduced the term will automatically remove itself.

It is also illuminating to examine the actual weight changes imposed by the
perceptron training rule, and the effect of the second derivative terms in $\epsilon_{kp}$ on
these changes. For example, a perceptron-rule update on the hidden-output layer
would yield:

$$w_{kj} \rightarrow w_{kj} + \delta w_{kj} = w_{kj} - \eta \sum_p \epsilon_{kp} o_{jp} o'_{kp}$$

(3.30)
i.e. the standard perceptron update, where $\eta$ is the adaptation rate. Substituting
in the expanded error equation, gives:

$$< \delta w_{kj} > = -\eta \sum_p < \epsilon_{kp} o_{jp} o'_{kp} > - \eta \sum_p < o_{jp} o'_{kp} > \times \sum_{ab} w_{ab}^2 \frac{\partial^2 o_{kp}}{\partial w_{ab}^2}$$

(3.31)
i.e. the standard weight update (the first term) and also a second term containing
second derivative information. See Fig. 3-9 for the full derivation.

It is not immediately obvious what effect this extra term has on the weight
update and the resultant error. However, deductions can be made. The final term,
the weight squared and the second derivative, represents an aggregate value over
all of the weights and thus is an indicator of what is happening to the output,$o_{kp}$, due to the weights as a whole. From Fig. 3-8, it can be deduced that if the
term is positive then an increase in $o_{kp}$ due to the entire weight set will occur
and vice versa. The extra term, therefore, projects statistical information onto
the weight update equation. In particular, we can state, from (3.31), that:

1. If $\epsilon_{kp}$ is positive ($o_{kp}$ too large) the weight change from (3.30) will be negative.

If the second derivative term in (3.31) is also positive, it will tend to augment
this change. This is helpful, because the sign of the second derivative term
suggests that the net effect of other weight changes is likely to increase
the output $o_{kp}$, and a larger weight decrement than (3.30) is required to
compensate.
Taking the standard perceptron rule weight update:

\[ w_{kj} \rightarrow w_{kj} + \delta w_{kj} = w_{kj} - \eta \sum_p \epsilon_{kp} o_{jp} o'_{kp} \]  

(3.32)

i.e. considering a single layer perceptron network with \( J \) inputs, \( K \) units and an adaptation rate parameter \( \eta \). Substituting in the expanded error function (3.15), gives:

\[
\delta w_{kj} = -\eta \sum_{p'} \left[ \epsilon_{kp} + \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial o_{kp}}{\partial w_{ab}} \right) + \frac{1}{2} \sum_{ab, cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left( \frac{\partial^2 o_{kp}}{\partial w_{ab} \partial w_{cd}} \right) \right] \times o_{jp} o'_{kp} 
\]

(3.33)

Averaging over several training epochs (acceptable for small values of \( \eta \)) gives:

\[
<\delta w_{kj}> = -\sum_p \eta \left[ <\epsilon_{kp} o_{jp} o'_{kp}> + \frac{1}{2} \sum_{ab} w_{ab}^2 \Delta^2 \left( \frac{\partial^2 o_{kp}}{\partial w_{ab}^2} \right) <o_{jp} o'_{kp}> \right] 
\]

(3.34)

or

\[
<\delta w_{kj}> = -\eta \sum_{p'} <\epsilon_{kp} o_{jp} o'_{kp}> - \eta^2 \sum_{p} <o_{jp} o'_{kp}> \times \sum_{ab} w_{ab}^2 \frac{\partial^2 o_{kp}}{\partial w_{ab}^2} 
\]

(3.35)

i.e. a noise mediated perceptron update rule.

**Figure 3-9:** Details of the mathematical derivation of the noise mediated perceptron update equation.
2. If $\epsilon_{kp}$ is negative ($o_{kp}$ too small) the weight change from (3.30) will be positive. If the second derivative term in (3.31) is also negative, it will tend to augment this change. The sign of the second derivative term suggests that the net effect of other weight changes is likely to decrease the output $o_{kp}$, and a larger weight increment than (3.30) is required.

On the other hand :-

3. If $\epsilon_{kp}$ is positive ($o_{kp}$ too large) the weight change from (3.30) will be negative. If the second derivative term in (3.31) is negative, it will tend to reduce this decrement. This is also helpful, because the sign of the second derivative term suggests that the net effect of other weight changes is likely to decrease the output $o_{kp}$, and a smaller weight decrement than (3.30) will suffice.

4. If $\epsilon_{kp}$ is negative ($o_{kp}$ too small) the weight change from (3.30) will be positive. If the second derivative term in (3.31) is positive, it will tend to reduce this increment. The sign of the second derivative term suggests that the net effect of other weight changes is likely to increase the output $o_{kp}$, and a smaller weight decrement than (3.30) will suffice.

So, the effect of the noise term in (3.31) is to take account not only of the weight currently being updated, but to add in a term that estimates what the other weight changes are likely to do to the output, and adjust the size of the weight increment/decrement as appropriate.

From this analysis, improvements in the learning trajectory are predicted. Noise in the weights has already been shown to improve learning times experimentally, [49], on a classification problem. Here the mechanisms as to how this happens are examined. In chapter 7, the predicted effects discussed above will be considered by examining simulation results. By considering the training time required and the number of failures that occur, verification experiments will be carried out. The improvements predicted will, therefore, be quantified and the limits noted, affirming the results in [49]. The learning enhancements predicted
here, although secondary to other solution enhancements, show that noise on the weights has a positive effect.

3.4.4 Summary

Using the hardware model described in the previous section, a mathematical analysis has been carried out as to the effects of noise in the weights on the learning performance and the quality of the final solution. Using a simple expansion it has been inferred that extra terms are introduced into the cost function and by considering the affect of these on the error, predictions have been made in the three performance metric areas. The predictions have been made using the multiplicative noise model, where noise is added in proportion to the actual magnitude of the weights. The effects of purely additive noise have not yet been discussed, although they will be related. In later chapters it will be shown that similar performance enhancements can be expected.

3.5 Chapter Summary

The aim of this chapter was to produce and analyse a model of hardware implementations of neural algorithms, to allow simulation of the effects of the precision constraints on those algorithms. From a diversity of possible implementation inaccuracies and imprecisions, those affecting analogue hardware for feed-forward, in-the-loop implementations were chosen. In particular, a model looking at the effect of noise in the weights on the learning and learned solution was constructed. Having formed a model that is easily implementable in software for the simulation experiments, a mathematical analysis was carried out to analyse the possible effects of the noise on the main neural network performance metrics. Using an expanded version of the learning cost function, with noise induced terms, predictions have been made as to the effects of the noise. Performance enhancements
are predicted in fault tolerance, generalisation ability and learning trajectory and speed.

This chapter has predicted, therefore, that instead of the performance being limited by the analogue imprecision, positive enhancements will be seen. The extent of these predictions and the effects of the imprecision on iterative learning will be explored in later chapters, verifying and discussing the implications in detail.
Chapter 4

Simulation Requirements and Environment

It was then that I learned that a graduate student is less an object of suspicion than an undergraduate. Those were years when defending a thesis was considered evidence of respectful loyalty to the state, and you were treated with indulgence.

_Umberto Eco, Foucault’s Pendulum_

4.1 Introduction

This chapter describes a simulation environment and a set of requirements appropriate for testing the predictions of the previous chapter. Its objective is to provide a strategy for verifying these predictions and allowing an understanding of the underlying mechanisms to be gained. In addition, it selects a set of test problems that will allow a full testing to take place and the limitations of the technique to be probed. This chapter therefore presents an environment where the neural algorithms and imprecision model can be simulated, and a verification strategy in which the test requirements can be satisfied.

Chapter 3 constructed a model for analysing the effects of analogue hardware imprecision on MLPs. It also presented predictions on the effects of this im-
precision on the three basic performance metrics, using a simple mathematical expansion. The issues involved in assessing network performance in these areas have also been discussed in chapter 2, giving a background for the verification experiments and also acting as a guide for the requirements of the test strategy.

By examining the predictions made earlier and the background work discussed in chapter 2, the simulation requirements are assessed. Using this assessment, a database of test problems is compiled to place the simulation results in the context of the current literature. In this way a comparison can be drawn between the performance enhancements seen in the literature and the results of the prediction verification experiments carried out during the course of this project. The simulation environment is also discussed in this chapter providing a description of the algorithms and techniques used.
4.2 Simulation Requirements

This section discusses the requirements of the problem database to allow a comparison of the verification experiment results with the performance enhancements seen in the current literature. By considering each of the three performance metrics and how the precision limitations of the hardware model will effect the network solutions, this section forms an assessment of the test problem requirements. In particular, the problems will be required to allow an analysis of the mechanisms underlying the enhancements seen, and also the limitations of the technique as an enhancement scheme. Problems fulfilling these considerations are presented and discussed below. These problems will thus form a database on which experiments can be carried out to verify the predictions made in the previous chapter.

4.2.1 Introduction

The choice of a test problem is an important task when assessing the performance of a new algorithm or learning technique. In the work of this thesis the effects of the hardware imprecision model and subsequent performance enhancement technique are explored in the context of each of the three performance metrics. To analyse the enhancements predicted a set of test problems is required. Using the problem database, prediction verification experiments can be carried out.

The verification experiments, and the test problems on which they are carried out, have three aims. These are listed below:

- The first, simply, is to assess the validity of the basic prediction. To form this assessment, a problem that has a small pattern set and one that can be solved in few training epochs, is required. This type of problem allows a large number of simulations to be carried out, giving the results statistical significance.
The second aim is to carry out an analysis of the underlying enhancement mechanisms. The specific problem that will allow this will vary from one metric to another - one perhaps requiring a simple problem, another a more complex one to stretch the network’s capability.

The third aim is to probe the limitations of the technique. Again the type of problem required depends on the particular metric, but in general is “hard”.

The three sections below briefly examine the individual performance metrics to illustrate the type of problems required to satisfy the aims of the tests, described above. The discussion takes into account the current literature and the predictions themselves.

**Fault Tolerance**

To test fault tolerance performance, a weight configuration can be “damaged” according to some fault model. A comparison of the network performance on a test data set, with and without damage, gives a relative indication of the tolerance to faults. Therefore, obtaining an indication of the performance of a fault tolerance enhancement scheme is not dependent on the test problem being used.

The fault-tolerance enhancement prediction made in chapter 3 centred on the prediction that the synaptic weight noise would give the learned solution a highly distributed representation. The test problem required to investigate the underlying mechanisms of the fault tolerance enhancements must, therefore, be able to verify the nature of the internal representation. Bolt in his thesis [9], uses the fact that the extremes of the sigmoid curve in the hidden and output layer units can be used to imbue the trained network with an improved tolerance to activation variation. Here this effect is also predicted, noting that it will give greater tolerance to weight errors. It is therefore important for the test experiments to be able to differentiate between an improved tolerance due to a more distributed representation and one due to the use of the sigmoid extremes.
However, the experiments detailed in chapter 5 show that the test problem in this case is not critical.

To test the limitations of the improvements to fault tolerance, a hard problem must be used where all the network weights are required to solve the problem. This will reduce any redundancy naturally occurring in the network architecture.

**Generalisation**

Generalisation performance testing is critically dependent on the individual problem in question. Radically different results are seen for different types of problem. Although artificial problems allow tests to be carried out to determine whether noise on the input data affects the network output state, the value of this type of experiment is limited. To assess the basic generalisation performance of an enhancement method, "real world" data is required to test the network on previously unseen data patterns.

The enhancement predictions in this performance area deal with the computational capacity of the network and the required capacity of the problem. It is therefore necessary to have a problem where the required capacity is known, or one of sufficient complexity that the information cannot be completely encapsulated by the network during training. This second option would allow the required capacity to be 100% of the network capacity plus some extra amount proportional to the training error $\epsilon_t$. The change in computational capacity of the network can be measured indirectly, as the synaptic weight noise is injected by measuring the change in the $\epsilon_t$. This rather complicated idea is explained in more detail in chapter 6, while here it is important simply to note that a complex problem is required to analyse the mechanism of generalisation ability enhancement. A problem of this type also exposes the limitations of the technique.
Learning Trajectory and Speed

Testing the synaptic weight noise technique as a means of improving the learning trajectory and speed is a computationally intensive task involving many test problems and comparative work with other algorithms. The study in this thesis concentrates on showing that the hardware imprecision model does not have a detrimental effect on the learning performance at low noise levels, but can have positive effects. To do this, a test problem is required that is non-trivial, but also allows a significant number of trials to be carried out at varying precision levels to make the results statistically valid.

The predictions made in the previous chapter concerning learning trajectory and speed deal with second order information. Therefore, they are very complicated to verify in detail. It is, however, possible to measure the learning speed for a given problem as described above. In addition, tests can be carried out "tracking" the second derivative numerically to gauge the effects of the noise. This sort of experiment is not critically dependent on the problem.

It is also difficult to test the limitations of the enhancements seen during the course of this project. While it is obvious that, eventually, excessive amounts of noise in the learning will have a detrimental effect on the training and therefore speed of convergence, it is hard to know how different problems will be affected. To assess variations in the effect of synaptic noise a problem is required where, perhaps, it is necessary to have a higher degree of accuracy to find a solution. The experiments with a problem of this type should allow a judgement to be made as to whether it is accuracy or precision that is required in neural network algorithms.

To summarise the above discussion, a set of test problems is required that will test the performance enhancement predictions fully. Each of the performance metrics requires specific types of problems to allow the full assessment of the effect of the model of hardware imprecision. The sections below describe a problem database that fulfils the specifications of the tests stipulated above.
4.2.2 Problem Database

This section describes the set of test problems in the problem database that are used in the testing and analysis of the synaptic weight noise performance enhancement predictions. They are discussed giving the details of the network architecture and learning parameters that are used during the training process.

Character Encoder

The character encoder problem is an artificially generated, binary input, binary output, and yet non-trivial problem. The inputs, a grid of 5x5 pixels, are used to define a 25-dimensional vector describing the 26 alphabetic characters. The problem is to map these to a one-out-of-26 output coding, (see Fig. 4-1).

![Input Patterns](image)

![Output Targets](image)

Figure 4-1: The character encoder task.
This problem, with its small training pattern set, can be solved relatively quickly and so allows many simulations to be carried out to form a statistically significant set of results. The solutions were obtained using the virtual targets algorithm [49], a back-propagation derivative. This algorithm and the reasons for its use are discussed in section 4.3.3. The parameters used to solve the problem are given in Table 4–1.

<table>
<thead>
<tr>
<th>Hidden Layer Neurons $J$</th>
<th>Initial Weight Range $\beta$</th>
<th>Adaptation Rate $\eta$</th>
<th>Momentum $\alpha$</th>
<th>Sigmoid Temperature $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>$\pm 1$</td>
<td>0.4</td>
<td>0.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 4–1: Training parameters for the character encoder problem, as used in all the simulations.

**Eye/Not-eye classifier**

The eye/not-eye classifier is a feature location task using real world normalised grey-scale image data [70]. The task is to locate eyes in facial images - to class these as either an “eye” or a “not-eye” feature (see Fig. 4–2). The two binary outputs are labelled eye and not-eye. The network is trained on 16x16 sections of the images (approximately the size of an eye), classified as eyes and not-eyes. The not-eyes are random sections of the facial images avoiding the eyes.

![16x16 section](image)

"eye"

"not-eye"

Figure 4–2: An idealised view of the eye/not-eye classifier problem.
This problem has a very high level of redundant information in the input vector and the classification task is, therefore, not easy. To solve the problem intelligently the data would obviously be pre-processed to reduce the dimensionality. However using it in this "raw" state does allow true generalisation tests to be carried out on previously unseen facial images. Also, because the input data comprise of analogue values in the range $0 \rightarrow 1$, the accuracy of the classification required is increased. Again the virtual targets algorithm was used to obtain the solutions, where the parameters given in Table 4–2.

<table>
<thead>
<tr>
<th>Hidden Layer Neurons $J$</th>
<th>Initial Weight Range $\beta$</th>
<th>Adaptation Rate $\eta$</th>
<th>Momentum $\alpha$</th>
<th>Sigmoid Temperature $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>±1</td>
<td>0.4</td>
<td>0.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 4–2: Training parameters for the eye/not-eye classifier, as used in all the simulations.

Localisation Problem

The "localisation problem"\(^1\) is an artificial classification problem based on mobile robot navigation. The navigation takes place in an idealised L-shaped room, with six corners and two rectangular obstacles (see Fig. 4–3). The input data is a set of eight features extracted from a $360^\circ$ scan of the room. The eight features are as follows:

- The shortest ray length.
- The median ray length.
- The longest ray length.

\(^1\)Many thanks to Lionel Tarassenko at Oxford University for the use of this problem data set.
Figure 4–3: An idealised view of the L-shaped room in the localisation problem.

- The energy\(^2\) in the pattern.
- The size of the two largest discontinuities in the pattern.
- The angle subtended by the longest wall segment.
- The perpendicular distance from the longest wall segment.

