An Analysis of Overfitting in Particle Swarm Optimised Neural Networks

by

Andrich Benjamin van Wyk

Submitted in partial fulfilment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

November 2014

Publication data:

Andrich Benjamin van Wyk. An Analysis of Overfitting in Particle Swarm Optimised Neural Networks. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, September 2014.

Electronic, hyperlinked versions of this dissertation are available online at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

An Analysis of Overfitting in Particle Swarm Optimised Neural Networks

by

Andrich Benjamin van Wyk
E-mail: avanwyk@cs.up.ac.za

Abstract

The phenomenon of overfitting, where a feed-forward neural network (FFNN) over trains on training data at the cost of generalisation accuracy is known to be specific to the training algorithm used. This study investigates overfitting within the context of particle swarm optimised (PSO) FFNNs. Two of the most widely used PSO algorithms are compared in terms of FFNN accuracy and a description of the overfitting behaviour is established. Each of the PSO components are in turn investigated to determine their effect on FFNN overfitting. A study of the maximum velocity ($\mathbf{V}_{max}$) parameter is performed and it is found that smaller $\mathbf{V}_{max}$ values are optimal for FFNN training. The analysis is extended to the inertia and acceleration coefficient parameters, where it is shown that specific interactions among the parameters have a dominant effect on the resultant FFNN accuracy and may be used to reduce overfitting. Further, the significant effect of the swarm size on network accuracy is also shown, with a critical range being identified for the swarm size for effective training. The study is concluded with an investigation into the effect of the different activation functions. Given strong empirical evidence, an hypothesis is made that stating the gradient of the activation function significantly affects the convergence of the PSO. Lastly, the PSO is shown to be a very effective algorithm for the training of self-adaptive FFNNs, capable of learning from unscaled data.

Keywords: Particle Swarm Optimisation, Feedforward Neural Networks, Overfitting, Adaptive Neural Networks

Supervisors : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."

Edsger Dijkstra

"Everything that civilisation has to offer is a product of human intelligence; we cannot predict what we might achieve when this intelligence is magnified by the tools that AI may provide, but the eradication of war, disease, and poverty would be high on anyone's list. Success in creating AI would be the biggest event in human history. Unfortunately, it might also be the last."

Stephen Hawking

Acknowledgements

I would like to express my immense gratitude to the following people for their continued support and belief in me during the production this thesis:

- Professor Andries P. Engelbrecht for his supervision, mentorship, guidance and support throughout my research journey.
- My parents for their unwavering support and love.
- My friends and colleagues in the research group CIRG for their advice and friendship in good times and bad.

Contents

List of Figures v

List of Algorithms vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Objectives . . . 2
- 1.3 Contributions . . . 3
- 1.4 Dissertation Outline . . . 3

2 Background 5
- 2.1 Artificial Neural Networks . . . 5
  - 2.1.1 Multi-layer Perceptrons . . . 6
  - 2.1.2 Neural Network Training . . . 8
  - 2.1.3 FFNN Applications . . . 10
- 2.2 Overfitting . . . 10
  - 2.2.1 Measuring Overfitting . . . 12
  - 2.2.2 Architecture Selection . . . 12
- 2.3 Particle Swarm Optimisation . . . 16
  - 2.3.1 The PSO Algorithm . . . 16
  - 2.3.2 Algorithm Control Parameters . . . 20
  - 2.3.3 Swarm Diversity . . . 23

2.3.4 Neighbourhood Topologies 24
2.3.5 Guaranteed Convergence Particle Swarm Optimisation 25
2.3.6 Application of PSO to FFNN Training 26
2.4 PSO Neural Network Applications 27
2.5 Summary 30

3 Comparing PSO and GCPSO **31**
3.1 Architecture Selection 32
3.2 Data Sets 32
3.3 Experimental Methodology 35
3.3.1 Hypothesis and Significance Testing 35
3.3.2 PSO Configurations 35
3.3.3 Experimental Procedure 36
3.4 Results 37
3.5 Summary 45

4 PSO Maximum Velocity **46**
4.1 Experimental Methodology 46
4.1.1 $\mathbf{V}_{max}$ selection 47
4.1.2 Experimental Procedure 47
4.2 Results 48
4.2.1 Diversity 49
4.2.2 Accuracy 51
4.2.3 Discussion 54
4.3 Summary 56

5 PSO Velocity Parameters **61**
5.1 Experimental Methodology 61
5.1.1 Data Sets 63
5.1.2 PSO Configuration 64
5.1.3 Experimental Procedure 64
5.2 Results 65

5.2.1 Individual Data Set Results . . . 65
5.2.2 Favourable Parameter Regions . . . 66
5.2.3 Adverse Parameter Regions . . . 69
5.2.4 Discussion . . . 72
5.3 Summary . . . 73

6 PSO Swarm Size **74**
6.1 Experimental Methodology . . . 74
6.1.1 Swarm Size Selection . . . 74
6.1.2 Data Sets . . . 75
6.1.3 PSO Configuration . . . 75
6.1.4 Experimental Procedure . . . 77
6.1.5 Algorithm Measurements . . . 78
6.2 Results . . . 78
6.2.1 Training Accuracy . . . 78
6.2.2 Generalisation Accuracy . . . 80
6.2.3 Diversity . . . 83
6.3 Summary . . . 86

7 Considering Activation Functions **88**
7.1 Activation Functions . . . 88
7.1.1 Sigmoid . . . 88
7.1.2 Hyperbolic Tangent . . . 90
7.1.3 Linear . . . 90
7.2 Experimental Methodology . . . 91
7.2.1 Data Sets . . . 91
7.2.2 PSO Configuration . . . 91
7.2.3 Algorithm Measurements . . . 92
7.2.4 Experimental Procedure . . . 92
7.3 Results . . . 93
7.3.1 Training Accuracy . . . 94
7.3.2 Generalisation Accuracy . . . 97

7.3.3 Diversity and Convergence . . . 101
7.4 Summary . . . 107

8 Adaptive Activation Functions **109**
8.1 Sigmoid Adaptation . . . 110
8.2 Lambda-Gamma Learning . . . 111
8.2.1 Generalised *lambda-gamma* Learning . . . 111
8.2.2 PSO *lambda-gamma* Learning . . . 111
8.3 Experimental Methodology . . . 113
8.3.1 Data Sets . . . 113
8.3.2 PSO Configuration . . . 114
8.3.3 Algorithm Measurements . . . 114
8.3.4 Experimental Procedure . . . 114
8.4 Results . . . 116
8.4.1 Static Activation Functions . . . 117
8.4.2 Adaptive Activation Functions . . . 125
8.5 Summary . . . 129

9 Conclusions **131**
9.1 Summary of Conclusions . . . 131
9.2 Future Work . . . 133

Bibliography **136**

A Acronyms **149**

B Symbols **151**
B.1 Chapter 2: Background . . . 151
B.2 Chapter 3: Comparing PSO and GCPSO . . . 153
B.3 Chapter 7: Considering Activation Functions . . . 153
B.4 Chapter 8: Adaptive Activation Functions . . . 153

C Derived Publications **154**

List of Figures

2.1 Abstract Representation of a Multi-Layer Perceptron. 7
2.2 An illustration of overfitting. 11

3.1 E_T and E_G over 2000 training iterations on the Diabetes data set. 41
3.2 E_T and E_G over 2000 training iterations on the Glass data set. 42
3.3 E_T and E_G over 2000 training iterations on the Henon Map data set. . . 42
3.4 PSO $\mathcal{D}_{avg}$ over 2000 training iterations on the WDBC data set. Graph shows 30 individuals samples with median result indicated in red. 43
3.5 GCPSO $\mathcal{D}_{avg}$ over 2000 training iterations on the WDBC data set. Graph shows 30 individuals samples with median result indicated in red. 44
3.6 PSO $\mathcal{D}_{avg}$ over 2000 training iterations on the Iris data set. Graph shows 30 individuals samples with median result indicated in red. 44

4.1 Swarm diversity, per sample, vs. iterations for the WDBC data set with increasing $\mathbf{V}_{max}$ values. The median result is shown in red (diamonds) . 50
4.2 E_T for each value of $\mathbf{V}_{max}$ on the Logistic Map and Wine data sets. . . 52
4.3 E_G (log scaled) for each value of $\mathbf{V}_{max}$ on the Logistic Map and Wine data sets. 54
4.4 ρ_F for each value of $\mathbf{V}_{max}$ on the Logistic Map and WDBC data sets. . . 55

5.1 Example parallel coordinate plot of 7-dimensional data. 63
5.2 Parallel coordinate plot of all data set results, ranked according to E_G. Top 20% parameter configurations by rank are highlighted. Stippled lines indicate parameter configurations with absolute lowest E_G. 68

5.3 Parallel coordinate plot of all data set results, ranked according to E_G. Bottom 20% parameter configurations by rank are highlighted. Stippled lines indicate parameter configurations with absolute highest E_G. 70

6.1 E_T results over 1000 iterations for $n_s > 25$ on the Glass data set. 81
6.2 E_G results over 1000 iterations for $n_s > 25$ on the Glass data set. 82
6.3 $\mathcal{D}_{avg}$ results over 500 iterations on the Glass data set. 85

7.1 Sigmoid activation function with $\lambda = \gamma = 1$ 89
7.2 Hyperbolic Tangent 91
7.3 Training errors (log scaled) per iteration for linear FFNN samples on the Henon Map data set. 95
7.4 Training errors (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set. 96
7.5 Training errors (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set. 96
7.6 Generalisation errors (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set. 98
7.7 Generalisation errors (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set. 99
7.8 Generalisation errors (log scaled) per iteration for linear FFNN samples on the Henon Map data set. 100
7.9 Diversity (log scaled) per iteration for linear FFNN samples on the Henon Map data set. 103
7.10 Diversity (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set. 104
7.11 Diversity (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set. 104

8.1 Activation function gradients. 119
8.2 Gradient function input and output to activation functions with shallower ($\lambda = 0.5$) and steeper ($\lambda = 1.5$) gradients for a single input pattern. . . . 121

List of Algorithms

1 The *gbest* PSO algorithm. 19

List of Tables

3.1 Data sets and FFNN layer sizes used in the empirical work. 33
3.2 Comparison of PSO and GCPSO training errors on classification and regression problems. 38
3.3 Comparison of PSO and GCPSO generalisation errors on classification and regression problems. 39
3.4 Comparison of PSO and GCPSO generalisation factors on classification and regression problems. 40

4.1 PSO swarm diversity for varying $\mathbf{V}_{max}$ parameter values. 57
4.2 PSO training errors for varying $\mathbf{V}_{max}$ parameter values. 58
4.3 PSO generalisation errors for varying $\mathbf{V}_{max}$ parameter values. 59
4.4 PSO generalisation factors for varying $\mathbf{V}_{max}$ parameter values. 60

5.1 Best performing velocity parameters per data set. 66
5.2 Worst performing velocity parameters per data set. 67
5.3 Velocity parameters overfitting example. 68
5.4 Top 20% parameter configurations per data set. 69
5.5 Bottom 20% parameter configurations per data set. 71

6.1 Swarm sizes and corresponding iteration counts used in experimental work. 76
6.2 Optimal velocity parameter values per data set. 77
6.3 Training error results for various swarm sizes. 79
6.4 Training error pairwise Wilcoxon test results across swarm sizes. 80
6.5 Generalisation error results for various swarm sizes. 81
6.6 Generalisation error pairwise Wilcoxon test results across swarm sizes. . . 82

6.7 Generalisation factor results for various swarm sizes. 83
6.8 Generalisation factor pairwise Wilcoxon test results across swarm sizes. . 84
6.9 $\mathcal{D}_{avg}$ results for various swarm sizes. 84
6.10 $\mathcal{D}_{avg}$ pairwise Wilcoxon test results across swarm sizes. 85

7.1 Training error results for various activation functions. 94
7.2 Wilcoxon Rank Sum test results for training errors across various activation functions. 95
7.3 Generalisation error results for various activation functions. 97
7.4 Generalisation factor results for various activation functions. 100
7.5 Wilcoxon Rank Sum tests for generalisation factor results 101
7.6 Swarm diversity results for various activation functions. 102
7.7 Wilcoxon Rank Sum tests for diversities of divergent swarms across various activation functions. 102

8.1 Swarm diversity with a static sigmoid function using various λ values. . . 118
8.2 Wilcoxon Rank Sum tests for FFNNs using static sigmoid function with various λ values. 119
8.3 Training errors with a static sigmoid function using various λ values. . . 123
8.4 Generalisation errors with a static sigmoid function using various λ values. 124
8.5 Training accuracy of PSO *lambda gamma* trained FFNNs 125
8.6 Wilcoxon Rank Sum tests for adaptive network training errors 126
8.7 Generalisation accuracy of PSO *lambda gamma* trained FFNNs 127
8.8 Wilcoxon Rank Sum tests for adaptive network generalisation errors . . . 128
8.9 Generalisation Factor of PSO *lambda gamma* trained FFNNs 128
8.10 Wilcoxon Rank Sum tests for adaptive network generalisation factors . . 128

Chapter 1

Introduction

Swarm intelligence (SI) can be defined as the emergent, problem-solving behaviour that stems from the interaction of a communicative group of agents acting on their local environment [7, 26, 59]. Computational swarm intelligence is the collection of algorithmic models of such behaviours [26].

One such algorithmic model is the particle swarm optimisation (PSO) algorithm which has enjoyed much attention within the computational intelligence (CI) community, largely due to its effectiveness on a vast number of optimisation problems [21, 25, 26, 103]. The PSO algorithm is a population-based global optimisation algorithm that explores a given problem space by modelling the social behaviour of a flock of birds [19, 58].

Neural networks (NN), one of the seminal fields of CI [6, 17, 39, 77, 120], are mathematical models inspired by natural neural networks such as the human brain. One of the very first applications of the PSO algorithm was the training of feedforward neural networks (FFNN) [19, 57], a particular type of artificial neural network that consists of multiple, forwardly connected layers of artificial neurons [89].

Although the PSO algorithm has been studied both theoretically and practically as a neural network training algorithm [103], little has been done to investigate the effect of PSO training on the *generalisation* and *overfitting* capability of the the FFNN, which is the main objective of this study.

The motivation for this study is given in Section 1.1. The research objectives are given in Section 1.2. The primary contributions of the study are given in Section 1.3.

Finally, an outline of the thesis is given in Section 1.4.

1.1 Motivation

This section briefly discusses previous work that serves as the motivation for this study.

Generalisation accuracy, within the context of NNs, describes the network's ability to *generalise*, that is, to apply concepts learned from the training data to previously unseen data. Overfitting is the phenomenon where a NN has adequate performance on the training data, but poor performance on unseen data and thus also in real-world application where the data is sampled from [40].

Various techniques have been developed to reduce overfitting, such as: regularisation [8, 16], architecture selection [34, 44, 65] and early stopping [92]. However, these techniques have been developed in the context of gradient based training methods [103]. Furthermore, research by Lawrence and Giles [64] has shown that overfitting in neural networks is specific to the algorithm used to train the networks.

The motivation for this thesis is the understanding of the effect of PSO training on FFNN accuracy, in particular generalisation and overfitting. Further, how accuracy and overfitting may be improved or controlled by adapting the PSO algorithm or its parameters.

1.2 Objectives

The primary objective of this thesis is an empirical analysis of the effect of the PSO algorithm and its components on accuracy and overfitting when training FFNNs. The following sub-objectives are identified:

- To provide an overview of FFNN training and overfitting.
- To review the PSO algorithm, the algorithm's components and parameters and the approach used to train FFNNs.
- To establish a baseline of FFNN accuracy and overfitting behaviour when trained by PSO algorithms.

- To analyse the effect each of the major PSO algorithm parameters has on FFNN accuracy and overfitting.

1.3 Contributions

The novel contributions, in the context of FFNN training, of this study include:

- An empirical comparison of the PSO with the Guaranteed Convergence Particle Swarm Optimisation (GCPSO) in terms of FFNN accuracy and overfitting.
- An analysis of the effect of the $\mathbf{V}_{max}$ PSO control parameter on FFNN accuracy and the parameter's control of overfitting through the swarm diversity.
- A study of the PSO *inertia* and *acceleration coefficient* control parameters and their effect on FFNN accuracy and overfitting.
- An analysis of the PSO swarm size and its effect on swarm diversity and FFNN accuracy and overfitting.
- A comparison of FFNN activation functions and their indirect effect on PSO swarm divergence and thus the network's accuracy.
- An analysis of PSO training of FFNNs using adaptive activation functions and their use in improving FFNN accuracy and overfitting on scaled and unscaled data.

1.4 Dissertation Outline

The remainder of the thesis is structured as follows.

- **Chapter 2** provides a background review of NNs, FFNN training, overfitting and the PSO algorithm.
- **Chapter 3** gives an empirical analysis and comparison of two PSO algorithms: the *lbest* PSO and the GCPSO as well as establishing a baseline of FFNN accuracy and overfitting on a set of data sets used throughout the thesis.

- **Chapter 4** provides an analysis of the first major PSO control parameter: $\mathbf{V}_{max}$. The use of the parameter to control FFNN accuracy through the swarm's diversity is discussed.
- **Chapter 5** presents a study of the *inertia*, *social acceleration coefficient* and *cognitive acceleration coefficient* control parameters and their effect on FFNN accuracy and overfitting.
- **Chapter 6** discusses the PSO swarm size and the effect on swarm diversity when training FFNNs.
- **Chapter 7** compares three FFNN activation functions and their effect on the PSO during FFNN training.
- **Chapter 8** presents a PSO algorithm for training FFNNs with adaptive activation functions along with a comparison with regular FFNNs. The benefit and use of adaptive activation functions on overfitting and learning from unscaled data is also discussed.
- **Chapter 9** concludes the study with an overview of conclusions reached and possible future work.

The following appendices to the thesis are included:

- **Appendix A** provides a list of acronyms used in the work along with their definitions.
- **Appendix B** presents a list of mathematical symbols used throughout the thesis and their associated meaning.
- **Appendix C** lists the publications derived from this work.

Chapter 2

Background

This chapter introduces and gives background on the CI paradigms relevant to this study. NNs are discussed in Section 2.1, with specific focus on FFNNs and their training. Section 2.2 reviews overfitting and related literature. This is followed by a discussion on the workings of the PSO algorithm in Section 2.3 and the PSO algorithm's applications in Section 2.4. The chapter is summarised in Section 2.5.

2.1 Artificial Neural Networks

This section discusses artificial neural networks (ANN) with specific focus on multi-layer perceptrons (MLP) in Section 2.1.1. NN learning and FFNN applications are discussed in Sections 2.1.2 and 2.1.3 respectively.

ANNs, or simply NNs [5], are mathematical data models inspired by studies of biological NNs like the human brain. In general, NNs provide a method for learning arbitrary mappings from an input space (or data set) to an output space (or data set). These spaces may be real valued or discrete, contain noisy or missing values or be ill defined in some other way. It is exactly this versatility of NNs that make them useful for a large variety of tasks.

Most ANNs typically consist of an arrangement of artificial neurons (crude functional abstractions of biological neurons), often in layers, that are connected to each other with weights. As a mathematical model, these weights are the NN's free or adjustable

parameters. In order for the network to perform a particular mapping, appropriate values for the weights need to be found.

NNs are a cornerstone of the CI field and as such have been studied in great detail. This lead to the existence of many different types of NNs differing mostly in arrangement and function of the neurons as well as the methods used to learn the weights. An exhaustive list of NN types is beyond the scope of this study and the interested reader is referred to books by Bishop [5, 6] and Haykin [41] for more information.

2.1.1 Multi-layer Perceptrons

The particular NNs relevant to this study are called MLPs, or more commonly standard FFNNs. FFNNs consist of a number of layers of neurons where the first layer is called the *input* layer and the last the *output* layer. All layers in between the input and output layers are known as *hidden* layers. Although a FFNN can have any number of hidden layers, it has been shown that NNs with monotonically increasing, differentiable activation functions and enough hidden units can approximate any continuous function [47]. An illustration of a FFNN is given in Figure 2.1. It consists of three layers that are fully connected — weights exist between all units in consecutive layers. The input and hidden layers contain *bias* units, shown with dashed outlines in Figure 2.1. Bias units have no incoming signals but are fully connected to the next layer. The purpose of bias units is to adjust the threshold value of the activation functions in the units of the subsequent layer and thereby, assuming classification, offset the decision planes constructed by the NN [5](chapter 3). Typically, bias units have a constant activation value of negative one, which is then adjusted by their outgoing weights.

As shown in Figure 2.1 the hidden and output units contain *activation functions*. An activation function serves as an abstraction for the action potential found in biological neurons. The input layer units do not normally have activation functions but simply pass a pattern's input vector through.

In most real-world cases it is not possible to obtain a data set that completely describes the true population it is sampled from. Therefore, a NN typically learns from a data set sampled from the true data population. Let a data pattern consist of an input vector $\mathbf{z}_p$ of dimension I and an associated target vector $\mathbf{t}_p$ of dimension K. The output

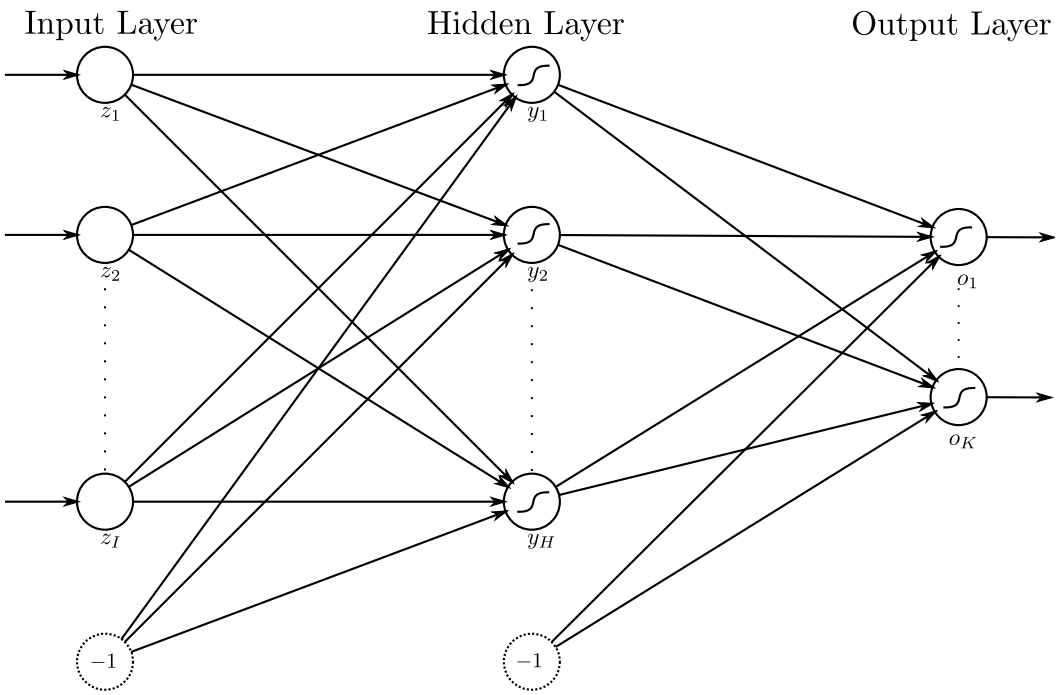


Figure 2.1: Abstract Representation of a Multi-Layer Perceptron.

neural network

of the NN is calculated by feedforward pass of the input vector. A feedforward pass computes the activation of each hidden and output unit as follows [5](chapter 4):

$$y_{j,p} = f_{y_j}(\sum_{i=1}^{I+1} w_{ji} z_{i,p}), \quad \forall\, j \in \{1, ..., J\} \tag{2.1}$$

$$o_{k,p} = f_{o_k}(\sum_{j=1}^{J+1} w_{kj} y_{j,p}), \quad \forall\, k \in \{1, ..., K\} \tag{2.2}$$

where o_k is the k^{th} output unit, w_{ji} is the weight from input unit z_i to hidden unit y_j and w_{kj} is the weight from hidden unit y_j to output unit o_k; functions f_{o_k} and f_{y_j} represent the activation functions for units o_k and y_j respectively.

A typical activation function that is used is the sigmoid function since it is differentiable, monotonically increasing and yields an output in $(0, 1)$:

$$f(net) = \frac{1}{1 + e^{-\lambda(net)}} \tag{2.3}$$

where λ controls the function's steepness.

Application areas of FFNNs include amongst others image analysis [23], real-time biometric data recognition, for example face detection [45], and game learning [4].

2.1.2 Neural Network Training

Learning in FFNNs is the process of finding a set of weights that realise a particular task. In general, the FFNN is presented with a pattern's input vector which is fed through the network yielding an output. The weights are then adjusted such that the error between the NN's output and the target output is minimised. This type of learning is known as *supervised learning*, as the learner is minimising the error between its output and a known target output. Formally, the task of learning a data set D_T of size P_T is finding the function $f_{NN} : \mathbb{R}^I \longrightarrow \mathbb{R}^K$ by minimising the empirical error:

$$\mathcal{E}(D_T; \mathbf{w}) = \frac{1}{P_T} \sum_{p=1}^{P_T} (F_{NN}(\mathbf{z}_p, \mathbf{w}) - \mathbf{t}_p)^2 \tag{2.4}$$

where F_{NN} is the FFNN described by the weight set $\mathbf{w}$, and $\mathbf{t}_p$ is the target output of pattern p [26](chapter 3).

In practice it is necessary to estimate the network's performance on data patterns not contained in the training set. Performance on unseen data is often referred to as the NN's *generalisation performance*. The data set D is divided into a training set D_T, and a generalisation set D_G. The latter is used to estimate the generalisation performance. Let $\mathcal{E}_T$ and $\mathcal{E}_G$ denote the errors on D_T and D_G respectively.

The process of training the NN is automated by a *training algorithm* (also called a learning algorithm). The goal of the learning algorithm is to minimise the objective function given by Equation (2.4) by adjusting the NN weights. Minimisation ceases when an acceptable error or a maximum number of epochs have been reached. A large variety of NN training algorithms have been developed and can broadly be classified into two classes: local and global optimisation algorithms. With local optimisation algorithms optimisation usually starts at a single point in the search space and uses only local search space information. Local optimisation algorithms are therefore more prone to getting stuck in a local optimum. Backpropagation training using gradient descent (GD) is an

example of a local optimisation NN training algorithm [112]. With global optimisation the algorithm attempts to find the global optimum by searching a larger part of the search space through the use of global search space information, usually starting at a number of different points. LeapFrog optimisation is an example of a global optimiser [95, 96].

Regardless of the optimisation method used, the weights of the NN must be initialised to appropriate values. *Weight initialisation* is dependent on the activation function. A simple strategy, applicable to activation functions with domains centred around zero, is random initialisation in the range $(-0.5, 0.5)$. Initialisation with small values around 0 is appropriate since this will lead to mid-ranged activation output, avoiding bias toward a particular solution [26](chapter 7). Work by Wessels and Barnard [113] has shown that a good range for initialisation is $(\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}})$, where $fanin$ is the number of incoming weights for a specific unit.

An important consideration in NN training is the selection of an appropriate network architecture for the learning problem at hand. The number of units contained in the input and output layers is derived from the problem. For the input layer the number of units is usually equal to the dimension of the pattern input vector — the number of independent variables, I, plus a bias unit. For the output layer size a number of strategies can be followed. For a function approximation task the output layer size is the dimensionality of the target space — the number of dependent variables. For classification problems, considering a data set with T number of classes, the researcher can use a single output unit and scale the class target values to the activation range of the unit, dividing the range into T intervals. The scaling of output targets to numerically smaller ranges has been shown to have the disadvantage of increasing training time [28]. Alternatively, T separate output units may be used, which has the disadvantage of increasing the number of weights in the network and therefore the dimensionality of the search space of the training algorithm. A compromise is to use a binary encoding for the T classes, where each class is represented by a bit string. In this case the number of output units K is equal to the number of bits it takes to represent the classes i.e. $K = \lceil \log_2 T \rceil$. The result is fewer than T output units and each unit uses the full activation range.

Although also problem specific, choosing the hidden layer size is not as straightfor-

ward as for the input and output layer sizes. This is because the number of hidden units is the primary factor that determines the network's learning ability — the complexity of the function it can approximate, or mapping it can learn [5](chapter 1). No rule exists for determining the number of hidden units necessary to learn complex real world tasks. However, VC-dimensions may be used to determine an upper bound for the number of hidden units [5]. Intuitively, the simplest method of selecting an appropriate architecture would be an exhaustive search using trial and error: select an architecture and evaluate the performance. If the generalisation error is too large, a different architecture is chosen [5](chapter 9). Sietsma and Dow [94] have, however, shown that smaller networks will yield better generalisation performance on average. If too few hidden units are chosen though, the network will be incapable of accurately modelling the data. An excess of hidden units on the other hand may cause the network to accurately model the noise in the data set or memorise training patterns, resulting in poor generalisation. This phenomenon is called *overfitting*. A detailed discussion of overfitting and network architecture selection is given in Section 2.2.

2.1.3 FFNN Applications

FFNNs have the ability to learn non-linear and arbitrary mappings between input and target spaces of a data set. Because of this, FFNNs have a wide area of application. The two tasks they are used for in this study are classification and time series regression. With a classification task the aim is to predict the class of an input data vector. Examples of classification data sets can be found at the UCI machine learning repository [1]. Time series regression/forecasting is an example of function approximation with the exception that the function output values are autocorrelated. In this case the NN is trained to approximate the functional mapping.

2.2 Overfitting

This section defines the concept of overfitting along with the causes and consequences thereof. This is followed by an overview of NN architecture selection in Section 2.2.2.

Overfitting is the statistical phenomenon where a model has adequate performance

on the training data set, but relatively poor performance on unseen data. Hawkins describes overfitting as the violation of the principle of parsimony, or Ockham's Razor, in that the model uses an over complicated architecture to describe the data mapping [40].

Fundamentally, overfitting occurs when the NN memorises the patterns and the noise contained in the training set. Consequently the network has poor performance on the generalisation data set.

Figure 2.2 illustrates the occurrence of overfitting during NN training. Initially, both the training and generalisation errors decrease over time as the network learns. At a specific point, the training error continues to decrease while the generalisation error starts to increase. This is the point of overfitting.

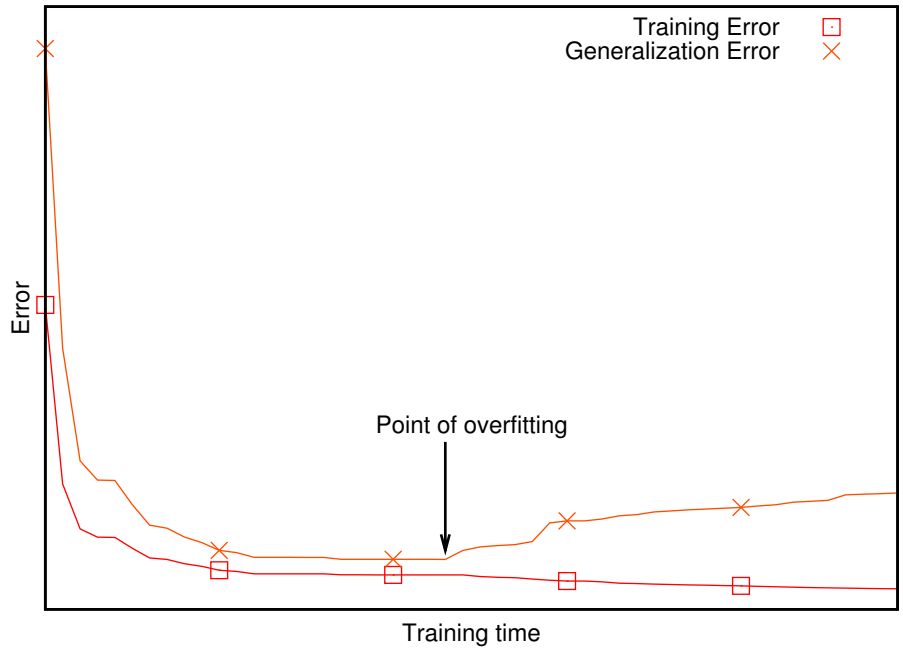


Figure 2.2: An illustration of overfitting.