These eight features are then mapped to a one-out-of-6 coding of the nearest corner.

This problem is extremely hard, requiring a significant number of hidden layer neurons. As the problem is artificial (i.e. has no errors, unless specifically introduced) and an unlimited number of training patterns are available, a complete solution can be found. However, by limiting the number of patterns, only a partial solution is obtainable. In the simulations described below, 500 patterns are used in training and a further 500 as a validation pattern set, to stop adaptation

\(^2\)The pattern energy is a measure of the “brightness” of the reflection, i.e. how much of the signal is reflected.
Chapter 4. Simulation Requirements and Environment

of the weights at a minimum generalisation error $\epsilon_g$. Full details of the problem are given in [71] and the techniques developed to halt training are discussed in section 4.3.4. Table 4–3 presents the parameters used during learning which was carried out using the back-propagation algorithm.

<table>
<thead>
<tr>
<th>Hidden Layer</th>
<th>Initial Weight</th>
<th>Adaptation Rate $\eta$</th>
<th>Momentum $\alpha$</th>
<th>Sigmoid Temperature $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neurons $J$</td>
<td>$\pm 0.1$</td>
<td>0.2</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4–3: Training parameters for the localisation problem, as used in all the experiments unless explicitly stated.

Henon Map - TSP Problem

The Henon map Time Series Prediction (TSP) problem, [26][19], is also a hard problem. This problem takes as its input a window of a two-dimensional time series (i.e. $x(t)$ and $y(t)$ at several values of $t$) and as a target a prediction of the subsequent pair of values for $x(t)$ and $y(t)$. The series is defined by the following equations:

$$x(t + 1) = y(t) + 1 - 1.4x(t)^2$$  \hspace{1cm} (4.1)  

and,

$$y(t + 1) = 0.3x(t)$$  \hspace{1cm} (4.2)  

For initial conditions of $x(0)$ and $y(0)$ near unity, the series exhibits chaotic behaviour and, therefore, conventional linear prediction techniques are incapable of solving the problem. Fig. 4–4 shows the locus of $x(t)$ against $y(t)$ for the target values of the output.

This problem is chosen to find the limitations of learning using the hardware imprecision model and hence some measure of the required accuracy and precision of calculation. The parameters used to define the series and those used during training are given in Table 4–4 and Table 4–5 respectively. The window length
Figure 4-4: A locus plot of $x(t)$ against $y(t)$ for the Henon map problem.

parameter states how much of the series is used at the input. With a 6 input window for $x(t)$ and a 6 input window for $y(t)$, there are 12 inputs. As with the localisation problem the back-propagation algorithm was used to obtain the solutions. The reasons for this are discussed in section 4.3.3.

<table>
<thead>
<tr>
<th>$x(0)$</th>
<th>$y(0)$</th>
<th>Window Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.95</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4-4: Parameters defining the Henon map series used to generate the training patterns.

4.2.3 Summary

This section has discussed the specification of the problem database that is used to test the predictions made in chapter 3. The results from the verification experiments carried out on these problems are reported in later chapters, which present
Chapter 4. Simulation Requirements and Environment

<table>
<thead>
<tr>
<th>Hidden Layer Neurons $J$</th>
<th>Initial Weight Range $\beta$</th>
<th>Adaptation Rate $\eta$</th>
<th>Momentum $\alpha$</th>
<th>Sigmoid Temperature $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$\pm 0.1$</td>
<td>0.05</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4-5: Training parameters used during training in all the simulations using the Henon map problem.

4.3 Simulation Environment

This section describes the simulation environment in which all the prediction verification experiments were carried out. The environment varies slightly for each of the four test problems and so the differences are discussed. Issues involving the use of noise in learning are also raised and techniques for stopping the learning are introduced.

4.3.1 Introduction

It is important to note from the start that the levels of noise used during most of the experiments (for the multiplicative noise case at least) were far higher than would be expected in a genuine hardware-realised system. As the experiments were carried out, the predicted enhancements were seen to increase with the level of the noise used for most, if not all of the test problems. Therefore, the noise levels used were much greater than those that would have been used in experiments merely to investigate the hardware implications of the imprecision model. The use of noise levels, such as a multiplicative level of 40%, also exceeds the levels in which assumptions, made in the Taylor expansion, hold. The detailed
predictions at this level of noise, and indeed levels significantly below this, are invalid. However, the performance improvements are still seen.

The basic environment was that of a MLP network with one hidden layer of non-linear units. For the three classification problems a layer of non-linear output units were incorporated, while for the Henon Map problem the outputs were linear. The networks were trained under the influence of noise using standard algorithms. The details of the noise, learning algorithms, stopping criteria and other issues are presented in the following sections.

4.3.2 Noise Generation

In chapter 3 the noise was stated as being taken from a uniform distribution. In simulation, this was carried out, using a random number generator producing numbers with equal probability in the range from +1 to -1. For the multiplicative noise model this value was then multiplied by the weight in question and a noise level term. In this case the noise level can be easily defined as a percentage of the weight's value itself. However, in the additive noise simulations the level is an absolute number unconnected to the weight value. The noise level is, therefore, hard to determine in real terms. In these simulations it is defined as a percentage of the mean absolute weight value in the initial weight distribution. To clarify the above descriptions, a multiplicative noise level of 10% on a weight of value 1.0, implies that in the forward pass the weight can take any value in the range of 0.9 to 1.1 with uniform probability. For additive noise of level 10% on a weight taken from a initial weight distribution $\beta$ of $\pm 1$ (i.e. a mean absolute value of 0.5, as the initial weights are also taken at random from a uniform distribution), a weight of 1.0 can vary in the range from 0.95 to 1.05 in the forward pass.

In the learning phase, weights are corrupted by a defined amount of noise. For full convergence to a good solution to take place it is usually necessary to reduce the noise level as the weights approach a solution. In the following simulations, where the problem is a classification one and a complete solution expected (i.e. the character encoder and the eye/not-eye classifier), the noise is reduced when
the maximum bit error is less than 0.4. The maximum bit error - $mbe$, as opposed to the mean square error - $mse$ (used in the cost function), is defined as the largest single error occurring at any one output neuron. When this mbe level is reached, the noise is reduced exponentially to a minimum value of 1%.

### 4.3.3 Learning Algorithm

It transpires that the precise choice of learning algorithm is not critical for training the noise-affected networks. Initially the virtual targets algorithm was chosen to train networks on all the test problems. It was chosen in preference to the more standard back-propagation because of its ability to "climb hills" to escape from local minima and also as it had already been used in the earlier work with synaptic noise [48]. However, while its performance was satisfactory for the character encoder problem and the eye/not-eye classifier, the other two "harder" problems were solved more consistently by networks trained with back-propagation. The back-propagation algorithm gives a much closer approximation to gradient descent on the error surface, even when used in stochastic mode, (adapting the weights after each pattern presentation, rather than only after the presentation of the whole pattern set) as was the case here.

The change in the learning algorithm raises the question of the validity of any comparison between results. Results obtained from a problem solved using back-propagation cannot be directly compared with results of problems trained with the virtual targets algorithm for that comparison to hold up in any scientific manner. However, during the course of the work carried out for this thesis the direct comparison of various networks is only carried out between those trained with and without synaptic noise included, on the same problem and hence with the same algorithm. Only general conclusions are drawn when comparing results from different problems, because of the inherent differences between the problems and the requirements to solve them, as well as any algorithm change. In addition it is noted that the virtual targets algorithm is similar to back-propagation as evidenced by Fig.4–5. The two learning curves in the diagram start at the same
point in weight space and both arrive at the same error, if taking slightly different paths down the error gradient.

![Graph showing a comparison between virtual targets and back-propagation on the character encoder problem.]

**Figure 4-5:** Graph showing a comparison between virtual targets and back-propagation on the character encoder problem.

The virtual targets algorithm, [49], was designed to give uniformity in the update rule of the hidden-input weights and the output-hidden weights to ease hardware implementation. This is done by using a set of virtual targets from which an error term can be calculated from the hidden node outputs. Using this term the hidden-input weights can be updated in the same manner (i.e. using the same circuitry) as the output-hidden weights. The major difference between back-propagation and virtual targets is seen in the way that the error, calculated from the actual network outputs and the “real” training targets, is propagated back through the hidden layer. With back-propagation this happens every iteration of the pattern set in the calculation of the gradient of the error with respect to the hidden-input weights. For virtual targets the error propagation happens in two steps, the first to the virtual target and then from there on the next iteration to the hidden input weights. Therefore the hidden-input weight update is in effect
calculated from the previous calculation of the local gradient. Assuming that the gradient is locally smooth and the steps are small then the approximation should be valid whilst giving a greater ease of hardware implementation. For the majority of problems the inaccuracy in the gradient due to the "delay" in the propagation of the error through the layers does not cause any problem and in fact is reported to give the possibility of escaping from local minima [49]. However some "hard" problems (see the two described earlier) require the pure gradient calculation of back-propagation for solutions to be consistently found.

From the above argument it is seen that the differences between the two algorithms is significantly less than the differences between the problems used for the work of this thesis. Therefore for the purpose of this thesis, any general conclusions drawn from the comparison of results obtained from different problems are assumed to be valid.

4.3.4 Stopping Criteria

The criteria used to stop training on a particular problem are essentially based on the task in question and the solution expected. In the character encoder, with its binary output coding and the fact that a complete solution is expected, a stringent stopping criterion can be used. However, in a problem where the solution is incomplete and varies from simulation to simulation, it is much harder to define a stopping criterion. Also, when the imprecision model is used during learning, the outputs are noisy and there are often significant fluctuations in the error even if the general trend is falling. A low error may be due simply to a particular set of noise-affected weights rather than the underlying network. These effects must be taken into account when deciding on criteria for stopping learning. As mentioned above noise levels need, if possible, to be reduced towards the end of training, requiring a second set of error measuring criteria.

As stated above, the stopping criterion is straightforward for an artificial problem such as the character encoder. In the verification experiments "hard" criteria of $\text{mbe} < 0.1$ and a noise level of $1\%$ - reduced exponentially from a higher
level after the mbe reached 0.4 - are used. The eye/not-eye classifier problem is also expected to be completely solvable and so criteria of mbe < 0.1 and a noise level of 1% are also used.

Deciding upon a stopping criterion for the localisation problem is much harder as the final solution will be incomplete and the outputs very noisy. To decide when learning has reached an optimum level, a validation set can be used and the validation error, \( \epsilon_v \), averaged over a window in the learning error curve, see Fig. 4-6. In fact two windows are used of equal size. If the average error in window 1 decreases below that in window 2 then the \( \epsilon_v \) has reached a minimum and learning has finished. By running a series of experiments at varying levels of noise and window lengths, a window size of 400 epochs was found to give good results. One problem with this technique is that, when a minimum is detected in \( \epsilon_v \), the actual weights have moved on 400 epochs and so have degraded. However, as on average the learning takes much longer than 400 epochs and the error \( \epsilon_v \) tends to only increase slowly after the minimum, this is not considered a problem. As a result of this averaging technique it is also unnecessary to reduce the noise level during training as the underlying trend in the weights is used as a stopping criterion and spurious results are ignored.

The purpose of the Henon map experiments is to test the required accuracy and precision of the calculation needed in the learning phase - in other words, how the hardware imprecision model affects the learning trajectory of an accurate and precise problem. The experiments carried out using this problem were halted when the error had stabilised. It was found that this had always happened after a period of 10,000 epochs and so this "time" limit was used to stop the training. In addition, no noise reduction was required in these experiments as an assessment of the learning performance was taken from an average mse and hence the effect of spurious results was minimised.
Figure 4–6: An idealised view of a validation curve and the windowing system used for stopping learning in the localisation problem.

4.3.5 Error Measures

Various measures of error have been discussed above to explain the criteria used to stop simulation and reduce the noise levels. These have involved the use of the mean square error (mse) and the maximum bit error (mbe) on the training pattern set. Similar error measures have also been used on the validation pattern set. The definitions of these errors have been stated above. When testing a trained network, the error measures are different from those used during the learning phase. In classification problems where a one-out-of-N coding is used, such as the character encoder, the eye/not-eye classifier and the localisation problem, if the unity target output corresponds to the actual output with the largest response to a pattern, then that pattern is said to have been classified correctly. For the Henon map problem the performance measure is much harder to define, as described above, and so only relative mse values can be calculated.
4.3.6 Testing Environment

The method of testing a trained weight configuration depends on the type of information required. In general in this project, a forward pass of some pattern-set through the network was carried out, followed by a calculation of the error. The specific test environments are discussed in detail in the following chapters along with their objectives. Here it is enough to note that the environments are designed to test the trained networks fully on the three performance metrics and on pattern sets applicable to the particular test.

4.3.7 Summary

This section has detailed the simulation environment. The environment varies for each of the four test problems depending on the type of problem and the aims of the simulation experiments. The environment is such that the experiments will produce results that are comparable, where necessary, and will yield qualitative and quantitative measures of performance for varying levels of synaptic noise.

4.4 Chapter Summary

This chapter has presented a discussion and specification of the simulation environment. From this discussion of the needs and issues, a test problem database has been assembled to allow full testing of the predictions made in chapter 3. The simulation environment has also been presented, putting forward means of carrying out the verification experiments using the test problems. From the results of these experiments, evidence of the validity of the predictions can be assessed, along with an understanding of the underlying mechanisms and a knowledge of the limitations of the techniques. Conclusions can therefore, be drawn on the accuracy and precision requirements of MLP neural algorithms and the software performance enhancement method of synaptic weight noise. The simulation
experiments carried out using the simulation environment and test problems detailed here are presented in the following three chapters.
Chapter 5

Fault Tolerance – Performance Implications

All animals are equal, but some are more equal than others.
George Orwell, Animal Farm

5.1 Introduction

This chapter presents results verifying the prediction that synaptic noise, injected during the learning phase, enhances the fault tolerance of the resultant network. In chapter 3 it was explained that the mathematical analysis of the expanded cost function led to the prediction that the salience of individual weights would have a lower standard deviation when trained under the influence of noise. In other words, it was predicted that the computation would be spread evenly throughout the network. This chapter reports on the simulation experiments carried out to verify that prediction.

In the current literature, fault models are discussed in conjunction with methods of testing network robustness. A review and critique of this work has been reported in chapter 2, and the work of this thesis placed in that context. In that wide arena, chapter 3 then presents predictions about the effects of the hardware
imprecision model on the fault tolerance performance. In chapter 4 the simulation requirements and environment were discussed to allow verification of the predictions, and analysis of the underlying mechanisms and limitations, to take place. Thus using all this information, verification experiments can be carried out.

This chapter details the fault model used to test the networks for fault tolerance performance. In addition, a test environment is presented that will allow a full analysis of the technique as an enhancement scheme. Finally, results of the simulations are reported along with a discussion of their significance.

5.2 Test Environment

This section describes the environment in which simulation experiments were carried out to verify the prediction of enhanced fault tolerance. Following the discussion in chapter 2, a fault model is presented to fulfil the aims of chapter 4. In addition, the experiments carried out to test the noise trained networks for fault tolerance performance, are detailed.

5.2.1 Introduction

The fault model and the experimental method are key to extracting the correct information from the test subject and results. In particular the fault model must be an accurate representation of real hardware faults. Also, experimental methods and details are important as they must produce results that will fulfil the test requirements. The details of the experiments are therefore vital in producing results that are not inconclusive or misleading. In this section the test environment is described, taking into account the requirements and aims of the experiments.
5.2.2 The Fault Model

The issue of defining a fault model for testing an operational mode neural network was discussed in chapter 2, where four possible models were listed. To test a network fully, each of these would be used individually and in combination. Here we are interested in the test aims that were discussed in chapter 4:

- Basic verification of the prediction.
- Analysis of the underlying mechanisms.
- Testing the limitations of the technique.

To achieve the first - validating the claim that training with synaptic noise enhances the network fault tolerance - testing with just one of the four models described in chapter 2, will suffice. In the experiments detailed later, a fault model of weights-stuck-at-zero was used. Testing networks with this model gives an indication of their tolerance to faults in general, while also verifying the prediction. In the simulation experiments, the second and third test aims were fulfilled using a faulting method of persistent weight adjustment or agitation. The simulation experiments are described in detail in the following sections.

5.2.3 Random Damage

The random-damage experiment was designed to fulfil the first test aim, that of verifying the basic prediction. It used the stuck-at-zero fault model on weights selected at random from the network. As weights in a trained network have differing importance and there are often many redundant weights, choosing individual redundant weights would give misleading results. One way of overcoming this problem would be to test every weight and every combination of weights, but this would be computationally impractical. Here, a reasonable compromise of selecting a number of weights at random and testing these allows the underlying sensitivity to faults to be measured. Therefore, damaging the network at random a number of times
and averaging gives a good statistical measure of the fault tolerance. Also taking a number of networks trained at each level of noise removes the possibility of the results of individual networks being pathological freaks. In the experiments carried out here, 35 weight sets, at each of 5 levels of multiplicative noise and 7 levels of additive noise were tested at 5 levels of random damage. The exact number of weight sets, used here and elsewhere, was chosen to be computationally viable and still give statistically clear results. The level of damage was measured as a percentage of the whole network and each level was inflicted on the network 100 times. The results of these experiments are presented in section 5.3.