Three conditions are necessary for overfitting to occur. First, it is necessary for the NN to have too many weights (and therefore an excess of free parameters), due to having too many hidden units and redundant or irrelevant input units [5](chapter 1). It is these additional parameters that the NN uses to model the noise or memorise the training set.

Secondly, it is necessary for the data set to contain noise. Finally, there must be enough training time for the NN to learn the noise contained in the data set. The amount of training time necessary for the NN to start overfitting is dependent on the problem.

Overfitting is disadvantageous for a number of reasons. Overfitting directly leads to the network performing worse on any data not included in the training set and may therefore perform badly during real world application. Not only does the network memorise the training data, it also learns noise and outlier data. This could lead to variation in the correct prediction of non-noisy patterns [40]. Another consequence of overfitting is that the training error is no longer a valid measure of accuracy. This is problematic in cases where the generalisation error is not a reliable estimate of generalisation ability, for example in cases where the generalisation data set is too small or unavailable.

2.2.1 Measuring Overfitting

The Röbel generalisation factor was developed by Röbel as a method for measuring the degree of overfitting given a training and generalisation error [88]. The generalisation factor is calculated as the ratio between the training and generalisation errors, i.e. $\rho_F = \frac{MSE_G}{MSE_T}$. A ratio of $\rho_F \leq 1$ is indicative of valid generalisation as it implies $MSE_G \leq MSE_T$. Conversely, a $\rho_F > 1$ means the generalisation performance is worse than the training performance which indicates overfitting. This study uses the generalisation factor ρ_F as both an indicator and measure of the overfitting of an algorithm on a particular data set.

2.2.2 Architecture Selection

Architecture selection broadly concerns the selection of an appropriate number of input and hidden units as well as the number of weights for the learning problem at hand — enough to accurately learn the mapping, but less than is necessary for overfitting to occur. Three main approaches exist for selecting optimal network architectures:

Regularisation: Regularisation relies on the assumption that smoother network mappings will tend to avoid overfitting. Regularisation attempts to create smoother

network mappings by adding a penalty term to the error function being optimised:

$$\mathcal{E} = \mathcal{E}_T + v\Omega \tag{2.5}$$

where $\mathcal{E}_T$ is the standard error being optimised (refer to Equation (2.4)) and v is a parameter controlling the influence of the penalty term, Ω [5]. One of the simplest forms of regularisation is *weight decay* where the intention is to drive small weights to zero [8, 16, 38], effectively removing the weight from the architecture. The penalty function for weight decay is defined as the sum of the squares of the weights in the weight set, $\mathbf{w}$:

$$\Omega = \frac{1}{2}\sum_i w_i^2 \tag{2.6}$$

In terms of conventional curve fitting, weight decay is a form of *ridge regression* and it has been shown empirically that this form of regularisation can lead to improved generalisation [43]. Further heuristic justification of weight decay is given in [5](chapter 9). A disadvantage of the method is that small weights and large weights are penalised equally. A potential solution to this problem is the use of hyperbolic or exponential penalty functions that penalise smaller weights more than large weights [38]. A further disadvantage is the possible addition of local minima due to the additional term in the objective function.

A closely related alternative to ridge regression that also limits the complexity of the network is *early stopping*, also known as *stopped training* [92]. With early stopping a large number of hidden units is chosen for the network. The network is then trained as usual. As soon as an increase in the generalisation error is witnessed (as in Figure 2.2), training is stopped, leading to a network with optimal estimated generalisation performance [92].

An example of an early stopping condition developed for gradient-based learning algorithms is the GL_5 stopping criterion — a specific instance of the more general GL_α stopping criterion [81]. The criterion measures the *generalisation loss* (GL) during training:

$$GL(t) = 100\left(\frac{\mathcal{E}_g(t)}{\mathcal{E}_{opt}(t)} - 1\right) \tag{2.7}$$

where $\mathcal{E}_g(t)$ is the generalisation error at time step t and $\mathcal{E}_{opt}(t)$ is defined as

$$\mathcal{E}_{opt}(t) = \min_{t' \leq t} \{\mathcal{E}_g(t')\} \tag{2.8}$$

Training is stopped as soon as the generalisation loss exceeds a threshold, α.

Advantages of early stopping include that it requires less training time, and that it can be applied to large networks with no additional overhead in computational costs. An analysis of early stopping has been conducted by Sarle [92].

Many other forms of regularisation and penalty functions exist and thorough studies into network regularisation have been conducted by Girosi *et al.* [36] and Williams [115].

Network Construction (Growing): With NN growing training starts on a network with a small number of hidden units. Training continues until the optimisation algorithm becomes stuck in a local minimum at which point additional hidden units are added with randomly initialised weights [34, 44, 50]. A number of difficulties exist in a network construction process: when exactly a new unit needs to be added, when to stop growing the network, and how to connect the new unit while avoiding a restart of the training process [26](chapter 7). An example of a learning architecture that uses growing is a *cascade-correlation* network [29]. With the cascade-correlation learning architecture a learning algorithm is used that creates and installs new units. Although new units are fully connected in the architecture, the new unit's incoming weights are frozen and only the outgoing weights are trained. At the time a unit is added, an attempt is made by the training algorithm to maximise the correlation between the output of the new unit (determined by the fixed weights) and the residual error signal. Training is stopped when no significant improvement is witnessed after a number of iterations (controlled by a user set parameter).

Kwok and Yeung [63] conducted a survey of NN construction techniques for regression problems and concludes that computational complexity is a concern in growing algorithms. In networks where complex hidden units (for example *adaptive spline network* units or *projection pursuit regression* units) are used, it is often necessary

to freeze the weights of previously installed hidden units in order for the learning algorithm to cope with a new unit. Learning weights layer-by-layer or restricting the hidden unit activation function to a parametric form can also be used to reduce computational complexity.

Network Pruning: NN pruning involves removing unnecessary weights, hidden units or input units from the network. Training starts on an oversized network from which weights or neurons are removed either during training or after solution convergence [5](chapter 9). The decision of which parameters to remove is in most cases related to a pruning heuristic that measures the parameter's *salience*, i.e. the relevance the parameter has to the network's accuracy. A simple measure of salience is the increase in the training error that occurs when the specific parameter is removed [5](chapter 9). If parameters with a small salience are removed the training error will not drastically increase, but the network is more likely to generalise well [101].

An example of a pruning algorithm is the Optimal Brain Damage (OBD) algorithm developed by Le Cun *et al.* [65]. The OBD algorithm uses second-order derivative information (an approximation of the network's Hessian matrix) to measure the salience of network weights. It is, however, necessary for the network to be trained to a local minimum before the approximation can be computed. Weights are sorted in order of decreasing salience and a number of low salience weights are removed by setting them to a constant zero value. The process of training and removal is repeated while an acceptable error is still obtained by the network.

The OBD algorithm is a sensitivity analysis based pruning technique. A variety of other pruning techniques exists that include evolutionary approaches [62, 83, 114] and statistical hypothesis testing based techniques [3, 81, 99].

Selecting the optimal architecture is however not necessarily enough to prevent overfitting from occurring. Lawrence and Giles have shown that the degree of overfitting is dependent on the training algorithm even when an appropriate number of hidden units is used for the problem [64]. In their empirical analysis a NN was used to approximate a sinusoidal function perturbed with noise. Four hidden units were sufficient for a GD

algorithm to learn the function with very good generalisation error. Even when using a large hidden layer size (100 units) no noticeable overfitting occurred. When the same four hidden unit network was trained with a conjugate gradient (CG)[2] training algorithm significant overfitting occurred becoming worse as the hidden layer size increased.

These observations show that overfitting as a phenomenon is specific to the *training algorithm* used, and that additional mechanisms may be necessary to obtain good generalisation performance and to avoid overfitting even when an optimal architecture has been selected [103]. In order to develop any such mechanisms it is first necessary to study the overfitting behaviour in the context of a specific NN training algorithm. This study investigates the overfitting phenomenon in the case where PSO is used as a FFNN training algorithm.

2.3 Particle Swarm Optimisation

This section describes the PSO algorithm. A detailed explanation of the algorithm is given in Section 2.3.1. Sections 2.3.2, 2.3.3 2.3.4 discuss the PSO control parameters, swarm diversity and PSO neighbourhood topologies respectively. Finally, the GCPSO algorithm is described in Section 2.3.5.

The PSO algorithm is a stochastic function optimisation algorithm first published in 1995 by Eberhart and Kennedy [19, 58]. It is a *population-based* search algorithm maintaining a group of entities called *particles* in a collection known as a *swarm*. The algorithm is inspired by the complex movements and social interaction of birds within a flock [58]. Conceptually, particles are flown with specific velocity through the search space, attracted by potentially good solutions found either individually or by the rest of the swarm. The swarm is usually arranged in a predefined structure that governs the communication between the particles called a *neighbourhood topology*.

2.3.1 The PSO Algorithm

The PSO algorithm described here is the Shi and Eberhart modified PSO algorithm that uses a global best neighbourhood topology and includes the use of an *inertia* weight [93]. The inertia weight is discussed in Section 2.3.2. The algorithm is a single solution PSO

algorithm, meaning there is a single objective for the optimisation process, which yields a single solution as the result.

Particles have a number of properties associated with them; for a particle i the properties are:

Position: Let the current position of particle i at time step t be denoted by $\mathbf{x}_i(t)$. This is a position within the optimisation problem's hyper-dimensional solution space (also called a search space). Let n denote the search space dimensionality. The particle's current position represents a solution to the optimisation problem.

Velocity: Each particle has a velocity vector, $\mathbf{v}_i(t)$, that represents a step size of the particle in the search space.

Solution Quality: A particle's solution quality is calculated from the particle's current position using the optimisation problem's *objective function*, f. Let $f(\mathbf{x}_i(t))$ denote the quality of solution $\mathbf{x}_i(t)$.

Personal Best Position: Each particle also maintains the best position it has found during the entire optimisation process, denoted by $\mathbf{y}_i(t)$.

Neighbourhood Best Position: Each particle references the best solution found in the particle's neighbourhood. The neighbourhood topology is responsible for tracking the best solutions within each neighbourhood. In the case of the global best topology, this is the best solution within the entire swarm. Let $\hat{\mathbf{y}}(t)$ represent the neighbourhood best position.

At the start of the PSO algorithm the particles are initialised to random positions within the search space: $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})^n$, with $\mathbf{v}_i = \mathbf{0}$ and $\mathbf{y}_i(0) = \mathbf{x}_i(0)$. The algorithm proceeds by updating each particle's position as follows:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t), \quad \forall\, j \in \{1, ..., n\} \tag{2.9}$$

The velocity is calculated as:

$$\begin{aligned} v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + \\ c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)], \quad \forall\, j \in \{1, ..., n\} \end{aligned} \tag{2.10}$$

The velocity update has three distinct terms that influence the overall result [103]:

Previous Velocity (Momentum), $\omega \mathbf{v}_i(t)$: In essence this term represents the particle's momentum. It forces the particle to maintain a consistent direction, preventing drastic change of the velocity. The term's influence is weighed by the inertia control parameter, ω.

Cognitive Component, $c_1 \mathbf{r}_1(t)\big(\mathbf{y}_i(t) - \mathbf{x}_i(t)\big)$: The cognitive component represents the particle's personal experience. The term drives the particle back to the best solution the particle has found so far. The cognitive component term is stochastically weighed with random numbers $\mathbf{r}_1 \sim U(0,1)^n$ and the cognitive acceleration coefficient c_1, an algorithm control parameter. The exact impact of the term therefore varies from time step to time step.

Social Component, $c_2 \mathbf{r}_2(t)\big(\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)\big)$: The social component draws the particle to the best solutions found in the particle's neighbourhood. In the case of the global best topology, all particles are drawn to the best solution found by the entire swarm. Just like the cognitive component, the term is stochastically weighed with random numbers $\mathbf{r}_2 \sim U(0,1)^n$ and the social acceleration coefficient c_2.

At each time step the quality of the solution represented by each particle is calculated and the personal best and neighbourhood best positions are updated as necessary. A number of criteria (stopping conditions) can be used to terminate the algorithm:

- **A fixed number of iterations have been completed**. This criterion is typically used when the algorithm has a restricted time period to execute in. Naturally, allocating too few iterations might not give the algorithm adequate time to find a good solution.
- **A fixed number function evaluations have taken place**. Objective function evaluations are almost always the most expensive operation that occurs during algorithm execution. This stopping condition is often used to limit the number of expensive operations or in cases where algorithms of different type or configuration are being compared.

- **An adequate solution has been found**. Often it is not necessary to find the absolute optimal solution in the search space and the algorithm may be terminated when an acceptable solution has been found.
- **No further improvement in the solution is observed**. It is possible for the swarm to become trapped in a local optimum, in which case optimisation can be terminated.

Pseudo code for a PSO is given in Algorithm 1 [93].

Algorithm 1 The *gbest* PSO algorithm.

Initialise the swarm S, of size n_s, within the n-dimensional search space.
while *all stopping conditions are false* **do**
  for each particle $i = 1, ..., n_s$ **do**
    Calculate the particle's fitness $f(\mathbf{x}_i(t))$
    if $f(\mathbf{x}_i(t)) < f(\mathbf{y}_i(t))$ **then**
      $\mathbf{y}_i(t) = \mathbf{x}_i(t)$
    end if
    if $f(\mathbf{y}_i(t)) < f(\hat{\mathbf{y}}_i(t))$ **then**
      $\hat{\mathbf{y}}_i(t) = \mathbf{y}_i(t)$
    end if
  end for
  for each particle $i = 1, ..., n_s$ **do**
    Update the particle's velocity $\mathbf{v}_i(t)$ using Equation (2.10)
  end for
  for each particle $i = 1, ..., n_s$ **do**
    Update the particle's position $\mathbf{x}_i(t)$using Equation (2.9)
  end for
end while

The PSO algorithm described above has a number of parameters that control various aspects of the optimisation process. The algorithm parameters drastically affect the performance of the PSO and are discussed in the next section.

2.3.2 Algorithm Control Parameters

Control parameters aim to balance the *exploration* and *exploitation* of the search process. Exploration refers to the algorithm's ability to explore a larger part of the search space. Exploitation is the ability of the algorithm to search in a focused area around a good solution with the intent of further refining the solution. Exploration and exploitation are contradictory objectives and ideally an algorithm should seek an optimal balance between them [26](chapter 16).

The Acceleration Coefficients

The acceleration coefficients c_1 and c_2 (along with a stochastic variable) respectively control the influence of the cognitive and social terms of the velocity update equation. Indirectly, however, they also influence the balance between exploration and exploitation of the search space. If c_1 is set to be much larger than c_2, the particle's personal best position is a greater attractor than the neighbourhood best. In this case particles act as multiple hill-climbers and focus their search in the areas local to their initialised positions [57]. Although this gives the PSO better opportunity to explore the search space the swarm could fail to converge, leading to a large number of sub-optimal solutions — a property useful for a niching algorithm [9]. On the other hand, if c_2 is much larger than c_1, the swarm better exploits the area around the best solution in a neighbourhood. This expedites the optimisation process, but might lead to premature swarm convergence [57].

Although the optimal values of c_1 and c_2 are problem specific, often the choice is made to set $c_1 = c_2$, balancing exploration and exploitation.

In most cases the values for c_1 and c_2 are static throughout the optimisation process. Ratnaweera *et al.* [82] propose a strategy where c_1 decreases linearly over time while c_2 linearly increases. This is potentially beneficial to the optimisation process as it encourages exploration near the start, but focuses on solution exploitation towards the end.

Maximum Velocity

When the PSO algorithm was first proposed, it was found that in certain situations it is possible for particle velocity values to become inappropriately large during the algorithm's execution. This is known as swarm explosion and often occurs when there are frequent changes in the neighbourhood best position, forcing some particles to have to change regions. The maximum velocity parameter, $\mathbf{V}_{max}$, was introduced later to specifically address this problem [22](chapter 6). The parameter specifies a maximum value that velocity values are clamped to if they exceed it. Clamping occurs after calculation of the velocity, but before the position update:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } |v'_{ij}(t+1)| < V_{max,j} \\ -V_{max,j} & \text{if } v'_{ij}(t+1) \leq -V_{max,j}, \quad \forall\, j \in \{1, ..., n\} \\ V_{max,j} & \text{if } v'_{ij}(t+1) \geq V_{max,j} \end{cases} \tag{2.11}$$

where $\mathbf{v}'_i(t+1)$ is calculated using Equation (2.10).

Appropriate settings of $\mathbf{V}_{max}$ can prevent swarm explosion, but also has a large effect on the granularity of the search. If $\mathbf{V}_{max}$ is set to small values, particles make small jumps in the search space, thereby exploiting the local area they are in [22](chapter 6). The risk associated with too small values for $\mathbf{V}_{max}$ is that a particle may become stuck in a local optimum, unable to generate the necessary velocity to escape. A second disadvantage is that a maximum velocity slows the optimisation process as particles will take longer to reach good optima. Large values for $\mathbf{V}_{max}$ allow the particles to make large jumps through the search space which better facilitates exploration [22](chapter 6) and faster search. But again, too large values for $\mathbf{V}_{max}$ is disadvantageous as the risk of swarm explosion occurring is increased and may lead to particles missing optima by flying past them.

Optimal values for $\mathbf{V}_{max}$ is problem specific, but similar to the acceleration coefficients, adaptive strategies have been proposed. An example of such a strategy is to decay $\mathbf{V}_{max}$ values exponentially [30]:

$$V_{max,j}(t+1) = [1 - (\frac{t}{T})^h V_{max,j}(t)], \quad \forall\, j \in \{1, ..., n\} \tag{2.12}$$

where T is the maximum number of time steps and h is a positive constant controlling

the degree of decay.

Inertia

The inertia parameter, ω, controls the influence of the previous velocity. The parameter's purpose is two-fold: to balance the exploration and exploitation objectives of the swarm, and to eliminate the need for $\mathbf{V}_{max}$ [21]. The experimentation showed that, although it is useful for balancing the objectives, the $\mathbf{V}_{max}$ parameter could not be eliminated completely. Intuitively, large values of ω allow larger jumps (assuming no velocity clamping) and promotes exploration, with small values aiding exploitation.

A second, very important aspect of ω is its interaction with the other control parameters in ensuring convergent swarm behaviour: If $\omega \geq 1$ then velocities increase during algorithm execution, eventually reaching maximum velocity if $\mathbf{V}_{max}$ is used [107]. This leads to a divergence of the swarm as it is difficult for particles to move toward promising areas. With $\omega < 1$, velocities tend towards 0 over time (depending on the values of c_i and c_2) *aiding* in both exploitation and swarm convergence [107]. An empirical study by Shi and Eberhart [93] showed that $\omega \in [0.8, 1.2]$ resulted in faster swarm convergence; with $\omega > 1.2$ the swarms failed to converge.

Due to the interaction between ω and the acceleration coefficients, an $\omega < 1$ is not enough to *ensure* swarm convergence. Van den Bergh and Engelbrecht have derived an inequality that guarantees convergent particle trajectories [103, 107], with similar findings shown in [102]:

$$\omega > \frac{1}{2}(c_1 + c_2) - 1 > 0 \tag{2.13}$$

However, both the studies of [102] and [107] use fixed points for the stochastic components in the PSO model. A more recent theoretical analysis by Cleghorn and Engelbrecht [13] leave the stochastic components unfixed, adding the following restriction for convergence:

$$0 < c_1 + c_2 < 4 \tag{2.14}$$

If the inequalities are not satisfied, particles may exhibit divergent or cyclic trajectories, irrespective of whether $\omega < 1$ [13, 103, 107].

Strategies have also been developed to dynamically adapt the inertia weight during the optimisation process. An example is to linearly decrease inertia, thereby encouraging

exploration initially and focusing on exploitation near the end of the process [73, 82, 100]:

$$\omega(t) = (\omega(0) - \omega(T))\frac{(T-t)}{T} + \omega(T) \tag{2.15}$$

For further reading on adaptive inertia strategies the reader is referred to [26](chapter 16).

Swarm Size

The swarm size refers to the number of particles in the swarm. The size of the swarm determines the initial diversity of the solutions, provided the particles are initialised in the search space using a uniform random number generator [26](chapter 16). A large swarm size facilitates that a larger part of the space can be searched per iteration at the cost of greater computational complexity due to an increase in function evaluations. It might therefore be necessary to reduce the number of algorithm iterations for the complexity to remain consistent to that of a smaller swarm. In reducing the number of iterations it is however possible that the algorithm will not be given a sufficient amount of time to refine a good solution.

Keeping the number of fitness evaluations constant therefore forces a trade-off between the swarm size and the number of iterations. Malan and Engelbrecht [70] conducted an empirical study investigating this trade-off using a *gbest* PSO on a number of benchmark optimisation problems. As expected, the results showed that the trade-off is problem specific, in some cases a smaller swarm size with more iterations fared better than a large swarm size and vice versa. However, the results also showed that a large swarm size often outperformed a smaller swarm size with highly multi-modal problems or problems with a high dimensionality.

2.3.3 Swarm Diversity

The swarm diversity is the degree to which the particles are dispersed within the search space [75]. Diversity is directly related to exploration and exploitation of the PSO algorithm as a large diversity implies a large area of the search space is explored, whereas a small diversity implies that a small area of the search space is being exploited by the swarm.

A number of methods for calculating swarm diversity have been developed [42, 61, 87]. A comparison of these methods is given by Olorunda and Engelbrecht [75]. Their analysis concludes that the *average distance around the swarm centre* measurement [61] is a valid indicator of swarm diversity, accurately representing the particle dispersion while remaining robust against outliers [75].

The *average distance around the swarm centre* is calculated as follows:

$$\mathcal{D}_{avg} = \frac{1}{|n_s|} \sum_{i=1}^{|n_s|} \sqrt{\sum_{k=1}^{n} (x_{ik} - \bar{x}_k)^2} \tag{2.16}$$

where $|n_s|$ is the swarm size, n is the dimensionality of the search space, $\mathbf{x}_i$ and $\bar{\mathbf{x}}$ represent the position of particle i and the swarm centre respectively. For the reasons given above, the *average distance around the swarm centre* measurement was used in the empirical work of this study.

2.3.4 Neighbourhood Topologies

Particles are connected to each other in a social structure called a neighbourhood topology (also called a social network structure). The neighbourhood topology controls the flow of information within the swarm. Particles are grouped together in neighbourhoods and information is shared with other particles in the same neighbourhood via the social term of the velocity update equation (refer to Equation (2.10)). A number of social structures exist [72], three of which are well studied and widely used within the field:

Global Best (*gbest*): With the *gbest* topology all particles are fully connected to each other in a star structure. As all particles are in a single neighbourhood, the global best solution serves as a single global attractor, and because of the fully connected structure all particles receive the information instantly. Compared to other topologies, the *gbest* topology has the fastest flow of information, which has been shown to lead to fast swarm convergence [19, 22, 60].

Local Best (*lbest*): The *lbest* topology connects particles on a one-dimensional ring lattice. A particle's neighbourhood is defined as the n_{ns} closest (based on index) particles to it on the lattice, including the particle itself. Instead of a single global

best, a neighbourhood best is selected from each neighbourhood. Information flow is much slower in this topology as a good solution has to travel through each neighbourhood via the overlapping particles, around the lattice to reach all particles. Convergence speed is therefore slower, with the benefit that the *lbest* topology has more time to explore the search space because of the multiple social attractors. The *lbest* topology has been shown to be less susceptible to local minima [19, 22, 60], and is thus well suited to multi-modal environments.

Von Neumann (*VN*): The *VN* social structure arranges the particles in a two-dimensional lattice. A neighbourhood is defined as the particles to the left, right, top and bottom of a particle on the lattice, including the particle itself. The rate of information flow balances the objectives of exploration and exploitation. Empirical studies have shown that this topology regularly outperforms the *gbest* and *lbest* topologies [60, 78]. A study by Franken in the domain of PSO NN training for game learning has, however, shown that, in the presence of a $\mathbf{V}_{max}$ parameter, the *lbest* topology performed better than the *VN* topology [33].

2.3.5 Guaranteed Convergence Particle Swarm Optimisation

It is possible for the standard PSO algorithm to convergence to a point in the search space that is not an optimum [106]. A cause of such stagnation is due to particle positions being equal to their personal and global best positions, i.e. $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$, where i is the particle index. From Equation (2.10) it is clear that in this case the particles' next velocity update will depend solely on the $\omega\mathbf{v}_i(t)$ term. If $\omega < 1$, this term tends to 0 as long as the condition remains, thereby causing the swarm to converge to arbitrary points in the search space.

This problem can be addressed by ensuring that the global best position is continually forced to change, which causes at least the social component of the velocity update to be non-zero thereby preventing stagnation. One of the first algorithms to solve this problem is the GCPSO algorithm. The GCPSO algorithm prevents stagnation of the global best particle until at least a local optimum is found [106]. This is accomplished through modification of the position and velocity update equations of the global best

particle [106]:

$$x_{\gamma j}(t+1) = \hat{y}_j(t) + \omega v_{\gamma j}(t) + \rho(t)(1 - 2r_2(t)), \quad \forall\, j \in \{1, ..., n\} \tag{2.17}$$

$$v_{\gamma j}(t+1) = -x_{\gamma j}(t) + \hat{y}_j(t) + \omega v_{\gamma j}(t) + \rho(t)(1 - 2r_{2j}(t)), \quad \forall\, j \in \{1, ..., n\} \tag{2.18}$$

where γ is the index of the global best particle and $\rho(t)$ is an additional algorithm control parameter. The $-\mathbf{x}_\gamma(t)$ term reverses the particle's previous position update and returns the particle to the $\hat{\mathbf{y}}$ position. The current search direction is maintained by the $\omega v_{\gamma j}(t)$ term. To this the randomly generated component, $\rho(t)(1 - 2r_{2j}(t))$, sampled from a space with side lengths $2\rho(t)$, is added. The PSO therefore randomly searches for a better position within the bounded sample space [106]. The bounds are continually adapted depending on the PSO's success or failure to find a better position within the area. This is done by adapting the ρ parameter as follows [106]:

$$\rho(t+1) = \begin{cases} 2\rho(t) & if\ \#successes > s_c \\ 0.5\rho(t) & if\ \#failures > f_c \\ \rho(t) & otherwise. \end{cases} \tag{2.19}$$

where s_c and f_c are control parameters defining threshold values for the number of consecutive $\#successes$ or $\#failures$ to find a better global best position. The $\#failures$ count is reset to 0 immediately after a better position is found; similarly, $\#successes$ is returned to 0 as soon as a failure to do so occurs. The initial values for ρ, s_c and f_c are dependent on the objective function. An empirical study has however shown that a default ρ value of 1.0 produces acceptable results [106]. Similarly, an $f_c = 5$ and $s_c = 15$ are recommended for high dimensional search spaces [106].

The GCPSO has been formally proven to converge to at least a local optimum [103]. It is important to note that except for the global best particle, all particles continue to use the standard position and velocity update equations as defined in Section 2.3.1.

2.3.6 Application of PSO to FFNN Training

This section describes the procedure of supervised learning for FFNNs using PSO for the purposes of this study's experimental work.

In addition to the algorithmic components defined above, a further two need to be defined in order to apply PSO to FFNN training: the particle representation and the objective function. Each particle in the swarm represents a complete FFNN. This is accomplished by serialising the FFNN's *weights and biases* to a multi-dimensional vector which forms a particle's position.

The objective function is defined as the mean squared error (MSE) over the training set D_T, of size P_T:

$$MSE_T = \frac{\sum_{p=1}^{P_T} \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2}{P_T K} \tag{2.20}$$

The MSE is deemed an appropriate objective function as it measures the network's training accuracy while remaining independent of both the algorithm and data set size.

2.4 PSO Neural Network Applications

The application areas of PSO are vast and diverse. A few examples include biomedical optimisation, data mining and clustering, process control, visualisation, video analysis, robotics, scheduling and many more. A survey of PSO applications has been done by Poli [80]. PSO has also been used successfully in training a variety of NNs, including radial bases function networks [49, 68, 84], recurrent networks [10, 55, 56, 85, 90] and product unit NNs [52]. One of the first PSO applications however was in training FFNNs [19, 57], and it is also the area of application most relevant to this study. The rest of this section gives a brief overview of PSO development and application with regards to FFNNs.

Comparison against gradient based algorithms. Mendes *et al.* [72] compared PSO against a number of other FFNN training algorithms, namely backpropagation (BP), QuickProp and RProp (which are all gradient based) [86] as well as an evolutionary programming (EP) technique [31]. Their results show that the RProp algorithm outperformed the PSO algorithm on the problems used, but that PSO showed promise when compared against the EP. They also state that the PSO algorithm is better suited to multi-modal problems such as FFNN training. Similar results are reported on an electromagnetic load forecasting problem in [37], where PSO trained FFNNs were compared

against networks trained by a genetic algorithm (GA) and a BP algorithm.

Comparison against other global optimisation algorithms. The usefulness of PSO as a global optimisation algorithm became very evident in the training of product unit FFNNs [52, 105]. Product unit FFNNs differ from summation unit FFNNs in calculating the weighted product of the input signal during the feedforward process as opposed to the weighted sum [18, 67]. Product unit FFNNs are capable of learning higher-order functional mappings than summation unit FFNNs with the same number of hidden units [52]. They are, however, significantly harder to train due to the search space containing steep gradients and a large number of local optima. Studies by Ismail and Engelbrecht [52] and later Van den Bergh and Engelbrecht [105] showed that the PSO algorithm is very successful at training product unit FFNNs, outperforming other global optimisation algorithms including a GA and the LeapFrog optimisation algorithm.

A thorough theoretical study of PSO that includes an investigation into the feasibility of PSO FFNN training has been conducted by Van den Bergh [103]. The work compared four PSO variants against two GAs and two gradient based learning algorithms. The algorithms trained both summation unit FFNNs as well as product unit FFNNs on a variety of classification and regression problems. The investigation showed that, with appropriate parameter choices, the PSO had competitive performance against the gradient algorithms in training the summation FFNNs and generally yielded superior generalisation performance over the GAs when training the product unit FFNNs.

Modified PSOs. A number of studies have investigated modifications to the standard PSO algorithm specifically to improve performance in FFNN training. Liu *et al.* [69] achieved moderate success using an *lbest* PSO with a variable neighbourhood size (increasing from two to the swarm size during the course of optimisation). The results showed that the so called *vbest* PSO had faster convergence than a standard *lbest* PSO and a lower total error than a *gbest* PSO on the iris classification problem.

Zhao *et al.* proposed a modification to the position update equation in [119]. In their work a particle's position is constructed by (per dimension) choosing parts from the previous, personal best or global best positions. The decision is made using the velocity vector, the components of which are stochastically generated from $U(0, 1)$ and

act as probabilities. This strategy is similar to that of a crossover operation in a GA. The new algorithm showed some improvement when compared against BP training and a standard PSO algorithm on a number of classification problems.

PSO hybrid algorithms. In an attempt to further improve FFNN training, a number of hybrid optimisation algorithms that include PSO as a sub-algorithm have been developed. One such algorithm, a PSO incorporating BP, has been developed by Zhang *et al.* [118]. With their approach FFNNs are trained using a PSO with adaptive inertia. If no improvement in the global best is shown for a number of generations, the global best NN is further refined using BP. The hybrid approach improved both convergence and overall error on the test problems.

Carvalho and Ludermir investigated the performance of a number of PSO-RProp hybrids in [11]. The hybrid algorithms were compared against their individual component algorithms on three classification problems. The results showed that, although the hybrid PSO algorithms were successful in training the NNs, the RProp algorithm by itself yielded the best performance on the problems used.

Generalisation of PSO trained FFNNs. Very little work has been done to investigate PSO trained NN generalisation performance and overfitting, and in some publications pertaining to PSO FFNN training the generalisation error is not reported at all.

One approach that has been investigated involves optimising the network architecture alongside the network weights. This has successfully been done in [116, 117] using an evolutionary algorithm (EA) and PSO hybrid algorithm.

Carvalho and Ludermir [11] directly addresses overfitting in their investigation of early stopping using a PSO. It was shown that the GL_5 early stopping criterion is inappropriate for PSO as it hampers the algorithm's exploration of the search space and leads to significantly worse results. Later work by the same authors added weight decay to the PSO training process [12], this resulted in better generalisation by the NNs.

2.5 Summary

This chapter provided background information on the algorithms and methods used in this study. Section 2.1 introduced NNs and FFNNs along with a detailed discussion of the elements involved in their training. This was followed by a discussion on the phenomenon of overfitting in Section 2.2 which included a review of methods that attempt to prevent overfitting from occurring. Motivation and supporting literature was given surrounding why it is deemed necessary to study the phenomenon in the context of particle swarm optimisation.

Section 2.3 focused on the PSO algorithm and its control parameters. Finally a brief overview of PSO applications was given in Section 2.4.

The next chapter marks the beginning of the experimental work of this study and provides a baseline analysis of overfitting in PSO trained FFNNs.

Chapter 3

Comparing PSO and GCPSO

Chapter 2 discussed two widely used PSO algorithms: the modified PSO that includes the inertia term (for the purposes of this chapter referred to as the standard PSO) in Section 2.3.1 and the GCPSO algorithm in Section 2.3.5.

Although the GCPSO solves a particular deficiency in the algorithm, it also introduces additional algorithm complexity and a number of additional control parameters. The primary purpose of this chapter is to empirically compare the standard PSO and GCPSO algorithms on the task of FFNN training. The results and subsequent discussion presented in this chapter will serve as motivation for algorithm and parameter choices for the rest of this study.

The chapter also aims to establish an empirical baseline to characterise and demonstrate 'typical' overfitting behaviour by the PSO algorithm.

The remainder of this chapter is structured as follows: the FFNN architectures used for the empirical work are given in Section 3.1. The test data sets and data set preparation methods are given in Section 3.2. The experimental methodology is discussed in Section 3.3. This is followed by an empirical comparison of the PSO and GCPSO algorithms in Section 3.4. The chapter is concluded with a summary in Section 3.5.

3.1 Architecture Selection

For each of the data sets an appropriate network architecture has been selected. As discussed in Section 2.2, a number of techniques exist for selecting an optimal network architecture. Overfitting is however specific to the training algorithm and selecting an optimal architecture is not guaranteed to prevent overfitting [64]. As the purpose of this study is to analyse overfitting specifically for the PSO, network architectures that have the potential to allow overfitting were selected. The generalisation performance was therefore not considered as a factor for the purposes of architecture selection and no attempt at optimisation was made in this regard.

All networks had a single hidden layer. A bias unit was included in both the input and hidden layers of each network. The sigmoid activation function (as defined in Section 2.1.1) was used as the activation function in the hidden and output layers of the FFNNs. The size of the input layer was set to the number of input variables associated with the data set. For all data sets a single neuron was used in the output layer, a strategy discussed in Section 2.1.2.

3.2 Data Sets

This section details the data sets selected for the empirical work of this chapter as well as the methods used to pre-process and prepare the data sets.

Data Set Selection

A set of 10 different and well-studied data sets were chosen for use in the empirical study. The sets are split evenly between *classification* and *regression* problems. Furthermore, the data sets vary in the number of input variables (pattern dimensionality), number of patterns and the scale of the input and output ranges. For the regression problems, time series were selected as the functions to approximate. Time series are appropriate regression functions as they inherently contain noise, which is a necessary condition for overfitting to occur.

The chosen data sets, corresponding network architectures, resulting search space

dimensionality and number of patterns in each set are given in Table 3.1. The equations for the various time series are listed below:

Table 3.1: The data sets used in the experimental work. The number of patterns in each set (P), chosen neural network input (I), hidden (J) and output layer (K) sizes and search space dimensionality (n) is shown.

Data Set		**P**	**I**	**J**	**K**	**n**
Classification problems						
Glass Identification	UCI MLR [1]	214	9	12	6	133
Iris	UCI MLR [1]	150	4	8	3	49
Pima Indians Diabetes	UCI MLR [1]	768	8	20	2	201
Wisconsin Diagnostic Breast Cancer	UCI MLR [1]	569	30	25	2	801
Wine	UCI MLR [1]	178	13	10	3	151
Regression problems						
Henon Map	Equation (3.1) [103]	100	2	10	1	41
Logistic Map	Equation (3.2)	981	4	10	1	61
Mackey-Glass Equation	Equation (3.3) [79]	981	4	20	1	121
Sunspots(Annual)	NGDC [74]	305	4	10	1	61
TS5	Equation (3.4)	291	10	5	1	61

- Henon Map:

$$z_t = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2;\ z_0 \text{ and } z_1 \sim U(-1, 1) \tag{3.1}$$

- Logistic Map:

$$z_t = rz_{t-1}(1 - z_{t-1});\ z_0 = 0.01,\ r = 4 \tag{3.2}$$

- Mackey-Glass Equation:

$$z_t = (1 - b)z_{t-1} + a\frac{z_{t-\tau}}{1 + z_{t-\tau}^{10}};\ b = 0.1,\ a = 0.2,\ \tau = 30 \tag{3.3}$$

- TS5:

$$z_t = 0.3z_{t-6} - 0.6z_{t-4} + 0.5z_{t-1} + 0.3z_{t-6}^2 - 0.2z_{t-4}^2 + n_t;\ n_t \sim N(0, 0.05) \quad (3.4)$$

For the Henon Map, Logistic Map, Mackey-Glass and TS5 time series, 1000 points were sampled from each equation. Training patterns were then constructed from the sampled points by taking, for a target point $t+1, t > I$, a window of the I preceding points as input variables; where I is the size of the input layer as given in Table 3.1. The Mackey-Glass equation was an exception to this, where instead the t, $t-6$, $t-12$ and $t-18$ points were used as input parameters, as in [79].

Data Set Preparation

The data sets were preprocessed in the following manner: the values of the input vector were scaled to the range $[-2, 2]$ which is a rational approximation of $[-\sqrt{3}, \sqrt{3}]$, the active domain of the sigmoid function [28]. Although scaling of the input values is not strictly necessary, it has been shown to improve training performance [28]. The data set's target values were scaled to $[0.1, 0.9]$ which falls within the output range of the sigmoid function. In the case of classification problems, nominal class values were converted to a numeric representation and then scaled to the range $[0.1, 0.9]$ such that the scaled target values are numerically equidistant. Although this particular method of encoding class values has been shown to increase training time [28] it allows a FFNN to use a single output neuron to classify any number of classes, avoiding an increase in network complexity as well as aiding deeper analysis of the FFNN output values across problems.

The data sets were split in a $2:1$ ratio into a training set D_T and a generalisation set D_G, with D_G being used to estimate the generalisation ability of the NN. In the case of the classification data sets, the patterns were assigned randomly to D_T and D_G and shuffled. Whereas with the time series the patterns were assigned in order to each set. Also, for the classification data sets, while calculating both the training and generalisation errors, the patterns are presented to the NN stochastically, preventing inadvertent memorisation of the pattern sequence.

3.3 Experimental Methodology

This section gives the general methodology used for this experiment. The hypothesis and significance testing methods are given along with the algorithm configurations and experimental procedure.

3.3.1 Hypothesis and Significance Testing

For this experiment, the following null and alternative hypotheses were used:

- H_0: There is no significant difference in performance between the standard PSO and the GCPSO.
- H_1: There is a significant difference in performance between the standard PSO and the GCPSO.

To determine statistical significance a two-tailed Mann-Whitney U non-parametric test was performed [71]. A significance level of 5% ($\alpha = 0.05$) was used for all tests. Where multiple comparisons were made per test (as opposed to a binary comparison), the significance level was adjusted using Bonferroni correction [46]: $\beta = \frac{\alpha}{n}$, where β is the adjusted significance level and n is the number of comparisons performed.

3.3.2 PSO Configurations

The following parameter values were used in the experimental work for both the standard PSO and GCPSO. The c_1 and c_2 control parameters were set to 1.496180, with *inertia* (ω) set to 0.729844. These values are known to be appropriate for a number of problems and correspond to those used in [20]. More importantly, the values ensure convergent particle trajectories [103]. For both algorithms a swarm size of 25 particles was used. The effect of the swarm size on FFNN training is investigated in a later chapter.

For the GCPSO, a $\rho = 1$ was used with a success threshold (s_c) of 15 and a failure threshold (f_c) of 5. These values correspond to the recommended values for high dimensional search spaces [106].

The *lbest* neighbourhood topology with a neighbourhood size of 5, as described in Section 2.3.4, was used for both the standard PSO and GCPSO, thereby making both algorithms less susceptible to local minima.

In order to reduce the number of experiment variables, no $\mathbf{V}_{max}$ parameter was defined for either algorithm for this experiment. The effect of the $\mathbf{V}_{max}$ parameter on overfitting is investigated in detail in a later chapter.

A fixed number of 50 000 objective function evaluations was used as stopping criterion for both PSO algorithms, resulting in 2000 iterations with a swarm size of 25.

Algorithm Initialisation

The particle swarms and NNs were initialised as follows: the network weights (and therefore the particle positions, as each particle position represents a network's weight vector) were randomly initialised in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, where $fanin$ is the number of incoming connections of the corresponding neuron. This weight initialisation strategy has been shown to aid in avoiding local minima during training [113].

The particles were initialised stationary, that is, velocities were initialised to the zero vector $\mathbf{0}$. This initialisation strategy has been shown to lead to superior optimisation speed compared to other strategies [27].

3.3.3 Experimental Procedure

For each of the problems the algorithms were executed 30 times, with each run using a Mersenne twister pseudo-random number generator (PRNG) with a unique seed. The means and standard deviations are reported across all 30 samples. Where stated, the sample median is also reported; the median is a better indication of population locality in cases where the population is highly skewed.

Algorithm Measurements

The following algorithm measurements were calculated during each run of an algorithm:

- E_T: The *re-scaled* mean squared error over the training data set as per Equation (2.20).

- E_G: The *re-scaled* mean squared error over the generalisation data as per Equation (3.5).
- ρ_F: The Röbel generalisation factor as given in Section 2.2.1.
- $\mathcal{D}_{avg}$: The swarm diversity as per Equation (2.16).

Similar to the training error, the generalisation error is calculated as:

$$E_G = \frac{\sum_{p=1}^{P_G} \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2}{P_G K} \tag{3.5}$$

where P_G is the number of patterns in the generalisation set and K is dimensionality of the target pattern.

It is important to note that the errors were calculated on re-scaled data. That is, the NN's output and pattern's target value is scaled back to the target value's original range. The re-scaling results in a finer indication of the NN's accuracy relative to the original form of the data sets.

3.4 Results

The results of the comparison between the two algorithms are given in this section. The results are presented in four parts, one per performance measurement.

Table 3.2 shows the re-scaled training error results for the PSO and GCPSO algorithms run on the five classification and five regression problems.

The standard PSO obtained a marginally lower error than the GCPSO for four of the five classification problems, the WDBC data set being the exception. For all five classification problems the difference in training error performance was not statistically significant. The PSO obtained an insignificantly lower training error on the Logisitics Map, Mackey-Glass equation and TS5 data sets. The reported standard deviations were low (relative to their scale) on all the data sets indicating that both algorithms showed consistent performance over the 30 samples.

From these results it is clear that the algorithms did not differ significantly and H_0 is not rejected for any of the data sets, based on training error.

Table 3.2: Re-scaled training error (E_T) means ($\bar{\mu}$), medians ($\tilde{x}$) and standard deviations (σ) for the PSO and GCPSO algorithms after 50 000 objective function evaluations. The p-value indicates the result of the significance test.

Problem	**PSO**			**GCPSO**			
	$\bar{\mu}$	$\tilde{x}$	σ	$\bar{\mu}$	$\tilde{x}$	σ	**p-value**
Diabetes	0.11841	0.1178	0.00528	0.11884	0.11818	0.00394	0.542
Glass	0.298	0.30054	0.05045	0.30241	0.29927	0.06605	0.994
Iris	0.00852	0.00849	0.00478	0.00927	0.00826	0.00505	0.612
WDBC	0.01235	0.01223	0.00263	0.01207	0.01135	0.0026	0.878
Wine	0.00179	0.00166	0.00116	0.00183	0.00125	0.00155	0.752
Henon Map	0.00101	0.0009	0.00057	0.00097	0.00081	0.00058	0.476
Log. Map	0.0008	0.00079	0.00019	0.00081	0.00081	0.00016	0.912
M.-Glass	0.00079	0.00077	0.00016	0.00082	0.00081	0.00016	0.413
Sunspots	155.602	162.201	18.037	155.203	155.746	15.171	0.592
TS5	0.00203	0.00203	0.00015	0.00208	0.00207	0.00014	0.26

The re-scaled generalisation error results for both algorithms are summarised in Table 3.3. The GCPSO obtained marginally lower generalisation errors for all of the classification problems with the exception of the Diabetes set. The PSO obtained slightly lower errors on the Henon Map, Mackey-Glass equation, Sunspots and TS5 regression data sets. The results also show consistent performance in terms of the standard deviations for all 30 samples.

Similar to the training error results, H_0 is not rejected and it is concluded that no significant difference in generalisation performance exists between the PSO and GCPSO for any of the classification or regression problems.

The generalisation factor results for both algorithms are given in Table 3.4. For the Iris, Wine and Henon Map data sets very large standard deviations were obtained for both the PSO and GCPSO. This that the obtained generalisation factor differed greatly between samples on these three data sets. As a result, the median was considered a

Table 3.3: Re-scaled generalisation error (E_G) means ($\bar{\mu}$), medians ($\tilde{x}$) and standard deviations (σ) for the PSO and GCPSO algorithms after 50 000 objective function evaluations. The p-value indicates the result of the significance test.

Problem	**PSO**			**GCPSO**			
	$\bar{\mu}$	$\tilde{x}$	σ	$\bar{\mu}$	$\tilde{x}$	σ	**p-value**
Diabetes	0.16925	0.16516	0.01227	0.16942	0.16936	0.00892	0.676
Glass	1.03654	1.0316	0.25657	0.99216	0.95951	0.30607	0.552
Iris	0.03733	0.03399	0.02032	0.03666	0.03292	0.01869	0.832
WDBC	0.03318	0.03256	0.00896	0.03259	0.03346	0.01002	0.878
Wine	0.04289	0.04069	0.0185	0.04267	0.03972	0.01582	0.912
Henon Map	0.01077	0.00564	0.01371	0.01225	0.00722	0.01614	0.665
Log. Map	0.00144	0.00126	0.0007	0.00133	0.00126	0.00055	0.697
M.-Glass	0.00129	0.00131	0.00053	0.0013	0.0014	0.00049	0.831
Sunspots	231.801	222.978	43.871	233.922	229.986	40.366	0.602
TS5	0.0044	0.00408	0.00114	0.00472	0.00461	0.00138	0.35

better indicator of performance.

For both algorithms a $\rho_F > 1$ was obtained on all problems, which shows that both algorithms overfitted the training set to some degree. However, from the large standard deviations it is clear that the degree of overfitting greatly differed between samples. This indicates that the stochastic conditions has a potentially large effect on overfitting, either during swarm initialisation or over the course of the algorithm. Swarm position initialisation (weight initialisation) is investigated in a later chapter.

Similar to the training and generalisation error results there was no significant difference in terms of generalisation factor between either algorithm; H_0 is not rejected with $p > 0.05$ for all of the problems.

A question remains whether the algorithms had similar accuracy and performance over time. In order to investigate this aspect, the training and generalisation errors were plotted against the iterations of the algorithms. Showing these results for all data

Table 3.4: Results showing the generalisation factor (ρ_F) means ($\bar{\mu}$), medians ($\tilde{x}$) and standard deviations (σ) for the PSO and GCPSO algorithms after 50 000 objective function evaluations. The p-value indicates the result of the significance test.

Problem	**PSO**			**GCPSO**			
	$\bar{\mu}$	$\tilde{x}$	σ	$\bar{\mu}$	$\tilde{x}$	σ	**p-value**
Diabetes	1.43409	1.39204	0.14529	1.42813	1.42547	0.10333	0.774
Glass	3.69748	3.13202	1.60842	3.58291	3.45989	1.69696	0.786
Iris	10.831	4.198	18.382	9.85419	4.18937	18.265	0.665
WDBC	2.87078	2.75135	1.14998	2.92848	3.0237	1.30513	0.832
Wine	52.348	26.678	87.873	39.127	27.529	34.97	0.697
Henon Map	10.762	8.424	11.188	15.974	12.161	19.557	0.39
Log. Map	2.06341	1.60321	1.402	1.78381	1.5952	0.96168	0.513
M.-Glass	1.78528	1.46487	1.0028	1.73461	1.67639	0.9086	0.994
Sunspots	1.53515	1.43052	0.45634	1.53766	1.45766	0.39045	0.581
TS5	2.1867	2.00878	0.61799	2.28335	2.25962	0.68583	0.561

sets is infeasible and therefore a number of figures that are considered representative or interesting are shown below. Unless otherwise stated, all figures show the mean over 30 samples.

Figure 3.1 shows the training and generalisation performance over time for both algorithms on the Diabetes classification data set. As seen in the figure there is no discernible difference in either training or generalisation performance, which indicates that the algorithms not only obtained near equal final errors, but also behaved very similarly over time. Figure 3.1 is representative of most of the other classification as well as regression problems, with the exception of the Glass and Henon Map data sets. Also clearly visible in the Figure 3.1 is the divergence of the training and generalisation errors, indicating overfitting of the training set after approximately 200 iterations. This behaviour was consistent over all data sets.

The E_T and E_G graphs for the Glass data set are given in Figure 3.2. Similar

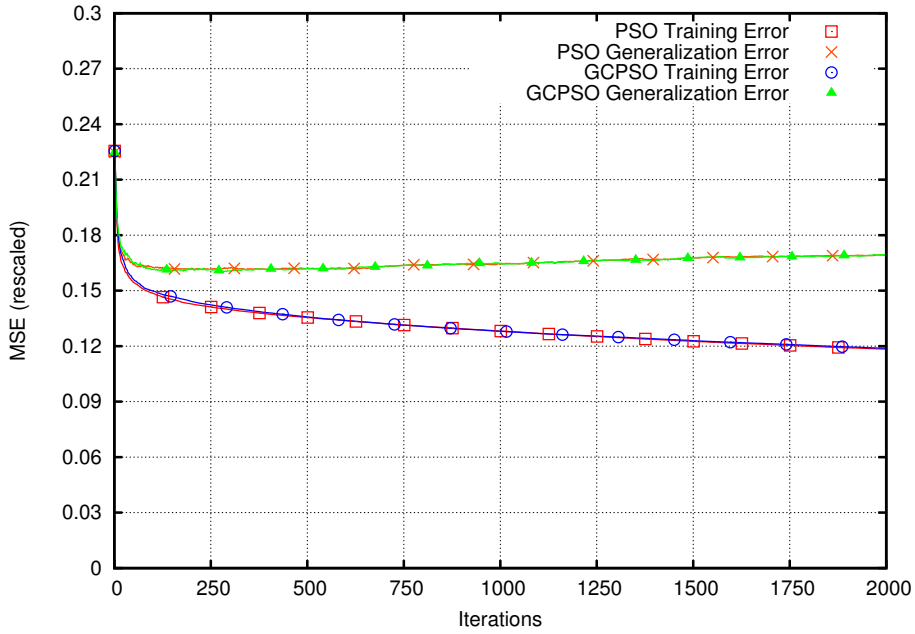


Figure 3.1: E_T and E_G over 2000 training iterations on the Diabetes data set.

to the results for all other data sets, the algorithms had near identical training errors throughout the run. The Glass data set was the only data set where, in the case of the generalisation errors, the PSO obtained a marginally higher error throughout the execution of the algorithm. However, as shown in Table 3.3, the difference was not significant.

The results for the Henon Map data set are illustrated in Figure 3.3. Similar to the results for the Glass data set, the PSO generalisation error began to deviate from the GCPSO generalisation error early in the execution of the algorithm. Although worse than the training error throughout the run, the generalisation error improved over time, lessening the extent of the overfitting. The PSO later succeeded in obtaining a generalisation error lower than that of the GCPSO.

The above graphs illustrate that the algorithms had very similar performance throughout the optimisation process, with minor variance shown on some, unrelated data sets.

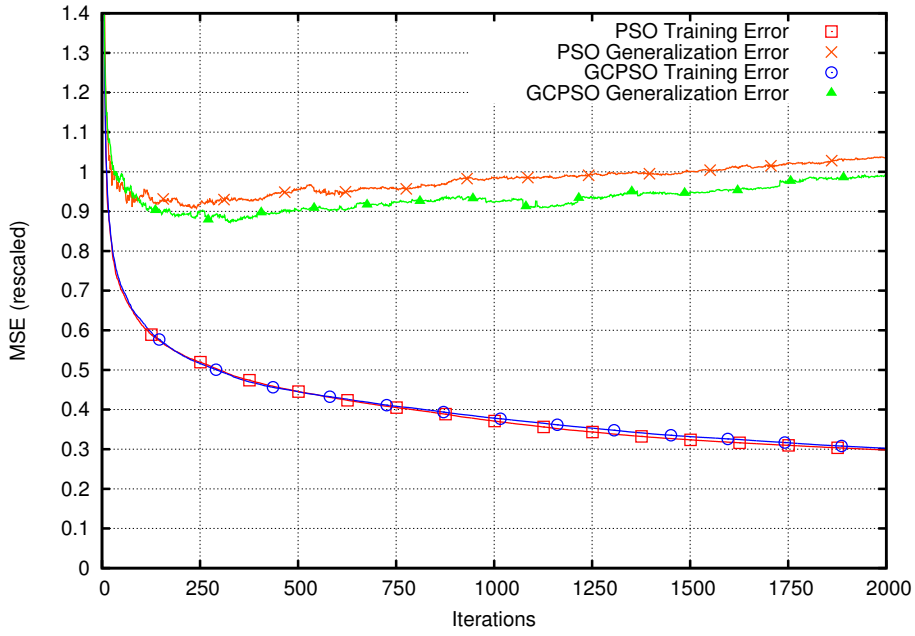


Figure 3.2: E_T and E_G over 2000 training iterations on the Glass data set.

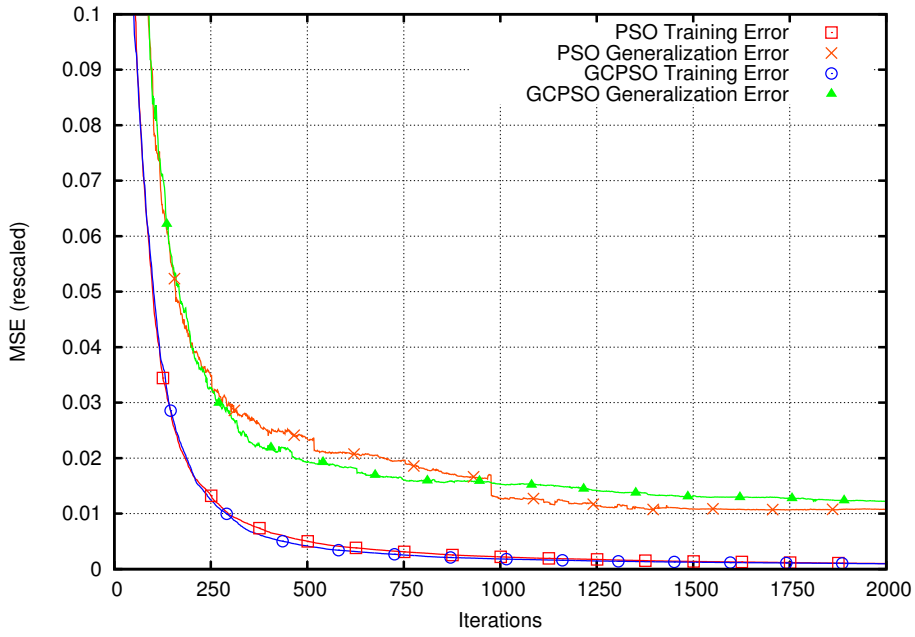


Figure 3.3: E_T and E_G over 2000 training iterations on the Henon Map data set.

Diversity

The diversity results for both algorithms showed that swarm divergence occurred on all data sets (both classification and regression) during the course of optimisation. Swarm divergence can be seen as an increase of diversity over time as measured by the average distance around the swarm centre.

Figures 3.4 and 3.5 show the diversity per iteration on the WDBC classification data set, for the PSO and GCPSO respectively. The WDBC diversity was representative of the other classification problems, for comparison, the Iris diversity results are also shown in Figure 3.6. The individual samples varied widely and also differed greatly in size. Therefore samples are shown individually and have been *log* scaled with the median result shown in red (along with diamond shaped markers).

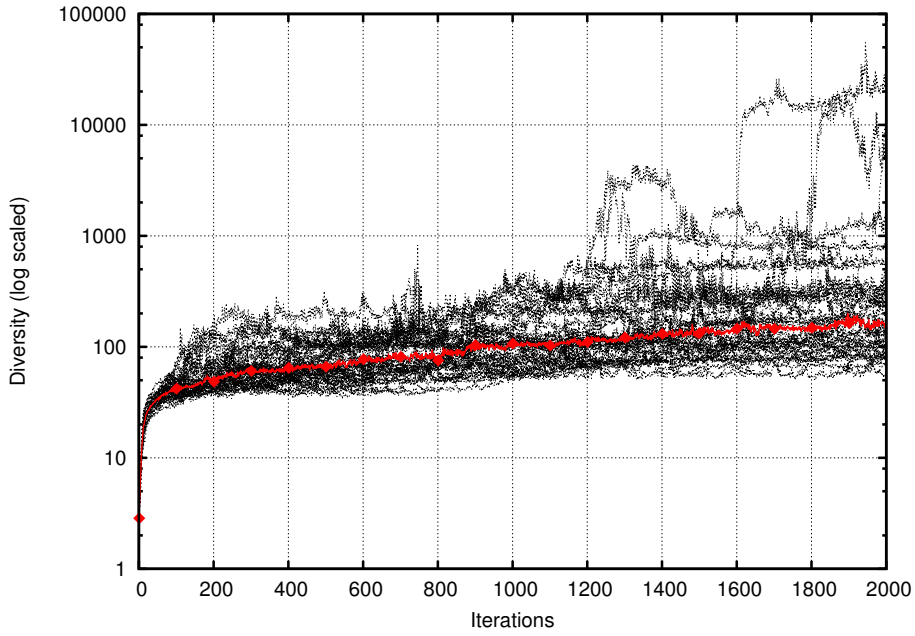


Figure 3.4: PSO $\mathcal{D}_{avg}$ over 2000 training iterations on the WDBC data set. Graph shows 30 individuals samples with median result indicated in red.

For both algorithms, the figures show that although some samples showed relatively stable diversity (a value of approximately 50), other samples diverged widely. When comparing the algorithms, the standard PSO showed less divergence, with samples showing

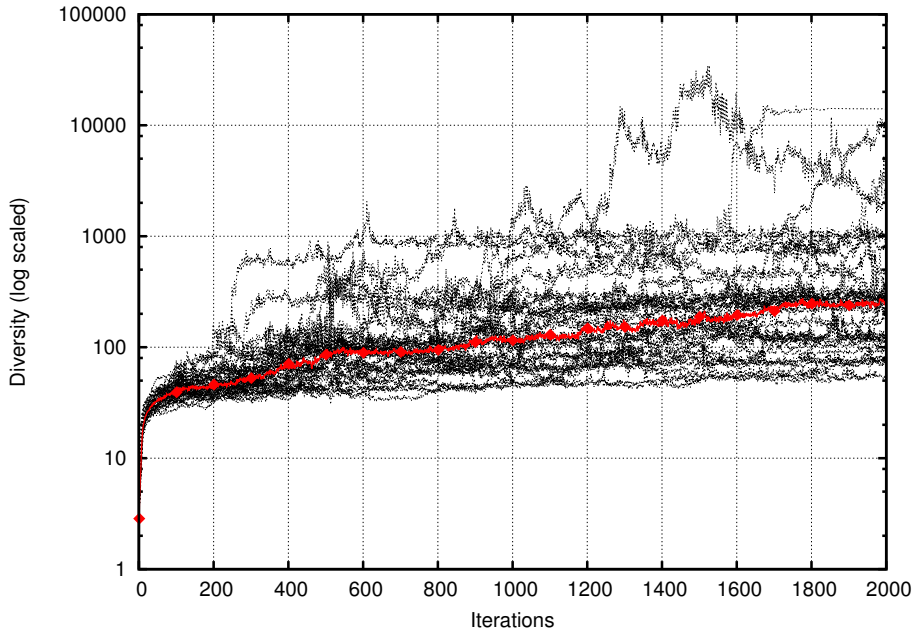


Figure 3.5: GCPSO $\mathcal{D}_{avg}$ over 2000 training iterations on the WDBC data set. Graph shows 30 individuals samples with median result indicated in red.

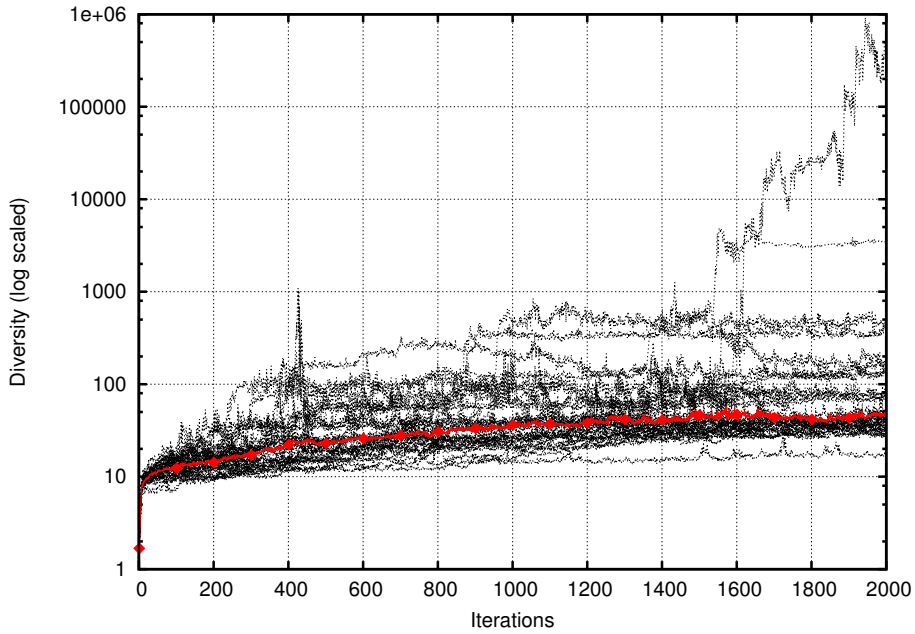


Figure 3.6: PSO $\mathcal{D}_{avg}$ over 2000 training iterations on the Iris data set. Graph shows 30 individuals samples with median result indicated in red.

tighter clustering around the median as compared to the GCPSO.

The diversity results indicate that, although the algorithms obtained a solution, the swarm did not fully converge on the solution. One method for addressing swarm divergence is the correct choice of control parameters, most notably the $\mathbf{V}_{max}$ parameter. As this initial experiment was a baseline comparison, the control parameters were not optimised, nor their effect on controlling overfitting investigated.

The effect of swarm convergence and the $\mathbf{V}_{max}$ parameter, as well as other control parameters, on overfitting is analysed in the following chapters.

3.5 Summary

This chapter provided an empirical comparison of two widely publicised PSO algorithms: the standard PSO that includes an inertia term and the GCPSO. The NN architectures and data sets used for the comparison were given in Sections 3.1 and 3.2. The experimental methodology was given in Section 3.3.

The results were given in Section 3.4. An empirical baseline for overfitting behaviour on the selected data sets was established. It was observed that overfitting by both PSO algorithms occurred early in the optimisation process, in most cases, no later than 200 iterations or 5000 objective function evaluations with a swarm size of 25.

It was further shown empirically that there was no significant difference between the GCPSO and PSO in terms of algorithm performance and overfitting behaviour either over the course of optimisation or in the final result obtained.

However, analysis of the diversity results showed that the particle swarms for both algorithms diverged over the course of the optimisation process. This is most likely due to choice of control parameters for this experiment. The control parameters may also contribute to the overall degree of overfitting, which is investigated in the following chapters.