5.2.4 Weight Agitation

The weight-agitation test method involves adjusting individual weights of a trained network by a constant to measure the importance of that weight to the output. Although it is extremely difficult to ascertain an absolute measure of the importance of a weight, the error due to the agitation, $\epsilon_{agit}$, allows an approximate saliency to be gauged. In particular, the variation in this saliency is of interest. While the variation in the mean value of this error, due to differing levels of noise, gives an indication of the tolerance to faults (where the fault is the weight agitation), the standard deviation across the network gives a measure of the spread of the computation in the network. In other words, by measuring the standard deviation, $\sigma_{agit}$, of $\epsilon_{agit}$ for networks trained with different levels of synaptic noise, the variation in the spread of weight saliency can be gauged. Taking the average of $\sigma_{agit}$ over all 35 weight sets, mentioned earlier, gives evidence of the mechanisms of fault tolerance enhancement. It also allows the limitation of the technique to be analysed if the test is carried out on a hard problem. Experiments using this technique were carried out and the details and results are presented in section 5.3.
5.2.5 Analysis of the Weight Distribution

The third test method involves the analysis of the spread of the weight values. By looking at this distribution a qualitative indication of the internal representation of the problem in the network can be seen. While it is true that the absolute value of a weight cannot give definite evidence of its actual importance to the final solution, the proportion of the weights at zero will indicate the proportion of the network involved in the computation. For a weight to be at or near zero, the input to that connection must be redundant. This should not be the case in a network where all the weights are being “used” and the computation is spread evenly. Therefore, looking at the final weight value distribution gives a qualitative measure of how dispersed the computation is throughout the network. Weight distributions for the networks trained with differing levels of multiplicative and additive noise are presented in section 5.3.

5.2.6 Summary

This section has detailed the test environment used in verifying the prediction of enhanced fault tolerance. It defines the fault model used and the experimental methods for fulfilling the test requirements. Using the results of these experiments the validity of the prediction will be concluded and an analysis of the underlying mechanisms and technique limitations will be carried out.

5.3 Results

This section presents the results and experimental details of the simulations carried out to verify the fault-tolerance prediction. The experiments use the test environment and methods described above, and the results presented below fulfil the test requirements defined in chapter 4.
5.3.1 Introduction

Compared to other performance improving techniques, fault tolerance enhancement methods described in the literature are few and far between, [9][16][54][57]-[65]. The majority of these techniques, where results have been published showing improvements, have required significantly more processing than standard learning algorithms. The method described above is simple and requires minimal external supervision. The results presented here are, therefore, a significant step forward in providing a training technique that produces a maximally robust solution.

Using the background of the literature review, the technique of training with synaptic weight noise in the forward pass is explored. The results presented below give a statistically significant verification of the basic prediction of fault tolerance enhancement. From this foundation an understanding of the technique is developed and results are presented to show the mechanisms by which the improvements are achieved. Finally, experiments testing the limitations of the enhancement method are carried out and the results discussed.

5.3.2 Verification of the Basic Prediction

In chapter 3 the mathematical analysis of the effects of synaptic weight noise on the learning and learnt solution, led to the prediction that the fault tolerance performance would be enhanced. To verify this prediction the random damage experiment, described in section 5.2.3, was carried out on the character encoder problem. The results of this experiment are shown in Fig. 5-1. The graph shows a significant drop in the classification error, $e_t$, as the level of multiplicative noise, used during training, is increased. The error bars plot the standard deviation of the results and show that the underlying behaviour has been extracted from the 100 random weight selections and also that the results are repeatable. The different “curves” show varying levels of damage inflicted on the network. Taking, for example, the 6% damage case, a drop of approximately 40% in the classification error is seen by using a 40% level of noise in the learning phase. Levels of noise
above 40% continued to give performance improvements, but the learning became erratic in behaviour. Clearly, the prediction of enhanced fault-tolerance has been verified for the multiplicative noise case.

Figure 5-1: Graph showing the reduction in classification error with increasing levels of multiplicative synaptic weight noise, injected during training, at different levels of random damage. i.e. fault tolerance performance enhancement on the character encoder problem.

### 5.3.3 Enhancement Mechanisms

Having shown that the fault tolerance is improved, it is now important to examine the underlying causes: in effect, to answer the question as to why training with noise produces this beneficial result. In chapter 3 the mathematical analysis predicted that the noise would have the effect of spreading out the computation in the network, thus giving a more distributed representation of the problem solution in the network. The implication of this is that if an individual unit was then damaged, or removed, the effect would be less catastrophic than if only
a few units were important and a fault occurred in one of those. To test this theory, weight agitation experiments were carried out on the trained networks, as described in section 5.2.4, to examine the individual weight saliency and to gauge the spread of the network computation. Fig. 5-2 shows the results of these experiments carried out on the 35 weight sets generated to solve the character encoder problem. The graph shows a drop in the standard deviation of the error across the network as synaptic noise is increased. With networks trained without noise there is a wide spread, \( \sigma_{\text{agit}} \approx 10 \), of errors caused by agitating weights, i.e. agitating some weights causes a large error, while the agitation of others has no effect. Contrastingly, networks trained with 40% multiplicative noise show a much lower spread of weight saliency, \( \sigma_{\text{agit}} \approx 5 \). Therefore, this somewhat crude test indicates that a more computationally dispersed representation is achieved for networks trained with noise than those trained without. From this the conclusion is drawn that the prediction, of a more distributed and therefore fault tolerant representation, is true.

This theory is further confirmed by looking at the distributions of the final weight values for the networks trained with various levels of noise. This experiment was described in section 5.2.5. Fig. 5-3 shows the final weight distributions for networks trained on the character encoder problem for noise levels of 0% to 40%. From the graphs, it is clear that the proportion of weights at zero decreases as the noise level increases. In fact, the weight values tend towards a bimodal distribution. It can therefore be surmised that more of the network weights are being used actively in the input-to-output mapping at the higher levels of noise.

Therefore, the underlying mechanisms of the fault-tolerance enhancement can be seen to be in the effect of the noise giving a solution that is distributed throughout the network. Simulation results showing the implications of additive rather than multiplicative noise are presented in section 5.3.5.
Figure 5-2: Graph showing the variation in standard deviation of the error caused by the agitation of individual weights across the network, for networks trained on the character encoder problem.
Figure 5-3: Final weight value distributions following training with multiplicative synaptic noise for the character encoder problem.
5.3.4 The Limitations of the Improvement

To test the limitations of the improvement technique, simulations were carried out using the localisation problem. Networks were trained using levels of multiplicative synaptic noise in the range from 0% to 25%, with 5 weight sets at each level. The complexity of the problem is such that the classification error on the training data, $\epsilon_t$, for the “solutions”, was not zero at the end of the learning phase. Apart from the problem complexity, this was due to the limitations of the size of the training pattern set and the limited capacity available in the network. While with the character encoder problem $\epsilon_t = 0$ at the end of training, here $\epsilon_t > 0$. The implication of this on the saliency of the weights will be discussed in depth in chapter 6, examining the relationship between network capacity and the required computation of the problem. Here the variation in the saliency is of interest. Weight agitation experiments were, therefore, carried out to examine the representation of the solution. Fig. 5-4 shows the standard deviation of the error due to the agitation across the network for the localisation problem. The difference in the two problems is evident when examining the standard deviation at 0% noise in Fig. 5-2 and Fig. 5-4. In the localisation problem this level of standard deviation shows that the majority of weights are required in the attempt to find a solution. In fact the level of 0% noise in Fig. 5-4 is lower than the level of standard deviation for 40% for the character encoder problem, Fig. 5-2, signifying the greater contribution required from all of the weights to solve this harder problem. Clearly, however, the use of noise again causes the computation to be spread evenly throughout the network for the localisation problem. The value of $\sigma_{agit} \approx 0.6$ for 25% noise.

From these results it is clear that the standard learning algorithms can produce networks where the salient weights are few. For hard problems many of the weights may be required, but some will still be more significant than others. The use of noise, even for these problems, spreads the network computation around the network producing individual weights with approximately equal saliency. This effect naturally leads to an enhancement in the fault tolerance performance of the
Figure 5-4: Graph showing the variation in standard deviation of the error caused by weight agitation across the network, for networks trained on the localisation problem.

trained networks. Therefore, the technique is only limited by the ability of the algorithm to achieve an adequate $e_t$ with noise injected. This ability will be discussed in the following two chapters.

5.3.5 Additive Noise

All the results presented thus far in this chapter have been recorded from experiments on networks trained with multiplicative synaptic noise. This section looks at the implications of additive synaptic noise on the fault tolerance performance of an MLP.

The random-damage experiments carried out on the character encoder problem were repeated, this time using the additive noise model. Noise levels of up to 300% were probed, where the level is defined as a percentage of the mean absolute value of the initial weight range. Using the parameters defined in Table 5-1, 35 weight sets were generated at levels of noise 0% to 300% at 50% intervals. Using
these weight sets the random-damage experiments were carried out as before and the results of these simulations are shown in Fig. 5-5.

<table>
<thead>
<tr>
<th>Hidden Layer Neurons ( J )</th>
<th>Initial Weight Range ( \beta )</th>
<th>Adaptation Rate ( \eta )</th>
<th>Momentum ( \alpha )</th>
<th>Sigmoid Temperature ( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>±0.1</td>
<td>0.1</td>
<td>0.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 5-1: Training parameters for the character encoder problem, as used in the initial additive noise simulations.

![Graph showing the classification error with increasing levels of additive synaptic weight noise injected during training at different levels of random damage. These results were obtained using a set of weights trained from an initial weight range of 0.1.](image)

From the graph it is clear that the fault tolerance of the networks has been enhanced. For 6% damage at 0% noise a classification error of approximately 96% is seen, while with 200% noise the error drops to approximately 33%. It is
interesting to note that at higher levels (> 200%) the error begins to increase. This effect, apparently going against the trends of the previous results, can be explained by looking back at the predictions made in chapter 3. The mathematical analysis was carried out on the multiplicative noise model, although the additive noise case was also briefly discussed. The major difference is that, with additive noise, the random values are added irrespective of the actual size of the weights. It will be seen later that with additive noise the weights have a tendency to find a solution where weight values are large. From this behaviour it is suggested here that additive noise has a detrimental effect on the learning characteristics of small weights. Where small weights are required in the solution, the proportionally very large levels of noise “swamp” the actual weight value. Multiplicative noise is added in proportion to the weight size, therefore allowing even small weights to learn. From this argument, one theory for the “saturation” effects seen in Fig. 5-5, would be that only large weights can learn under the influence of noise and so there is a limited number of weights that are “usable” in the final solution. To accumulate more evidence for this hypothesis, a further set of simulations was carried out using a lower initial range of weights (\( \beta = 0.01 \)). The results of these experiments are shown in Fig. 5-6. Again the increase in error is seen for the higher levels of noise and in this case the “cut-off” point happens at a lower level of absolute noise. The absolute noise level is defined as the maximum actual noise value added to the weights, rather than the proportion of the initial weight range. In Fig. 5-6, the “cut-off” point occurs at the calculated percentage noise level of 1500% taken in combination with the initial weight range gives an absolute level of 0.150, while with the initial weight range of 0.1 the “saturation” occurs at an absolute noise level of 0.2 (see Fig. 5-5). The effect of the reduction in the initial weight size is thus seen.

As with the multiplicative noise case the distributions of the final weight values are also interesting. Fig. 5-7 shows the distributions for the networks trained from an initial weight range of 0.1. Following the sequence of increasing noise used during training, the distribution, as in the multiplicative noise case, becomes more spread out. In this case, the use of the parameters \( \beta = 0.1 \) and \( \eta = 0.1 \) give
Figure 5-6: Graph showing the classification error with increasing levels of additive synaptic weight noise injected during training at different levels of random damage. These results were obtained using a set of weights trained from an initial weight range of 0.01.
a very “narrow” distribution at 0% noise, showing that these training parameters also affect the spread of saliency. This was also evident in the random damage experiment results. Considering again the “saturation” effect which occurs at noise levels > 200% for $\beta = 0.1$, the distributions, at these levels, show a marked increase in low negative weights and a trend towards larger valued weights in the solutions. The precise reason for this is unclear, although following on from the previous discussion, the spread of the weight saliency is reduced at high levels of additive noise. This implies that some weights are unable to learn at these levels. The hypothesis presented above suggests that smaller weights are “swamped” by the noise level and are unable to learn.

5.3.6 Summary

The results presented in this section verify the prediction of improved fault tolerance with the use of synaptic weight noise during the learning phase. As well as verifying the basic prediction the results allow an extraction of the underlying mechanisms of the enhancement through an examination of the spread of the weight saliency in the networks. By using a hard problem the limitations of the technique have been defined as depending more on the ability of the training algorithm to be able to find a solution in the noise affected training, than on the technique itself. Even on the very hard localisation problem, the technique has been shown to give a solution with a distributed representation, giving good fault tolerant characteristics. The effect of the noise on the final training error will be discussed in the following chapters.

Results have also been presented from experiments using an additive noise model and equally good fault tolerance enhancements noted. In the additive noise case, however, the error increases at very high levels of noise. A hypothesis has been presented to explain this increase.
Figure 5-7: Final weight value distributions following training with additive synaptic noise for the character encoder problem. The initial weight range was 0.1.
5.4 Chapter Summary

This chapter has presented a fault model and a test methodology for validating the fault tolerance enhancement predictions made in chapter 3. A series of simulations has been carried out, the results of which prove the accuracy of the predictions. In other words the injection of noise during learning increases a network's tolerance to faults. All the test aims, outlined in chapter 4, have been achieved.

The implications of these results are wide ranging and the technique is much simpler than many other fault tolerance improvement methods, only being limited by the ability of the learning algorithm to find a solution in the noise affected training phase. In appendix A this issue is addressed by examining the extra term in the cost function, forming the basis of the mathematical analysis. The term is presented as a more heuristic means of improving fault tolerance and a learning algorithm incorporating it is derived using the basic error back-propagation method.

In summary, the conclusive fault tolerance enhancement results reported here are at the centre of the performance improvements presented by this thesis. Chapters 6 and 7 examine the implications of the noise on the generalisation performance and learning trajectory and speed.
Chapter 6

Generalisation Ability – Performance Implications

‘What a remarkable phenomenon!’ said the Professor ... And he wrote a long letter about it to the local newspaper. Everyone quoted it, it was full of so many words that they could not understand.

Oscar Wilde, The Happy Prince

6.1 Introduction

This chapter presents results verifying the predictions made concerning the effects of learning with synaptic weight noise on the generalisation ability of the resultant network. In chapter 3, the mathematical analysis of the expanded cost function led to the prediction that training with noise would de-stabilise solutions with hidden units using the slope of the sigmoid function. It was inferred that this training condition would cause the hidden layer units to be either fully ON or OFF. It was also predicted that in this state solutions would be more robust to variation in the inputs and that the computational capacity would be reduced along with the overall error. The combined effect of these two implications is an improvement in the generalisation ability of the network. This chapter reports on the simulation experiments carried out to verify that prediction and their results.
Many techniques are reported in the current literature for enhancing the generalisation ability of multilayer networks. Chapter 2 reports on a selection of this work and places the synaptic weight noise technique in that context. As described above, chapter 3 then predicts generalisation ability improvements from the effects of the hardware model. In preparation for the simulation experiments, the results of which are presented below, chapter 4 examines the simulation environment and requirements to allow a complete verification of these predictions. Therefore, experiments can be carried out analysing the predictions for their validity, underlying mechanisms and limitations.

In summary, this chapter details the experimental methods used to verify the claims of chapter 3. Following this, results are presented and discussed and conclusions are drawn about the technique as a generalisation enhancement scheme.

6.2 Method and Aims

This section details the test environment used to verify the generalisation ability predictions. The experiments used to carry out this objective are discussed individually and their aims presented.

6.2.1 Introduction

In chapter 3, predictions were made concerning the hardware precision model and in particular, for this chapter at least, the effects on generalisation ability. The predictions centred on the effect of the synaptic noise causing solutions to be preferred where the hidden layer units were in the fully ON or OFF horizontal regions of the sigmoid. From this characteristic it was surmised that the generalisation ability would be enhanced. In chapter 2, generalisation enhancement techniques were categorised into three areas and the synaptic noise technique classified into the section on "in-training" complexity constraints. This is a group
of enhancement methods that improve generalisation by controlling the capacity (or complexity) of the training network to stop "over-fitting" of the training patterns occurring. This chapter verifies these predictions and places them in a context of other complexity constraining techniques.

In chapter 5, the objectives of the verification experiments were presented. These can be summarised as:

- Basic verification of the prediction.
- Analysis of the underlying mechanisms.
- Testing the limitations of the technique.

Thus a set of experiments is required that fulfils these aims and allows the technique to be discussed in the context of the current literature. Following the guidelines set up at the start of chapter 5 concerning the requirement for careful consideration of the experiments to avoid misleading or ambiguous results, the following sections detail the simulation experiments used to analyse the generalisation ability.

6.2.2 Problem Generalisation

To assess the basic ability of a trained network to generalise from the "knowledge" it has extracted from the training pattern set, test patterns are required. For most real world applications these will be patterns not included in the training set, although ones which will have been represented by the training data - the generalisation being an interpolation task rather than an extrapolation outside the bounds of the training data. By presenting these patterns to the trained network and measuring the error in classification, $\epsilon_g$, an indication of how well the network has extracted the "underlying rules" of the problem can be measured. Also, to a lesser extent, the ability of the network to operate in a working system will have been gauged. Therefore all that is required to assess the generalisation ability is a set of test patterns.
In the case of the eye/not-eye classifier, described in chapter 4, test patterns are available from images of faces previously not encountered by the network. These patterns give a quantitative measure of the generalisation performance and the effect of the synaptic weight noise, as the results of using multiple weight sets are collected and statistics are accumulated. For the character encoder problem, there is no true measure of generalisation, as the problem is artificial and all the patterns are used in the training phase. Despite this, an indication of the robustness to changes in the input data can be achieved by introducing corruption into the input patterns and presenting them to the trained networks. The sensitivity of the output to small changes in the inputs can therefore be measured, giving some assessment of generalisation [6]. In the experiments carried out for this thesis, noise taken from a normal distribution at a fixed standard deviation was added to the input patterns. Each pattern was presented to the network 200 times with its associated noise being re-generated each time to build up statistics. The results for this experiment and the eye/not-eye classifier experiment are reported and discussed in section 6.3.2.

6.2.3 Assessment of the Internal Representation

In chapter 3 predictions were made concerning the internal state of the trained networks. In particular it was stated that the hidden node outputs were more likely to be at the extremes of the sigmoid curve (i.e. firmly ON or OFF). If this is indeed the case then it would explain the improved generalisation results seen in the work of this thesis and also reported previously, [48]. To test this prediction two schemes were devised to probe the characteristics of the weight sets and the internal representations produced following training with synaptic weight noise.

The first of these two methods carries out a direct numerical measurement of the region of the sigmoid curve a particular node is using. By calculating the average derivative of the \( J \) hidden node outputs for all the \( P \) patterns, (6.1), a
comparison can be made between one network and another.

\[ F(o_p) = \frac{1}{J_P} \sum_{j_P} o_{j_P}(1 - o_{j_P}) \]  

(6.1)

Alternatively, a comparison can be made between a group of networks trained at one level of synaptic weight noise and another group trained at a different level. From the results a measure can be taken of whether the neurons are firmly ON or OFF, or in the linear sloping region. As the output nodes, \( o_k \), are driven firmly ON or OFF, for the classifier problems at least, the measure for these nodes is less significant. For this reason, the calculation only includes the hidden layer nodes. Results from this experiment, for the character encoder and the eye/not-eye classifier, are reported in section 6.3.3.

The second method involves an adjustment of the gain (or "temperature" \( \theta \)) term of the sigmoid function (6.2).

\[ F(\text{net}) = \frac{1}{1 + \exp \frac{-\text{net}}{\theta}} \]  

(6.2)

This is a less direct method in that it induces errors in trained networks where the neuron outputs (particularly the hidden node outputs) are on the steep section of the sigmoid curve. It is however of more practical significance than the straightforward numerical measurement in the first method, as it effectively relates directly to the resilience of the trained network to imperfections in the activation level. Therefore, the method gives an indication of how robust the internal representations are to changes in the inputs (or weights). As an aside, it is interesting to note the relationship between this work and the fault tolerance analysis of the previous chapter. However, here the tolerance to input variation is of primary importance. Fig. 6-1 shows the effect of a change in \( \theta \), and indicates how a node with an activation value that places it on the slope of the sigmoid will be affected by the change in "temperature". If, for example, the activation level is above the threshold (Th) such that it produces an output of 0.95 at \( \theta = 0.2 \), then a change in "temperature" to \( \theta = 0.7 \) will cause a change in output to 0.7. Therefore, if the hidden layer units use the steep section of the sigmoid curve a large variation in the output will be seen for a small change in the activation or temperature.
Using this method, simulations were carried out on 35 weight sets trained on the character encoder problem and the results are reported in section 6.3.3.

**Figure 6-1**: Graph showing the sigmoid curve with varying "temperature", $\theta$.

### 6.2.4 Summary

This section has described experiments that were designed to analyse the predictions made in chapter 3 and to fulfil the test requirements. The results of these experiments will allow the predictions to be verified, and an analysis of the underlying mechanisms and technique limitations to be made. The following section presents the results and discussion of the generalisation performance tests, and conclusions are drawn.
6.3 Results

This section presents the results of the simulation experiments carried out to verify the generalisation ability predictions. The experiments are those described in the previous section. From the results and ensuing discussion, the predictions are verified and the enhancement technique placed in the context of the current literature.

6.3.1 Introduction

The current literature provides a wealth of methods that can be used to improve the generalisation ability of a network. Included in this literature are experimental results of simulations carried out with synaptic weight noise [48]. These results show improved generalisation performance for networks trained with synaptic noise on two classification problems, one artificial and the other a "real world" example. Chapter 3 presented a theory for this enhancement and the results reported below support the arguments raised. Therefore, the results in this section are not new, but confirm those in [48]. Following on from this confirmation they also allow a discussion of the underlying mechanisms of the technique to be carried out and the limitations assessed.

Using the same artificial problem used in [48] and a different "real world" problem, the predictions of enhanced generalisation ability are verified. Utilising these problems and the experiments described above, the processes underlying the improvements are analysed. Then, considering a much harder problem, the limitations of the technique are assessed. Finally, following the discussion in the previous chapter, results of experiments carried out using additive synaptic noise are presented and discussed.
6.3.2 Verification of the Basic Prediction

Following on from the mathematical analysis of the hardware model, presented in chapter 3, predictions were made concerning the generalisation ability of networks trained with noise in the synaptic multiplication. To verify these predictions the experiments discussed in section 6.2.2 were carried out on the character encoder problem and the eye/not-eye classifier. The generalisation ability of the character encoder was tested using the “input corruption” method. For each level of synaptic noise used during training, 35 weight sets were produced, where the noise levels from 0% to 40% were probed in detail. Above this noise level improvements were still noted, but the training became erratic. The results of the simulations are presented in Fig. 6–2 and clearly show a reduction in the classification error, $e_t$, as the synaptic noise level is increased. The error bars show the standard deviation of the results across the 35 weight sets and the “curve” is an average. For the experiments the noise on the input patterns had a standard deviation of 0.1. At 0% noise in the weights during training, the percentage of corrupted patterns classified incorrectly is approximately 22%, while at the level of 40% noise only 1% are incorrectly classified. A significant improvement is therefore achieved.

For the eye/not-eye classifier a true test of generalisation ability can be carried out. Again, 35 weight sets were used at each level of synaptic noise (0% to 40%). In this case, the networks were tested on data taken from facial images previously unseen by the networks and “asked” to classify them as “eye” or “not-eye”. The results of this experiment are given in Table 6–1. Here the results again show that synaptic noise improves the generalisation ability of the networks - an improvement of 6% is seen from the networks trained at 0% noise to those trained at 40% noise. While this enhancement is clear in the average value for the classification error, the standard deviation ($\sigma$) shows that the performance of individual networks varied dramatically. The precise reason for this is unclear, although some of the test patterns contained sections of facial images of people wearing glasses, or having beards - both of which cases were not represented by the training set. These generalisation tasks could be said to be extrapolation and not
Figure 6-2: Graph showing the reduction in classification error due to "input corruption" with increasing levels of multiplicative synaptic weight noise, injected during training for the character encoder problem.

interpolation, perhaps explaining to some extent the reason for the wide variation in the generalisation performance. On closer inspection, the classification errors for patterns from individual facial images did indeed have a tendency to have a lower variation than those from different images. This shows that for the networks some images were easy to generalise to while others, perhaps but not necessarily explicitly contained in the training set, were harder. Despite this variation, on average the generalisation ability is improved by the noise and the results are consistent with those reported earlier in [48].

6.3.3 Enhancement Mechanisms

The results presented in the previous section show the enhancement in generalisation ability predicted for networks trained with synaptic noise. Therefore, having verified the result of the prediction in chapter 3, the trained networks can now be examined to look at the underlying causes of the improvement. By analysing the
Table 6-1: Table giving the percentage classification error for different levels of multiplicative synaptic noise for the eye classification task.

expanded cost function, chapter 3 proposed that the noise would have the effect of penalising solutions with outputs on the linear sloping section of the sigmoid. Hence, it was concluded that if possible the network would find a solution where the extremes of the sigmoid curve were used. Generally, in the case of the classifier problems such as the character encoder and the eye/not-eye classifier, the output nodes will be at the extremes in any case. However, when training with noise the hidden layer neurons will also be more likely to have outputs that are fully ON or OFF. This characteristic should have the effect of giving a more robust internal representation as small changes in the activation level will have less effect on the overall network output. Also, as argued above, the computational capacity will be reduced.

The capacity of a network is a concept that is very hard to quantify. Although it is known that the capacity is connected to the number of parameters for example, there are many other factors that are much harder to determine. At the same time it is known that having a capacity that is greater than that required by the underlying rules of the problem will cause over-fitting to the idiosyncrasies of the training patterns and a subsequent drop in the generalisation performance. Therefore techniques that reduce the computational capacity of the network during learning have the potential to improve the generalisation ability.
If the hidden layer neurons can only take on two values, 0 or 1 in this case (or at least values close to this and not changing significantly for changes in the input), then the network has a reduced capacity from one whose hidden nodes can have values changing in a linear relationship with changes in the input. In other words the computation is restricted.

Although as discussed above, the actual capacity of the network at any point in time is extremely hard to calculate, the region of the sigmoid curve being used by a particular solution can be gauged using the experiments detailed in section 6.2.3. By carrying out these experiments the underlying mechanisms of the generalisation performance improvements, predicted in chapter 3, can be verified. Taking first the character encoder problem and the 35 weight sets obtained for the generalisation tests in the previous section, a numerical measurement of the average hidden layer node output derivative can be made. The results of this experiment are given in Fig. 6-3, where the solid line plots the mean level for all 35 weight sets of the average derivative value, and the error bars show the standard deviation. The graph clearly shows that for networks trained with noise, there is a definite trend towards using the sigmoid extremes where the derivative is low. This experiment was repeated for the eye/not-eye classifier and the results are given in Fig. 6-4. Again the trend is seen and the predictions are confirmed.

Having shown that the use of synaptic noise causes the trained network to be more likely to use the sigmoid extremes than the sloping region, the second experiment detailed in section 6.2.3 can be carried out. Using the same network weights as before, the sigmoid function was systematically changed by varying the “temperature” $\theta$. At 6 levels of $\theta$, greater than the value of 0.2 used during training, the training pattern set was presented and the classification error measured. Fig. 6-5 shows the results of this experiment. The 3-dimensional graph shows how, as the value of $\theta$ is increased from the one used during training, the error initially stays at zero. At $\theta = 0.4$ the weight sets trained with high levels of synaptic noise still classify the majority of patterns correctly, while at levels less than 10% all the patterns are incorrectly classified. This result clearly shows that
Figure 6-3: Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected noise during training for the character encoder problem.

Figure 6-4: Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected noise during training for the eye/not-eye classifier.
the extremes are being used and that small changes in the activations will have negligible affect on these robust networks. Therefore these two experiments have verified the prediction of chapter 3 and give an explanation for the generalisation ability enhancements seen in the previous section.

Figure 6-5: Graph showing the reduction in classification error $\epsilon_t$ as the synaptic noise level is increased during training. The temperature parameter is varied in the sigmoid function for networks trained on the character encoder.

6.3.4 The Limitations of the Improvement

In the previous two sections the synaptic weight noise technique has been shown to give improvements in the capability of the trained network to generalise. This ability has been shown to result from the effect of the hidden layer outputs being forced to use the extremes of the sigmoid curve, rather than the "unstable" steep slope. In some problems, however, the linear region is required for the network to obtain a solution and the preferential selection of the horizontal region over the sloping linear region can have a negative effect on the learning process. To illustrate this argument, the localisation problem was used. For each level of synaptic noise (0% to 25% at 5% intervals in this case) 5 weight sets were trained
until the generalisation ability reached a maximum as shown by a cross-validation technique described in chapter 4. These weight sets were then tested as before to analyse the generalisation performance for this hard problem using patterns previously unseen by the networks. Fig. 6-6 shows the average levels of the classification error for the weight sets at each level of synaptic noise. The two “curves” (the solid line representing the average value of the error and the error bars, the standard deviation of the distribution of errors) represent the generalisation error, \( \varepsilon_g \), and the training set error, \( \varepsilon_t \). The graphs show that after a small improvement at 5% synaptic noise, the error begins to rise. In this case, therefore, the synaptic noise has a negative effect on the generalisation performance and also the training set error.

To test the hypothesis that this negative effect is due to the capacity reduction caused by the hidden layer units with high derivative values being de-stabilised, a measurement of that derivative was taken. Fig. 6-7 presents the results of this test and shows how the hidden layer outputs, as in the previous experiments are more likely to use the sigmoid extremes. In this case however, the effect is less marked and the increase in the derivative at high noise levels gives an indication of the trade off that there is with the competing pressures of the reduction of the mean square error and the hidden layer derivative. Although these experiments do not give any definite proof of the mechanisms for the detrimental effects on learning, an indication is seen. However, from the results it is clear that with some problems the synaptic noise has a detrimental effect on the training performance. If, as suggested above, the cause of this is the preference for the use of the sigmoid extremes, then judicious use must be made of noise as a performance enhancement scheme for problems that are known or suspected to require this region of the curve. In the case of a hardware implementation the effects noted must also be taken into account when trying to map a hard problem onto a particular hardware system. It would perhaps be useful to oversupply the hardware with hidden nodes and then train with noise to get good performance. This issue is addressed below. In most classification problems a binary internal representation (or an approximation of it by the extremes of the sigmoid function) may be
Figure 6–6: Graph showing the change in the training and generalisation classification error at increasing levels of synaptic noise, for the localisation problem.

Figure 6–7: Graph showing the average value of the hidden layer derivative against noise for the localisation problem.
adequate and also a positive characteristic. However, problems where the inputs
and outputs are analogue, such as most TSP problems, often require hidden units
where the sigmoid slope is used. These problems may be detrimentally affected
by the noise and, therefore, any attempt to realise the solution in hardware may
prove to be extremely difficult.

To confirm that the detrimental performance effects are caused by a noise-
induced reduction in the network capacity, another experiment was carried out.
The previous experiment using the localisation problem was repeated, this time
increasing the number of hidden units. If the hypothesis that the reduction in
network capacity causes an increase in error is correct, then the introduction
of extra nodes at the start of training should overcome the problem. Using a
synaptic noise level of 20%, 5 networks were trained for each hidden layer size
and the classification errors, $\varepsilon_g$ and $\varepsilon_t$, measured. Fig. 6–8 shows the results
of this experiment. It can clearly be seen that the increased level of network
capacity allows more of the patterns to be correctly classified, as shown by the
training set error, $\varepsilon_t$. The generalisation pattern set error, $\varepsilon_g$, also improves as
the capacity increases until a minimum is reached. Therefore, the introduction of
more hidden units shows that the noise reduces the capacity of the network to a
level where it has a detrimental effect on the network performance. At numbers
of hidden units above 60, the generalisation error rises again, possibly due to
the network having more capacity and being able to "over-fit" the training data
to some extent. However, the important result from this experiment is that the
error is initially seen to decrease as the capacity (related to the number of hidden
units) is increased. This result can also be applied when realising the solution
to a hard problem onto imprecise hardware. By using an increased hidden layer
size than would normally be required for a more precise software solution, better
results will perhaps be evident.

Noise has therefore been shown to have the effect of penalising solutions that
have activation levels using the sloping section of the sigmoid curve. For many
problems, particularly in the field of pattern classification, this will have a positive
Figure 6-8: Graph showing the change in the training and generalisation classification error at increasing numbers of hidden layer units for the localisation problem.
effect on the generalisation ability. This has clearly been shown in the results of the experiments using the character encoder and the eye/not-eye classifier problems. However, on other problems the noise can have a detrimental effect causing performance degradation. In these problems, the noise level is a significant factor in determining whether a solution is found. In the case of a hardware implementation, where the noise level is a constant reality, improvements may be seen by increasing the number of hidden layer neurons. This has the effect of increasing the capacity of the network so that the sigmoid extremes can be used to "store" the problem solution as well as giving a robust solution. In the software case, judicious use of the noise must be made otherwise degradation in performance will be seen rather than the required enhancements.