Chapter 4

PSO Maximum Velocity

Chapter 3 established a baseline for the overfitting behaviour of FFNNs when trained using particle swarms, comparing the *lbest*- and GCPSO algorithms.

Two primary observations were made, the first is that there is no significant difference in behaviour or performance of the algorithms. The second was that swarm divergence occurred on all problems in the case of both PSOs. It is likely that the cause of the divergence is due to the lack of the use of a *maximum velocity* ($\mathbf{V}_{max}$) parameter in the initial experiment.

The purpose of this chapter is to investigate the effect, if any, of the $\mathbf{V}_{max}$ parameter on the overfitting behaviour of the FFNNs. The hypothesis that the lack of the $\mathbf{V}_{max}$ parameter in the initial experiment was the cause of the divergent swarms will also be investigated.

The remainder of the chapter is structured as follows: Section 4.1 describes the methodology used for the investigation, including the method used to explore the $\mathbf{V}_{max}$ parameter space, algorithm configurations and measurements. This is followed by the results in Section 4.2. A summary of the chapter is given in Section 4.3.

4.1 Experimental Methodology

This section describes the methods used to analyse the potential effect of $\mathbf{V}_{max}$ on overfitting.

4.1.1 $\mathbf{V}_{max}$ selection

In order to analyse the effect of $\mathbf{V}_{max}$, a number of candidate $\mathbf{V}_{max}$ values needed to be chosen for study. Considering the problem domain, that is, the training of FFNNs using the sigmoid activation function (2.1.1), the $\mathbf{V}_{max}$ parameter effectively limits the maximum size of the updates to the neural network weights (i.e. the particle positions). The optimal range for $\mathbf{V}_{max}$ is therefore tied to the specific activation function used by the FFNN, as it is the activation function that determines the effective domain of the weights. In the case of the sigmoid function, the active domain is defined as $[-\sqrt{3}, \sqrt{3}]$ [28]. It therefore follows that an appropriate range for $\mathbf{V}_{max}$ would be relatively small, perhaps within, or slightly larger than the active domain.

Assuming the use of a sigmoid activation function, the $\mathbf{V}_{max}$ parameters to be used for the analysis were generated using the following exponential equation:

$$V_{max_i} = 0.008e^{4.83\frac{s}{S}}, \quad \forall\, i \in 1, ..., n;\ s \in [1..S]. \tag{4.1}$$

where n is the dimensionality of the search space, s is the sample to generate, and S is the number of $\mathbf{V}_{max}$ samples to generate. All components (V_{max_i}) of each $\mathbf{V}_{max}$ vector were equal. For this analysis, 10 $\mathbf{V}_{max}$ parameters ($S = 10$) were sampled. This yields $\mathbf{V}_{max}$ parameters with components in the range $(0.0, 10.017]$. Due to the exponential nature of Equation (4.1) the majority of the generated parameter values are small, falling within the domain of the activation function. Larger values (closer to 10.0) are included for the sake of completeness. The analysis therefore focuses on smaller $\mathbf{V}_{max}$ parameters, which is deemed appropriate.

4.1.2 Experimental Procedure

The data sets used for the experiment comprised of the same ten data sets specified in Section 3.2, split evenly between classification and regression problems. Similarly the NNs used the same architectures given in Table 3.1.

The *lbest* PSO was then executed on each of the data sets using each of the 10 selected $\mathbf{V}_{max}$ parameters and the results recorded. Each execution of the algorithm on a specific problem with a specific $\mathbf{V}_{max}$ value was repeated 30 times using a Mersenne twister

PRNG with a unique seed. The means and standard deviation (and, where stated, the median) across the 30 samples are reported.

PSO Configuration

As shown in Chapter 3 there is no significant advantage in using the GCPSO algorithm over the standard PSO. The GCPSO also adds additional complexity and parameters. As such, the PSO algorithm used for the $\mathbf{V}_{max}$ analysis was an *lbest* PSO with $\omega = 0.729844$. The c_1 and c_2 parameters were set to 1.496180 and a neighbourhood size of 5 was used. The swarm size was set to 25 particles. The number of objective function evaluations performed by the PSO were limited to 50000.

As described in Section 3.3.2 the particle positions were randomly initialised in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, with velocities initialised to $\mathbf{0}$.

This is the exact same configuration used in Chapter 3 which makes the results, especially swarm convergence, comparable.

Algorithm Measurements

The algorithm measurements as defined in Section 3.3.3 were used. That is, the training error E_T, generalisation error E_G, Röbel generalisation factor ρ_F, and swarm diversity $\mathcal{D}_{avg}$.

4.2 Results

This results of the experimental work of this chapter is given in this section. The purpose of the experiment was two-fold: firstly, to determine whether $\mathbf{V}_{max}$ has an effect on overfitting and secondly, whether the lack of $\mathbf{V}_{max}$ lead to the divergent behaviour seen in Chapter 3. The results are therefore divided into two subsections: Section 4.2.1 focuses on diversity, with the accuracy and overfitting results discussed in Section 4.2.2. A general discussion and analysis of the results is given in Section 4.2.3.

4.2.1 Diversity

The diversity results for each of the data sets and $\mathbf{V}_{max}$ values are given in Table 4.1. The results show that the lowest diversities coincided with the lowest tested $\mathbf{V}_{max}$ value of 0.129 for all the data sets, with the exception of the Diabetes data set, where diversity was lowest for the slightly higher $\mathbf{V}_{max} = 0.21$. Diversity also increased monotonically as $\mathbf{V}_{max}$ increased. The standard deviations of the results show that the diversities were consistent across samples.

These results are expected, as a smaller $\mathbf{V}_{max}$ hampers the speed of the particles during the optimisation process, thereby leading to smaller position changes which reduces the distance a particle may travel from the swarm centre within a finite number of steps. A smaller $\mathbf{V}_{max}$ therefore reduces the exploration potential of the swarm. A higher $\mathbf{V}_{max}$ will in turn allow a particle to potentially move further away from the swarm centre in fewer iterations, thereby allowing more exploration of the search space, leading to a higher diversity. However, the question remains on whether the swarms *stagnated* to a higher final diversity in the presence of a higher $\mathbf{V}_{max}$, or whether there was an initial phase of convergence followed by divergence, due to increased velocities allowed by the higher $\mathbf{V}_{max}$ [15].

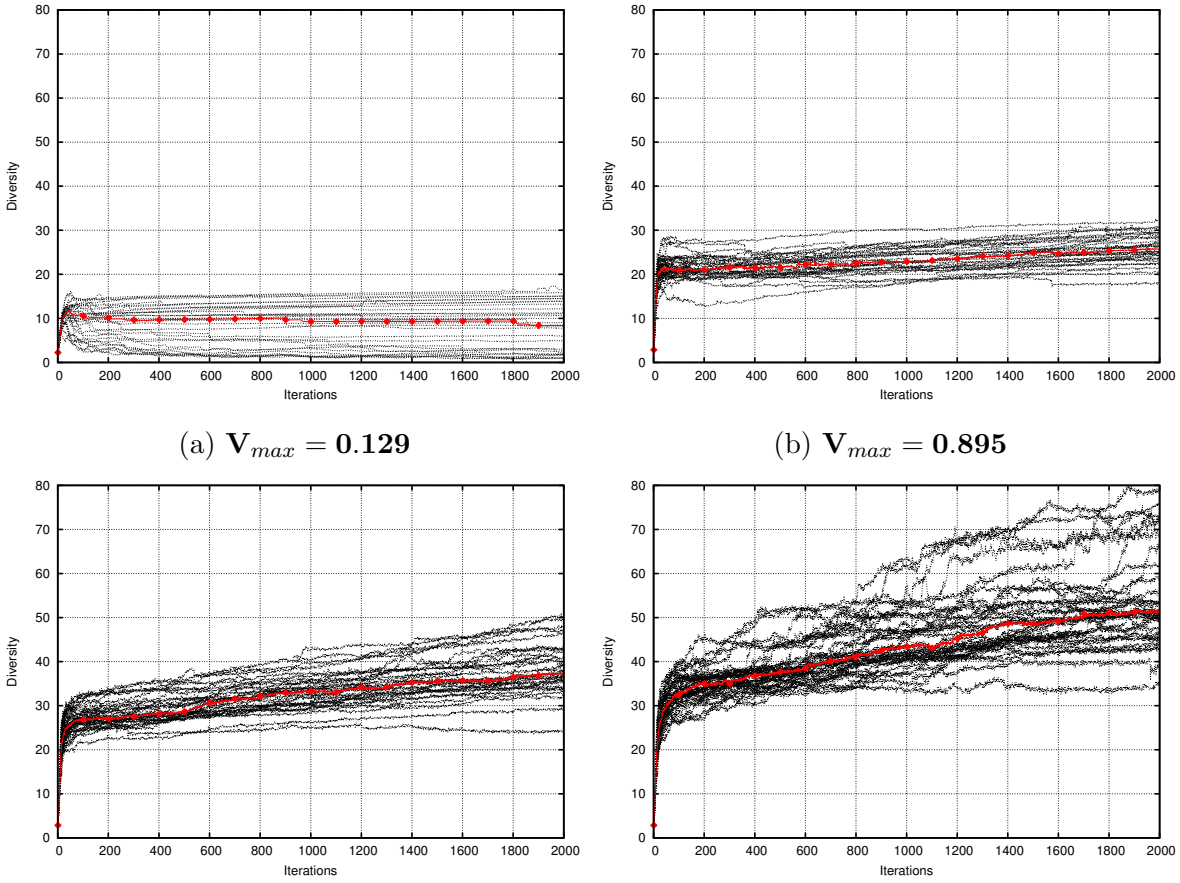
Figure 4.1 illustrates the effect of $\mathbf{V}_{max}$ on the swarm diversity over time. The figure shows the diversity, per sample, on the WDBC data set for increasing values of $\mathbf{V}_{max}$. When compared to the results in Section 3.4, the introduction of the $\mathbf{V}_{max}$ parameter has indeed had the desired effect of reducing the swarm divergence. As seen in Figure 4.1a, for small $\mathbf{V}_{max}$ values, the swarm goes through an initial phase of intense exploration, ending within approximately 100 iterations. After which the swarm diversity decreased until finally stagnating. However, the final diversity values are still relatively large for some samples, indicating the swarm did not converge on a single point.

When compared with Figures 4.1b, 4.1c and 4.1d, it is seen that for larger $\mathbf{V}_{max}$ values (generally for $\mathbf{V}_{max} > 1.0$, depending on the data set), the swarm has a far shorter (if any) period where diversity decreases and instead continues to explore throughout the optimisation process, with the divergence increasing in severity as the value of $\mathbf{V}_{max}$ increased. This behaviour was consistent across all data sets.

Due to the large diversities seen for $\mathbf{V}_{max} > 1$, which continue to increase as $\mathbf{V}_{max}$

increases, it is clear that the lack of a $\mathbf{V}_{max}$ parameter is not the underlying cause of the swarm divergence. Instead, the use of a sufficiently small (depending on the data set) $\mathbf{V}_{max}$ limits any divergence that does occur.

The effect of $\mathbf{V}_{max}$, or indeed the convergence of the swarm, on overfitting, training and generalisation accuracy is investigated next.



(a) $\mathbf{V}_{max} = \mathbf{0.129}$ (b) $\mathbf{V}_{max} = \mathbf{0.895}$

(c) $\mathbf{V}_{max} = \mathbf{3.812}$ (d) $\mathbf{V}_{max} = \mathbf{10.02}$

Figure 4.1: Swarm diversity, per sample, vs. iterations for the WDBC data set with increasing $\mathbf{V}_{max}$ values. The median result is shown in red (diamonds)

4.2.2 Accuracy

Training Accuracy

Table 4.2 shows the training error means for each of the $\mathbf{V}_{max}$ values across all data sets. The standard deviations indicate the results were consistent with the reported mean values.

The effect of the $\mathbf{V}_{max}$ parameter on the training accuracy was dependant on the data set. Primarily two behaviours were observed.

For the Diabetes, Glass, WDBC and Wine classification data sets the lowest training errors were achieved for a $\mathbf{V}_{max} < 0.2$. Furthermore, as seen in Table 4.2 a non-monotonic *increase* in the training errors were observed as the $\mathbf{V}_{max}$ parameter increased.

In the case of the other data sets (all regression data sets as well as the Iris data set), the observed training errors remained stable, merely fluctuating for the different values of $\mathbf{V}_{max}$. Notwithstanding, for the Iris, Mackey-Glass and TS5 data sets the PSO still achieved the lowest training errors for a $\mathbf{V}_{max} < 1.0$.

The difference in the two PSO behaviours on the data sets is clearly shown in Figure 4.2. As shown in the figure, there is an initial decrease in training error for the Logistic Map data set for $\mathbf{V}_{max} < 0.1$ after which it remains stable around $E_T = 0.0008$. However, for the Wine data set there is a clear increase in training error as $\mathbf{V}_{max}$ increases, peaking around $\mathbf{V}_{max} = 6.0$.

Also illustrated in Table 4.2, any effect $\mathbf{V}_{max}$ did have on the training error diminished beyond, approximately, a value of 6 regardless of data set.

In summary, the results show that lower $\mathbf{V}_{max}$ values, $\mathbf{V}_{max} < 1.0$, lead to an improvement in training accuracy for eight of the ten data sets tested, especially in the case of the larger data sets. The severity of the effect is however largely dependent on the data set. Considering the diversity results for the $\mathbf{V}_{max}$ values in this range, it is likely that the reduced divergence of the swarm allowed for exploitation and improvement of any optima found, thereby decreasing the training error. The effect of $\mathbf{V}_{max}$ on the training errors also seemed to reached a plateau for $\mathbf{V}_{max} > 6.0$. Considering the high swarm diversity for $\mathbf{V}_{max} > 6.0$, it follows that the swarms likely failed to effectively find and exploit optima, reducing the training accuracy of the FFNNs. Most importantly, despite

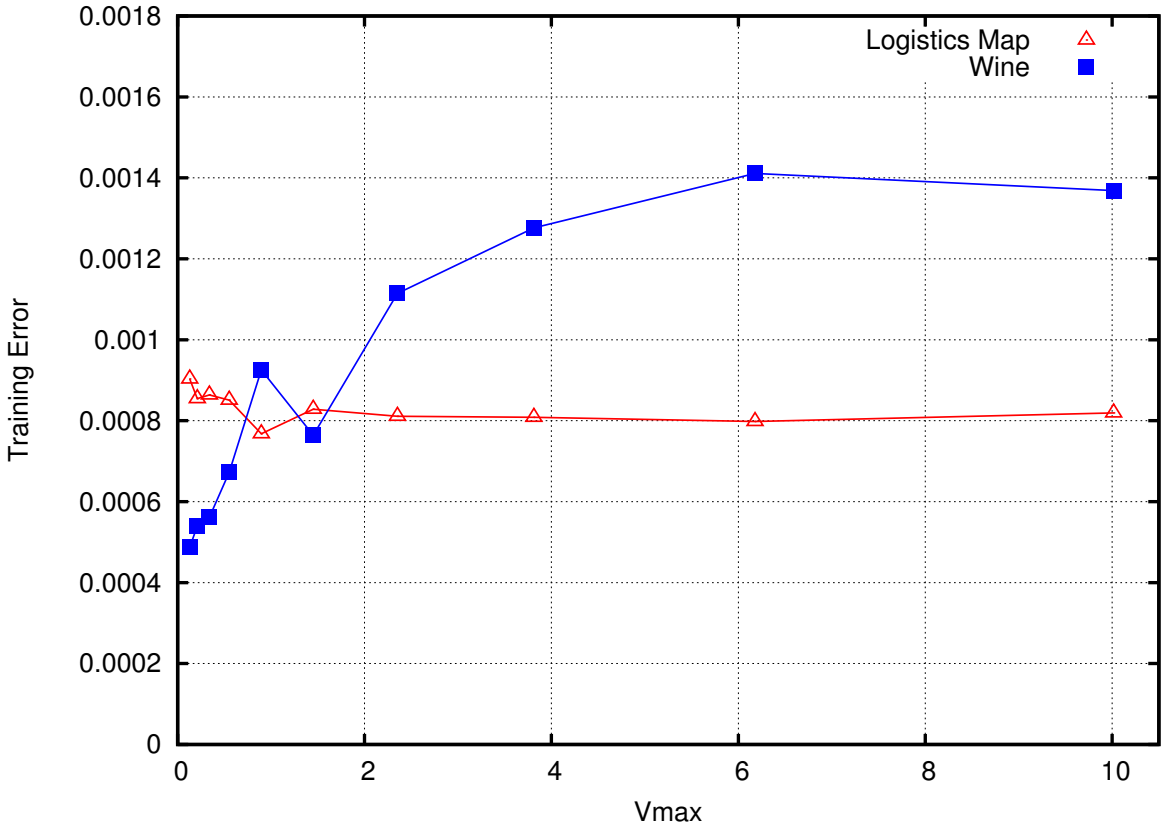


Figure 4.2: E_T for each value of $\mathbf{V}_{max}$ on the Logistic Map and Wine data sets.

small $\mathbf{V}_{max}$ values improving swarm convergence, the swarms still failed to converge to a single point within the available iterations.

Generalisation Accuracy

The generalisation error results are given in Table 4.3. Similar to the training error results, the standard deviations indicate results were consistent with the mean value across all data sets for each value of $\mathbf{V}_{max}$.

For most data sets, $\mathbf{V}_{max}$ had little effect on the generalisation error, although a marginal *decrease* in the generalisation error is noticeable for the Diabetes data set as $\mathbf{V}_{max}$ is increased. The specific $\mathbf{V}_{max}$ value for which the lowest generalisation error was obtained differed greatly between data sets.

For illustrative purposes the generalisation errors for the Wine and Logistic Map data sets are shown in Figure 4.3. The errors are logarithmically scaled to account for the difference in magnitude. Though some fluctuation in the errors can be seen, no clear trend emerges. Similar to the training error, any change in generalisation error seems to lessen in severity as the $\mathbf{V}_{max}$ value increases. This was characteristic across all the data sets.

Generally, the results show that $\mathbf{V}_{max}$ and by extension the diversity had little effect on the generalisation performance. Since the generalisation error is not directly optimised by the PSO (as opposed to the training error which is used as the objective function), it would follow that a change in the swarm's optimisation behaviour, in this case, the divergence, could potentially have little effect on the actual generalisation of the FFNNs, as is the case here.

However, considering an increase in training error relative to an increasing $\mathbf{V}_{max}$ and a constant generalisation error, it is expected that overall, overfitting should decrease with an increase in $\mathbf{V}_{max}$, an effect more clearly seen through the generalisation factor.

Generalisation Factor

Finally, the generalisation factor results are given in Table 4.4. The Diabetes, Glass, Wine and WDBC data sets showed a marked decrease in the generalisation factor, which indicates a reduction in overfitting, as $\mathbf{V}_{max}$ increased. However, as mentioned above, this is expected, as a decrease in E_T (as was the case with these four data sets) along with a (near) constant E_G will result in a decrease of the generalisation factor.

This shows a weakness in the use of the generalisation factor as the sole indicator of overfitting, since, as shown here, generalisation was unaffected and the decrease in ρ_F was due to the reduced accuracy on the training data.

In terms of the regression data sets, lower generalisation factors were generally obtained for lower $\mathbf{V}_{max}$ values, as opposed to higher values as with the classification data sets. On the Logistic Map, Mackey-Glass, Sunspot and TS5 data sets, the generalisation factor remained roughly the same across the range of $\mathbf{V}_{max}$, with only marginal increases as $\mathbf{V}_{max}$ increased. This is consistent with expectations, as $\mathbf{V}_{max}$ did not have as drastic an effect on the training accuracy on these data sets.

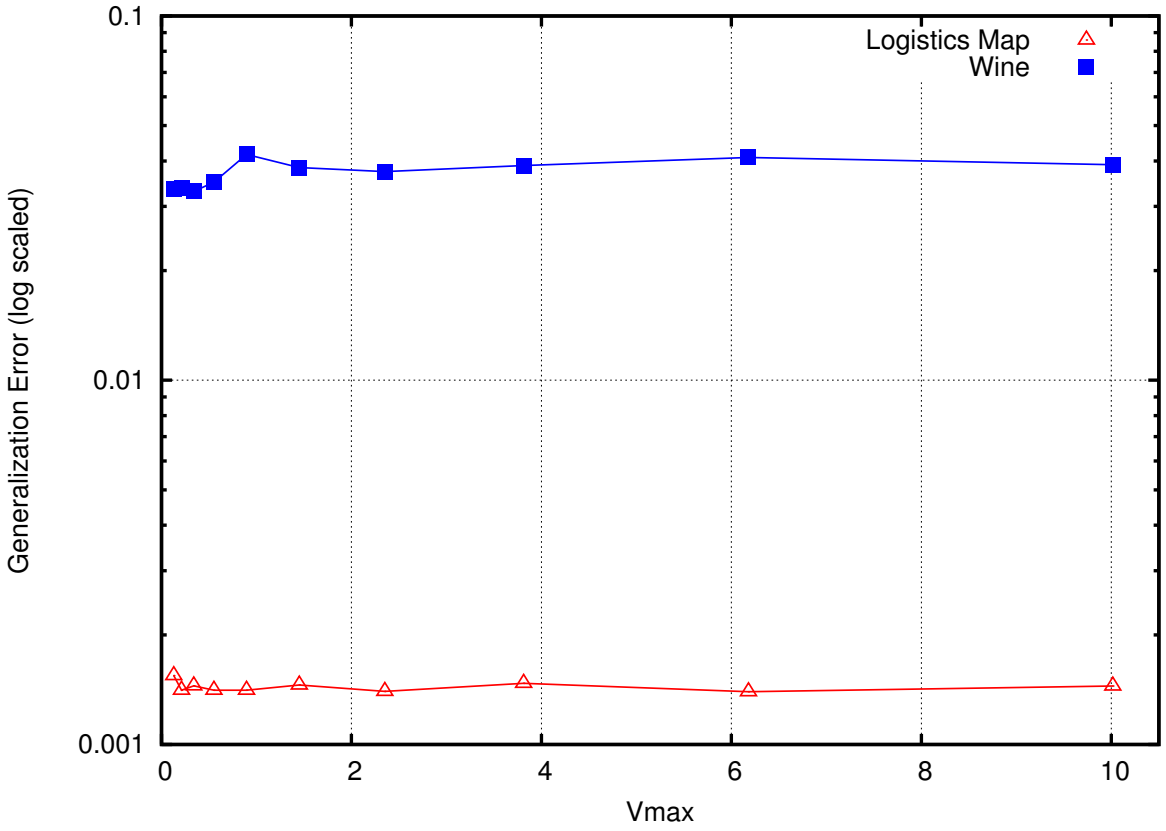


Figure 4.3: E_G (log scaled) for each value of $\mathbf{V}_{max}$ on the Logistic Map and Wine data sets.

The effect of the generalisation factors on a classification and a regression data set is graphed in Figure 4.4 for the WDBC and Logistic Map data sets. The figure clearly shows the decrease in overfitting on the WDBC data set and the slight increase in generalisation factor on the Logistic Map data set. Again, any effect became less pronounced as $\mathbf{V}_{max}$ increased.

4.2.3 Discussion

In terms of $\mathbf{V}_{max}$'s effect on the PSO algorithm itself it was shown that the lack of a $\mathbf{V}_{max}$ parameter was not the cause of the divergence of the swarms seen in the results of Chapter 3, but divergence was reduced by the use of a $\mathbf{V}_{max}$ in the range $(0.0, 1.0]$. Generally, for $\mathbf{V}_{max} > 1.0$ the swarms failed to converge during the course of optimisation, with the

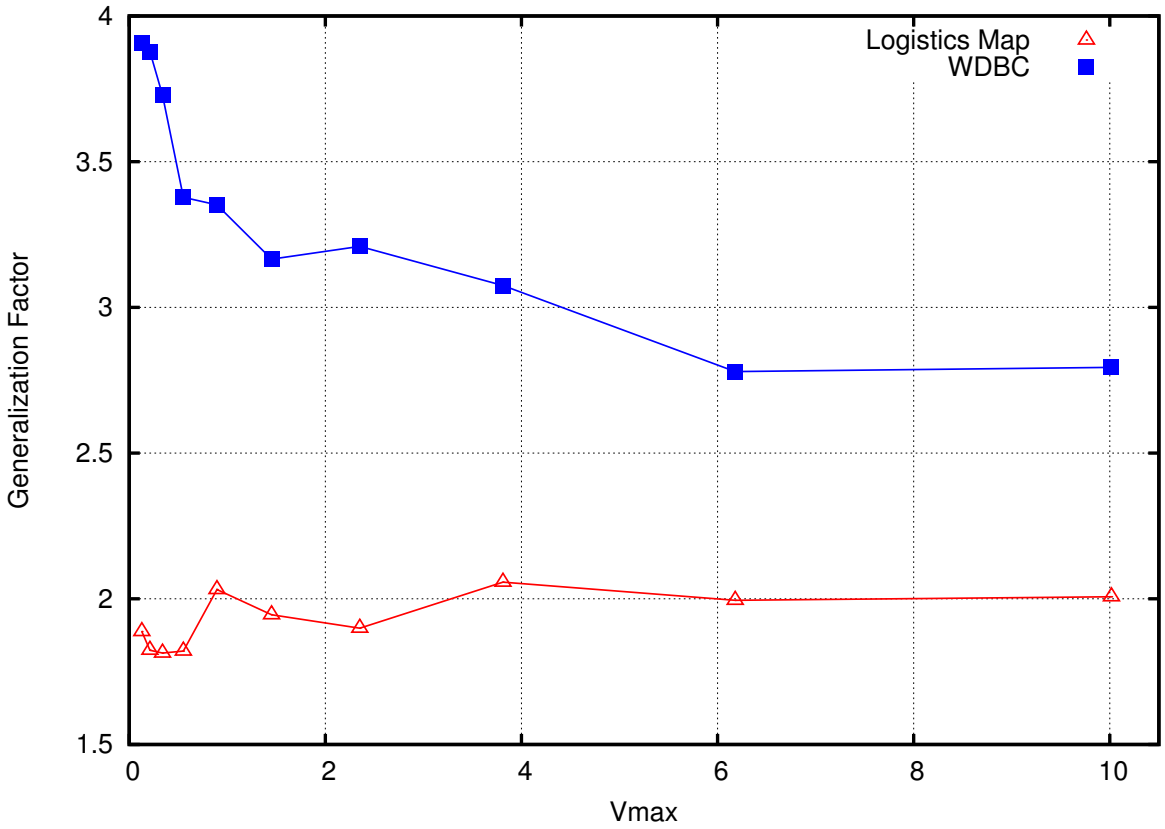


Figure 4.4: ρ_F for each value of $\mathbf{V}_{max}$ on the Logistic Map and WDBC data sets.

effect becoming worse as $\mathbf{V}_{max}$ was increased.

An assumption of this investigation was the activation function used, in this case the standard sigmoid function. Further investigation is required into the effect of modified sigmoid and other activation functions, including whether a small $\mathbf{V}_{max}$ is still appropriate. This is discussed in a later chapter.

Further, it was shown that $\mathbf{V}_{max}$ largely effects only the training error, likely due to the parameter's effect on the diversity. On the larger (both in terms of the number of patterns and the search space dimensionality) data sets, the training accuracy was improved for smaller $\mathbf{V}_{max}$ values. However, the generalisation accuracy remained unaffected regardless of the value of $\mathbf{V}_{max}$.

On the less complex data sets, $\mathbf{V}_{max}$ had little effect on accuracy (both training and generalisation), with all errors remaining roughly the same for the values of $\mathbf{V}_{max}$ that

were tested.

4.3 Summary

This chapter investigated the effect of the PSO $\mathbf{V}_{max}$ parameter on swarm diversity and overfitting in the context of training FFNNs. Section 4.1 described the methodology used for the investigation, particularly the method of exploring the $\mathbf{V}_{max}$ parameter space along with other algorithm considerations. The results and a discussion thereof was given in Section 4.2.

A $\mathbf{V}_{max} \approx 1.0$ offered accuracy results comparable to higher values, while reducing divergence. None of the tested $\mathbf{V}_{max}$ parameters could however guarantee convergence of the swarm.

The next chapter similarly investigates the effect of the other control parameter involved in the PSO's velocity update equation, namely the *inertia* and *social* and *cognitive acceleration* coefficients.

Table 4.1: Results showing $\mathcal{D}_{avg}$ means with standard deviations in parenthesis for the PSO algorithm after 50 000 objective function evaluations. Figures in bold indicate lowest value for the data set.

Problem	$\mathbf{V}_{max}$									
	0.129	**0.21**	**0.34**	**0.552**	**0.895**	**1.451**	**2.352**	**3.812**	**6.179**	**10.02**
Diabetes	6.373	**5.499**	7.402	10.34	15.26	20.53	23.97	29.80	35.69	43.10
	(3.612)	(3.876)	(4.571)	(4.191)	(3.652)	(2.860)	(4.424)	(5.919)	(6.700)	(9.223)
Glass	**4.602**	5.693	8.388	9.620	14.17	16.41	20.74	25.09	27.55	33.02
	(3.440)	(3.975)	(4.373)	(3.988)	(3.972)	(2.352)	(3.366)	(7.063)	(9.348)	(10.45)
Iris	**4.692**	6.855	7.247	9.437	12.68	16.70	19.45	22.20	23.88	28.98
	(3.920)	(3.817)	(4.922)	(3.878)	(6.257)	(5.395)	(7.645)	(5.417)	(5.692)	(12.32)
WDBC	**8.267**	9.784	12.77	16.32	22.39	25.92	33.55	38.55	46.66	54.69
	(5.455)	(5.371)	(6.158)	(5.620)	(4.427)	(3.251)	(4.023)	(6.018)	(6.516)	(11.31)
Wine	**3.494**	4.724	6.469	8.878	10.44	13.32	15.43	19.49	21.05	25.48
	(3.482)	(3.716)	(3.390)	(3.114)	(2.945)	(2.525)	(3.870)	(4.070)	(4.395)	(8.558)
Henon	**2.299**	3.467	4.799	6.575	8.932	8.578	12.14	13.26	19.83	22.75
Map	(2.249)	(2.383)	(2.581)	(2.002)	(1.676)	(3.075)	(3.268)	(5.178)	(9.348)	(1.020)
Log.	**2.933**	4.099	4.913	6.021	7.224	8.707	10.29	11.82	12.79	14.17
Map	(1.561)	(1.528)	(1.364)	(1.485)	(1.310)	(1.935)	(2.204)	(3.251)	(4.122)	(5.782)
M.-Glass	**4.083**	4.441	5.880	7.853	9.303	11.13	13.29	14.63	17.84	18.12
	(1.381)	(1.738)	(1.332)	(1.054)	(1.445)	(1.605)	(2.603)	(2.748)	(5.817)	(4.439)
Sunspots	**3.760**	4.475	5.419	6.751	9.078	11.65	13.26	16.49	23.49	22.30
	(2.050)	(2.311)	(2.136)	(2.789)	(2.311)	(4.411)	(3.196)	(4.850)	(1.378)	(1.280)
TS5	**2.443**	2.614	4.357	6.051	7.745	9.109	12.58	16.30	19.35	27.84
	(1.768)	(2.054)	(1.817)	(1.722)	(3.297)	(4.433)	(6.691)	(1.118)	(1.013)	(2.146)

Table 4.2: Results showing E_T means with standard deviations in parenthesis for the PSO algorithm after 50 000 objective function evaluations. Figures in bold indicate lowest value for the data set.

Problem	$\mathbf{V}_{max}$									
	0.129	**0.21**	**0.34**	**0.552**	**0.895**	**1.451**	**2.352**	**3.812**	**6.179**	**10.02**
Diabetes	**1.116e-1**	1.121e-1	1.133e-1	1.134e-1	1.130e-1	1.151e-1	1.151e-1	1.162e-1	1.156e-1	1.174e-1
	(5.35e-3)	(5.73e-3)	(6.36e-3)	(5.31e-3)	(5.48e-3)	(4.83e-3)	(5.96e-3)	(5.81e-3)	(4.86e-3)	(5.22e-3)
Glass	2.629e-1	2.691e-1	**2.549e-1**	2.634e-1	2.773e-1	2.733e-1	2.815e-1	2.922e-1	2.895e-1	2.885e-1
	(4.80e-2)	(5.22e-2)	(5.16e-2)	(5.13e-2)	(5.19e-2)	(5.59e-2)	(6.66e-2)	(5.55e-2)	(6.06e-2)	(6.18e-2)
Iris	9.330e-3	9.659e-3	8.756e-3	8.715e-3	**7.538e-3**	8.465e-3	7.738e-3	8.302e-3	8.824e-3	8.907e-3
	(5.07e-3)	(4.68e-3)	(5.40e-3)	(4.78e-3)	(4.24e-3)	(4.67e-3)	(3.71e-3)	(4.26e-3)	(4.60e-3)	(4.86e-3)
WDBC	**8.268e-3**	8.627e-3	8.725e-3	9.517e-3	1.011e-2	1.059e-2	1.100e-2	1.135e-2	1.210e-2	1.144e-2
	(1.77e-3)	(1.93e-3)	(1.94e-3)	(1.80e-3)	(2.10e-3)	(2.35e-3)	(2.34e-3)	(2.31e-3)	(2.42e-3)	(2.34e-3)
Wine	**4.876e-4**	5.397e-4	5.612e-4	6.725e-4	9.255e-4	7.642e-4	1.114e-3	1.276e-3	1.411e-3	1.368e-3
	(3.85e-4)	(3.11e-4)	(4.00e-4)	(5.24e-4)	(6.94e-4)	(5.95e-4)	(7.62e-4)	(1.06e-3)	(1.22e-3)	(8.58e-4)
Henon	1.160e-3	1.241e-3	1.147e-3	1.047e-3	1.097e-3	1.056e-3	9.333e-4	**7.164e-4**	8.012e-4	8.790e-4
Map	(4.19e-4)	(4.24e-4)	(4.18e-4)	(3.29e-4)	(4.71e-4)	(4.80e-4)	(4.28e-4)	(4.06e-4)	(3.85e-4)	(4.06e-4)
Logistic	9.031e-4	8.545e-4	8.635e-4	8.506e-4	**7.679e-4**	8.283e-4	8.111e-4	8.083e-4	7.982e-4	8.194e-4
Map	(1.92e-4)	(1.78e-4)	(1.60e-4)	(1.74e-4)	(1.61e-4)	(1.68e-4)	(1.74e-4)	(2.06e-4)	(1.87e-4)	(1.98e-4)
Mackey-	8.506e-4	8.236e-4	8.095e-4	8.163e-4	**7.442e-4**	7.955e-4	7.980e-4	7.893e-4	7.547e-4	7.993e-4
Glass	(1.86e-4)	(1.82e-4)	(1.53e-4)	(1.61e-4)	(1.65e-4)	(1.69e-4)	(1.63e-4)	(1.55e-4)	(1.64e-4)	(1.71e-4)
Sunspots	157.3	157.5	156.7	156.8	153.9	**152.0**	155.4	154.3	158.3	155.4
	(15.30)	(15.33)	(17.30)	(15.05)	(15.23)	(16.11)	(14.61)	(18.13)	(18.77)	(15.31)
TS5	1.983e-3	1.991e-3	1.995e-3	1.971e-3	**1.962e-3**	1.987e-3	2.039e-3	2.027e-3	2.027e-3	2.021e-3
	(1.42e-4)	(1.36e-4)	(1.64e-4)	(1.49e-4)	(1.18e-4)	(1.46e-4)	(1.56e-4)	(1.42e-4)	(1.36e-4)	(1.44e-4)

Table 4.3: Results showing E_G means with standard deviations in parenthesis for the PSO algorithm after 50 000 objective function evaluations. Figures in bold indicate lowest value for the data set.

Problem	$\mathbf{V}_{max}$									
	0.129	**0.21**	**0.34**	**0.552**	**0.895**	**1.451**	**2.352**	**3.812**	**6.179**	**10.02**
Diabetes	1.770e-1	1.749e-1	1.748e-1	1.735e-1	1.730e-1	1.742e-1	1.721e-1	1.694e-1	1.716e-1	**1.690e-1**
	(1.20e-2)	(1.31e-2)	(1.24e-2)	(1.24e-2)	(1.24e-2)	(1.34e-2)	(1.05e-2)	(1.21e-2)	(1.14e-2)	(1.01e-2)
Glass	1.067	1.105	1.081	1.073	1.031	1.045	1.063	1.047	**1.020**	1.088
	(3.19e-1)	(2.78e-1)	(3.27e-1)	(2.66e-1)	(3.17e-1)	(2.60e-1)	(3.10e-1)	(3.26e-1)	(2.99e-1)	(2.93e-1)
Iris	3.471e-2	3.335e-2	3.656e-2	3.583e-2	3.611e-2	3.711e-2	3.552e-2	3.355e-2	**3.328e-2**	3.544e-2
	(1.96e-2)	(1.52e-2)	(1.81e-2)	(2.12e-2)	(1.72e-2)	(1.82e-2)	(1.71e-2)	(1.74e-2)	(1.60e-2)	(1.95e-2)
WDBC	**2.987e-2**	3.121e-2	2.998e-2	3.002e-2	3.149e-2	3.084e-2	3.293e-2	3.252e-2	3.142e-2	2.993e-2
	(7.78e-3)	(7.40e-3)	(8.27e-3)	(7.50e-3)	(8.37e-3)	(7.58e-3)	(1.01e-2)	(9.35e-3)	(9.21e-3)	(8.16e-3)
Wine	3.339e-2	3.377e-2	**3.300e-2**	3.496e-2	4.168e-2	3.836e-2	3.740e-2	3.886e-2	4.087e-2	3.908e-2
	(1.54e-2)	(1.57e-2)	(1.47e-2)	(1.49e-2)	(1.78e-2)	(1.59e-2)	(1.77e-2)	(2.20e-2)	(1.94e-2)	(1.70e-2)
Henon	8.766e-3	**6.401**e-3	9.558e-3	6.439e-3	1.630e-2	8.337e-3	1.083e-2	1.320e-2	1.660e-2	1.404e-2
Map	(1.20e-2)	(4.96e-3)	(1.32e-2)	(6.38e-3)	(3.43e-2)	(7.54e-3)	(1.53e-2)	(2.54e-2)	(4.54e-2)	(3.96e-2)
Logistic	1.545e-3	1.410e-3	1.449e-3	1.409e-3	1.410e-3	1.456e-3	1.398e-3	1.472e-3	**1.396**e-3	1.447e-3
Map	(6.13e-4)	(6.06e-4)	(5.90e-4)	(5.60e-4)	(5.77e-4)	(5.99e-4)	(6.22e-4)	(6.20e-4)	(7.10e-4)	(6.83e-4)
Mackey-	1.372e-3	1.395e-3	1.363e-3	**1.305**e-3	1.372e-3	1.395e-3	1.329e-3	1.354e-3	1.353e-3	1.330e-3
Glass	(6.10e-4)	(6.19e-4)	(5.96e-4)	(5.19e-4)	(5.61e-4)	(6.07e-4)	(5.80e-4)	(5.91e-4)	(5.95e-4)	(5.22e-4)
Sunspots	227.3	226.2	**224.3**	228.1	231.9	230.0	233.0	232.6	235.4	234.7
	(38.35)	(39.63)	(36.19)	(41.29)	(43.32)	(47.16)	(34.00)	(41.11)	(50.83)	(46.78)
TS5	4.499e-3	**4.223e-3**	4.546e-3	4.464e-3	4.799e-3	4.584e-3	4.930e-3	4.419e-3	4.557e-3	4.557e-3
	(1.28e-3)	(1.15e-3)	(1.32e-3)	(1.05e-3)	(1.32e-3)	(1.05e-3)	(1.44e-3)	(9.98e-4)	(1.57e-3)	(1.11e-3)

Table 4.4: Results showing ρ_F means with standard deviations in parenthesis for the PSO algorithm after 50 000 objective function evaluations. Figures in bold indicate lowest value for the data set.

Problem	$\mathbf{V}_{max}$									
	0.129	**0.21**	**0.34**	**0.552**	**0.895**	**1.451**	**2.352**	**3.812**	**6.179**	**10.02**
Diabetes	1.594	1.568	1.552	1.536	1.538	1.519	1.503	1.465	1.489	**1.444**
	(1.78e-1)	(1.89e-1)	(1.91e-1)	(1.62e-1)	(1.73e-1)	(1.72e-1)	(1.60e-1)	(1.62e-1)	(1.49e-1)	(1.42e-1)
Glass	4.313	4.392	4.600	4.318	4.015	4.126	4.133	3.865	**3.875**	4.056
	(1.77)	(1.82)	(2.16)	(1.61)	(2.07)	(1.80)	(1.89)	(1.87)	(1.90)	(1.70)
Iris	8.781	8.669	11.39	10.95	11.02	10.00	7.666	**7.129**	8.052	9.932
	(13.2)	(15.9)	(16.3)	(17.6)	(17.2)	(18.4)	(9.03)	(8.98)	(11.2)	(18.8)
WDBC	3.907	3.878	3.728	3.378	3.352	3.166	3.209	3.075	**2.780**	2.794
	(1.61)	(1.44)	(1.58)	(1.37)	(1.43)	(1.33)	(1.31)	(1.33)	(1.13)	(1.15)
Wine	144.4	94.73	126.6	102.2	139.6	81.33	56.05	65.73	68.1	**48.70**
	(176.0)	(95.5)	(166.0)	(159.0)	(328.0)	(66.9)	(53.4)	(69.7)	(144.0)	(54.2)
Henon	10.26	**5.866**	10.15	7.476	14.64	10.01	15.16	36.31	20.59	14.95
Map	(20.5)	(5.40)	(13.7)	(9.23)	(21.4)	(10.6)	(24.8)	(114.0)	(37.7)	(32.6)
Logistic	1.888	1.823	**1.814**	1.821	2.032	1.945	1.899	2.057	1.995	2.007
Map	(1.01)	(1.05)	(9.53e-1)	(1.00)	(1.14)	(1.13)	(1.10)	(1.19)	(1.40)	(1.39)
Mackey-	1.788	1.878	1.843	**1.752**	2.052	1.972	1.848	1.881	1.978	1.834
Glass	(1.03)	(1.06)	(1.04)	(9.32e-1)	(1.17)	(1.19)	(1.09)	(1.08)	(1.14)	(1.05)
Sunspot	1.478	**1.467**	1.467	1.486	1.536	1.553	1.523	1.552	1.528	1.547
	(0.383)	(0.378)	(0.379)	(0.394)	(0.394)	(0.478)	(0.330)	(0.444)	(0.474)	(0.448)
TS5	2.288	**2.142**	2.319	2.282	2.474	2.319	2.446	2.202	2.283	2.272
	(7.00e-1)	(6.43e-1)	(7.89e-1)	(5.87e-1)	(7.66e-1)	(5.43e-1)	(7.78e-1)	(5.56e-1)	(9.03e-1)	(6.00e-1)

Chapter 5

PSO Velocity Parameters

Chapter 4 investigated the effect of the $\mathbf{V}_{max}$ control parameter on FFNN accuracy and overfitting. The purpose of this chapter is to similarly investigate the influence of the additional control parameters involved in the calculation of a particle's velocity, that is, the *inertia* weight, *cognitive* and *social* acceleration control parameters. The goal of the analysis is not merely to find the absolute best parameters for each data set, but rather to attempt to discover any commonality in parameter choices or defining interactions across all data sets that favourably or adversely affect FFNN performance, in terms of accuracy and overfitting.

The experimental methodology used to investigate the effect of the control parameters is given in Section 5.1. The results and discussion of the findings are given Section 5.2. Section 5.3 summarises the chapter.

5.1 Experimental Methodology

This section discusses the specific methods used in this chapter to explore and investigate the effect of the inertia weight (ω), cognitive (c_1) and social (c_2) acceleration control parameters on FFNN accuracy and overfitting.

The effect and purpose of the acceleration coefficient and inertia control parameters in the PSO algorithm has been discussed in Section 2.3.2. An important aspect of these parameters, as detailed in Section 2.3.2, is the co-dependent interaction between the

parameters and their affect on swarm convergence. Due to this interaction, it is inappropriate to analyse any one of the parameters in isolation, as specific combinations of parameter values might affect performance in specific ways. As such, a method was required where all three parameters are analysed simultaneously, including any interaction between the parameters.

To this end, an exploratory data analysis technique that makes use of low-discrepancy sequences (LDS) and high dimensional data visualisation was used [32]. This method of exploring the parameter space holds a number of advantages suitable to the investigation of the control parameters in this chapter, namely [32]:

- Discovering any interactions between parameters.
- Avoiding biased step sizes in the parameter space.
- Analysing a representative sample of the parameter search spaces.
- Limiting the required number of experiment simulations that needs to be performed.

LDSs are used to generate values in the control parameter search space and subsequently parallel coordinate plots to visualise the parameters along with the algorithm performance measurements. These techniques are briefly discussed below with their application in the experimental work discussed in Section 5.1.3.

Low-Discrepancy Sequences

Low-discrepancy sequences, also known as quasi-random numbers, are numerical sequences for which the elements have a low discrepancy (are roughly equidistributed) for any size of the sequence [35]. The purpose of LDSs are to *uniformly* fill the unit hypercube as evenly as possible while still maintaining the appearance of randomness [35]. This effectively samples a given area in a uniform manner while avoiding potential aliasing which occurs when using a fixed step size to sample the space. For the purposes of this study, Sobol sequences (a specific type of LDS) were used [97, 98], primarily due to the efficiency of their generation and their previous use in analysis of the parameter space in CI algorithms [32, 54, 108].

Parallel coordinates

Parallel coordinates, or parallel coordinate plots, is a method which "induces a nonprojective [sic] mapping between n-dimensional and two-dimensional sets", thereby enabling the visualisation of high dimensional data on a two-dimensional plot [51, 111]. The primary advantage of using parallel coordinates in analysing the effect of parameters is that it greatly aids in the identification of clusters or trends of data through human pattern recognition. An example of a parallel coordinate plot of 7-dimensional data is given in Figure 5.1, showing clustering of the data in the fourth and fifth dimensions.

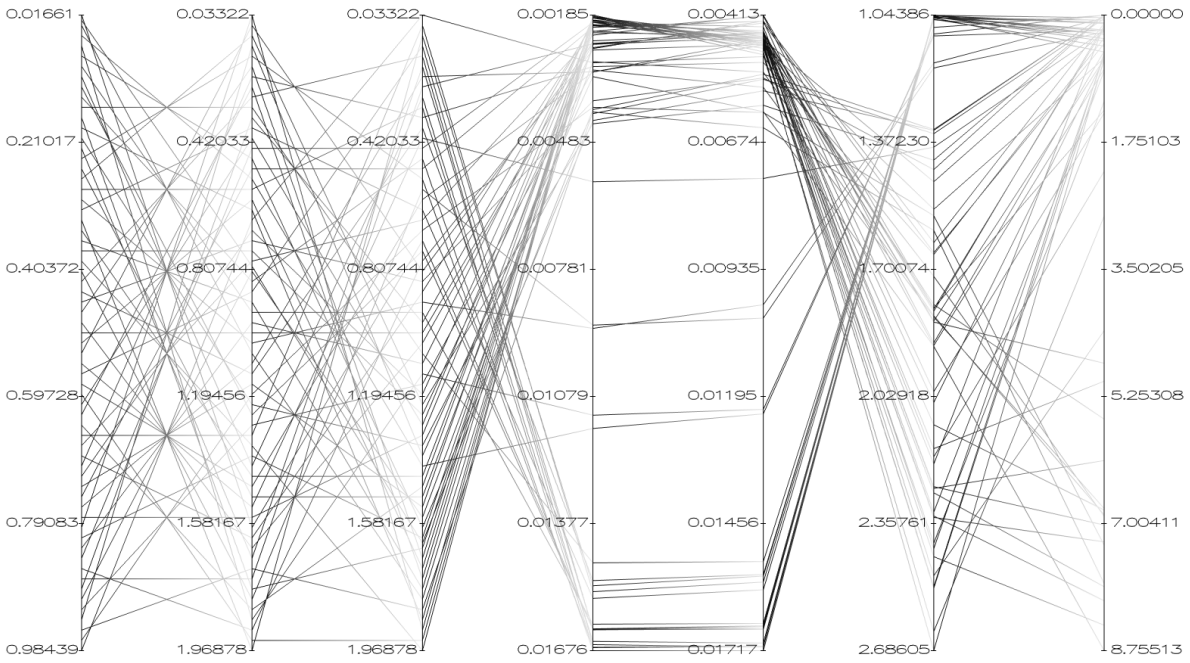


Figure 5.1: Example parallel coordinate plot of 7-dimensional data.

5.1.1 Data Sets

The ten data sets (five classification and five regression data sets) described in Section 3.2 were used for the purposes of this analysis. The NN architectures given in Table 3.1 were again used, making results comparable to those of previous chapters.

5.1.2 PSO Configuration

Similar to Chapter 4, the PSO algorithm used in this chapter was an *lbest* PSO with a neighbourhood size of 5. The swarm size was set to 25 particles. The number of fitness evaluations performed by the PSO were limited to 50000.

The particle positions were randomly initialised in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, with velocities initialised to $\mathbf{0}$ as described in Section 3.3.2.

A $\mathbf{V}_{max}$ value of $\mathbf{1}$ was used for all problem data sets. As shown in Chapter 4, this value aids in preventing swarm divergence from occurring whilst still allowing adequate swarm exploration.

5.1.3 Experimental Procedure

Analysis proceeded as follows: for a data set, a set of 64 3-dimensional Sobol points were generated. Each Sobol point represented a single control parameter configuration, with the first dimension representing ω, the second c_1 and the third c_2. Each dimension was scaled to a range known to be appropriate for the relevant parameter [103]: $0 < \omega < 1$; $0 < c_1 < 2$ and $0 < c_2 < 2$. The algorithm was then executed 30 times for each of the 64 parameter configurations using a Mersenne Twister PRNG with a unique seed. The mean performance over the 30 samples is reported. The 64 aggregated result sets were then visualised using parallel coordinates and analysed for any patterns or trends. This procedure was repeated for all data sets.

Algorithm Measurements

The training (E_T) and generalisation error (E_G) as well as the generalisation factor (ρ_F) as described in Section 3.3.3 were used to measure the performance of each execution of the algorithm. The diversities of the swarms were also recorded for the purpose of ensuring drastic swarm divergence (a swarm explosion) did not occur.

5.2 Results

The results of the parameter analysis are given in this section, followed by a discussion thereof.

Despite some parameter configurations violating the inequality given in Equation (2.13), the diversity results showed no drastic swarm divergence occurred for any of the parameter configurations on any of the data sets. This is likely due to the choice of $\mathbf{V}_{max} = 1.0$ which, as shown in Chapter 4, sufficiently aids in preventing drastic swarm divergence within the permitted 50000 fitness evaluations.

5.2.1 Individual Data Set Results

The parameters that lead to the *lowest* generalisation errors per data set are shown in Table 5.1. For eight of the ten data sets a ω value in the range of $(0.59, 0.74)$ lead to the lowest generalisation error. Unlike ω, the optimal values for c_1 and c_2 could not be consolidated to a common range for all data sets, with optimal values occurring throughout the range $(0.22053, 1.93756)$.

Table 5.1 also shows a $\rho_F > 1$ for all data sets, with exceptionally high ρ_F values occurring for the Iris, Wine, Henon Map and Mackey-Glass data sets. It is clear that for these data sets, the parameter configurations not only lead to the lowest E_G results, but also lead to very low training errors, to a point at which severe overfitting is shown to occur.

Table 5.2 shows the parameter configurations that lead to the highest generalisation errors per data set. Unlike the optimal parameter results, regions that lead to the absolute highest generalisation error for all data sets could be identified for each of the parameters: $\omega \in (0.14, 0.24)$, $c_1 \in (0, 1.0)$ and $c_2 \in (0, 0.5)$. These regions make intuitive sense as a low inertia and acceleration coefficients restrict particle movement, preventing effective optimisation.

These results are further discussed in Section 5.2.4.

Table 5.1: Parameter configurations that lead to the lowest generalisation error per data set.

Problem	ω	c_1	c_2	E_T	E_G	ρ_F
Diabetes	0.70342	0.22053	0.34541	0.14366	0.15887	1.10940
Glass	0.70342	0.22053	0.34541	0.55144	0.84202	1.61516
Iris	0.85952	1.90634	0.78247	0.01645	0.02870	2.13375
WDBC	0.25075	1.50050	1.50050	0.01674	0.02777	1.75793
Wine	0.59416	1.93756	1.93756	0.00118	0.03317	57.2235
Henon Map	0.71903	0.68881	1.18831	0.00064	0.00693	12.9213
Logistic Map	0.73464	1.15709	1.53172	0.00075	0.00127	1.86779
Mackey-Glass	0.62538	0.25175	1.75025	0.00060	0.00126	2.28892
Sunspots	0.46928	0.18931	1.68781	141.038	224.351	1.61574
TS5	0.67220	0.78247	0.65759	0.00232	0.00413	1.80070

5.2.2 Favourable Parameter Regions

The results shown in Tables 5.1 and 5.2 show only the single highest and lowest generalisation error results of the 64 configurations tested per data set. In order to more comprehensively identify appropriate ranges for the parameters and further analyse their interactions, the parallel coordinate plots for each data set were used.

An aggregate parallel coordinate plot of all tested data set results is shown in Figure 5.2. Errors are shown normalised to the range $[0.0, 1.0]$ and results are ranked according to the generalisation error.

Considering the training and generalisation error axes, the figure shows two clusters of results grouped at the upper and lower regions of each axis. This indicates that the tested parameter configurations had a type of binary effect on the overall performance: parameters either lead to the PSO finding an acceptable solution, and then refining the solution, or alternatively not finding a solution at all (or an exceptionally poor solution). Few of the parameter configurations tested fell in between these two extremes. This implies that the PSO is sensitive to the correct choice of inertia and acceleration parameters in order to find an acceptable solution when training FFNNs. The highlighted

Table 5.2: Parameter configurations that lead to the highest generalisation error per data set.

Problem	ω	c_1	c_2	E_T	E_G	ρ_F
Diabetes	0.14148	0.84491	0.47028	0.22527	0.22449	0.99702
Glass	0.14148	0.84491	0.47028	2.78991	2.88053	1.03811
Iris	0.23514	0.15809	0.53272	0.58071	0.61542	1.06298
WDBC	0.14148	0.84491	0.47028	0.22489	0.22682	1.00930
Wine	0.23514	0.15809	0.53272	0.55863	0.56210	1.00956
Henon Map	0.28197	0.56394	0.31419	0.41303	0.42738	1.05953
Logistic Map	0.14148	0.84491	0.47028	0.06436	0.06571	1.02358
Mackey-Glass	0.14148	0.84491	0.47028	0.06355	0.06490	1.02369
Sunspots	0.42245	0.28297	0.15809	1658.6	1754.8	1.06720
TS5	0.14148	0.84491	0.47028	0.01676	0.01717	1.04500

area, which shows the top 20% configurations by rank, is shown to nearly completely cover each of the parameter axes. This shows that no specific parameter regions can be identified for any of the parameters that lead to better FFNN accuracy across all data sets, further indicating the data set dependent nature of optimal parameters.

Figure 5.2 also shows the generalisation factor as the fourth axis. For the sake of comparison across the data sets, the generalisation factor results are normalised to the range $[0.0, 1.0]$. However, an absolute (non-normalised) $\rho_F > 1$, which would potentially indicate overfitting, was observed for all data sets and the normalised ρ_F serves as an indicator of the severity of overfitting.

The figure shows specific parameters which lead to both low training and generalisation errors, and therefore low generalisation factors, indicating a lack (or reduction) of overfitting. An example of such a parameter configuration is shown in Table 5.3 for the Diabetes data set. As seen in the table, configuration (1) obtained a significantly better generalisation error and lower generalisation factor compared with configuration (2).

Although certain specific parameter configurations such as these were found *on a per data set basis*, the figure again shows that the entire ρ_F axis falls in the highlighted

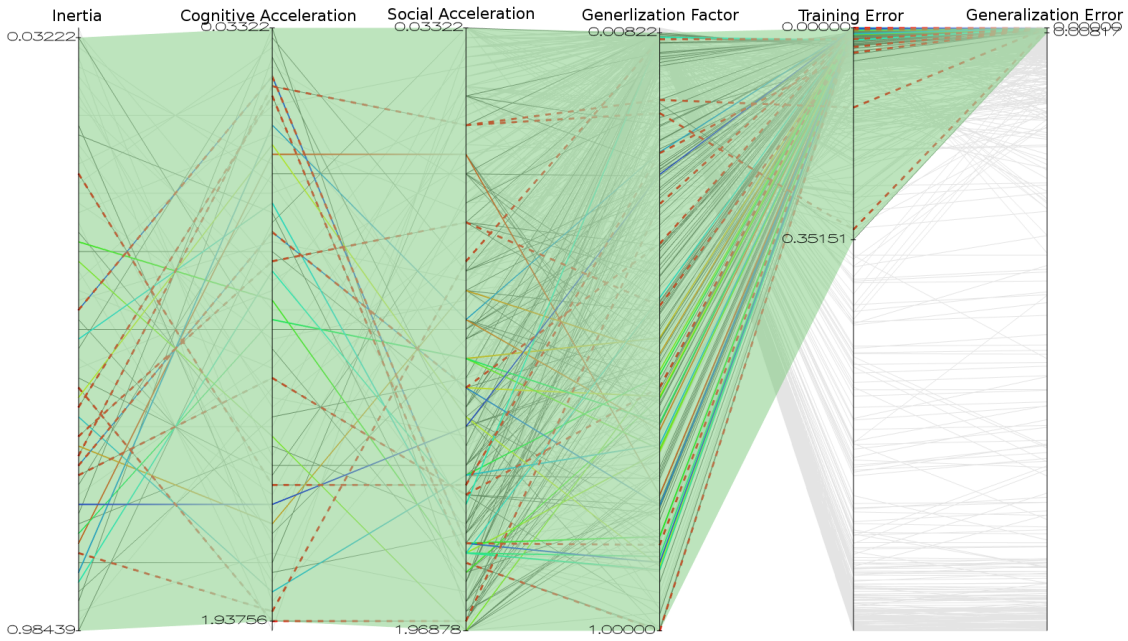


Figure 5.2: Parallel coordinate plot of all data set results, ranked according to E_G. Top 20% parameter configurations by rank are highlighted. Stippled lines indicate parameter configurations with absolute lowest E_G.

Table 5.3: Diabetes data set example of specific parameters leading to increased overfitting. Errors indicate mean over 30 samples.

	ω	c_1	c_2	E_T	E_G	ρ_F
1.	0.76586	1.09466	0.22053	0.14498	0.15978	1.10589
2.	0.62538	0.25175	1.75025	0.10266	0.18338	1.7966

area. This indicates that no specific pattern or interaction where parameters lead to good accuracy and reduced ρ_F values (such as in the example shown in Table 5.3) could be identified for all tested data sets.

Upon further analysis, one distinctive parameter *interaction* could however be identified consistently for all of the data sets. Although difficult to see in Figure 5.2, it was observed that for configurations where the *inertia* parameter was low, good generalisation was still obtained if the corresponding social acceleration value was high and

vice versa. In cases where both parameter values were high, generalisation performance was also good. More specifically, in order for the FFNN to obtain good generalisation accuracy, the PSO required either a $\omega > 0.5$ or a $c_2 > 1.0$, but not necessarily both. This observation was common for all tested data sets.

Table 5.4 breaks down the observations from Figure 5.2 for each data set, showing the extent of the parameter regions of the top 20% results individually for each of the data sets. The individual breakdown shows that, for four of the five regression data sets, the top 20% results fell in the smaller, as compared with the classification data sets, inertia region $(0.36, 0.89073]$, indicating that in general better network accuracy was obtained with slightly higher inertia values on the regression data sets.

Table 5.4: Extents of parameter regions associated with the top 20% generalisation error results per data set.

Problem	ω	c_1	c_2
Diabetes	$[0.02441, 0.98439]$	$[0.06444, 1.90634]$	$[0.03322, 1.81269]$
Glass	$[0.02441, 0.87513]$	$[0.06444, 1.90634]$	$[0.22053, 1.64098]$
Iris	$[0.17270, 0.93756]$	$[0.06444, 1.90634]$	$[0.25175, 1.65659]$
WDBC	$[0.03222, 0.87513]$	$[0.62638, 1.93756]$	$[0.25175, 1.96878]$
Wine	$[0.03222, 0.78147]$	$[0.18931, 1.93756]$	$[1.18831, 1.96878]$
Henon Map	$[0.39123, 0.89073]$	$[0.18931, 1.93756]$	$[0.50150, 1.96878]$
Logistic Map	$[0.36002, 0.89073]$	$[0.03322, 1.84391]$	$[0.43906, 1.90634]$
Mackey-Glass	$[0.36002, 0.89073]$	$[0.18931, 1.84391]$	$[0.43906, 1.96878]$
Sunspots	$[0.39123, 0.89073]$	$[0.18931, 1.93756]$	$[0.43906, 1.96878]$
TS5	$[0.03222, 0.89073]$	$[0.06444, 1.78147]$	$[0.43906, 1.81269]$

5.2.3 Adverse Parameter Regions

When considering the parameter configurations that lead to poor generalisation performance, in contrast to the best performing results, clear regions were identified that applied to all data sets tested. This is best illustrated in Figure 5.3 showing the results

for all data sets with the bottom 20% results highlighted, as ranked by generalisation error. E_T, E_G and ρ_F were again scaled to the range $[0.0, 1.0]$. The highlighted area shows a grouping of results on the lower half ($\omega < 0.65659$) of the inertia axis and a similar grouping on the low end of the c_2 axis ($c_2 < 0.93856$). This is complimentary to the configurations that lead to the overall best results shown in Table 5.1, where higher ω and c_2 values lead to the most accurate networks. Similar to the best performing configurations shown previously, the highlighted area contains nearly the entire c_1 axis, further indicating the parameter's non-dominance in terms of network accuracy. From this figure it is clear that configurations with both a low *inertia* and *social acceleration* lead to poor generalisation performance across all data sets.

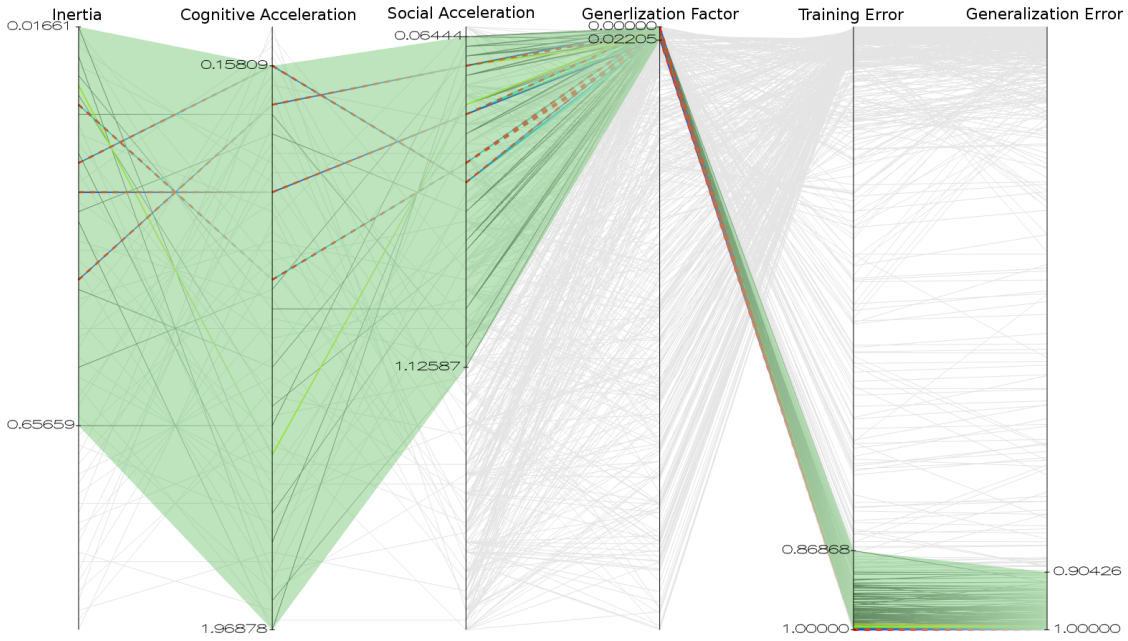


Figure 5.3: Parallel coordinate plot of all data set results, ranked according to E_G. Bottom 20% parameter configurations by rank are highlighted. Stippled lines indicate parameter configurations with absolute highest E_G.