6.3.5 Additive Noise

The results reported in the previous sections and the ensuing discussion all referred to multiplicative noise. This section returns to the topic of additive synaptic noise, added at a constant level irrespective of the weight magnitude. The results presented show that the improvements and mechanisms are very similar to those for the multiplicative noise case.

Experiments were carried out, as with multiplicative noise, to verify the predictions made in chapter 3 that the noise would enhance the generalisation ability of the networks. Using the character encoder task 35 weight sets were generated for levels of additive noise from 0% to 150%, using an initial weight range $\beta = 0.1$. Although by using the character encoder only an indication of generalisation ability is seen as the networks are trained using the entire set of patterns, the weight sets can be tested using the "input corruption" method. This technique gives an indication of the ability of the network to generalise in some given distribution around the input patterns. Therefore, the weight sets were tested and the results of these experiments are shown in Fig. 6-9. As in the multiplicative noise case, the input was corrupted using noise from a normal distribution with a mean of 0.0 and a standard deviation of 0.1. The graph clearly shows that as noise is used
during training the weight sets become more resistant to input variation. With 0% noise the classification error is approximately 13.3%, while at a noise level of 100% the error has fallen to approximately 0.5%. This improvement continues up until high noise levels are reached where the error begins to increase. However, a significant improvement is seen and the prediction of enhanced generalisation ability is verified.

![Classification Error vs. Synaptic Noise Level](image)

**Figure 6–9:** Graph showing the reduction in classification error due to “input corruption” with increasing levels of additive synaptic weight noise, injected during training for the character encoder problem.

Considering the mechanisms for this improved performance, experiments were carried out on the weight sets produced for the previous experiment to analyse the hidden layer output derivatives. From the results, shown in Fig. 6–10, it is again clear that the injection of noise has de-stabilised solutions using the sloping section of the sigmoid function and favoured those that use the extremes. The mechanisms are therefore the same as those seen for the multiplicative noise case.
Figure 6–10: Graph showing the reduction in the average value of the hidden unit output derivative for different levels of injected additive noise during training for the character encoder problem.

The saturation effects reported in chapter 5 have not been as evident in the experiments carried out here. The drop in the proportion of salient weights in the network, noted in the fault tolerance tests, does have some affect on the generalisation ability. The slight increase in classification error noted in Fig. 6–9 is perhaps an indication of this. However, as stressed above, the key result of this section is that an improvement is seen in the generalisation ability for additive noise injected during the training phase.

6.3.6 Summary

The results presented in this section have verified the predictions of chapter 3. The generalisation ability has been shown to be enhanced by the use of synaptic noise during training on a number of problems. This supports the results reported in [48]. Also the mechanisms for the enhancements have been investigated and have been discussed in terms of network capacity and required computation of
the problem. This discussion has given a qualitative understanding of the generalisation ability improvements and also assessed the limitations of the technique. Results have also been presented from the additive noise model and show similar enhancements to the multiplicative noise case.

6.4 Chapter Summary

The results presented in this chapter have verified the predictions made in chapter 3 that training with noise in the weights enhances the generalisation ability of the resultant network. This result supports the work reported in [48]. An analysis is carried out as to the underlying mechanisms of the technique and also the limitations. In chapter 5, fault tolerance enhancements were shown to be general for a variety of problems and only depended on the ability of the network to be able to learn under the influence of noise. Here, however, it has been seen that the implications of noise are not always beneficial. For some problems where the sloping section of the sigmoid curve is required, the use of noise can have a negative effect on the ability of the network to solve the problem. Although this does not affect the generalisation performance directly, if the classification error on the training pattern set cannot be reduced then the ability to generalise is affected. Despite this, for many problems particularly in the field of classifiers, enhancements to the generalisation ability can be obtained by training with synaptic weight noise.
Chapter 7

Learning Trajectory and Speed – Performance Implications

...if not a present remedy, at least a patient sufferance.

William Shakespeare, Much Ado about Nothing

7.1 Introduction

This chapter assesses the effects of the analogue hardware model, described in chapter 3, on the learning trajectory and speed. The mathematical analysis carried out in chapter 3 led to the prediction that the noise would have a positive effect on the learning performance. This somewhat non-intuitive prediction arose from the examination of the second derivative term in the expanded cost function and the noise affected weight-update equation. This chapter presents results confirming that prediction for one simple problem and investigates the mechanisms behind the improvement and the limitations of the technique.

The previous two chapters confirmed the predictions of chapter 3 concerning the effects of synaptic weight noise on the performance of the trained network. They also examined the underlying mechanisms of the enhancements by investigating the effects of the noise on the network during the learning phase. Following
on from this work, this chapter looks at the training process directly to verify the prediction of chapter 3 that the noise would give an improved learning trajectory. This assessment will be carried out in the context of the learning time improvement techniques reviewed in chapter 2.

In summary, this chapter presents a discussion of the concept of assessing the quality of the learning trajectory and speed, and presents test experiments to quantify the implications of the synaptic weight noise. Simulation experiments are carried out and the results are reported and discussed.

### 7.2 Test Aims and Method

This section presents the methodology for testing the third performance metric, that of learning trajectory and speed. It first discusses the issues involved and then goes on to detail experiments that will give a quantitative assessment of the effects of the hardware model.

#### 7.2.1 Introduction

The quality of a learning trajectory is very difficult to define. To some extent it can only really be measured by testing the performance of the final solution. In the previous two chapters it has been shown that for the synaptic weight noise technique the final solution has been enhanced for the fault tolerance and generalisation ability performance metrics. However, there must also be some control over the rate at which the solution is obtained as in the limit a complete search of the entire parameter space to find a global optimum is unacceptable. Ideally the learning scheme should be able to find a global optimum in a "short" period of time and in a way that is repeatable or robust for a wide variety of problems. In reality, a solution that meets the required criteria for the given problem in "an acceptable time" is adequate. Chapter 2 reviewed a selection of the optimisation techniques that are in use in the field. Depending on how
stringent are the criteria for judging the solution and as to what constitutes an "acceptable time", these techniques can be utilised. The synaptic weight noise technique must therefore be examined using this background. Essentially, it is assumed that the underlying back-propagation learning that approximates gradient descent is an acceptable learning scheme. If the synaptic noise does not degrade the learning to a significant extent, then the performance enhancements reported previously would be significantly more valuable.

This section addresses the issues of obtaining a quality measure for the learning trajectory of the noise affected network. The test experiments that are detailed seek to fulfil the requirements discussed in chapter 4. In summary, these were to verify the basic prediction of chapter 3, to analyse the underlying mechanisms and to assess the limitations of the technique. Further analysis of the additive noise model will also be carried out in comparison with the multiplicative noise results that are at the centre of the work of this thesis.

7.2.2 Measuring Learning Speed

The simplest method of assessing the quality of a learning scheme for its trajectory is to measure the rate at which it reaches the point of convergence to an acceptable solution. This is a measure that is widely used in the artificial neural network literature and is usually reported in terms of the number of *epochs* (iterations of the entire pattern set) taken for the network to converge to a solution. However, there are a number of considerations that need to be taken into account. These are listed below:

1. The first of these considerations is how the starting point in weight space affects the final solution. The common way of initialising the weights in a MLP is to use small random values. Therefore, any single trajectory is only of limited significance and may in fact be misleading in the general case. It is essential for many random starting points to be taken into account so that the underlying training mechanisms can be probed.
2. Another issue is that of the acceptability of the solution. For the localisation problem, for example, a classification error of zero is very difficult to obtain and in fact may not necessarily be the optimal solution anyway. Therefore, careful consideration of the stopping criteria must be made. Here the same stopping criteria that were used for the previous experiments, detailed in chapter 4, are used.

3. Continuing from the last issue, how should weight sets that find a local minimum with an unacceptable error be represented by the quality measure? Here it was decided to record simply the number of runs that failed to converge within an acceptable period and report the result alongside the average learning speed as a percentage "success factor".

4. A final issue is how to assess the distribution of the convergence times (in terms of epochs) to form a meaningful result for a learning scheme. The variance of the distribution will depend on how random the distribution of initial weights is. If for instance the initial weight range \( \beta \) is approximately zero then the variance in learning speed will be low; however for a greater range the variance will increase significantly. In this thesis the mean value of the distribution is used as an assessment measure along with the success factor discussed previously.

Therefore, taking all these considerations into account the learning speed can be used as a measure of learning trajectory quality. In section 7.3.2 the results for the character encoder at varying levels of synaptic noise are reported. These results confirm those reported previously in [48].

7.2.3 Analysis of the Enhancement Mechanisms

In chapter 3 predictions were made concerning the learning trajectory of a MLP trained with synaptic noise in the weights. These predictions stated that enhancements would be seen in the quality of the trajectory since the noise effectively
Chapter 7. Learning Trajectory and Speed – Performance Implications

distributes second order information into the learning mechanisms. In the analysis the second derivative term in the expanded cost function was described as a constructive term that could actually reduce the error via the noise injection. By using the second order information as a “look-ahead” mechanism, weight changes may be introduced that could increase the raw error, while indicating a move to a more promising area of weight space. In addition, by analysing the effect of the expanded cost function on the weight update equation the noise was described as producing an extra noise term giving statistical information at each iteration. The effect of the noise term is to take account not only of the weight currently being updated, but to add in a term that estimates what the other weight changes are likely to do to the output, and adjust the size of the weight increment/decrement as appropriate. These predictions form an explanation of the improvement in learning times reported in [48]. However, to verify explicitly that they are indeed the cause of the surprising enhancements is extremely difficult. Despite this, the second derivative term discussed in the noise analysis can be monitored for networks trained with and without synaptic noise and conclusions drawn. Results from this experiment for the character encoder problem are reported in section 7.3.3.

7.2.4 Assessing the Limitations of the Technique

To ascertain the limitations of the enhancements predicted in chapter 4 and verified in section 7.3.2, a harder problem is required. The character encoder problem was initially chosen because of its simple although non-trivial nature. However, during the simulation experiments carried out in the course of this thesis the problem has been shown to be robust to noise in the weights during the training phase. In the previous chapter synaptic noise was shown to have the effect of de-stabilising solutions that use the sloping section of the sigmoid curve. The character encoder problem has been shown to have solutions that do not require these sections. However, at higher levels of noise (> 40%) the learning becomes erratic and eventually the learning algorithm fails to find solutions.
Therefore, for different problems there will be a different acceptable noise level where performance-enhancing effects may be evident and where the noise will not be detrimental to the final solution. This can be explained by using the mathematical analysis of the expanded cost function where the terms can be viewed as having stabilising or de-stabilising effects on the learning trajectory. For high levels of noise (for any problem) there will inevitably be a disruptive influence on the learning trajectory, as the extra mechanisms of the noise mask the real task of reducing the error on the training set. For some problems these mechanisms will also be disruptive at lower noise levels. An example of this is the preference for solutions using the sigmoid extremes having a de-stabilising effect on the learning trajectory of problems that require the sloping region. Therefore, the limitations of the technique as a performance enhancer are problem dependent. To show this, the Henon map time series prediction problem is used to show a problem with a much lower acceptable noise level. The results of this experiment are reported in section 7.3.4.

7.2.5 Summary

This section has discussed the issues for assessing the quality of a learning trajectory. Experiments have been detailed to verify the predictions presented in chapter 3. The results of the experiments and the following discussion allow the underlying mechanisms to be analysed and the limitations of the technique to be assessed. The tests therefore fulfil the aims presented in chapter 4. The results of these experiments are reported in the following section.
7.3 Results

This section reports on the results of the experiments detailed in the previous section. The results and the ensuing discussion allow the predictions of chapter 3 to be verified and the underlying mechanisms to be understood. In addition, the limitations of the technique are probed.

7.3.1 Introduction

Many researchers have investigated ways of improving the learning trajectory of MLPs in the training phase, to enhance the reliability and reduce the time taken to converge to a solution. A selection of these techniques has been discussed in chapter 2. The predictions made in chapter 3, concerning the hardware model, suggested that improvements would be seen in the learning trajectory. The results presented here verify this prediction, although the enhancements seen are small compared with some of those reported for other techniques, reviewed in chapter 2. Also as a software enhancement scheme for accelerating learning, the computation time to calculate the random numbers required for the noise generation exceeds any gain made by the underlying mechanisms of the technique. However, the results do show that there is no price to pay in terms of excessive training times for the performance enhancements reported in the previous two chapters.

Using the background of the literature review and the aims of the simulation experiments detailed in chapter 4, this section presents verification simulations. Therefore carrying out the experiments detailed in the previous section results were obtained to verify the predictions of chapter 3, extract the underlying mechanisms and assess the limitations of the technique.
7.3.2 Learning Speed

It is not a straightforward task to verify the claim that synaptic noise will enhance the learning trajectory of a MLP in the training phase. However, by examining the learning time for networks trained at different levels of noise, some indication of the effect of that noise will emerge. As the starting point of the training is a random position in weight space, many hundreds of different cases must be considered to allow the actually properties of the training to become evident. Some initial weight configurations will naturally allow a solution to be found quickly, while others will inhibit the learning process. Fig. 7-1 shows two learning curves for networks trained with different amounts of synaptic noise. From this graph it could be concluded that the noise improves the learning time dramatically. However, despite the fact that both curves started at the same place in weight space other curves could easily be produced to confirm the exact opposite! Fig. 7-2 shows the distribution of 1000 training runs all starting from different random positions for the character encoder problem. Therefore it is clear that many runs are required to allow the underlying properties to emerge.

To test the predictions of chapter 3 and to confirm the results reported in [48], 1000 starting points were assessed for each level of synaptic noise. As discussed in the previous section, an upper limit was placed on the learning times above which the “run” was said to have failed. This “cut off limit”, depicted in Fig. 7-2, was set to 1000 epochs. The number of learning “failures” for each noise level is tabulated in Table 7-1. The mean of the remaining learning times is shown in Fig. 7-3. The reduction in training time seen for noise levels between 0% and 30% gives a clear indication of the learning enhancements predicted in chapter 3 and reported previously in [48]. Above this level the mean convergence time increases as the competing pressures of the other regularising processes mask the mean square error reduction. The significance of the variation in the success rate of the runs with different levels of noise is unclear and it is apparent that there are a number of different processes at work. The virtual targets algorithm has been shown to have the “hill-climbing” ability needed to escape from local minima,
Figure 7-1: Graph showing a typical pair of learning curves for the character encoder problem trained with the virtual targets algorithm.

Figure 7-2: Probability density function for learning times (in terms of epochs) for networks with different random starting points for the character encoder problem at 10% multiplicative noise.
[49], while it is clear from examining the distributions of the training times at each noise level that with more noise the variance in the distribution is increased. At higher levels of noise the stochastic process could also have the effect of aiding the learning trajectory in jumping out of minima, see Hanson [23]. Therefore, there are many processes at work and the exact reasons for the variation in the success rate is unclear.

<table>
<thead>
<tr>
<th>Noise Level (%)</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>98</td>
<td>202</td>
<td>159</td>
<td>105</td>
<td>59</td>
<td>74</td>
</tr>
<tr>
<td>Success Rate (%)</td>
<td>90.2</td>
<td>79.8</td>
<td>84.1</td>
<td>89.5</td>
<td>94.1</td>
<td>92.6</td>
</tr>
</tbody>
</table>

Table 7–1: Failures recorded at each noise level and the subsequent “success rate”.

![Graph showing the variation in mean learning time verses the multiplicative synaptic noise level used during the training phase for the character encoder problem.](image)

Figure 7–3: Graph showing the variation in mean learning time verses the multiplicative synaptic noise level used during the training phase for the character encoder problem.
Chapter 7. Learning Trajectory and Speed – Performance Implications

1.0 0.0 0.5 : -0.5 -1.0 -1.5 -2.0 -2.5 -3.0

Learning Epochs

Figure 7-4: The second derivative x error term trajectory for injected synaptic noise levels 0% and 7%

7.3.3 Enhancement Mechanisms

Having shown that the prediction of enhanced learning trajectory through the use of synaptic weight noise during learning is true for the character encoder problem at least, it is now important to assess the mechanisms by which this improvement is achieved. In chapter 3 the second derivative term in the expanded cost function was analysed and seen to be a constructive term in the learning process. The term was noted as including second order information to give a “look-ahead” property to the training. In addition, an extra term was seen to be added to the weight update equation which included statistical information about the effect of individual weight changes on the other weights. These two effects were concluded to be the mechanisms for the learning trajectory enhancement, previously reported in [48]. To substantiate these claims is extremely difficult. However, the second derivative term can be monitored numerically during a training run and might be expected to be reduced more rapidly with the introduction of noise. Fig. 7-4 shows a graph with two trajectories of the second derivative term calculated numerically - one for 0% noise and the other for 7%. The plots were produced using
a single starting position for the character encoder problem and clearly show that with noise injected into the training the second derivative term is reduced much more quickly. This may be interpreted as a tendency to make "better" weight changes as described in chapter 3. The implications of this graph are limited in that they show only two trajectories from one starting position. Nevertheless, they do indicate that the second order processes, being incorporated into the training by the noise, are taking part in the learning process. At levels of noise greater than 7%, the effect shown in Fig. 7-4 is exaggerated until at high levels of noise the learning becomes unstable and the increased learning times result. The level of 7% was chosen because it is visually clear what is happening and is also typical.