Figure 5.3 also shows that the 20% parameter configurations that lead to poor generalisation performance also very clearly corresponded with significantly lower ρ_F values. The reason for the reduction can be seen in the training and generalisation errors: both errors were similarly poor and nearly equal (as illustrated by the straight lines between

the two error axes) producing a ρ_F ratio close to one. Whilst $\rho_F \approx 1.0$, the exceptionally poor errors are in this case indicative of underfitting.

Table 5.5 shows the extent of the parameter regions associated with the bottom 20% results broken down on a per data set basis. It is very clear from the table that the observations made in Figure 5.3 hold for all the tested data sets: a $\omega < 0.65659$ coupled with a $c_2 < 1.06344$ lead to poor FFNN accuracy on all tested data sets.

Table 5.5: Extents of parameter regions associated with the bottom 20% generalisation error results per data set.

Problem	ω	c_1	c_2
Diabetes	$[0.01661, 0.65659]$	$[0.15809, 1.96878]$	$[0.06444, 0.93856]$
Glass	$[0.01661, 0.56294]$	$[0.15809, 1.96878]$	$[0.09566, 0.93856]$
Iris	$[0.01661, 0.56294]$	$[0.28297, 1.68781]$	$[0.12687, 1.06344]$
WDBC	$[0.01661, 0.65659]$	$[0.15809, 1.96878]$	$[0.06444, 0.93856]$
Wine	$[0.01661, 0.65659]$	$[0.28297, 1.68781]$	$[0.06444, 0.93856]$
Henon Map	$[0.01661, 0.65659]$	$[0.15809, 1.68781]$	$[0.06444, 0.93856]$
Logistic Map	$[0.01661, 0.56294]$	$[0.15809, 1.68781]$	$[0.12687, 1.06344]$
Mackey-Glass	$[0.01661, 0.56294]$	$[0.15809, 1.96878]$	$[0.09566, 0.93856]$
Sunspots	$[0.01661, 0.65659]$	$[0.15809, 1.96878]$	$[0.06444, 0.93856]$
TS5	$[0.01661, 0.65659]$	$[0.15809, 1.96878]$	$[0.06444, 0.93856]$

5.2.4 Discussion

From the results presented here, a number of observations can be made. It is clear that c_1 is a non-dominant parameter in terms of trained FFNN accuracy. Considering either the best performing of worst performing configuration results, no specific region or interaction with other parameters could be identified for c_1 that lead to improved generalisation performance. Optimal c_1 values were shown to be specific to each data set.

In terms of the ω and c_2 parameters, neither of the parameters were dominant on their own, instead it was found that one specific interaction of the two parameters adversely affected performance. That is, for parameter configurations where $\omega < 0.5$ and $c_2 < 1.0$, the PSO was unable to effectively optimise the FFNN weights, leading to large training and generalisation errors. If either parameter fell outside these bounds, that is to say $\omega \geq 0.5$ or $c_2 \geq 1.0$, the PSO was still capable of training the FFNN to an acceptable error. It was also shown that the absolute lowest generalisation errors (regardless of overfitting) for each data set were obtained with a ω in the range $(0.59, 0.74)$. However, similar accuracy with improved overfitting results were obtained with ω values outside this range in specific cases. The results also showed that a slightly higher ω value lead to improved accuracy on the regression data sets tested, suggesting that these data sets benefited from increased exploration.

A $\rho_F > 1$ was reported for nearly all parameter configurations on all data sets tested. While this is indicative of overfitting, it was shown that in the case of parameter configurations that lead to extremely poor generalisation, high ρ_F values were obtained, even though the FFNNs in these cases instead underfitted the data (as evident from a high E_T). While the generalisation errors were still much higher than the training errors, these cases should not be considered as overfitting.

However, in most instances, the ρ_F measurement was a valid indication of overfitting and no specific parameter regions or interactions were identified across all data sets that lead to a reduction in ρ_F while maintaining good generalisation accuracy. Specific parameter configurations that obtained good generalisation and overfitting results were unique to each data set tested. This illustrates that it is both possible and necessary to optimise the ω, c_1 and c_2 parameters on a per data set basis to reduce overfitting.

5.3 Summary

This chapter provided an analysis of the effect of the inertia and acceleration coefficient control parameters on the accuracy and overfitting behaviour of PSO trained FFNNs.

The experimental methodology and a description of the analytical use of low-discrepancy sequences and high dimensional visualisation techniques were given in Section 5.1.

Section 5.2 gave the results of the parameter analysis. Optimal and adverse parameter regions for each data set as well as specific interactions of parameters leading to adverse performance were identified, concluded by a discussion in Section 5.2.4.

The next chapter analyses the effect on FFNN training of another PSO control parameter, the particle swarm size.

Chapter 6

PSO Swarm Size

Chapters 4 and 5 investigated the effect of major PSO control parameters on accuracy and overfitting when training FFNNs, specifically, the effect of *maximum velocity*, *inertia* and the *acceleration coefficient* control parameters were analysed. This chapter seeks to similarly determine the effects of the last remaining major control parameter: the swarm size.

The chapter comprises of the following sections. The methodology used to investigate the swarm size's effect is given in Section 6.1, with the experimental configuration and procedure given in Sections 6.1.3 and 6.1.4 respectively. Section 6.2 gives the results and a discussion of the experiments performed. The chapter is summarised in Section 6.3.

6.1 Experimental Methodology

This section discusses the methodology used to investigate the effect of the *swarm size* (n_s) on FFNN accuracy.

6.1.1 Swarm Size Selection

Formally, the objective of this chapter is to determine if a significant difference in FFNN accuracy exists, both on a training set as measured by the training error and a generalisation set via the generalisation error, when trained using particle swarm optimises with different swarm sizes.

Swarm size, as a PSO control parameter, was discussed in Section 2.3.2. As mentioned in Section 2.3.2 the work of Malan and Engelbrecht [70] showed that the effect of the parameter *is dependent on the optimisation problem*, with larger swarms often having an advantage in highly multi-modal and high-dimensional search spaces. Many NN training problems are high-dimensional, and as such it is possible that an increase in swarm size might benefit training. Furthermore, the paper notes the inversely proportional relationship that exists between the number of iterations the PSO is executed for and the swarm size [70]. That is, as the swarm size increases, the iterations the algorithm is executed for has to decrease in order for the optimises to maintain the same amount of computational complexity as measured by the number of *objective function evaluations*. With larger swarm sizes there is therefore a risk that the particles are not given a sufficient amount of time (iterations) to adequately explore the search space and converge to an optimum.

For the experiments in this chapter, similar to those of Malan and Engelbrecht [70], computational complexity is kept similar between each of the PSOs. As such, PSOs with larger swarm sizes were limited by the number of iterations, and therefore the number of objective function evaluations were kept constant. Table 6.1 lists the swarm sizes used in the experiments to follow.

6.1.2 Data Sets

As with the previous chapters the five classification and five regression data sets as described in Section 3.2 were used for the experimental work in this chapter. The FFNN architectures are given in Table 3.1.

6.1.3 PSO Configuration

Due to the findings in Chapter 3, and, for the sake of comparison with previous chapters, the PSO algorithm used in this experiment was an *lbest* PSO with a neighbourhood size of five.

Particles were randomly initialised in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, with velocities initialised to $\mathbf{0}$ as described in Section 3.3.2.

Table 6.1: Swarm sizes and corresponding iteration counts used to compare PSOs. Note that, where the goal number of evaluations was not exactly divisible by the swarm size, that is $n_s = 75, n_s = 150$ and $n_s = 300$, the iterations were rounded up to the nearest integer which lead to a small additional number of evaluations in those cases.

Swarm Size (n_s)	**Iterations**	**Objective Evaluations**
10	5000	50000
25	2000	50000
50	1000	50000
75	667	50025
100	500	50000
150	334	50100
200	250	50000
300	167	50100
500	100	50000

A $\mathbf{V}_{max} = \mathbf{1}$ was used for all data sets. As was shown in Chapter 4, a $\mathbf{V}_{max} = \mathbf{1}$ is appropriate for maintaining exploration and preventing drastic swarm divergence while training FFNNs. Chapter 4's experimental work was, however, carried out under the condition of a swarm size of 25 and the investigation in this chapter will also serve to determine if the choice of $\mathbf{V}_{max}$ is appropriate for other swarm sizes.

As shown in Table 6.1, a fixed number of iterations was used as stopping condition for the algorithm while keeping the objective function evaluations constant at approximately 50000.

The results of the experimental work in Chapter 5 showed that specific values of the ω, c_1 and c_2 control parameters improved FFNN accuracy on a per data set basis. In order to eliminate these parameters as a factor in this experiment, the optimal values found in Chapter 5 for each of the data sets were used. These values are summarised in Table 6.2.

Table 6.2: Optimal values for the ω, c_1 and c_2 parameters per data set.

Data set	ω	c_1	c_2
Diabetes	0.70342	0.22053	0.34541
Glass	0.70342	0.22053	0.34541
Iris	0.85952	1.90634	0.78247
WDBC	0.25075	1.50050	1.50050
Wine	0.59416	1.93756	1.93756
Henon Map	0.71903	0.68881	1.18831
Logistic Map	0.73464	1.15709	1.53172
Mackey-Glass	0.62538	0.25175	1.75025
Sunspot	0.46928	0.18931	1.68781
TS5	0.67220	0.78247	0.65759

6.1.4 Experimental Procedure

For this experiment, the following null and alternative hypotheses were used:

- H_0: There is no significant difference in performance between any of the swarm sizes.
- H_1: There is a one-tailed significant difference, lower values in the cases of E_T and E_G and higher values in the cases of ρ_F and $\mathcal{D}_{avg}$, between any of the swarm sizes.

A significance level of $\alpha = 0.05$ was used for both the Friedman and Wilcoxon rank sum tests that were performed.

The experimental analysis proceeded as follows. For each of the swarm sizes listed in Table 6.1 the PSO was executed on all 10 data sets. Each execution on a data set for a swarm size was repeated 30 times with a unique seed for a Mersenne Twister PRNG. The results of the 30 executions were aggregated, and the aggregated results are reported. The results are shown in a matrix of 10 blocks (one row per data set) over the 10 swarm sizes. Four such result matrices were produced, one for each of the algorithm measurements discussed in Section 6.1.5.

A Friedman test was then performed on each of the result matrices. If the result of the Friedman test proved significant, one-tailed, pairwise Wilcoxon rank sum tests were conducted with the p-values adjusted using the Holm-Bonferroni method to determine whether a significant difference exists between each of the swarm sizes.

6.1.5 Algorithm Measurements

The measurements described in Section 3.3.3 were used to quantify the performance of the algorithm execution. That is, the training and generalisation errors, E_T and E_G, along with the generalisation factor, ρ_F. The PSO swarm diversity $\mathcal{D}_{avg}$ was also measured.

6.2 Results

The results and a discussion of the experimental analysis are given in this section. Unless otherwise stated, values indicated in tables and figures show the mean over the 30 samples.

6.2.1 Training Accuracy

Table 6.3 shows the training error results for all swarm sizes and data sets. The Friedman test produced a p-value of 7.327e-9 indicating a significant difference between at least two swarm sizes over all of the data sets. The results of the Wilcoxon tests, shown in Table 6.4, revealed that, over all of the data sets, the training error results for swarm sizes $n_s \in \{10, 25, 50, 75\}$ did not differ significantly, although *higher* training errors were obtained for most data sets with $n_s = 10$. A swarm size of either $n_s = 25$ or $n_s = 50$ lead to the lowest training errors, depending on the data set. For $n_s > 75$ the PSO obtained significantly worse training errors when compared to smaller swarm sizes, which continued to degrade up until $n_s = 500$.

From the results given above it is clear that the PSO's swarm size has a significant impact on the *training error* when training FFNNs. The E_T results show that there exists a critical value for the swarm size, approximately in the range of $(25, 50)$ for the

Table 6.3: E_T results for classification and regression data sets.

	Swarm Size								
Data set	**10**	**25**	**50**	**75**	**100**	**150**	**200**	**300**	**500**
Diabetes	0.20314	0.14335	0.13799	0.13995	0.14208	0.14278	0.14441	0.14607	0.14998
Glass	1.70402	0.55411	0.51303	0.51256	0.52097	0.54499	0.55917	0.58564	0.62495
Iris	0.01639	0.01606	0.01855	0.01861	0.01933	0.02066	0.02049	0.02194	0.02348
WDBC	0.10606	0.01674	0.01549	0.01656	0.01691	0.01939	0.02038	0.02252	0.02529
Wine	0.00081	0.00108	0.00208	0.00257	0.00291	0.00459	0.00609	0.01021	0.01674
Henon Map	0.00051	0.00067	0.00092	0.00116	0.00166	0.00195	0.00245	0.00356	0.00527
Logistic Map	0.00063	0.00075	0.00088	0.00100	0.00106	0.00119	0.00126	0.00140	0.00161
Mackey-Glass	0.00055	0.00060	0.00073	0.00085	0.00091	0.00103	0.00112	0.00125	0.00145
Sunspot	163.731	142.298	147.793	160.082	170.101	178.861	185.302	196.209	208.297
TS5	0.00499	0.00233	0.00235	0.00238	0.00246	0.00252	0.00262	0.00277	0.00292
Friedman p-value: 7.327e-9									

data sets tested here, for which the best training accuracy was obtained by the networks. Values *smaller or larger* than the critical range were shown to lead to significantly worse training accuracy up to and including $n_s = 500$.

Upon further examination of the E_T results it was found that the reason for the significant degradation in training accuracy is the trade-off that exists between an increased n_s and the number of iterations the PSO may execute for due to a bounded number of objective function evaluations. This is better illustrated in Figure 6.1 which shows the training error results for $n_s > 25$ for the Glass data set. Similar results were observed for the other data sets. The figure shows that as the swarm size increases, the training error initially decreases at a steeper rate when compared to smaller swarm sizes. The improvement in the optimisation speed of E_T is most likely due to the increased exploration capability of the larger swarm. However, in keeping the fitness evaluations constant, the PSO finally fails to converge to a solution with a lower *final* training error as optimisation is prematurely stopped. It is possible that larger swarm sizes might outperform smaller swarms if given the same number of iterations at the cost of significantly increased computational time. Such a study is left as future research.

Table 6.4: Holm adjusted p-values for one-tailed Wilcoxon rank sum tests on E_T results.

Swarm Size	10	25	50	75	100	150	200	300
25	1	-	-	-	-	-	-	-
50	1	1	-	-	-	-	-	-
75	1	1	0.209	-	-	-	-	-
100	1	1	0.035	0.035	-	-	-	-
150	1	0.961	0.035	0.035	0.035	-	-	-
200	1	0.035	0.035	0.035	0.035	0.096	-	-
300	1	0.035	0.035	0.035	0.044	0.035	0.035	-
500	1	0.035	0.035	0.035	0.035	0.035	0.035	0.035

6.2.2 Generalisation Accuracy

The generalisation error results are given in Table 6.5. Although the generalisation performance is remarkably similar across all swarm sizes for each of the data sets, the Friedman test indicated a significant difference exists between some of the swarm sizes. The Wilcoxon test results shown in Table 6.6 shows that the only significant difference was for $n_s = 500$ compared against $n_s = 200$, with $n_s = 500$ performing worse in terms of generalisation. In the case of all other swarm sizes, no significant difference in generalisation error was found.

The generalisation factor results are given in Table 6.7. The table shows that the generalisation factors decrease significantly as swarm size increases beyond $n_s = 25$ for most of the data sets. For an $n_s = 10$ the ρ values are shown to be comparatively low. The generalisation factor differed significantly between at least two of the swarm sizes as illustrated by the p-value of the Friedman test. The subsequent Wilcoxon comparisons, summarised in Table 6.8, show that for an $n_s > 50$ significantly *lower* generalisation factors were obtained by the PSOs with larger swarm sizes.

Figure 6.2 shows the generalisation error for the Glass data set for $n_s > 25$ plotted against the number of iterations. The results for other data sets were similar. Unlike Figure 6.1 showing the training error, the generalisation error behaviour does not differ

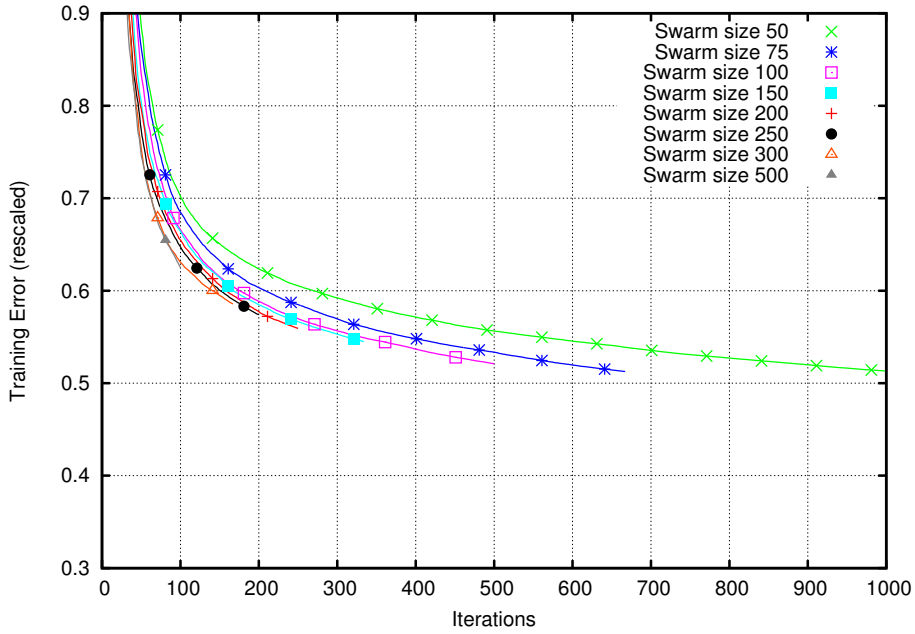


Figure 6.1: E_T results over 1000 iterations for $n_s > 25$ on the Glass data set.

Table 6.5: E_G results for classification and regression data sets.

	Swarm Size								
Data set	**10**	**25**	**50**	**75**	**100**	**150**	**200**	**300**	**500**
Diabetes	0.20260	0.15875	0.16179	0.16010	0.16103	0.15963	0.16111	0.15965	0.16206
Glass	1.82938	0.85734	0.90883	0.92014	0.90707	0.87381	0.89378	0.91873	0.90350
Iris	0.03064	0.02870	0.03005	0.03056	0.02891	0.03215	0.03059	0.03116	0.03201
WDBC	0.10696	0.02777	0.02777	0.02806	0.02744	0.02841	0.02859	0.02918	0.03056
Wine	0.03976	0.03299	0.03775	0.03459	0.03394	0.04218	0.03525	0.03715	0.04396
Henon Map	0.00555	0.00603	0.00835	0.00867	0.01310	0.01015	0.01524	0.01284	0.02262
Logistic Map	0.00137	0.00133	0.00141	0.00143	0.00147	0.00158	0.00166	0.00173	0.00185
Mackey-Glass	0.00130	0.00131	0.00131	0.00137	0.00152	0.00149	0.00154	0.00159	0.00172
Sunspot	235.575	225.509	226.598	228.88	234.247	238.515	236.967	245.344	256.717
TS5	0.00653	0.00453	0.00448	0.00442	0.00429	0.00434	0.00443	0.00434	0.00446
Friedman p-value: 0.00027									

Table 6.6: Holm adjusted p-values for one-tailed Wilcoxon rank sum tests on E_G results.

Swarm Size	10	25	50	75	100	150	200	300
25	1	-	-	-	-	-	-	-
50	1	0.303	-	-	-	-	-	-
75	1	0.146	1	-	-	-	-	-
100	1	0.806	1	1	-	-	-	-
150	1	0.094	1	1	1	-	-	-
200	1	0.068	1	0.806	0.806	1	-	-
300	1	0.068	1	1	0.659	1	1	-
500	1	0.068	0.659	0.806	0.52	0.094	0.035	0.806

greatly between the various swarm sizes; for all swarm sizes, the generalisation error stabilised after approximately 200 to 400 iterations. As stated above, no significant difference in generalisation was finally obtained by any of the swarm sizes.

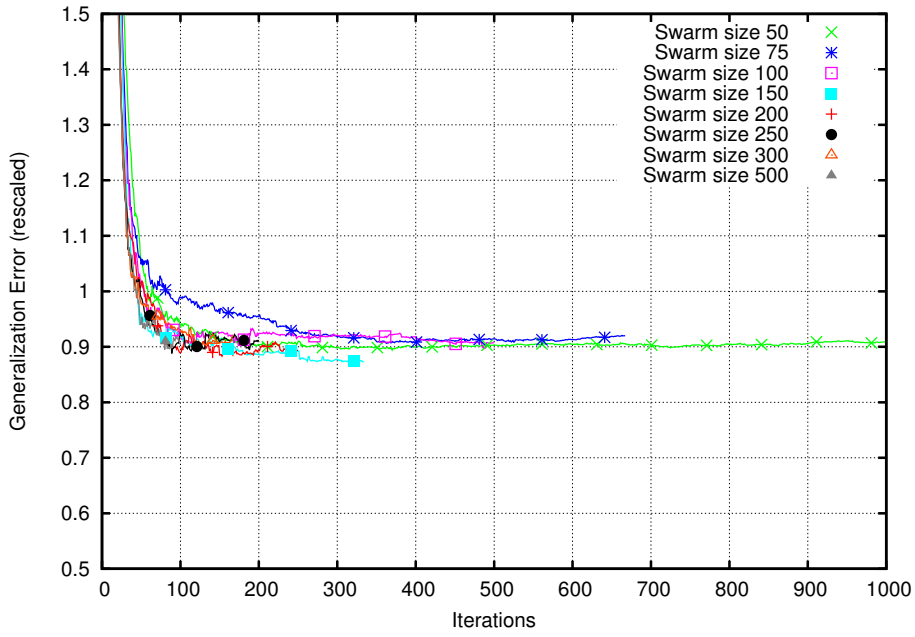


Figure 6.2: E_G results over 1000 iterations for $n_s > 25$ on the Glass data set.

Table 6.7: ρ_F results for each swarm size and data set.

	Swarm Size								
Data set	**10**	**25**	**50**	**75**	**100**	**150**	**200**	**300**	**500**
Diabetes	0.99792	1.11162	1.17737	1.14870	1.13693	1.12096	1.11927	1.09632	1.08350
Glass	1.08540	1.62497	1.91648	1.92037	1.84991	1.69007	1.69627	1.66246	1.53798
Iris	2.67587	2.24302	1.97799	1.89886	1.68726	1.76372	1.71207	1.57774	1.50169
WDBC	1.01025	1.75793	1.97068	1.80239	1.69814	1.53647	1.46284	1.34477	1.25176
Wine	105.734	56.5611	28.8316	17.7568	15.9741	10.9412	7.36232	4.29181	2.89652
Henon Map	13.5181	11.6379	10.2124	9.7817	7.26610	5.86085	8.21218	3.73871	5.03158
Logistic Map	2.44848	1.94669	1.77663	1.58415	1.52224	1.46944	1.44815	1.34001	1.23847
Mackey-Glass	2.53481	2.36546	1.90867	1.76864	1.82661	1.59759	1.51535	1.38039	1.29063
Sunspot	1.48232	1.63221	1.56991	1.46373	1.40504	1.36369	1.30899	1.28263	1.25487
TS5	1.38843	1.97498	1.91110	1.89127	1.75970	1.73867	1.71471	1.58349	1.53644
Friedman p-value: 1.111e-7									

The results for the generalisation factor can intuitively be explained from the combination of the E_T and E_G results. That is, as swarm size increases, the generalisation error remains unchanged but the training error worsens, by definition, this leads to an increase in ρ_F.

Due to the stability of the generalisation error, the increase in ρ_F proportionately to n_s is *therefore not an indication of increased overfitting*. This is similar to the earlier result of the effect of $\mathbf{V}_{max}$ in Section 4.2, where it was also found that the ρ_F measure increased, but not as a result of overfitting. These results further show that the use of ρ_F in isolation is a poor indicator of any overfitting during training, and indeed other errors and measurements should be used in conjunction with ρ_F.

6.2.3 Diversity

The final result set is given in Table 6.9, which shows the diversity results across all tested swarm sizes. The results of the significance tests in Table 6.10 show that significant increases in diversity took place as the swarm size increased up to $n_s = 50$, after which insignificant increases in diversity are still seen. In general, larger diversities were

Table 6.8: Holm adjusted p-values for one-tailed Wilcoxon rank sum tests on ρ_F results.

Swarm Size	**10**	**25**	**50**	**75**	**100**	**150**	**200**	**300**
25	0.589	-	-	-	-	-	-	-
50	0.589	0.589	-	-	-	-	-	-
75	0.589	0.317	0.039	-	-	-	-	-
100	0.589	0.146	0.035	0.056	-	-	-	-
150	0.589	0.083	0.035	0.035	0.146	-	-	-
200	0.589	0.083	0.035	0.035	0.58	0.58	-	-
300	0.589	0.056	0.035	0.035	0.035	0.035	0.035	-
500	0.589	0.035	0.035	0.035	0.035	0.035	0.035	0.387

obtained for the classification problems as opposed to the regression problems, most likely due to the larger dimensionality of the classification problems.

Table 6.9: $\mathcal{D}_{avg}$ results for each swarm size and data set.

	Swarm Size								
Data set	**10**	**25**	**50**	**75**	**100**	**150**	**200**	**300**	**500**
Diabetes	0.0	0.2414	2.6767	7.6161	10.437	11.976	12.191	12.256	12.078
Glass	0.0	0.0977	3.0424	7.1546	8.9806	10.318	9.6183	9.8767	9.6471
Iris	3.5309	8.5723	9.1306	9.8292	9.4852	9.4254	9.4436	9.1899	9.0111
WDBC	0.0	0.4434	7.3585	12.107	12.69	12.88	12.948	12.8592	12.42
Wine	0.99	11.291	13.423	13.8	13.987	13.501	13.329	12.8557	12.284
Henon Map	0.0463	3.2054	11.785	12.587	12.559	12.184	11.845	11.733	11.514
Logistic Map	0.2745	7.4001	8.7305	8.5207	8.2762	7.9432	7.8647	7.7442	7.7091
Mackey-Glass	0.0993	1.2201	11.03	10.974	11.052	10.957	10.776	10.641	10.527
Sunspot	0.0434	0.2434	1.9076	8.0335	9.0853	9.3730	9.1294	8.5388	8.0591
TS5	0.0	1.3488	4.5841	4.9437	5.0136	4.8548	4.8107	4.4047	4.1525
Friedman p-value: 8.759e-9									

Finally, Figure 6.3 illustrates the $\mathcal{D}_{avg}$ results for each of the swarm sizes for the Glass

Table 6.10: Holm adjusted p-values for one-tailed Wilcoxon rank sum test on $\mathcal{D}_{avg}$ results.

Swarm Size	10	25	50	75	100	150	200	300
25	0.035	-	-	-	-	-	-	-
50	0.035	0.035	-	-	-	-	-	-
75	0.035	0.035	0.103	-	-	-	-	-
100	0.035	0.035	0.137	1	-	-	-	-
150	0.035	0.035	0.353	1	1	-	-	-
200	0.035	0.035	1	1	1	1	-	-
300	0.035	0.035	1	1	1	1	1	-
500	0.035	0.035	1	1	1	1	1	1

data set. $\mathcal{D}_{avg}$ remained largely unchanged after approximately 500 iterations for the data set and as a result, and in order to improve readability, only the first 500 iterations are shown. Results for the other data sets were very similar.

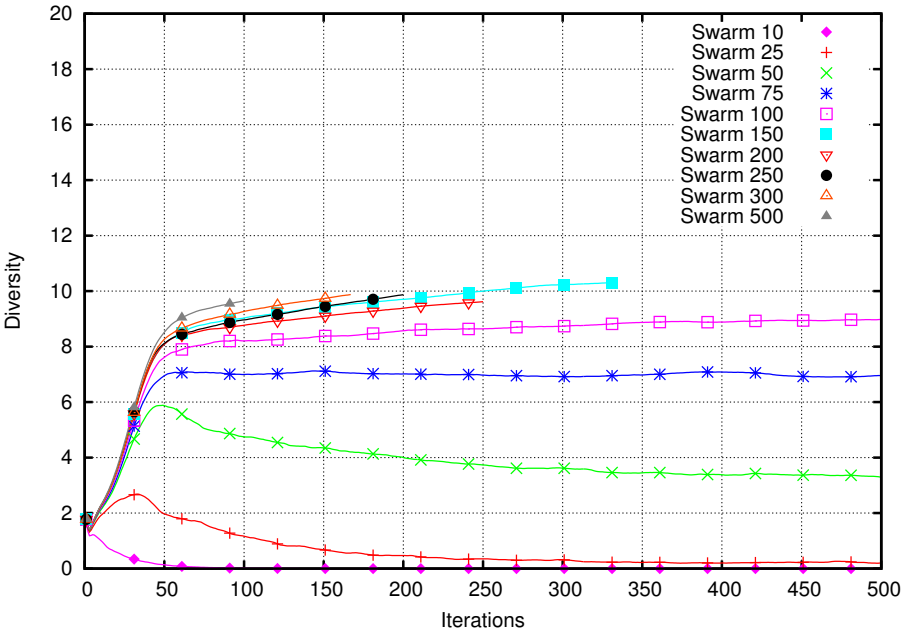


Figure 6.3: $\mathcal{D}_{avg}$ results over 500 iterations on the Glass data set.

For three of the 10 tested swarm sizes, that is, $n_s \in \{10, 25, 50\}$, a convergent diversity profile is shown. That is, diversity decreases over time and tends to 0 as the swarm converges on a solution. However, for $n_s > 50$ the diversity is shown increase over time. For $n_s = 75$, the diversity does again start to decrease after a number of iterations, oscillating between convergence and divergence as optimisation continues. However, for swarm sizes larger than 75, the swarm is shown to diverge completely as optimisation continues.

This divergent behaviour was previously seen in Chapters 3 and 4, with Chapter 4 specifically investigating whether $\mathbf{V}_{max}$ could prohibit this behaviour. Previously it was shown that the lack of a $\mathbf{V}_{max}$ lead to divergent swarms and that $\mathbf{V}_{max} \in (0.0, 1.0]$ was required for the swarms to converge. Larger $\mathbf{V}_{max}$ parameters were ineffective in preventing divergence. These results assumed a swarm size of 25 and the results of this chapter show that a $\mathbf{V}_{max} \in (0.0, 1.0]$ only produced non-divergent swarms for $n_s < 75$. This indicates that the appropriate value of $\mathbf{V}_{max}$ is relative to the swarm size. Further analysis of the relationship between $\mathbf{V}_{max}$ and the swarm size is left as a future work.

As mentioned in Chapter 4 the lack of a $\mathbf{V}_{max}$ parameter, or indeed a $\mathbf{V}_{max}$ that is too large, is not the cause of the divergent behaviour and the use of a small $\mathbf{V}_{max}$ merely addresses the symptom of the cause.

Since all other PSO parameters have been investigated in previous chapters and were optimised on the data sets used for the experimental work in this chapter, it is clear that some factor independent of the PSO itself and specific to the training of FFNNs must be the reason for the divergence. In Chapter 4 it was speculated that the specific activation function used for the FFNNs, which influences the optimisation landscape, could affect much of the PSO's behaviour. The next chapter investigates the influence of the activation function, with specific focus on the swarm's convergence.

6.3 Summary

This chapter investigated the effect of the swarm size algorithm parameter on the accuracy of FFNNs when trained using a PSO. The experimental methodology was given in Section 6.1.