7.3.4 The Limitations of the Technique

In the previous two chapters training with noise has been shown to give enhanced generalisation ability and fault tolerance. However, at the same time limitations have been seen in the effects. These limitations are not so much to do with the performance improvements, but rather are more to do with the ability of the network to learn under the influence of the noise. In chapter 3 the first derivative term in the expanded cost function was noted as having a de-stabilising effect on certain weight configurations. This de-stabilisation has been shown to have a positive effect on the final solution for a number of problems. However, if reduction of the mean square error and the stabilisation of the other noise induced terms (as seen in the expanded cost function) cannot be reconciled, then the learning itself will be detrimentally affected. In particular, in chapter 6 it was shown that the first derivative term in the expanded cost function had the effect of de-stabilising solutions that had activations on the slope of the sigmoid curve. For problems that can be solved using "hard-limited" outputs (in the hidden and output layers), this form of regularisation can be combined with the mean square error reduction during learning. However, if the solution requires the use of the linear section of the sigmoid curves then the regularisation has
a de-stabilising effect on the learning trajectory, perhaps causing a significantly longer training time, or possibly even a failure to converge. For problems where this is the case, any subtle underlying learning trajectory enhancements caused by the second derivative term, seen in the previous results of this chapter, will be far out-weighed by the de-stabilising effect of the regularising first derivative term.

To show that this is indeed the case, a problem is required that will be de-stabilised by the regulariser during the training process. The character encoder has been shown to be robust to training with noise and has solutions where all the hidden layer neurons have activations that use the sigmoid extremes. Therefore the second derivative term in the expanded cost function comes into effect - producing improved learning trajectories. Here a different sort of problem is required. The Henon map time series prediction problem was chosen because it requires the use of analogue outputs. In such a network, solutions are preferred that have hidden nodes that use the linear regions of the sigmoid and only have small excursions into the non-linear region [10]. Therefore, utilising the characteristics of this hard problem, simulation experiments were carried out to examine the limitations of the synaptic weights noise technique of learning trajectory enhancement.

Using the learning parameters and network structure detailed in chapter 4 and the back-propagation learning algorithm, simulations were carried out with varying levels of synaptic noise injected during the training process. Fig. 7-5 shows three typical mean square error trajectories taken from the same initial starting position for noise levels of 0%, 2% and 4%. It can clearly be seen that the trajectory is detrimentally affected by the noise giving a much higher final mean square error for networks trained with more noise. To give a more statistically significant result, 100 different starting points were then considered and the average final mean square error assessed. The individual networks were all trained for 10000 epochs as after this length of “time” the learning curves have stabilised. The mean square errors were calculated using a forward pass of the
training pattern set without any noise injected to give the result for the underlying network. Fig. 7-6 shows the results of these simulations, where the solid line plots the mean and the error bars show the standard deviation of the results.

![Graph showing the mean square error learning curves for three levels of synaptic noise, 0%, 2% and 4% for the Henon map problem.](image)

Figure 7-5: Graph showing the mean square error learning curves for three levels of synaptic noise, 0%, 2% and 4% for the Henon map problem.

Having shown that the learning trajectory has been detrimentally affected for this hard problem the underlying causes can now be examined. As argued above, the Henon map problem requires the use of the linear section of the sigmoid function to form a good solution. Therefore by measuring the derivative of the hidden layer outputs, in a similar way to the experiments carried out in chapter 6, the region of the sigmoid curve being utilised can be judged. Considering the three single learning trajectories plotted above, all starting from the same initial weights, the trajectory of the average value of \( \sigma_j(1 - \sigma_j) \), for all the hidden units and patterns in the training set, was recorded during the training process. The results of this experiment are shown in Fig. 7-7. This result shows that the higher the noise level the lower the average value of the hidden layer derivative and, considering Fig. 7-5, the higher the resulting mean square error. From this result it can be surmised that the significance of the first derivative term in the expanded cost function increases with the level of noise and for problems
Figure 7-6: Graph showing the average mean square error for networks trained on the Henon map problem for 10000 epochs using increasing levels of synaptic noise.

where the linear region of the sigmoid curve is required, or preferred, learning is detrimentally affected.

7.3.5 Additive Noise

The results presented in the previous sections have all referred to the injection of multiplicative synaptic weight noise. In this section additive noise is under investigation although, as seen in the previous two chapters, the results reported are similar to those accumulated previously. To test the learning trajectory enhancements predicted in chapter 3, simulation experiments were carried out to probe the effects at varying levels of additive noise. For each level of synaptic noise 1000 different random starting points in weight space were used to train networks on the character encoder problem. The weights were taken from an initial range, $\beta$, of 1.0 to give a comparison with the multiplicative noise results. The average time taken for the networks to converge to a solution is plotted in
**Figure 7-7**: Graph showing the trajectory of the average value of the hidden layer output derivative for the three learning trajectories plotted above.

Fig. 7-8 and the number of failures is recorded in Table 7-2. As with the multiplicative noise case the graph shows that for low levels of synaptic noise the learning trajectory is not detrimentally affected for this robust problem. As the noise level is increased to higher levels the corresponding detrimental effect on learning is again evident. The results are therefore similar to the ones presented previously for multiplicative noise in Fig. 7-3 and in [48]. However, considering the success rate, a marked increase in the number of failures is noted. Although the exact reason for this is unclear, the result shows that the use of additive noise does not give the repeatable performance of the multiplicative case.

---

1 As with the multiplicative noise case, the "cut off limit" for the additive noise experiments was set to 1000 epochs.
Table 7-2: Failures recorded at each additive noise level and the subsequent "success rate".

<table>
<thead>
<tr>
<th>Noise Level (%)</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>209</td>
<td>298</td>
<td>389</td>
<td>426</td>
<td>444</td>
<td>483</td>
<td>507</td>
<td>504</td>
<td>524</td>
</tr>
<tr>
<td>Success Rate (%)</td>
<td>79.1</td>
<td>70.2</td>
<td>61.1</td>
<td>57.4</td>
<td>55.6</td>
<td>51.7</td>
<td>49.3</td>
<td>49.6</td>
<td>47.6</td>
</tr>
</tbody>
</table>

Figure 7-8: Graph showing the variation in mean learning time versus the additive synaptic noise level used during the learning phase for the character encoder problem.
7.3.6 Summary

The results reported in the previous section have fulfilled the aims set out in chapter 4. The results have verified the predictions of chapter 3, showing an enhanced learning trajectory for networks trained with noise in the weights, also confirming the results of [48]. In addition the underlying mechanisms of the improvements have been investigated and the enhancements have been explained by the analysis of the second derivative term in the expanded cost function. Following this the limitations of the enhancements, and in fact the limitations of all the improvements reported in the previous chapters, have been assessed using the Henon map time series prediction problem. The limitations have been shown to be due to the de-stabilising effects of the first derivative term in the expanded cost function that was the mechanism for the improvements seen in generalisation ability and fault tolerance. In problems where the minimisation of this term cannot be reconciled with the minimisation of the raw problem error then learning is detrimentally affected. However, for a variety of problems where the competing pressures of the mixture of error terms can be reconciled then enhancements, shown here and previously, are evident.

7.4 Chapter Summary

This chapter addresses the implications of learning with noise in the weights on the learning trajectory and speed. The issues involved when examining the learning trajectory are discussed and simulation experiments are detailed. The results of these experiments are reported and verify the basic prediction of chapter 3. The chapter also investigates the underlying mechanisms, and assesses the limitations of the technique. In addition the results and ensuing discussion bring together the enhancements seen in chapter 5 and chapter 6 and explains them in terms of regularisation terms in the cost function. The enhancements seen in the area of learning trajectory and speed verify the predictions of chapter 3, confirm
the results reported in [48] and are consistent with the implications of the noise in chapters 5 and 6.
Chapter 8

Conclusions

Of making many books there is no end, and much study wearies the body. Now all has been heard, here is the conclusion of the matter. Fear God and keep his commandments for this is the duty of man.

Ecclesiastes 12, 12-13

8.1 Introduction

This chapter brings to a conclusion the work presented in this thesis. Having constructed a model of imprecise analogue hardware implementations of neural networks, this thesis has investigated the implications of the model on the performance of the MLP. The work has shown that the model does not have a detrimental affect. In fact, enhancements to the final solution have been demonstrated in the areas of fault tolerance and generalisation ability. In addition, the model has been shown to give an enhanced learning trajectory for certain problems. This chapter draws together all these results and considers the implications for analogue hardware designs of neural networks and the use of synaptic noise as a performance enhancement scheme.

In chapter 3 a model of analogue hardware was presented and a mathematical analysis carried out predicting performance enhancements in the three metrics - fault tolerance, generalisation ability and learning trajectory and speed.
Chapter 8. Conclusions

Chapters 5, 6 and 7 report results of simulation experiments carried out on a variety of problems, detailed in chapter 4, to verify these predictions. In addition they discuss the underlying mechanisms and the limitations of the enhancements seen in the light of other performance enhancing techniques reviewed in chapter 2. Therefore having investigated the implications of the hardware model on the three neural network performance metrics this chapter can now draw conclusions about the implications for analogue hardware design and also the use of synaptic weight noise as an enhancement scheme.

The following sections discuss these conclusions, analysing the model for its accuracy and looking at the implications for the precision constraints of analogue designs. The issue of using synaptic noise as a performance enhancer is also considered, discussing the improvements reported in previous chapters and examining how and when the technique can be used for beneficial effect. Finally the issue of using the enhancement technique in a hardware implementation is briefly addressed.

8.2 Implications for Analogue Hardware

This section examines the implications of the learning enhancements seen in the previous three chapters, for analogue hardware design. Initially it looks at the analogue hardware model, considering its validity and how the resulting performance enhancements affect design constraints. Then the issue of whether it is precision or accuracy that is important in neural networks, is again discussed.

8.2.1 Introduction

This thesis has set about the task of considering the implications of analogue hardware imprecision on MLPs. By constructing a model of analogue hardware, chapter 3 showed that the necessary compromises made in the implementation
task have a significant detrimental affect on the precision of multiplication functions. Therefore, incorporating this into a model and implementing it in software, simulation experiments were carried out to investigate the implications of the hardware model on the MLP neural network performance metrics. These experiments have shown that by injecting noise into the weights in the forward pass function of an MLP during learning gives a significant improvement in all three performance areas. In particular it has been shown that the fault tolerance and generalisation ability of the final solution have been improved. Also under certain conditions the learning trajectory can be improved to give faster learning times. Assuming that during the software implementation the hardware model remains valid, conclusions can be drawn as to the implication of these results on analogue hardware system design. Applying these results to hardware should enable an assessment of the precision and accuracy requirements to be made. Also the results have shown that for some problems the capacity of the network will diminish below a level required to solve them, i.e. a larger hidden layer may be needed to solve the problem at high levels of imprecision. Differences may therefore occur in the optimum architecture of MLPs implemented in imprecise analogue hardware to those implemented in full floating-point software. This section explores these issues.

### 8.2.2 An Assessment of the Hardware Model

In chapter 3 a model of imprecise analogue hardware was constructed. The model was put together taking into account as many of the implementation issues as possible. In doing this an attempt was made to create a general model. Therefore the results of the simulation experiments should have at least general implications for the hardware designer. The implementation errors which were not included in the model are as follows:

- Interfacing signal quantisation.

- Offsets in the neuron outputs.
Inaccuracy in the threshold function.

During the experiments a multiplicative noise model (adding noise in proportion to the weight size), uncorrelated and with a zero mean was used. Also a comparison has been made with purely additive noise and the results have shown similar trends. Depending on the exact weight storage mechanism one or other of these cases will be a valid model, with additive noise being more generally valid. The case of noise with a non-zero mean (or systematic errors) has not been fully addressed in this thesis. Appendix B carries out a brief investigation of the implications, finding that another term is added to the cost function if the same mathematical analysis, as carried out in chapter 3, is performed. Although it is unclear as to what specific mechanisms are occurring, the simulation experiments carried out show that, depending on the sign of the mean, different effects are seen. However with a mean value close to zero only a small difference is noted, sometimes giving a positive performance effect but generally a negative one. These experiments were carried out on the character encoder problem only and it is expected that the results will be problem specific. Appendix B details these experiments and discusses the results. Another question that could be raised is as to what level of noise it is reasonable to expect. Generally the literature does not specifically answer this question although "a few percent of the maximum weight value" is a quoted figure [53]. The noise level used in the simulation experiments above exceed this significantly. Therefore, the conclusion could be drawn that more noise is required to give the performance enhancements detailed above. This issue will be returned to later and discussed at greater length.

From the above discussion it is clear that the model used in the work of this thesis is at least generally representative of analogue hardware. Individual implementations will have their own idiosyncracies and in some cases may vary significantly. However, generally the model is valid and the results of the subsequent simulation experiments have implications for analogue hardware design.
8.2.3 Precision vs. Accuracy

The argument of whether it is accuracy or precision that is of major importance in the implementation of MLPs has formed one of the central themes of this thesis. In chapter 1 the discussion was initiated using Kirk's definitions of the two terms. Accuracy was said to be "the degree of conformity to some recognised standard value" and precision, "the degree of agreement of repeated measurements of a quantity" [40]. During the subsequent discussion digital and analogue hardware were categorised in these terms. Digital hardware was stated as having very high precision and an accuracy in proportion to the number of bits. Analogue hardware, on the other hand, often has low precision but can be highly accurate with good design. In this thesis it has been shown that high precision is not a necessary requirement for learning to take place in a significant number of problems. The results presented here are therefore indicative of the fact that low precision, but highly accurate analogue hardware, can be used to implement problems that do not require analogue outputs. For problems where the output is required to vary in an analogue manner, low precision can be detrimental to the final solution. The differences between these two cases will be discussed in more depth in later sections. Here it is only necessary to state that for problems that require binary outputs, such as classifiers, low precision does not have a detrimental effect, but in fact gives significant performance enhancements.

The requirement of greater precision for some problems where the outputs are analogue, or even sometimes for very hard classifier problems, that require the use of the linear section of the sigmoid curve in the hidden layer, has been shown. In addition it has also been seen in the experiments carried out with the localisation problem that this can be remedied, to some extent, by using a larger number of neurons in the hidden larger. This result can also be applied to the hardware implementation design. By allowing the use of more hidden layer units than would perhaps be necessary for a full floating point software implementation, a correct solution can still be obtained in an imprecise analogue hardware environment.
8.2.4 Summary

The implications of the results of this thesis on the design of analogue hardware can therefore be summarised:

- For many problems high precision is not required when accurate analogue hardware is used.
- For some problems a larger hidden layer can overcome the detrimental effects caused by low precision.

Therefore despite the problem dependency of the results it can be concluded that the low precision of analogue hardware is not always detrimental to the training process and can actually have beneficial effects.

8.3 Synaptic Noise as an Enhancement Scheme

This section considers the implications of the hardware model on the performance of MLPs and discusses whether the improvements reported in this thesis imply that the model could be used as a performance enhancing technique. Initially synaptic weight noise is considered as an enhancement scheme for software implementations. Then, following this, the issue of including an amplified noise model in a hardware implementation is briefly discussed.

8.3.1 Introduction

This thesis has reported that by incorporating a model of imprecise analogue hardware into the learning phase of a MLP, performance enhancements can be expected for a number of problems. Therefore conclusions have been drawn as to the implications of this on the actual design constraints of analogue hardware implementations of MLPs. The question has also arisen as to the extent of the
improvements noted and when and how the technique could be used as an enhancement scheme. It is clear from the results presented in the previous three chapters that for a software implementation, synaptic weight noise can be used to give a much enhanced solution and in some cases even an enhanced trajectory to reach that solution. Specifically the solution has been shown to have improved the fault tolerance and generalisation ability over networks trained without the model included. Therefore for the software implementation case, the conclusion is clearly that the technique can be used as a performance enhancing technique. Here the questions are tackled of when and how the noise should be used to give maximum benefit. Also the issue has been raised as to whether there is a case for incorporating the technique into a hardware implementation. The noise levels used in the simulation experiments carried out here have been in excess of those that would normally be evident in analogue hardware implementations. In fact the designer of analogue hardware would normally take great pains to keep the noise levels to a minimum by careful design. Here, however, the issue arises as to whether the incorporation of an increased amount of noise in the synaptic multiplication would be beneficial.

8.3.2 A Summary of the Software Implementation Enhancements

As discussed above, the results of the previous chapters have shown that the inclusion of the hardware imprecision model into the synaptic multiplication has produced enhanced network performance in the three performance metrics. This section summarises the use of synaptic noise as an enhancement scheme in each of these areas.