The results, given in Section 6.2, showed that the swarm size had a significant impact on the training error and generalisation factor. Under the condition of an equal number of objective function evaluations, obtained training accuracy deteriorated as the swarm size increased whilst the generalisation accuracy was unaffected. Furthermore, it was shown that swarm divergence occurred and worsened as the swarm size increased; the use of a $\mathbf{V}_{max}$ parameter proved ineffective in aiding swarm convergence for larger swarm sizes.

The next chapter investigates the hypothesis that the cause of the divergence is related to the activation function used by the FFNN neurons.

Chapter 7

Considering Activation Functions

This chapter investigates the behaviour of the PSO algorithm when training FFNNs using a number of different activation functions. Due to questions raised in previous chapters surrounding swarm divergence during optimisation, specific focus is placed on the swarm's diversity and convergence. The effect on the FFNN's accuracy is also analysed.

The activation functions used are described in Section 7.1. The methodology used to compare the activation functions' effect is given in Section 7.2. The results are given in Section 7.3 followed by a summary of the chapter in Section 7.4.

7.1 Activation Functions

The activation function and its role in the neurons of the FFNN was discussed in Section 2.1.1. This section briefly discusses the three activation functions used in the experimental work of this chapter: The *sigmoid function*, the *hyperbolic tangent function* and the *linear function*.

7.1.1 Sigmoid

The sigmoid activation function was first introduced in Section 2.1.1 and has been used as activation function for the experimental work of previous chapters. The sigmoid

function is defined as:

$$f_{sig}(net, \lambda, \gamma) = \frac{\gamma}{1 + e^{-\lambda(net)}} \tag{7.1}$$

where λ controls the steepness of the function and γ the range. For most FFNN applications $\lambda = \gamma = 1$.

As discussed in Section 2.1.1, the sigmoid function is a non-linear, differentiable and monotonically increasing function. The sigmoid function is an asymptotically *bounded* function, as illustrated in Figure 7.1, yielding an output in $(0, 1)$. During training, it is therefore necessary to scale the data target values to the range $(0, 1)$.

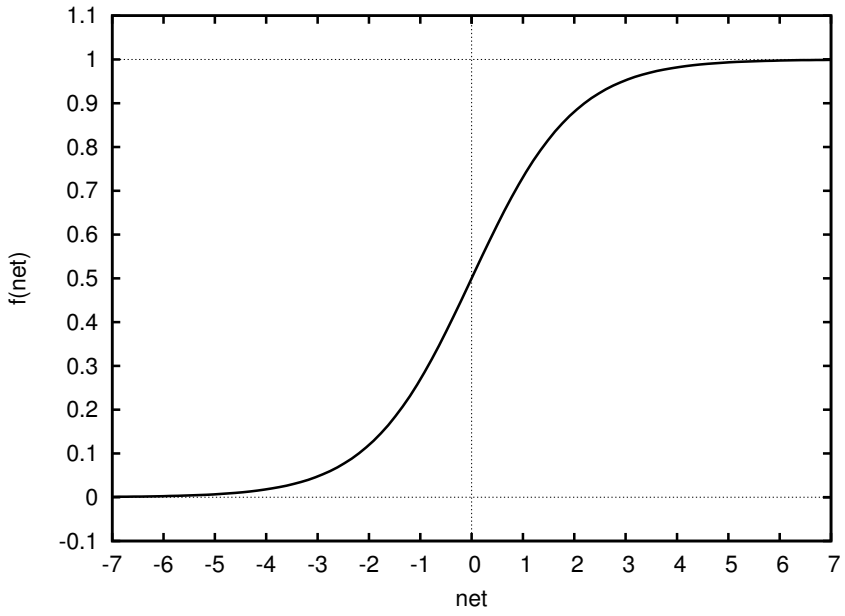


Figure 7.1: Sigmoid activation function with $\lambda = \gamma = 1$

A notable consequence of the asymptotically bounded nature of the function is that, as the absolute magnitude of the input, *net*, to the function increases, the rate of change in function output differs insignificantly. That is:

$$\lim_{net \to \pm\infty} f'(net) = 0 \tag{7.2}$$

where $f'(net)$ is the derivative of the function.

The *net* input of the function is defined as the sum of the weighted (in the case of PSO the particle positions represent the weights) input signals to the neuron. Due to

the limit defined in Equation (7.2) the data set input values are scaled to the range $[-\sqrt{3}, \sqrt{3}]$: its active domain. This range has been shown to correspond to the part of the function where the input has a large effect on the output of the function [28]. Similarly, the particle positions are initialised to small values, in a range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, to avoid *net* input signals that produce output near the bounds of the function. This methodology was also followed for all experimental work of the previous chapters.

7.1.2 Hyperbolic Tangent

The second activation function used is the hyperbolic tangent function:

$$f_{\tanh}(net) = \frac{e^{\lambda(net)} - e^{-\lambda(net)}}{e^{\lambda(net)} + e^{-\lambda(net)}} \tag{7.3}$$

where λ controls the steepness of the function.

As shown in Figure 7.2 the hyperbolic tangent function is also an asymptotically bounded activation function with asymptotes at -1 and 1 and a similar active domain to the sigmoid function. Target output values therefore also have to be scaled, but to a larger numerical range, which has the advantage of decreasing training time [28].

Equation (7.2) also holds true for $f_{\tanh}$ and as such training is benefited by scaling input values to the range $[-\sqrt{3}, \sqrt{3}]$.

7.1.3 Linear

The final activation function is the linear function:

$$f(net) = \lambda net. \tag{7.4}$$

where λ controls the gradient.

Unlike the sigmoid and hyperbolic tangent functions, the linear activation function is *unbounded*, avoiding the need for scaling of either input or output data values. However, due to the function being linear, a FFNN using linear activation functions in the hidden layer is a less capable approximator of a non-linear target function. Linear FFNNs will therefore require more hidden units than a FFNN using sigmoid or hyperbolic tangent functions to achieve the same accuracy on most problem sets.

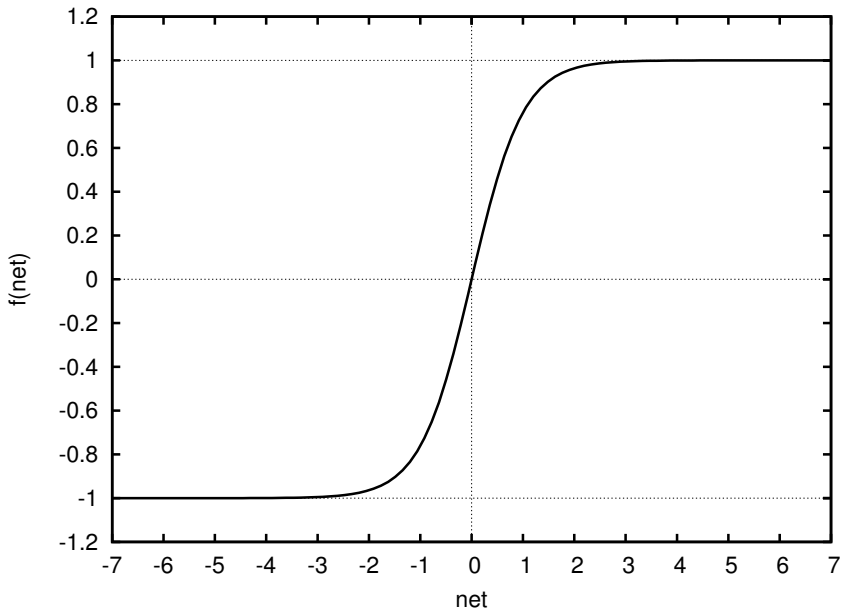


Figure 7.2: Hyperbolic Tangent

7.2 Experimental Methodology

This section discusses the methodology used to investigate the behaviour of particle swarms when training FFNNs using each of the activation functions discussed above.

7.2.1 Data Sets

In keeping with the work of previous chapters, the classification and regression data sets described in Section 3.2 were used for the experimental work in this chapter. Similarly, the FFNN architectures given in Table 3.1 were used for all activation functions.

7.2.2 PSO Configuration

The PSO algorithm used for the work in this chapter was an *lbest* PSO with a neighbourhood size of five and a swarm size of 25. As shown in Chapter 6, a swarm size of 25 lead to greater exploration compared with smaller swarm sizes while still converging over time.

As detailed in Section 3.3.2, the particle positions were initialised in the range: $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$, where $fanin$ is the number of incoming connections of the corresponding neuron. Particle velocities were initialised to the zero vector.

In terms of the $\mathbf{V}_{max}$ control parameter, the results of Chapter 6 have shown that in order to aid in preventing swarm divergence, the $\mathbf{V}_{max}$ parameter has to be adjusted according to the swarm size. Previous results have also shown that $\mathbf{V}_{max}$ is only instrumental in delaying eventual swarm divergence and, if given enough iterations, the swarm may still diverge. Since part of the objective of this chapter is to investigate the cause of the divergence, no $\mathbf{V}_{max}$ parameter was used in the experimental work of this chapter. Severe swarm divergence was therefore expected to occur.

The per data set optimised values for the ω, c_1 and c_2 control parameters, as summarised in Table 6.2, were used.

A fixed number of 50000 objective function evaluations was used as stopping condition for the algorithm, regardless of the activation function used by the FFNN.

7.2.3 Algorithm Measurements

The following measures were used to quantify the performance of the PSO as a training algorithm for the FFNNs with different activation functions. The training and generalisation errors, E_T and E_G and the generalisation factor ρ_F. The swarm diversity $\mathcal{D}_{avg}$ was also calculated. These measurements were discussed in Section 3.3.3.

7.2.4 Experimental Procedure

The goal of the experimental work of this chapter is two fold: Investigate if there is a significant change in swarm convergence when different activation functions are used for the FFNN. Secondly, is there a significant change in accuracy, generalisation or overfitting when different activation functions are used. In the case of both objectives, hypothesis testing was performed on the various algorithm measurements using the following hypothesis:

- H_0: There is no significant difference in performance between the FFNNs using different activation functions.

- H_1: There is a one-tailed significant difference, lower values in the cases of E_T and E_G and higher values in the cases of ρ_F and $\mathcal{D}_{avg}$, between FFNNs using a different activation function.

With specific reference to the diversity measurement, a significant difference *does not necessarily indicate decreased or increased divergence*, merely a more contracted or expanded swarm and as such, any tests using $\mathcal{D}_{avg}$ is accompanied by further analysis of the diversity over time.

The experiment proceeded as follows. For each of the activation functions, the FFNN was configured such that the neurons in both the hidden and output layers of the network used the function.

In the case of the sigmoid function, as with previous chapters, the target values were scaled to $[0.1, 0.9]$. For the hyperbolic tangent function networks the target values were scaled to $[-0.9, 0.9]$. These ranges were chosen to be well within the bounds of the functions. Although not strictly necessary, the target values for the linear function networks were also scaled to $[-0.9, 0.9]$.

The PSO algorithm was then used to to train the FFNNs on each of the 10 data set. Each execution on a data set was repeated 30 times using a unique seed, per sample, for a Mersenne Twister PRNG. The measurements discussed above were taken for each execution of the algorithm and the aggregated results for all 30 samples are reported.

A Friedman test at a significance level of $\alpha = 0.05$ was performed per algorithm measurement to determine if a significant difference exists between any of the activation functions for that measurement across all of the data sets. If successful, pairwise one-tailed Wilcoxon rank sum tests were performed, also at a significance level of $\alpha = 0.05$ with p-values adjusted using the Holm-Bonferroni method.

7.3 Results

The results and a discussion of the experimental work is given in this section. The results are divided into three subsections: the training accuracy, generalisation and overfitting, and finally the swarm diversity and convergence behaviour.

7.3.1 Training Accuracy

The means and standard deviations of the training errors for each of the activation functions for all 10 data sets are given in Table 7.1. The result of the Friedman test yielded a significant result of 0.0005. The subsequent Wilcoxon rank sum test are shown in Table 7.2. The Wilcoxon tests indicate a significant difference between the linear FFNN training errors compared with the other activation functions. This is an expected result since, as discussed in Section 7.1, the linear function is less powerful than its non-linear counterparts, and it would require many more hidden units to approximate the target function as effectively. Since the same FFNN architectures were used for all activation functions, it was expected that the linear FFNNs would perform worse.

Table 7.1: Training error mean and standard deviations for each of the activation functions.

	Linear		**Sigmoid**		**Tanh**	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	0.15735	0.00469	0.14315	0.00566	0.14254	0.00571
Glass	0.76000	0.08914	0.54763	0.07882	0.53611	0.10636
Iris	0.04727	0.00443	0.02917	0.00760	0.03065	0.00658
WDBC	0.05720	0.00361	0.01679	0.00371	0.01306	0.00540
Wine	0.05759	0.00550	0.00480	0.00268	0.00481	0.00237
Henon Map	0.34961	0.03722	0.00059	0.00030	0.00143	0.00064
Logistics Map	0.00130	0.00032	0.00078	0.00016	0.00085	0.00018
Mackey-Glass	0.00130	0.00032	0.00062	0.00015	0.00067	0.00011
Sunspot	268.882	18.2997	136.973	14.3489	140.304	12.0938
TS5	0.00256	0.00012	0.00232	0.00043	0.00225	0.00020
Friedman p-value: 0.0005						

The Wilcoxon test comparing the sigmoid function results with that of the hyperbolic tangent function yielded a p-value of 0.5771, indicating no significant difference in training performance.

Examining the training errors over time on a per sample bases revealed a significant difference in the optimisation behaviour between the different FFNNs. Graphs showing the log scaled training error per iteration for each of the activation function FFNNs on

Table 7.2: Wilcoxon rank sum test p-values for training error results across all data sets.

Activation Function	Linear	Sigmoid
Sigmoid	0.0029	-
Hyperbolic Tan	0.0029	0.5771

the Henon Map data set are given in Figures 7.3, 7.4 and 7.5 respectively. Results were similar for the other data sets.

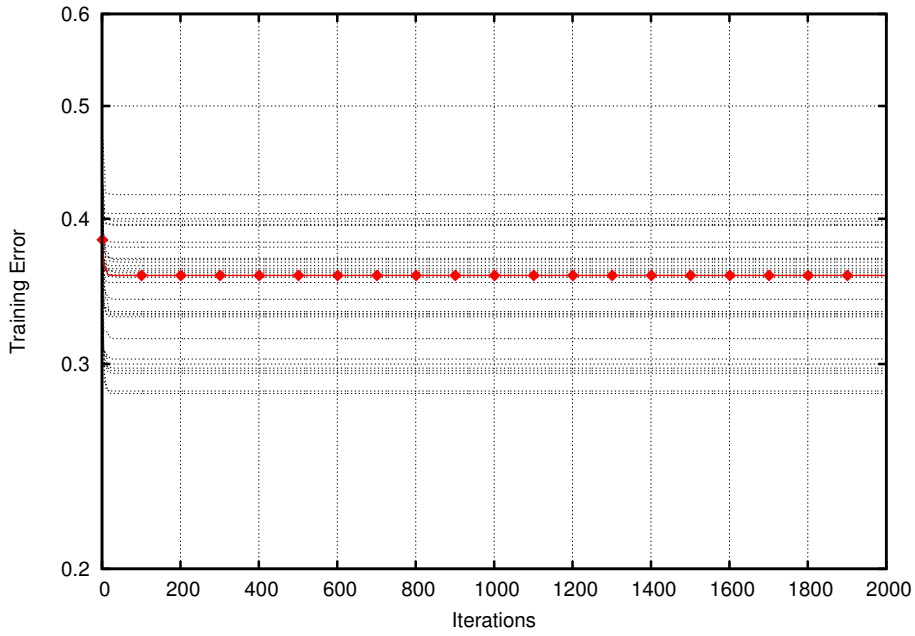


Figure 7.3: Training errors (log scaled) per iteration for linear FFNN samples on the Henon Map data set.

As seen in the figures, the sigmoid and hyperbolic tangent FFNNs have very similar training profiles, with the PSO continuously optimising the problem for the duration of the run. The linear FFNN, however, shows a much different graph: for all samples, there was a large decrease in training error in the first few iterations after which the error remains constant, indicating the swarm failed in finding any optima with a lower training error. In these cases, it is possible that the swarm became stuck in a local optimum,

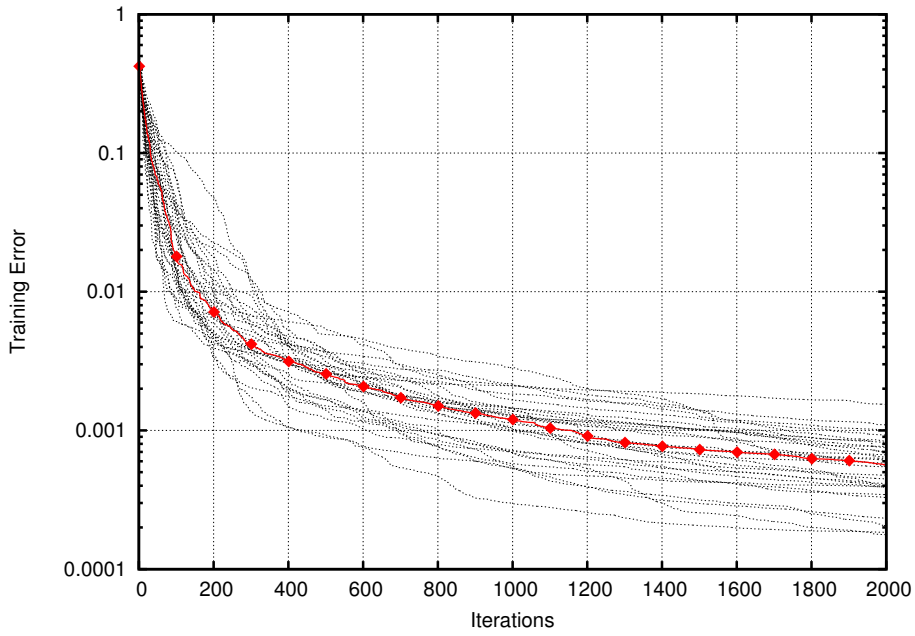


Figure 7.4: Training errors (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set.

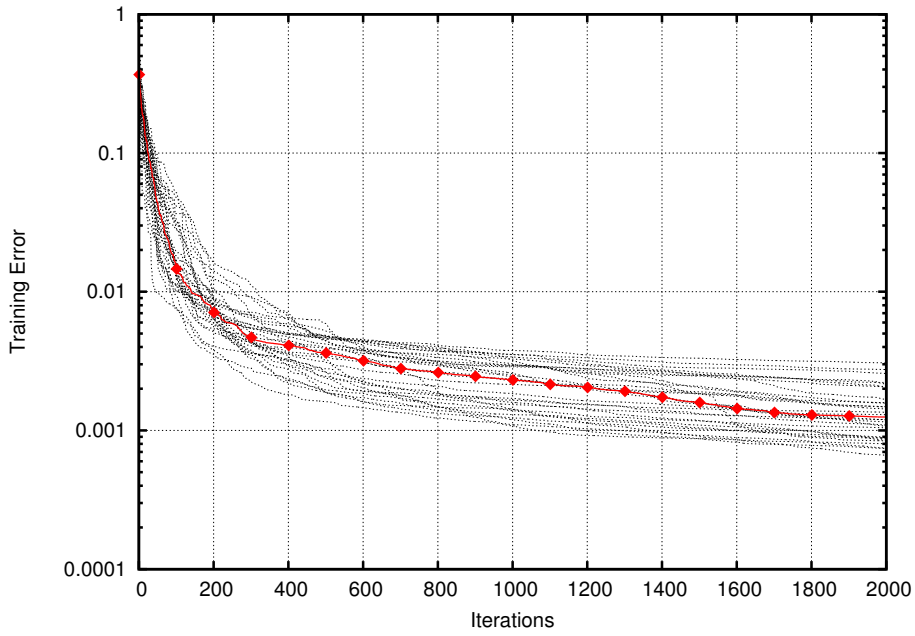


Figure 7.5: Training errors (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set.

and fails to escape the optimum over time. If this is true, the diversity of the swarms should tend to 0 as the optimum is exploited. Or alternatively, the linear FFNNs are simply incapable of accurately modelling the data and no better solution exists. This is analysed in further detail with the diversity results in Section 7.3.3.

7.3.2 Generalisation Accuracy

The generalisation error results for each of the activation functions are given in Table 7.3. All FFNNs obtained very similar generalisation errors, regardless of the activation function. The result of the Friedman test is a p-value of 0.05807, showing that there was no significant difference in generalisation accuracy across all of the data sets. This is a surprising result, as the linear FFNNs were clearly outperformed on the training data.

Table 7.3: Generalisation error mean and standard deviations for each of the activation functions.

	Linear		**Sigmoid**		**Tanh**	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	0.16389	0.01012	0.15836	0.00980	0.15942	0.01118
Glass	0.96249	0.21472	0.87905	0.24763	0.91431	0.26952
Iris	0.04956	0.01201	0.03709	0.01474	0.03930	0.01677
WDBC	0.06467	0.00765	0.02917	0.00694	0.02892	0.00785
Wine	0.07981	0.01374	0.04828	0.02095	0.05372	0.02615
Henon Map	0.42944	0.08615	0.01067	0.01681	0.02427	0.03896
Logistics Map	0.00137	0.00067	0.00139	0.00068	0.00138	0.00055
Mackey-Glass	0.00137	0.00067	0.00124	0.00049	0.00137	0.00054
Sunspot	285.49304	36.37461	236.70343	46.41062	236.82422	45.73935
TS5	0.00346	0.00118	0.00452	0.00108	0.00388	0.00077
Friedman p-value: 0.05807						

When looking at the generalisation errors over time, Figures 7.6 and 7.7 show the generalisation error of sigmoid FFNNs for each sample on the Henon Map data set. The graphs are similar in nature to the training errors for the sigmoid and hyperbolic tangent networks, though, with much larger variation between samples. This can also be noted

from the larger standard deviations in Table 7.3 compared to those for the training error results. Very similar generalisation behaviour was observed on the other data sets.

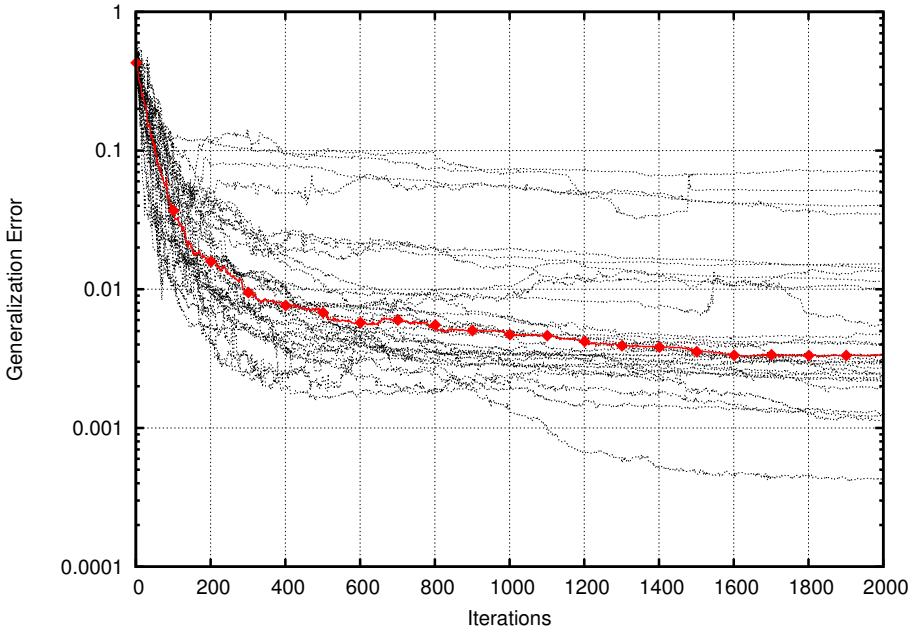


Figure 7.6: Generalisation errors (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set.

It can also be observed that for many of the sigmoid and hyperbolic tangent samples, there was an increase in the generalisation error as optimisation continued, characteristic of overfitting. However, the mean results do not reflect the increases over time, since it did not occur on all samples. The fact that overfitting only occurred for some of the samples, is an indication that the overfitting was dependent on the stochastic behaviour of the swarm.

Similar to the training results, the linear FFNNs showed very different behaviour from the other activation functions. Figure 7.8 shows the per sample generalisation errors for linear FFNNs on the Henon Map data set. Some fluctuation in the generalisation error occurred within the first 200 iterations, after which optimisation stagnated. Due to the stagnation, no overfitting is seen to occur as was the case with the sigmoid and hyperbolic tangent networks.

The overfitting in the non-linear FFNNs explains their poor generalisation accuracy

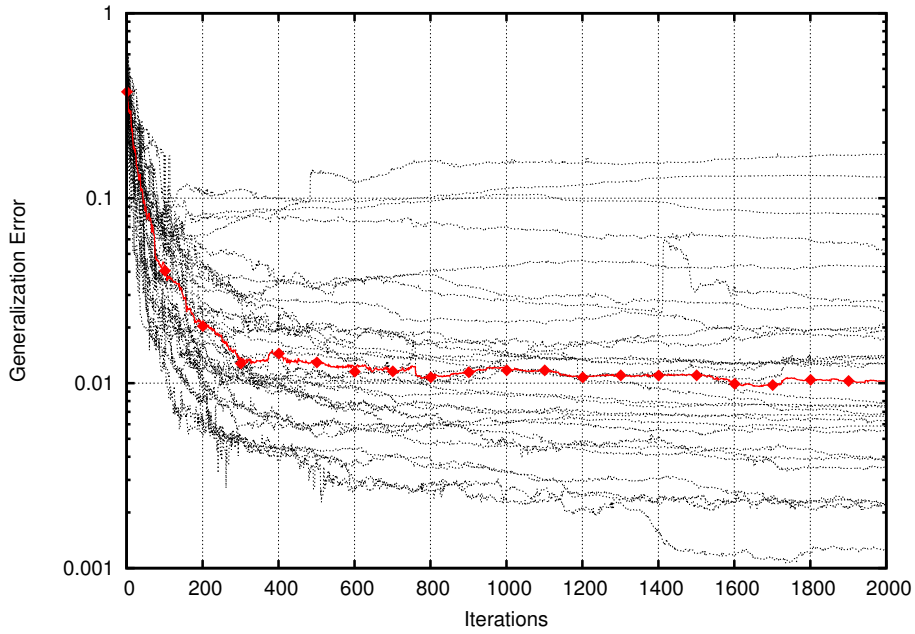


Figure 7.7: Generalisation errors (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set.

and why there was no significant difference in their generalisation performance compared with the linear FFNNs.

Table 7.4 summarises the means and standard deviations of the generalisation factor results. A Friedman test comparing the results returned a significant p-value of 0.00037, the results of the pair wise Wilcoxon tests are shown in Table 7.5. The significance tests show that worse generalisation factors were obtained by the sigmoid and hyperbolic tangent FFNNs compared with the linear FFNNs across all data sets. However, the generalisation factor did not differ significantly between the sigmoid and hyperbolic tangent FFNNs. The significantly higher generalisation factors validate the overfitting shown in the generalisation error results. The magnitude with which ρ_F differs between the linear and other networks is however compounded due to the linear FFNN's poor training performance.

This high, per sample, variation of the generalisation error and factor suggests that for the sigmoid and hyperbolic tangent FFNN, the stochastic nature of the algorithm itself has a significant effect on the generalisation and overfitting. Besides the activation

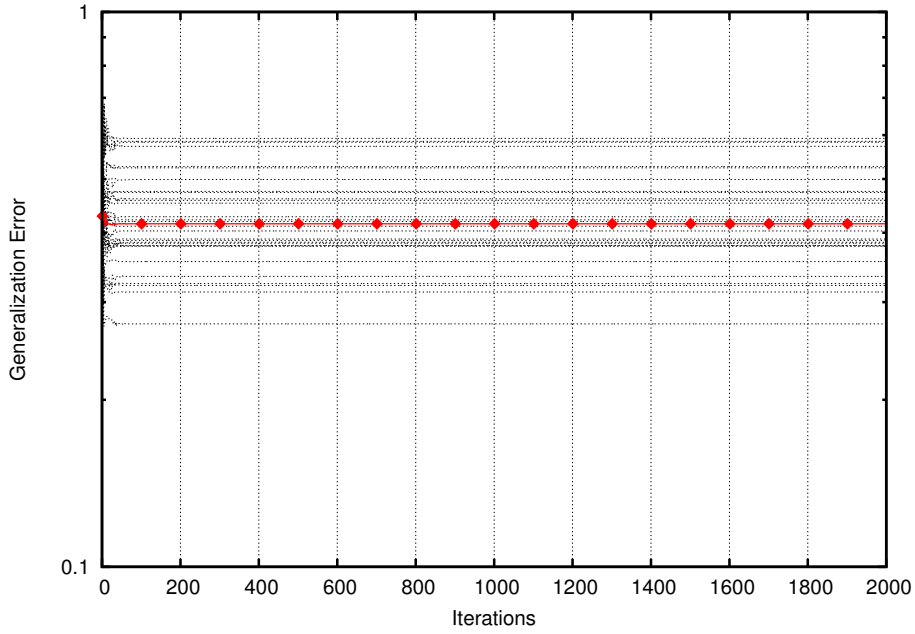


Figure 7.8: Generalisation errors (log scaled) per iteration for linear FFNN samples on the Henon Map data set.

Table 7.4: Generalisation factor mean and standard deviations for each of the activation functions.

	Linear		**Sigmoid**		**Tanh**	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	1.04426	0.09540	1.11003	0.10690	1.12260	0.11850
Glass	1.31634	0.45454	1.70208	0.74202	1.83511	0.85932
Iris	1.08019	0.37082	1.50652	1.09819	1.40890	0.83086
WDBC	1.13680	0.16669	1.84170	0.68848	2.62311	1.29353
Wine	1.41143	0.33855	15.10526	16.06418	13.62793	11.12940
Henon Map	1.26817	0.39660	16.8779	19.6601	14.9836	18.56449
Logistics Map	1.27605	0.96850	2.02382	1.55622	1.80276	1.03876
Mackey-Glass	1.27605	0.96850	2.25601	1.25912	2.18496	1.04314
Sunspot	1.07595	0.21891	1.76883	0.50394	1.72003	0.44527
TS5	1.35967	0.47621	1.99415	0.54696	1.73540	0.37774
Friedman p-value: 0.0003707						

Table 7.5: Wilcoxon rank sum test p-values for generalisation factor results across all data sets.

Activation Function	Linear	Sigmoid
Sigmoid	0.0029	-
Hyperbolic Tan	0.0029	0.29

function, worse overfitting occurred on certain data sets, specifically the the Iris, Wine, Henon-Map and Logistic Map data sets. This phenomenon was not observed for the linear FFNNs, likely due to the early stagnation of the illustrated in Figures 7.3 and 7.8.

7.3.3 Diversity and Convergence

The swarm diversity results for all three activation functions are discussed in this section. The diversity means and standard deviations for each of the data sets and FFNNs are shown in Table 7.6. Immediately noticeable is the extremely large standard deviations obtained for some data sets, especially in the case of the sigmoid and hyperbolic tangent FFNNs. This is indicative of outliers in the diversity data and as such the mean is not necessarily a good indicator of central tendency, the median is therefore also shown in the table.

With a p-value of 0.4066, the result of the Friedman test indicates no significant difference exists when comparing the *final* diversity values across all of the data sets.

Since $\mathcal{D}_{avg}$ has an absolute lower bound of 0 (negative diversities are not possible), the large standard deviations could only have been obtained if some swarms obtained excessively large final diversities, characteristic of severe divergence (i.e. swarm explosion). As mentioned earlier, this was expected, since no $\mathbf{V}_{max}$ parameter was used.

However, the divergence phenomenon did not occur on all data sets. Using the criterion of $\bar{\mathcal{D}_{avg}} > 1.0$ *for all activation functions*, divergence was said to have occurred on all data sets except the Diabetes, Glass and Sunspot data sets. Using only the results for the remaining seven data sets, a second Friedman test was performed which obtained a p-value of 0.01193. This indicates that, for data sets where swarm divergence

Table 7.6: Swarm diversity mean, median and standard deviations for each of the activation functions.