Fault Tolerance

The results presented in chapter 5 have reported that the fault tolerance performance of an MLP trained using noise in the synaptic multiplication is much im-
proved over networks trained without. This result has been shown to be achieved by the noise spreading the computation throughout the network so that individual weight saliencies are low. Therefore if damage occurs in the weights the effect on the network output is kept to a minimum. During the tests, networks were trained on two problems, one artificial and the other a "real world" problem, and were tested using a fault model of random weights being set to zero. For both of the problems the fault tolerance performance was significantly improved. Compared with other fault tolerance enhancement methods reviewed in chapter 2 the synaptic noise technique is very simple - requiring no additional supervision. Other techniques have been shown to be complicated and also possibly require a significantly longer training time.

Generalisation Ability

Chapter 6 presented the implications of including the hardware model in the learning phase on the generalisation ability of the subsequent solutions. The results show that with noise injected the generalisation ability of the networks is significantly improved over networks trained without noise. The mechanism behind this improvement was argued as being due to the solutions with activations on the linear section of the sigmoid curve being de-stabilised. In this way the internal representation of the problem is much more robust to small changes in the activation due to variations in the input patterns. This effect was also argued as reducing the capacity of the network and thus having the potential to reduce over-fitting. During the experiments on the localisation problem the results showed that for some hard problems, where the linear section is required to form a solution, the de-stabilising effect of the noise can have a detrimental effect on the performance, in some cases prohibiting a solution being found. Although the results of the experiments carried out on this hard problem did not prove conclusively that the reduction in network capacity, due to the noise, was the direct cause of the variation in the generalisation error, an indication was nevertheless seen that it was a significant factor in the results. However, for this
hard problem, by adding more neurons in the hidden layer, the capacity of the network was increased so that a better solution could be achieved. These results show that the noise-trained networks are imbued with a robust hidden layer giving immunity to small changes in the inputs. In addition the noise can be used as a mechanism for controlling the network capacity dynamically during the training process to avoid over-fitting.

Comparing this technique with other generalisation improvement methods is very difficult, as often the enhancement schemes require highly specialised knowledge of a particular problem to achieve any improvement. Generalisation performance is a completely problem-dependent issue and so any comparison must be carried out using the same set of data. Therefore, it is very difficult and time consuming to obtain any accurate measure of relative performance. Here, it can be said that the synaptic noise technique gives enhanced generalisation ability for a number of problems. In some cases, such as the localisation task, it is necessary to have a detailed knowledge of the problem. This allows a careful use of the noise to be made, to achieve any enhancement and to avoid reducing the capacity of the network below that required to find a solution.

Learning Trajectory and Speed

In chapter 7 the issues involved in analysing the learning trajectory and speed were discussed and the implications of the synaptic noise technique in this area reported. The results show that for some problems where the de-stabilising of solutions using the linear section of the sigmoid curve is not detrimental to learning, enhancements in the trajectory are achieved. If the de-stabilisation has a detrimental effect on the learning then significantly longer learning times can be the result and in some cases the competition between the cost function terms may not be resolved. The experiments carried out showed that this was the case by analysing two types of problem. For the character encoder problem improvements were achieved in the learning speed through noise injection. For the Henon map time series prediction problem with its analogue output, the noise has a negative
effect on the learning performance. Therefore, these results give an indication of where the synaptic noise technique can be used to give performance enhancements and also where careful consideration of the effects must be made before incorporating it into a learning scheme.

To compare the improvements seen in the learning trajectory with other schemes is difficult because of the variety of optimisation methods. The particular method of optimisation that performs best for one particular task will undoubtedly be different for a different sort of problem. Here the ability of the network to learn in a "reasonable time" and to find an "adequate" solution is all that is required. Chapter 7 has shown the type of problems where the use of synaptic noise is acceptable and will give performance enhancements, and also the types of problem where care must be taken or the use of noise avoided. One of the issues that has not been investigated in this thesis is that of local minima. Hanson in his related work, [23], uses synaptic noise during learning as a means of getting out of local minima. By adapting means and standard deviations of noise distributions for each weight he reports local minima avoidance. The work described in this thesis uses a strategy that is much simpler than Hanson’s scheme. However, whether the model used here is capable of escaping from local minima is an issue that is still to be addressed and is clearly a topic for further research.

8.3.3 A Hardware Implementation

The previous section summarised the issues raised throughout this thesis on the topic of using synaptic weight noise as a software implementation performance enhancer. This section returns to hardware implementations and discusses the issue of how analogue imprecision can be used in a design to give performance enhancements. In the final section of this chapter the issue of whether it is necessary for learning to be carried out with the hardware in the loop at all, will be considered. Here, however, the issue is as to how much noise there is in an analogue implementation, how this corresponds to the levels of noise used in the simulations experiments and whether there are methods for reconciling the two.
As has been stated through the course of this thesis the amount of imprecision found in analogue hardware implementations is extremely design dependent. Therefore it is very hard to make any general conclusions as to how the noise levels used in the simulation experiments correspond to reality. Also the issue of how the noise should be incorporated into the algorithm is unclear. However, assuming that the quoted figure of “a few percent of the maximum weight value”, [53], is a valid assessment of the situation then some tentative conclusions can be made. During the course of learning the weight values have a tendency to increase in magnitude from the “small random values” typically used as a starting point. In a hardware implementation it is expected that some initial training would be carried out prior to “down-loading” weights onto the hardware. Therefore, the actual weight magnitude may well have increased from the small initial values to perhaps the expected levels of the solution. These are expected to be of the order of units, where the range from $-5$ to $+5$ would be typical for some problem$^1$.

Taking the additive noise case as being more generally correct and the maximum weight value to be 5, and noting that the level of noise used was in the range from 0.0 to 0.2, a few percent of 5 would be somewhere at the centre of this range. Therefore the simulations are reasonably realistic. For the multiplicative noise case the noise levels are obviously much greater than those expected in reality. Depending on the implementation, the task in question and the range of the weights being trained, it may be advantageous to introduce a supplementary noise source to give the performance enhancements seen in the simulation experiments. The actual implementation of such a requirement is not altogether unknown (see [40] and included references) and would therefore be possible. However, at the same time this would increase the complexity of the design significantly.

Therefore it has been seen that the possibility exists for incorporating additional noise sources into a hardware implementation of a MLP to give some of

\[1\text{Note, again, that this is extremely problem dependent and that it is very hard to say exactly what the range will be.}\]
the performance enhancements seen in the software implementation experiments of this thesis. The question of how hard this would be and how the noise levels could be controlled are very complex and are outside the scope of this thesis.

8.3.4 Summary

This section has discussed the issues of using the synaptic weight noise technique as a performance enhancer. The improvements seen during the course of this project have been summarised and the problem dependency issue again raised. From the results it is clear that for the majority of problems with binary state outputs the noise will have a beneficial effect. For other problems, careful use of noise may give an improved solution, although there may be a cost in the subsequent learning time and reliability of finding a solution. Other problems with analogue outputs may be detrimentally affected. The issue of incorporating the technique into a hardware implementation has also been touched upon, although the issues are complicated and really the whole subject is outside the scope of this thesis. However, the possibility of such an implementation exists and may be a topic for research in the future.

8.4 General Conclusions

This chapter has presented some specific conclusions as to the implications of the work reported in this thesis. The results have been applicable to analogue hardware design of MLPs and also as a performance enhancing scheme. This section makes a return to subjectivity and the wider field of neural networks and hardware implementations.

The majority of work published in the neural network field uses learning techniques that in optimisation terms are at best old-fashioned. The reasons behind this are perhaps the unwritten aim of the field to become a global panacea for all problems. Ideally neural networks are required to be a "black box" that can be
used without any implicit knowledge of the problem or indeed the actual neural network itself. Unfortunately neural networks at present do not have such a magical problem-solving capability. Therefore, detailed knowledge of the problem, the mechanisms needed to solve it and the requirements of the resultant network, are necessary for a good solution to be found. Also it is the case that some of the work carried out in the field of neural networks is merely replicating work carried out in other areas. For the field to progress, rather than trying to compete, the techniques of these often much older and theoretically well established fields must be embraced so that the experience can be used to the mutual advantage of both. This is perhaps not the romantic ideal that the first neural network researchers envisioned but must be a reality.

This thesis was motivated initially by hardware implementations of neural algorithms although has progressed into the realms of neural computation. The field of hardware implementations of neural networks is still young and it is at present unclear where neural network techniques will be used in the future. From the results of this thesis and the literature, it is clear that there is a strong problem-dependency in the accuracy and precision requirements of the hardware. This is true for both the analogue and the digital case. As in the general neural network discussion above, there is no ideal implementation that will be able to solve all problems. The future of neural network hardware must lie in problem specific design for individual applications. Also whether a learning capability is a necessary requirement could also be argued. Obviously for low-level processing in remote sensors, where there are many sensors and it would be impossible to communicate directly with all of them, a learning capability "on-chip" and "stand-alone" would be an advantage. However for higher-level systems where neural networks are required, learning could possibly be carried out remotely and parameters for the solution down-loaded as required. Ideally no extra learning in-the-loop would be necessary if the solution could be tolerant of any variations in the hardware. At present this is not the case - hence one of the motivations for the work of this thesis. Therefore, it is thought that problem specific hardware
implementations are a necessary topic for further research, keeping track of the developments in the neural network algorithms.

In chapter 1 this thesis set out to explore analogue imprecision in MLPs, considering the implications for analogue hardware and using it as a technique for software enhancement. In the following chapters a theoretical investigation was carried out and then intensive simulation experiments to verify predictions made from that theory. In conclusion, low precision in analogue hardware is not always detrimental to the training process, and under the right conditions can have beneficial effects. In addition, synaptic weight noise can be used as a performance enhancement scheme in the software domain.
Bibliography


Appendix A

Penalty Terms for Fault Tolerance Enhancement

A.1 Introduction

This appendix takes the extra cost function term, discussed in chapter 3 as giving enhanced fault tolerance, and looks at how it could be incorporated into a learning algorithm to give enhanced fault tolerance. The term is first discussed as a regularisation or penalty term in a learning cost function. Then the issue of producing an algorithm to minimise it in conjunction with a standard error is addressed.

In chapter 5 it was shown that the limitations of synaptic weight noise as a fault tolerance enhancement method depended only on whether the learning algorithm could train the network to solve a particular problem under the influence of the noise. Taking the actual mechanism and using it directly in learning will give a more heuristic approach and could solve this problem. This appendix explores this issue.

A.2 Definition of the Penalty Terms

This section looks at the issue of producing a learning algorithm that achieves a fault tolerant solution. A penalty term to be included in the learning algorithm is presented and its requirements addressed.
A.2.1 Introduction

In chapter 5 results were presented that showed that the fault tolerance performance was enhanced if the computation was spread evenly around the network. There the standard deviation of the weight saliency gave some measure of the fault tolerance performance. If a cost function could include a term that when minimised would reduce the individual sensitivity of the weights, then the fault tolerance would be enhanced. Using the mathematical analysis of chapter 3 of this thesis, it was shown that the derivative of the output with respect to individual weights was such a term.

The cost function is of the form:

$$
\epsilon_{\text{tot}} = \epsilon_{\text{mse}} + \lambda \epsilon_{\text{FT}}
$$

(A.1)

i.e. $\epsilon_{\text{tot}}$ is the total error, or cost, $\epsilon_{\text{mse}}$ is the usual mean square error, $\epsilon_{\text{FT}}$ is the fault tolerance penalty term and $\lambda$ some parameter controlling the degree of fault tolerance. Looking directly at the fault tolerance penalty term for pattern $p$:

$$
\epsilon_{\text{FT},p} = \sum_{k=0}^{K-1} \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \left\{ \frac{\partial o_{kp}}{\partial w_{ab}} \right\}^2
$$

(A.2)

i.e. the magnitude of the sensitivity of each weight-output relationship in a network. For the rest of this section the pattern number will be dropped to avoid confusion.

To minimise this cost function term the gradient is required. In other words the derivative of the error term with respect to the minimising weight. i.e. $\frac{\partial (\epsilon_{\text{FT}})}{\partial w_{cd}}$.

By calculating this gradient a learning algorithm can be produced to train a network on the above cost function (A.1). The gradient is given by:

$$
\frac{\partial (\epsilon_{\text{FT}})}{\partial w_{cd}} = \frac{\partial}{\partial w_{cd}} \left\{ \sum_{k=0}^{K-1} \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \left\{ \frac{\partial o_k}{\partial w_{ab}} \right\}^2 \right\}
$$

(A.3)

$$
= \sum_{k=0}^{K-1} \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \left\{ 2 \frac{\partial o_k}{\partial w_{ab}} \frac{\partial^2 o_k}{\partial w_{ab} \partial w_{cd}} \right\}
$$

(A.4)
A.2.2 The 3-Layer MLP Case

Considering the case of a three layer \( I:J:K \) network, there are two versions of (A.2). Considering first the sensitivity \( s \) of the output \( k \) to the output weight \( w_{kj} \):

\[
s_{kj} = \frac{\partial o_k}{\partial w_{kj}} \tag{A.5}
\]

To evaluate the derivative, the chain rule is first used:

\[
\frac{\partial o_k}{\partial w_{kj}} = \frac{\partial o_k}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}} \tag{A.6}
\]

i.e. \( \text{net}_k \) is the activation of output \( k \). The first partial derivative can immediately be evaluated:

\[
\frac{\partial o_k}{\partial \text{net}_k} = f'(\text{net}_k) = o'_k \tag{A.7}
\]

Also expanding out the second partial:

\[
\frac{\partial \text{net}_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \left\{ \sum_{j=0}^{J-1} w_{kj} o_j \right\} = o_j \tag{A.8}
\]

Therefore, the sensitivity \( s_{kj} \) is given by:

\[
s_{kj} = o_j o'_k \tag{A.9}
\]

The second version of (A.2) can be calculated by considering the sensitivity of an output to the hidden weight \( w_{ji} \):

\[
s_{ji} = \frac{\partial o_k}{\partial w_{ji}} \tag{A.10}
\]

Again using the chain rule the derivative becomes:

\[
\frac{\partial o_k}{\partial w_{ji}} = \frac{\partial o_k}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{ji}} \tag{A.11}
\]

From (A.7) the first partial can be evaluated:

\[
\frac{\partial o_k}{\partial \text{net}_k} = o'_k \tag{A.12}
\]

\(^1\)The notation for this appendix has been changed here to avoid confusion.
Appendix A. Penalty Terms for Fault Tolerance Enhancement

Expanding the second partial gives:

\[
\frac{\partial net_k}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left\{ \sum_{j=0}^{J-1} w_{kj} o_j \right\}
\]  

(A.13)

From (A.9) the output-weight derivative can be substituted, giving:

\[
\frac{\partial net_k}{\partial w_{ji}} = o_i o_j' w_{kj}
\]  

(A.14)

Therefore, the sensitivity \( s_{ji} \) becomes:

\[
s_{ji} = o_i o_j' o_k' w_{kj}
\]  

(A.15)

A.2.3 Summary

A cost function has been defined that includes terms that control the sensitivity of outputs to individual weights. As well as the usual mean square error term, there are now two extra terms, one for the output weights and the other for the hidden weights. Minimising all these terms should give a solution to a given problem with enhanced fault tolerance performance.

A.3 Learning Algorithms

This section discusses the issue of minimising the cost function presented in the previous section. Various techniques have been used in the literature, such as a successive quadratic programming algorithm [54]. Bishop [6] uses a more standard back-propagation method to minimise his “smoothness” constraint. Here a similar method is presented.

A.3.1 Introduction

Using a back-propagation of the error technique involves finding the gradient of the error term with respect to the individual weights, as described above.
Appendix A. Penalty Terms for Fault Tolerance Enhancement

In a three layer MLP there are two sets of weights - the hidden layer weights and the output weights, both of which are used to minimise the penalty terms. Therefore, having two extra penalty terms, gradients are found for each of the two sets of weights giving four equations. The following two sections contain the mathematical derivations of the gradients for error minimisation.

A.3.2 Gradient With Respect to the Output Weights

First of all, taking the output weight sensitivity term (A.9) and finding its gradient with respect to the output weight $w_{KJ}$, where $K$ and $J$ are not necessarily the same as $k$ and $j$ :-

$$\frac{\partial (s_{kj})}{\partial w_{KJ}} = \frac{\partial (o_j o_k')}{\partial w_{KJ}} = o_j \frac{\partial o_k'}{\partial w_{KJ}} = o_j \frac{\partial o_k}{\partial w_{KJ}} (A.16)$$

Considering the case where $K = k$, using information from (A.9) and assuming use of the sigmoid function, the partial derivative can be evaluated :-

$$\frac{\partial (o_k')}{\partial w_{k,j}} = o_j o_k' (1 - 2o_k) (A.17)$$

Therefore, the gradient of the sensitivity of an output weight $w_{kj}$ with respect to another output weight connected to the same output $k$, is given by :-

$$\frac{\partial (s_{kj})}{\partial w_{k,j}} = o_j o_j o_k' (1 - 2o_k) (A.18)$$

The gradient of the sensitivity of the same weight with respect to a weight connected to a different output is given by :-

$$\frac{\partial (o_j o_k')}{\partial w_{KJ}} = o_j \frac{\partial o_k'}{\partial w_{KJ}} = o_j (1 - 2o_k) \frac{\partial o_k}{\partial w_{KJ}} = 0 (A.19)$$

Therefore the weight adaptation term, from (A.4) is given by :-

$$\Delta w_{k,j} = -2o_j o_k' (1 - 2o_k) \sum_{j=0}^{J-1} o_j^2 (A.20)$$

Secondly, taking the hidden weight sensitivity term (A.15) and finding its gradient with respect to the output weight $w_{k,j}$, where $K$ and $J$ are not necessarily the same as $k$ and $j$ :-

$$\frac{\partial (s_{ji})}{\partial w_{KJ}} = \frac{\partial (o_j o_j' o_k w_{k,j})}{\partial w_{KJ}} = o_j o_j' \frac{\partial (o_k w_{k,j})}{\partial w_{KJ}} (A.21)$$
Again, considering the case where $K = k$ first, and taking the two variables of the product term separately, the first from (A.17), becomes:

$$\frac{\partial (o'_k)}{\partial w_{kJ}} = o_J o'_k (1 - 2o_k) \quad (A.22)$$

The second has two cases, $J = j$ and $J \neq j$:

$$\frac{\partial (w_{kj})}{\partial w_{kJ}} = \begin{cases} 0 & J \neq j \\ 1 & J = j \end{cases} \quad (A.23)$$

Combining (A.22) and (A.23) using the product rule, substituting into (A.21) and rearranging, gives the gradient of the sensitivity of an output $k$ to a hidden weight $w_{ji}$ with respect to an output weight connected to the same output $k$:

$$\frac{\partial (s_{ji})}{\partial w_{kJ}} = \begin{cases} o_i o_J o'_j o'_k w_{kj} (1 - 2o_k) & J \neq j \\ o_i o'_j o'_k (1 + w_{kj} o_j (1 - 2o_k)) & J = j \end{cases} \quad (A.24)$$

Considering the case where the outputs are different, i.e. $K \neq k$, the two derivatives in the product from (A.21) go to zero and hence the gradient does not exist. Therefore there are two weight adaptation terms from (A.4). The term for $J \neq j$ is given by:

$$\Delta w_{kJ} = -2o_J o'_k \sum_{i=0}^{I-1} \sum_{j=0, j \neq J}^{J-1} o_i^2 w_{kj}^2 \quad (A.25)$$

while the term for $J = j$ is:

$$\Delta w_{kj} = -2o_j^2 o'_k w_{kj} (1 + w_{kj} o_j (1 - 2o_k)) \sum_{i=0}^{I-1} o_i^2 \quad (A.26)$$

### A.3.3 Gradient With Respect to the Hidden Weights

Calculating the gradient for the two sensitivity terms with respect to a hidden layer weight $w_{JI}$ involves the evaluation of the derivative with respect to those weights. Therefore, starting with the output weight sensitivity term (A.9) the derivative is given by:

$$\frac{\partial (s_{kj})}{\partial w_{JI}} = \frac{\partial (o_J o'_k)}{\partial w_{JI}} \quad (A.27)$$
Appendix A. Penalty Terms for Fault Tolerance Enhancement

Considering the case where \( J = j \) and taking the partial derivatives of the variables in the product separately, the first from (A.9) gives:

\[
\frac{\partial (o_j)}{\partial w_{jI}} = o_j' o_j'
\]  
(A.28)

The second can also be evaluated using information from (A.15):

\[
\frac{\partial (o_k')}{\partial w_{jI}} = o_j' o_k' w_{kj} (1 - 2o_k)
\]  
(A.29)

Using the product rule, substituting back into (A.27) and rearranging, gives the gradient of the sensitivity of an output \( k \) to an output weight \( w_{kj} \) with respect to a hidden weight \( w_{jI} \):

\[
\frac{\partial (s_{kj})}{\partial w_{jI}} = o_j' o_k' (1 + o_j (1 - 2o_k) w_{kj})
\]  
(A.30)

For the case \( J \neq j \) the partial derivatives in the product (A.27) become:

\[
\frac{\partial (o_j)}{\partial w_{jI}} = 0
\]  
(A.31)

and

\[
\frac{\partial (o_k')}{\partial w_{jI}} = o_j' o_k' w_{kj} (1 - 2o_k)
\]  
(A.32)

Therefore, the gradient of the sensitivity term with respect to \( w_{jI} \), becomes:

\[
\frac{\partial (s_{kj})}{\partial w_{jI}} = o_j' o_j (1 - 2o_k) o_k' w_{kj}
\]  
(A.33)

Therefore there are two weight adaptation terms, from (A.4) the term for \( J = j \) is given by:

\[
\Delta w_{jI} = -2o_j o_j' \sum_{k=0}^{K-1} o_k^2 (1 + o_j (1 - 2o_k) w_{kj})
\]  
(A.34)

while the term for \( J \neq j \) is:

\[
\Delta w_{jI} = -2o_j o_j' \sum_{k=0}^{K-1} o_k^2 (1 - 2o_k) w_{kj} \sum_{l=0,l\neq J}^{J-1} o_j'^2
\]  
(A.35)

Secondly calculating the hidden weight sensitivity term (A.15) and finding its gradient with respect to a hidden layer weight \( w_{jI} \):

\[
\frac{\partial (s_{ji})}{\partial w_{jI}} = \frac{\partial (o_i o_k' o_j' w_{kj})}{\partial w_{jI}} = o_i w_{kj} \frac{\partial (o_k' o_j')}{\partial w_{jI}}
\]  
(A.36)
Appendix A. Penalty Terms for Fault Tolerance Enhancement

Again, considering the case where \( J = j \) first, and taking the derivatives of the variables in the product separately, the first from (A.29), is given by:

\[
\frac{\partial (o_k')}{\partial w_{ji}} = o_I' o_j' o_k' w_{kj} (1 - 2o_k) \quad (A.37)
\]

The second part, using information from (A.28), is:

\[
\frac{\partial (o_j')}{\partial w_{ji}} = o_I' (1 - 2o_j) \quad (A.38)
\]

Using the product rule, substituting the term terms back into (A.36) and rearranging gives:

\[
\frac{\partial (s_{ji})}{\partial w_{ji}} = o_I o_i' o_j' o_k w_{kj} (1 - 2o_j + o_j' w_{kj} (1 - 2o_k)) \quad (A.39)
\]

For the case \( J \neq j \), the partial derivatives in the product (A.36) become:

\[
\frac{\partial (o_k')}{\partial w_{JI}} = o_I' o_j' o_k' w_{kJ} (1 - 2o_k) \quad (A.40)
\]

and

\[
\frac{\partial (o_j')}{\partial w_{JI}} = 0 \quad (A.41)
\]

Therefore, the gradient of the sensitivity term with respect to \( w_{JI} \), becomes:

\[
\frac{\partial (s_{ji})}{\partial w_{JI}} = o_I o_i' o_j' o_k w_{kJ} w_{kj} (1 - 2o_k) \quad (A.42)
\]

Therefore there are two weight adaptation terms from (A.4). The term for \( J = j \) is given by:

\[
\Delta w_{JI} = -2o_I o_j' \sum_{i=0}^{I-1} o_i' \sum_{k=0}^{K-1} o_k' w_{kj}^2 (1 - 2o_j + o_j' w_{kj} (1 - 2o_k)) \quad (A.43)
\]

while the term for \( J \neq j \) is:

\[
\Delta w_{JI} = -2o_I o_j' \sum_{i=0}^{I-1} o_i' \sum_{k=0}^{K-1} o_k' w_{kJ} (1 - 2o_k) \sum_{j=0,j\neq J}^{J-1} o_j' w_{kj}^2 \quad (A.44)
\]
A.3.4 Summary

This section considers the requirements of a learning algorithm and evaluates gradient expressions allowing a error back-propagation algorithm to be implemented. The gradients allow the sensitivity terms (A.9) and (A.15) to be minimised along with the "normal" mean square error required to solve the problem. By minimising all three terms a solution that is optimally distributed across the network can be achieved.

A.4 Appendix Summary

The work of this appendix has described a method of incorporating a means to improve the fault tolerance performance of a problem solution into the training algorithm itself. While with synaptic weight noise fault tolerance enhancement has been shown to be a useful side effect, here a learning algorithm has been defined to introduce similar conditions into the network directly.

The extra terms in the cost function and the subsequent gradient calculations require a significant amount of extra processing and there is a high likelihood of more local minima because of the competing pressure of the three terms. However, if the computational overheads are not a problem and the local minima can be avoided a solution where the problem is solved and distributed throughout the network can be achieved.
Appendix B

Systematic Errors

B.1 Introduction

This appendix briefly investigates the issue of a non-zero mean noise model (or systematic errors). All the results presented in the main body of this thesis were obtained using a uniform noise model with a zero mean (i.e. $\lambda = 0$ in Fig. B–1). Here an offset is applied to the mean of the distribution to analyse the effect of introducing systematic errors into the learning MLP. By examining the three main performance metrics as before, general conclusions should be able to be drawn.

![Figure B–1: A noise distribution with a mean of $\lambda$.](image)

Mean

Noise Value

-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1 1.2

Figure B–1: A noise distribution with a mean of $\lambda$. 
Initially a mathematical expansion is carried out using techniques similar to those used in chapter 3. This analysis shows that noise taken from a distribution with a non-zero mean introduces another term into the cost function. Following this analysis simulation experiments are carried out to analyse the effect of this extra term on the MLP performance metrics.

**B.2 Mathematical Analysis**

To analyse the effect of introducing a non-zero mean into the noise distribution the mathematics of chapter 3 were again used. A Taylor expansion of the network output around a noise free weight set is carried out for multiplicative noise, and then substituted into the cost function giving:

\[
\epsilon_p = \epsilon_p(\{w_{\text{nominal}}\}) + \frac{1}{2} \sum_{k=0}^{K-1} \sum_{ab,cd} w_{ab} \Delta_{ab} w_{cd} \Delta_{cd} \left[ \frac{\partial \theta_{kp}}{\partial w_{ab}} \left( \frac{\partial \theta_{kp}}{\partial w_{cd}} \right) \right] \\
+ \epsilon_{kp} \left( \frac{\partial^2 \theta_{kp}}{\partial w_{ab} \partial w_{cd}} \right) + \sum_{k=0}^{K-1} \epsilon_{kp} \sum_{ab} w_{ab} \Delta_{ab} \left( \frac{\partial \theta_{kp}}{\partial w_{ab}} \right) \quad (B.1)
\]

Taking the time average over the training phase and noting that each individual noise source does not average to zero, but to a constant \( \lambda \), \( < \Delta_{ab} > = \lambda \), after some algebraic manipulation, gives:

\[
< \epsilon_{\text{tot}} > = \frac{1}{P} \sum_{p=1}^{P} < \epsilon_p > = < \epsilon_{\text{tot}}(\{w_{\text{nominal}}\}) > + \\
\frac{1}{P} \sum_{p=1}^{P} \sum_{k=0}^{K-1} \sum_{ab} \left\{ \frac{1}{2} \Delta_{ab}^2 w_{ab}^2 \left[ \left( \frac{\partial \theta_{kp}}{\partial w_{ab}} \right)^2 + \epsilon_{kp} \left( \frac{\partial^2 \theta_{kp}}{\partial w_{ab}^2} \right) \right] + \lambda w_{ab} \epsilon_{kp} \left( \frac{\partial \theta_{kp}}{\partial w_{ab}} \right) \right\} \quad (B.2)
\]

Therefore, as well as the two extra terms discussed in chapter 3, a further term is included in the cost function. At present it is unclear as to the effect of this term on the cost function and the learning. However, it can be seen that any increase or decrease in the magnitude of the error is dependent on the sign of \( \lambda \), the noise mean offset from zero. Despite the lack of clarity in the mechanisms involved with this extra term, simulation experiments can be carried out to investigate the effect of the non-zero mean noise on the performance of the MLP.
B.3 Simulation Experiments

To investigate the implications of the extra cost function term introduced by the non-zero mean noise distribution, simulation experiments were carried out using the character encoder problem. The simulations investigated both of the hardware models - multiplicative and additive noise. For the multiplicative noise case a constant noise level of 20% was used, while for the additive case the noise level was set to 200% for an initial weight range of 0.1. By using just one weight level for each of the types of noise injection the dimensionality of the possible tests was reduced. During the experiments the offset was calculated as a percentage of the noise level. i.e. for an absolute additive noise level of ±0.1 (200% at an initial weight range of 0.1), $\lambda = 0.1$ is equivalent to a 10% shift in the distribution. Therefore using the standard network and learning parameters defined above, simulations were carried out to test the noise affected MLP in the three performance areas - fault tolerance, generalisation ability and learning trajectory and speed.

B.3.1 Fault Tolerance

To test the networks trained with noise taken from a non-zero mean distribution for fault tolerance, 35 weights sets were produced for a range of levels of $\lambda$. Using the same fault model as described in chapter 5, random damage was inflicted on the networks and the classification error measured. This was carried out 100 times per weight set to give a clear statistical result. Fig. B-2 and Fig. B-3 show the result of this experiment for multiplicative and additive noise respectively. The solid lines plot the average classification error, while the error bars show the standard deviation of the results. The 5 lines on each graph correspond to damage levels of 2%, 4%, 6%, 8% and 10%. For the multiplicative noise case the value of $\lambda$ has a positive effect if negative and a detrimental effect if positive. For the additive noise case the inverse is true. Although it is important to note for
both cases that the rate of variation in performance with respect to a change in \( \lambda \) is slow. The values of \( \lambda \) used in the experiments were very large because at low levels (which incidentally are probably more representative of actual hardware) no variation in performance was seen. However, at high levels of \( \lambda \) some changes are evident in the network fault tolerance performance for the character encoder problem.

### B.3.2 Generalisation Ability

To test the networks for their ability to generalise the same 35 weight sets generated for each level of \( \lambda \) noted above were used. Although, as discussed in detail in chapter 6, for the character encoder there is no true test of generalisation, an artificial test of "input corruption" can be carried out. For each weight set each pattern was presented 200 times with different corruption and a statistical set of results taken. Fig. B-4 and Fig. B-5 show the result of this experiment for the multiplicative and additive noise cases respectively. Again the solid line is an average classification error for a certain level of input corruption and the error bars plot the standard deviation of the results. The 5 lines on each graph show the classification error for corruption levels of 0.1, 0.15, 0.2, 0.25 and 0.3. The results show that for the multiplicative noise case the value of \( \lambda \) has a positive effect if negative and a detrimental effect if positive. For the additive noise case a detrimental effect is seen irrespective of the sign of \( \lambda \). Again, however, the effects are minimal at the low levels of \( \lambda \) that might be expected in reality.

### B.3.3 Learning Trajectory

To test the learning trajectory of the networks trained with noise taken from a non-zero mean distribution the learning speed was used. For each level of \( \lambda \), networks were trained and the average learning time was recorded. An initial
Appendix B. Systematic Errors

Figure B-2: Graph showing the variation in fault tolerance performance for varying $\lambda$ with multiplicative noise.

Figure B-3: Graph showing the variation in fault tolerance performance for varying $\lambda$ with additive noise.
Appendix B. Systematic Errors

Figure B-4: Graph showing the variation in tolerance to input corruption (or generalisation ability) for varying $\lambda$ with multiplicative noise.

Figure B-5: Graph showing the variation in tolerance to input corruption (or generalisation ability) for varying $\lambda$ with additive noise.
weight range of 0.1\(^1\) was used in these experiments and 200 runs were carried out for each level of \(\lambda\). Therefore the results are statistically significant. Fig. B-6 and Fig. B-7 show the results of this experiment for the multiplicative and additive noise cases respectively. Again it is clear that the mechanisms cause a minimal effect on the learning times at low levels, while at high levels some detrimental effects are seen. Also, as in the previous two sections, the sign of \(\lambda\) leads to a different effect on the network performance.

### B.4 Appendix Summary

This appendix has shown that taking a noise model with a non-zero mean (or introducing systematic errors) can be viewed as adding a further term to the cost function. Although it is unclear as to the exact mechanisms of this term the results of the simulation experiments show the effect of the distribution offset on the performance of the MLP on the character encoder problem. The results show that for low levels of \(\lambda\) only small variations in the performance are noted, some positive and some negative depending on the sign of \(\lambda\) and the specific experiment. It is therefore clear that the learning enhancements seen in the main body of this thesis are valid for low levels of systematic error in the distribution of the noise.

\(^1\)Note that in the previous experiments in chapter 7, the weight range was set to 1.0. Here 0.1 was used to reduce the training times.
Appendix B. Systematic Errors

Figure B-6: Graph showing the variation in mean learning time for varying \( \lambda \) with multiplicative noise.

Figure B-7: Graph showing the variation in mean learning time for varying \( \lambda \) with additive noise.
Appendix C

List of Publications