	Linear			Sigmoid			Tanh		
Data Set	μ	$\tilde{\mathcal{D}_{avg}}$	σ	μ	$\tilde{\mathcal{D}_{avg}}$	σ	μ	$\tilde{\mathcal{D}_{avg}}$	σ
Diabetes	0.71202	0.00155	0.94522	0.19723	0.12176	0.30356	0.30739	0.05128	0.82733
Glass	0.46935	0.16004	0.56920	0.12093	0.06787	0.18220	0.23769	0.04528	0.53634
Iris	743.109	198.678	1753.74	13726.8	2472	33115.8	5568.39	713.912	19927.1
WDBC	1.26690	1.05795	1.20240	1.31676	0.09986	2.68156	0.93136	0.12864	1.79627
Wine	17.1774	15.6	6.80043	15189.8	938.451	34209.6	218.438	91.2404	562.018
Henon Map	1.71339	1.72276	0.29838	140.868	6.72582	416.798	12.903	4.01229	36.0592
Logistic Map	2.94217	2.85352	0.41764	296.487	13.549	1516.2	6.93903	5.5107	6.78715
Mackey-Glass	2.76173	2.91764	0.80287	5.00045	1.42606	8.03390	15.2508	0.54958	70.1036
Sunspot	0.43416	0.00006	0.62378	0.42717	0.16483	0.69912	0.39433	0.07465	1.08090
TS5	2.83100	0.80964	10.6835	952.203	1.53622	4136.96	28867.2	0.63965	158035
Friedman p-value: 0.4066									

occurred, a significant difference in diversity exists between FFNNs using different activation functions. The subsequent Wilcoxon rank sum tests, the results of which are shown in Table 8.2, showed that the linear FFNNs obtained significantly smaller diversities when compared against the sigmoid and hyperbolic tangent FFNNs. The Wine data set is an example where this is clearly evident, the linear FFNN obtained a $\tilde{\mathcal{D}_{avg}}$ of 15.6 compared with the sigmoid and hyperbolic tangent FFNNs which obtained $\tilde{\mathcal{D}_{avg}}$ values of 938.451 and 91.2404 respectively.

Table 7.7: Wilcoxon rank sum test p-values for diversity results on data sets with $\mathcal{D}_{avg} > 1.0$ for all activation functions.

Activation Function	**Linear**	**Sigmoid**
Sigmoid	0.023	-
Hyperbolic Tan	0.032	0.812

From the above results it is clear that the activation function has a significant impact

on the swarm diversity and convergence. In particular, on data sets where swarm divergence occurred, the diversity was significantly reduced when a linear activation function was used.

The difference in swarm behaviour is better illustrated when looking at the swarm diversities on a per sample basis. Graphs showing the diversity of each sample for all three activation functions on the Henon Map data set are shown in Figures 7.9, 7.10 and 7.11. Similar results were obtained for other data sets.

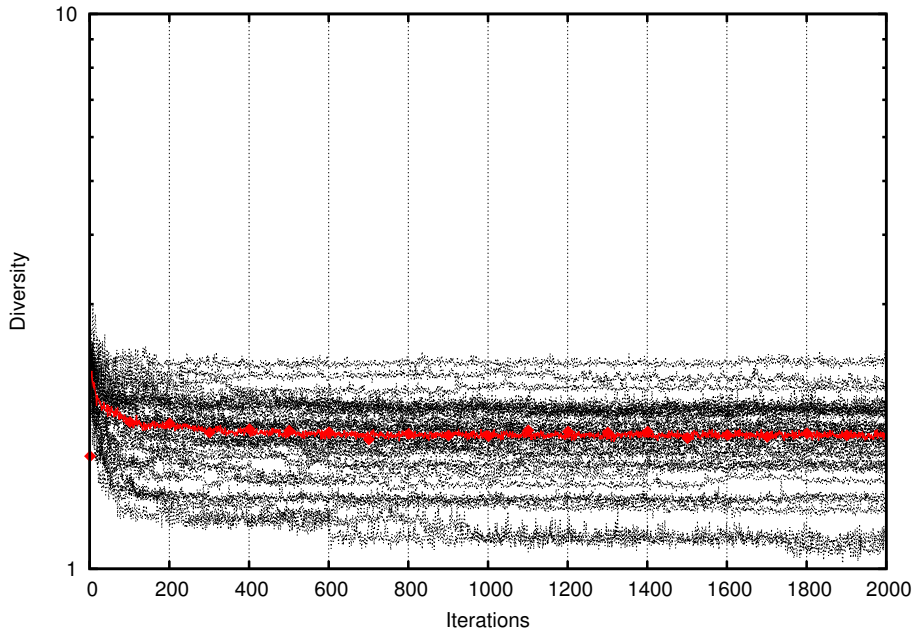


Figure 7.9: Diversity (log scaled) per iteration for linear FFNN samples on the Henon Map data set.

Even with the absence of a $\mathbf{V}_{max}$ parameter, the linear FFNN samples, shown in Figure 7.9, exhibited behaviour consistent with the PSO model: diversity decreases over time as the swarm converges to a solution. Samples are also shown to form a reasonably tight cluster around the median.

Although the PSO exhibited convergent behaviour when training linear FFNNs with a high degree of exploration, the training errors shown previously for the linear FFNNs indicate that the FFNNs were ineffective at modelling the training data, likely due to an insufficient number of hidden units. Further investigation is required to ascertain the

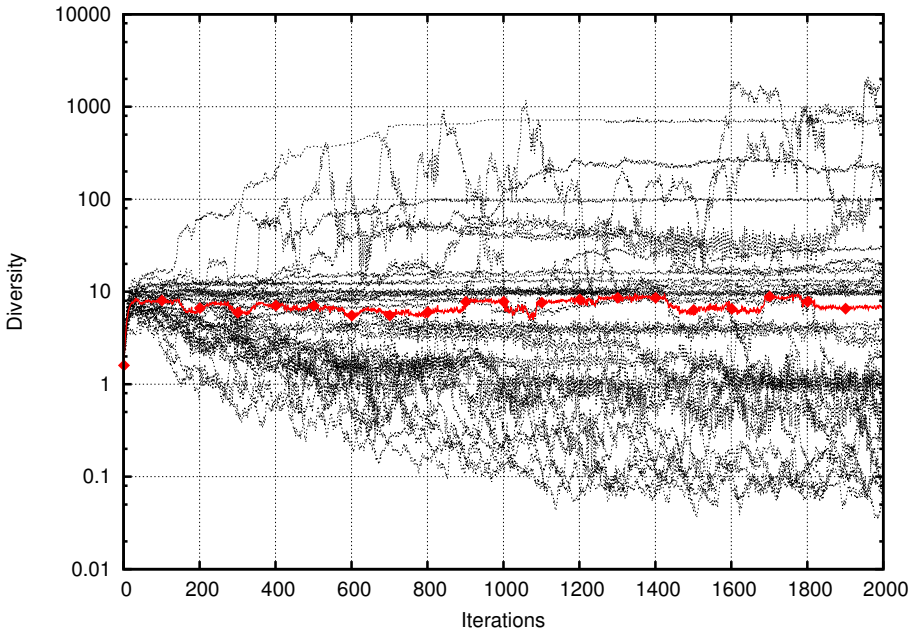


Figure 7.10: Diversity (log scaled) per iteration for sigmoid FFNN samples on the Henon Map data set.

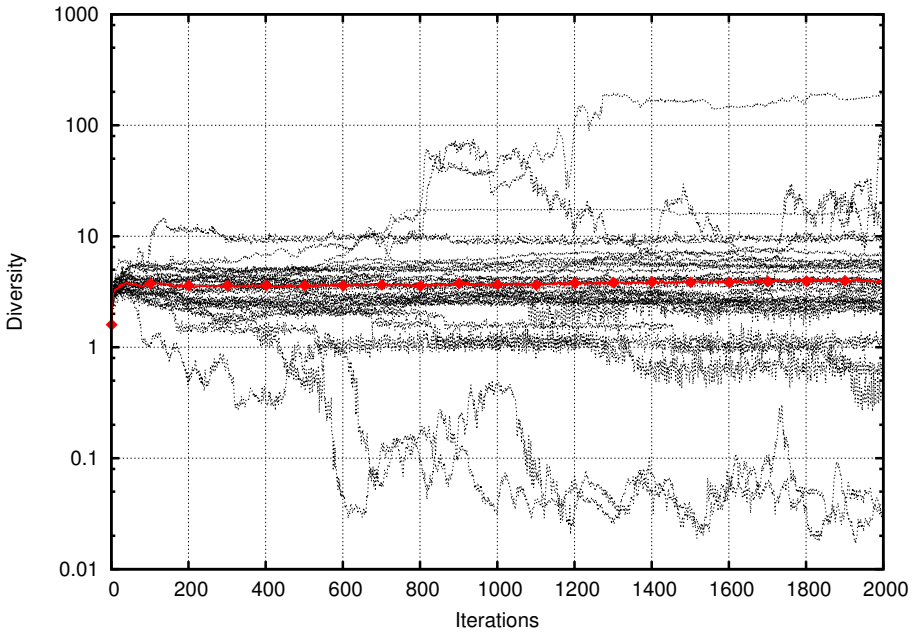


Figure 7.11: Diversity (log scaled) per iteration for hyperbolic tangent FFNN samples on the Henon Map data set.

behaviour of larger FFNNs using the linear activation function.

The sigmoid and hyperbolic tangent FFNNs, shown in Figures 7.10 and 7.11 shows distinctly different behaviour. Although some samples are shown to converge, for most samples, the diversity is erratic over time and, in many cases, is shown to *increase* as the optimisation progresses. This effect is more pronounced for the sigmoid FFNNs.

The lack of swarm convergence did not however seem to affect either the training or generalisation errors. As shown earlier, none of the Henon Map sigmoid or hyperbolic tangent FFNN samples were shown to have either stagnating or increasing errors. Similarly, no indication of any affect on either error was observed for any other data set where the swarms diverged. This disconnect between particle movement and the obtained optima is very irregular PSO behaviour.

Activation Function Bounds

As discussed in Section 7.1, one of the primary differences between the sigmoid and hyperbolic tangent functions compared with the linear function is the *bounded* nature of the functions described with Equation (7.2).

Considering a single neuron, a large *net* input would be a direct consequence of large particle positions in one or more dimensions, since the particle positions weigh the input values. However, due to the function asymptotes, significant increases in the magnitude of the *net* input signal *would not significantly affect the output of the activation function* as per Equation (7.2). Neighbourhood best particles could therefore potentially make large positional changes in one ore more dimensions with either extremely small changes, or no changes whatsoever, occurring in objective function evaluation.

Changes in the particle positions are determined by the velocities. As particles continue to drift to positions that lead to activations closer to the bounds of the function (with small changes in neuron activation), velocities increase without bound (assuming no $\mathbf{V}_{max}$ parameter). It therefore becomes increasingly difficult for the particles to change direction: the momentum term of the velocity update equation (Equation (2.10)) dominates the velocity calculation due to the size of previous velocities. As the velocities increase, the swarm diverges drastically and becomes unable to effectively exploit optima.

This hypothesis would explain the behaviour observed previously where large particle positions, as observed through large swarm diversities, were seen to have very little or no effect on the training and generalisation errors.

Further, the sigmoid function has a steeper gradient compared against the hyperbolic tangent function. Considering Equation (7.2), in the case of the sigmoid function, the limit tends to zero faster as the *net* input signal increases due to the steeper gradient. A PSO training a FFNN using sigmoid activation functions would therefore be more susceptible to divergence as an indirect result of the activation function bounds, and indeed, this was observed in the results.

The hypothesis is also consistent with observations from Chapter 4 where it was shown that, although an appropriate $\mathbf{V}_{max}$ value greatly aids the swarm's convergence, it only served to *delay* eventual divergence in some cases. This follows intuitive from the theory: if a $\mathbf{V}_{max}$ parameter is used, particles may still continue to move towards the bounds of the function, but will move much slower under a constrained velocity, thereby delaying divergence from the centre of the swarm.

However, severe swarm divergence did not occur for all runs of the algorithm, nor on all data sets, showing that the swarm diversity, and subsequently any swarm divergence, is also significantly influenced by both the data set and the stochastic conditions. This too is reflected in the results of Chapter 4 where a higher $\mathbf{V}_{max}$ value was found to be optimal on certain data sets.

Avoiding Swarm Divergence

Assuming the bounded activation function hypothesis holds true, there are a number of methods that could improve the PSO performance.

The simplest method is the use of an appropriate $\mathbf{V}_{max}$ parameter, relative to the swarm size, as shown in Chapters 4 and 6. However, the $\mathbf{V}_{max}$ parameter would have to be exceptionally small in order to effectively constrain particle velocities, which will have the unwanted side-effect of increasing the required training time. Further, the effect of the velocity equation in controlling the optimisation process is mitigated if all particles simply have a step size equal to $\mathbf{V}_{max}$.

As shown above, the hyperbolic tangent FFNNs had better convergence, hypotheti-

cally due to the hyperbolic tangent function's reduced slope. Using the function parameter λ (as in Equations (7.1) and (7.3)), the steepness of the function can be reduced by decreasing the value of λ. Changing the function's slope will increase the correlation between the net input signal and the activation value, but reduce the function's non-linearity.

Using sigmoid or hyperbolic tangent activation functions with varying degrees of gradient adjustments may also be used to further validate the bounded activation hypothesis made in this chapter. This is investigated further in the next chapter.

7.4 Summary

This chapter investigated the behaviour of particles swarms, in particular swarm convergence, when training FFNNs using a number of different activation functions. Section 7.1 described each of the functions, followed by the experimental methodology in Section 7.2.

The results of the chapter was given in Section 7.3. It was shown that the two, non-linear activation functions outperformed their linear counterpart on training data. However, no statistically significant difference in performance was found between the activation functions on the generalisation data across all tested data sets. This was prescribed to the overfitting that was shown to occur for the sigmoid and hyperbolic tangent networks on the data sets.

The results also showed that the generalisation accuracy differed significantly on a per sample basis, indicating that the stochastic conditions during optimisation has a significant impact on the overfitting of the network.

Finally, the diversity results showed that very different swarm behaviour was observed when comparing the unbounded linear function FFNN against the asymptotically bounded sigmoid and hyperbolic tangent function FFNNs. Based on the results, an hypothesis was made stating that the bounded nature of the functions leads to the swarm divergence observed in this and previous chapters. A number of methods to avoid such divergence was also discussed.

The next chapter further develops the idea of activation function adjustment by

adding the function parameters to the search space of the algorithm, thereby allowing the PSO to adapt both the FFNN weights and activation function to the data sets.

Chapter 8

Adaptive Activation Functions

The previous chapter investigated the PSO's behaviour when training FFNNs using various activation functions. A hypothesis was made about the use of asymptotically bounded activation functions: due to the asymptotic nature of the function, particles continued to move toward the asymptote without significantly affecting the accuracy of the FFNN. The unbounded acceleration of particles toward the asymptotes then lead to divergent swarms, preventing the effective exploitation of optima.

The purpose of this chapter is two-fold: firstly, to further investigate the hypothesis made in the previous chapter by analysing the convergence of swarms when adaptations to the activation function are made. Specifically, adaptations to the gradient and range of the functions are investigated. Secondly, to investigate the feasibility of the PSO as a training algorithm when using self-adaptive activation functions, that is, parameters of the activation functions are optimised alongside the network weights.

The remainder of the chapter is set out as follows: Section 8.1 discusses how the sigmoid function may be adapted to further analyse the effect of asymptotic activation function bounds on the PSO. This is followed by a brief review of previous work on self-adaptive activation functions for FFNN training in Section 8.2. The experimental methodology used to conduct the investigations is given in Section 8.3. This is followed by the results and summary in Sections 8.4 and 8.5 respectively.

8.1 Sigmoid Adaptation

The general form of the sigmoid function, as given by Equation (7.1), has two parameters which control the shape of the function, i.e. λ and γ. The derivative of the function is given by:

$$f'_{sig}(net, \lambda, \gamma) = \frac{\gamma\lambda e^{\lambda net}}{(e^{\lambda net} + 1)^2} \tag{8.1}$$

As illustrated by the function's derivative, both λ and γ affect the gradient of the function. As an activation function, adjusting the gradient allows control over the active domain of the function, and as such allows better accommodation of *net* input signals that do not correspond with the active domain of $(-\sqrt{3}, \sqrt{3})$ when $\lambda = \gamma = 1$. The γ parameter additionally controls the function's output range, which is defined as $(0, \gamma)$ if $\gamma > 0$ and $(\gamma, 0)$ if $\gamma < 0$.

Considering the results of Chapter 7, adjustment of the gradient of the sigmoid function, and thus the proximity of the asymptotic bounds to 0, also potentially affects the convergence and thus the FFNN training potential of the PSO.

Furthermore, appropriate adjustment of the function parameters of each activation function in a FFNN theoretically allows the network to model completely unscaled data. Besides the obvious benefit of eliminating the overhead associated with data scaling, the capability of learning from unscaled data also allows for incremental training of a network. With incremental training not all data patterns are known beforehand, and it is therefore very difficult or in some cases impossible to obtain values such as the maximum, minimum, mean, median or other descriptive statistics that would be required to scale the data [28].

Furthermore, in the context of back-propagation training, it has been shown that scaling the data into numerically smaller ranges necessitates longer training times of the FFNN to obtain a required accuracy on the unscaled data [28].

In the case of the sigmoid function, a notable restriction to any unscaled data is that if a data set or function has both positive and negative target output values, which is the case with many regression problems, a single sigmoid output neuron would not be able to accommodate all output targets, as the output range is always exclusively positive or negative, as shown above.

Although perhaps feasible for the output neuron, manually adapting each activation function for all of the hidden neurons is an intractable problem, especially since optimal values for the functions' parameters are dependent both on the input data and network weights. As such, a more feasible strategy is to algorithmically optimise the activation function parameters alongside the weights of the network using a technique such as the *lambda-gamma* learning rule [121].

8.2 Lambda-Gamma Learning

This section briefly discusses previous work on the adaptation of the sigmoid function during training. The modification to the PSO FFNN training algorithm to accomplish the same is given in Section 8.2.2.

8.2.1 Generalised *lambda-gamma* Learning

Zurada developed the generalised *lambda learning algorithm* for layered networks as an extension of the error back-propagation (BP) algorithm to allow for the learning of the λ values for each of the activation functions in addition to the network weights [121]. The addition of a *lamba* learning rule proved effective and the *lambda* learning algorithm has been shown to outperform the standard BP algorithm in certain cases.

The generalised *lamba-gamma* learning algorithm was later developed by Engelbrecht et al. to include a *gamma* learning rule [28]. The addition of a *gamma* rule allowed for self-scaling neurons in the network, theoretically enabling learning from unscaled data. The generalised *lambda-gamma* algorithm was shown to require fewer training cycles to achieve a desired accuracy when compared against the standard BP algorithm learning from scaled data [28].

8.2.2 PSO *lambda-gamma* Learning

As discussed above, self-adaptive FFNNs have many potential advantages, the most interesting of which in the context of this work is the possibility of improving the convergence of the swarm.

In order for the PSO algorithm to optimise the FFNN weights and the function parameters of each neuron, the following approach was taken. The particle position (candidate solution) is augmented with the λ and γ parameters:

$$\mathbf{x} = (x_0, x_1, ..., x_n, \lambda_0, \lambda_1, ..., \lambda_{J+K}, \gamma_0, \gamma_1, ..., \gamma_{J+K}) \tag{8.2}$$

where n is the number of network weights and biases, J is the number of hidden units and K is the number of output units. The appropriate function parameters are then used when calculating a feedforward pass through the network. For the parametric sigmoid activation $f(net, \lambda, \gamma)$ as given in Equation (7.1) and an input pattern $\mathbf{z}_p$:

$$y_{j,p} = f_{y_j}(\sum_{i=1}^{I+1} w_{ji} z_{i,p}, \lambda_j, \gamma_j), \quad \forall\, j \in \{1, ..., J\} \tag{8.3}$$

$$o_{k,p} = f_{o_k}(\sum_{j=1}^{J+1} w_{(J+k)j} y_{j,p}, \lambda_{J+k}, \gamma_{J+k}), \quad \forall\, k \in \{1, ..., K\} \tag{8.4}$$

Appropriate strategies are required for the initialisation of the λ and γ vectors for each particle. Considering the λ parameter, one potential strategy is to initialise λ uniformly in the range $(0.0, 2.0]$ allowing exploration of both gradual gradients less than 1 and much steeper gradients larger than 1.

Since the γ parameter controls the possible output range of the function, an appropriate initialisation strategy is uniform initialisation in the range $(0.0, max(t)]$, where $max(t)$ is the maximum target output value across all patterns and dimensions for the data set.

These strategies were used in the experimental work of this chapter, although a number of other initialisation strategies for both the λ and γ parameters may exist, and can be developed for future research.

The addition of the *lambda-gamma* parameters significantly increases the optimisation complexity in two distinct ways. The first is an increase in the dimensionality of the search space, which is now defined as the number of weights plus double the number of hidden and output units. The second complexity is the interaction of the λ and γ parameters as illustrated by Equation (8.1). That is, both variables affect the gradient of the function and are said to be non-linearly separable from each other, as seen through

Equation (8.1). Non-linear separability of parameters in the solution (position) vector is known as epistasis in the context of genetic algorithms (GA) and has been shown to significantly increase the hardness of the problem, particularly in the case of GAs, but also the PSO [70, 91].

The PSO algorithm has, however, been found to be an effective optimisation algorithm in both high-dimensional environments and environments with large degrees of variable inter-dependency [103]. As such, it is a good candidate as a training algorithm for the *lambda-gamma* augmented networks.

The approach described in this section to train *lambda-gamma* augmented networks using a PSO is referred to as the PSO Lambda-Gamma (PSO-LG) algorithm for the remainder of this study.

8.3 Experimental Methodology

This section discusses the methodology used in this chapter's experimental work. Two experiments are performed. The first, described in Section 8.3.4 involves predictable, static adaptations to the activation functions to investigate their effect on the PSO algorithm in an attempt to investigate the bounded activation function hypothesis.

The second experiment investigates and compares PSO *lambda-gamma* learning on a number of scaled and unscaled data sets in terms of training and generalisation accuracy and overfitting as detailed in Section 8.3.4.

8.3.1 Data Sets

The data sets given in Section 3.2 were used for all experimental work in this chapter. However, preparation for each of the data sets differed as follows.

For the experimental work that required *scaled* data sets, the scaling as described in Section 3.2 was used with the input values scaled to the range $[-2, 2]$ and the output values scaled to $[0.1, 0.9]$.

Where *unscaled* data was required, neither the input nor the output values of the data sets were scaled, regardless of size, with the exception of the Henon Map and TS5 data sets. The Henon Map and TS5 data sets have negative output values for some data

patterns, and due to the limitations of the sigmoid function discussed above, output scaling was always applied for these data sets to the range $[0.1, 0.9]$.

8.3.2 PSO Configuration

As in Chapter 7 an *lbest* PSO with a neighbourhood size of five and a swarm size of 25 was used for both experiments. Particle velocities were initialised to $\mathbf{0}$ as described in Section 3.3.2. Where static activation functions were used, particle positions were initialised to the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$. In the case of adaptive functions, the weight sub-vectors of the position vector was initialised in $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$ with the λ and γ sub-vectors each initialised as discussed in Section 8.2.2.

The per data set optimised ω, c_1 and c_2 control parameters as given in Table 6.2 were used for both experiments. Although these values were optimised for training standard FFNNs, they serve as a good starting point for training with the additional *lambda* and *gamma* parameters.

The algorithm execution was limited to 50000 objective function evaluations for both experiments.

8.3.3 Algorithm Measurements

The measurements given in Section 3.3.3 were used to quantify performance of the algorithm and NNs. As described in Section 3.3.3 the training and generalisation errors were calculated on the re-scaled data, that is, data that has been scaled to their original values (if applicable) before calculating the error. Error measurements on scaled and unscaled data sets are therefore comparable.

8.3.4 Experimental Procedure

The experimental procedure for both experiments are given in this section.

Static Activation Functions

Chapter 7 stated the hypothesis that the use of bounded activation functions within FFNNs that are trained by a PSO caused severe swarm divergence due to particles

moving towards the bounds of the function without it necessarily having a significant impact on the output of the FFNN (and thus the objective function evaluation). This was attributed to the flat gradient of the function at the function's asymptotic ends.

This hypothesis is supported by results shown in Chapter 7 where swarm divergence did not occur for the *unbounded* linear activation function. A feasible course of analysis is therefore to adapt the sigmoid function to be more linear in nature and analysing of swarm trained using such sigmoid functions.

Using the λ parameter, the gradient of the sigmoid function can be adapted, such that the function resembles the linear function, $y = x$. A $\lambda < 1$ reduces the gradient of the sigmoid function thereby expanding the active domain and creating a more gradual slope approaching the function asymptotes. Conversely, with a $\lambda > 1$ the gradient steepens and the active domain contracts as the function resembles the step function.

Both steeper and more gradual gradients and their effect on the PSO as FFNN training algorithm were tested. A FFNN was configured using a specific λ value for all activation functions in the hidden and output layers. Scaling was applied to all data sets as described in Section 8.3.1 and the PSO algorithm was used to train the FFNN on each of the data sets. Each execution on a data set was repeated for 30 samples, where each sample used an unique seed for a Mersenne Twister PRNG, and the performance recorded. This procedure was repeated for each $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.1, 1.3, 1.5, 1.7, 1.9\}$.

Results for each of the λ values were compared across all data sets using a Friedman test with $\alpha = 0.05$. The following hypothesis was evaluated:

- H_0: There is no significant difference in performance between any of the λ values.
- H_1: There is an one-tailed significant difference, lower values in the cases of E_T and E_G and higher values in the cases of ρ_F and $\mathcal{D}_{avg}$, between at least one pair of λ values.

If the Friedman test resulted in the acceptance of H_1, the pairwise Wilcoxon rank sum tests were performed comparing pairs of λ values.

As the purpose of the experiment is to investigate the divergence hypothesis with regards to the use of bounded activation functions, no $\mathbf{V}_{max}$ parameter was used for this

experiment, making the diversity results directly comparable with the results of Chapter 7.

Adaptive Activation Functions

The purpose of the second experiment is to investigate the capability of the PSO algorithm and the performance of the FFNNs when trained using the *lambda-gamma* training technique described in Section 8.2.2. Additionally, training on unscaled data is also investigated.

The PSO was configured with the *lambda-gamma* particle presentation and then used to train a FFNN on all data sets discussed in Section 8.3.1 with data scaling applied. Each execution of the algorithm on a data set was repeated 30 times with each sample using a unique seed for a Mersenne Twister PRNG.

This procedure was then repeated for unscaled data as described in Section 8.3.1.

The results of the adaptive FFNNs, for both scaled and unscaled data, were then compared against standard FFNNs trained on scaled data sets using the following hypothesis:

- H_0: There is no significant difference in performance between any of the algorithms.
- H_1: There is an one-tailed significant difference, lower values in the cases of E_T and E_G and higher values in the cases of ρ_F and $\mathcal{D}_{avg}$, between the algorithms.

A Friedman test with $\alpha = 0.05$ was performed to compare the algorithms across all data sets. If significant, pairwise Wilcoxon rank sum tests were performed, with $\alpha = 0.05$, comparing each algorithm against the other.

8.4 Results

The results of the experimental work of this chapter is given and discussed in this section. Section 8.4.1 discusses the results of the static activation function experiments. This is followed by the adaptive activation function experimental work in Section 8.4.2.

8.4.1 Static Activation Functions

The results of the static activation function experiment are given and discussed here. The results are divided into two sections: the diversity and convergence results followed by the training and generalisation accuracy results.

Diversity and Convergence

Table 8.1 presents the swarm diversity results for each of the sigmoid λ values tested. Similar to the results of Chapter 7, large standard deviations were obtained due to swarm divergence that occurred for most samples which in turn lead to outlying diversity values; the median result is therefore also given. Due to the magnitude of the outliers, the Friedman and Wilcoxon rank sum tests were also performed on the median diversity results.

A Friedman test produced a p-value of 2.181e-9, indicating a significant difference between at least two λ values across all data sets. The subsequent Wilcoxon rank sum test results are given in Table 8.2. The pairwise tests show that, with the exception of the $\lambda = 0.3$ vs. $\lambda = 0.5$ tests, as λ was increased from 0.1, the diversity decreased significantly up to $\lambda = 0.3$. Diversity continued to decrease for $\lambda > 0.3$, though insignificantly so.

Chapter 7 showed that when using the linear activation function, significantly lower final diversity were obtained by the swarms than when using sigmoid FFNNs. Further, no severe swarm divergence was seen to occur with linear activation functions. As explained in Section 8.3.4, it was expected that diversities should similarly decrease when using sigmoid functions with $\lambda < 1$. Indeed, the opposite is observed here: diversity and severe swarm divergence is shown to increase as λ decreases i.e. as the gradient of the sigmoid function decreases.

Section 8.3.4 made the statement that a sigmoid function with a small λ parameter becomes more linear in nature, shifting the bounds away from the origin. Figure 8.1 shows the derivative of five different activation functions: the linear function, hyperbolic tangent function and the sigmoid function for $\lambda \in 0.5, 1.0, 1.5$. The figure illustrates that, although for $\lambda = 0.5$ the derivative of the sigmoid function is shown to be flat in nature, similar to the linear function, the *magnitude* of the gradient is also significantly decreased.

Table 8.1: Swarm diversity median when training FFNNs with static sigmoid functions using indicated λ values.

Data Set	$\lambda = 0.1$	$\lambda = 0.3$	$\lambda = 0.5$	$\lambda = 0.7$	$\lambda = 0.9$	$\lambda = 1.0$	$\lambda = 1.1$	$\lambda = 1.3$	$\lambda = 1.5$	$\lambda = 1.7$	$\lambda = 1.9$
Diabetes	1.99521	0.41088	0.17380	0.15919	0.12891	0.12176	0.09075	0.06583	0.05545	0.05381	0.05000
Glass	1.17390	0.25586	0.12817	0.10212	0.08848	0.06787	0.06717	0.07412	0.03960	0.03480	0.03421
Iris	48319.4	8822.27	5082.16	5970.48	3611.53	2472.00	1810.25	2582.30	2841.42	6738.40	2670.18
WDBC	2.75500	0.80151	0.39409	0.21680	0.17552	0.09986	0.17003	0.09006	0.09742	0.09880	0.06054
Wine	13974.6	7909.55	4532.74	2336.60	1793.45	938.451	805.226	2411.50	3801.25	617.992	1050.97
Henon Map	48.5315	7.54401	13.7115	3.24757	9.13975	6.72582	4.37648	4.52302	3.89755	5.14714	3.83722
Logistic Map	131.699	40.7277	21.9249	16.3103	13.5399	13.5490	10.5616	9.20085	8.69144	7.15207	7.11017
Mackey-Glass	5.70172	2.25206	1.63601	1.32180	6.02527	1.42606	0.68286	0.81197	0.40705	1.66599	0.74923
Sunspot	3.10197	0.61781	0.40406	0.24837	0.17554	0.16483	0.20077	0.11973	0.09726	0.10911	0.08324
TS5	4.78350	1.17874	0.60122	0.55202	1.17234	1.53622	1.24607	0.90917	0.37968	1.62823	1.38401
Friedman p-value: 2.181e-09											

Table 8.2: Wilcoxon rank sum test p-values for swarm diversity results when training FFNNs using static sigmoid functions with indicated λ values.

λ	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1.0**	**1.1**	**1.3**	**1.5**	**1.7**
0.3	0.019	-	-	-	-	-	-	-	-	-
0.5	0.019	0.188	-	-	-	-	-	-	-	-
0.7	0.019	0.019	0.271	-	-	-	-	-	-	-
0.9	0.031	0.562	0.416	1.000	-	-	-	-	-	-
1.0	0.019	0.069	0.150	1.000	0.228	-	-	-	-	-
1.1	0.019	0.031	0.112	0.562	0.228	0.188	-	-	-	-
1.3	0.019	0.019	0.112	1.000	0.271	1.000	1.000	-	-	-
1.5	0.019	0.019	0.019	0.753	0.342	1.000	1.000	1.000	-	-
1.7	0.019	0.069	0.870	1.000	0.753	1.000	1.000	1.000	1.000	-
1.9	0.019	0.031	0.112	0.562	0.112	1.000	1.000	0.753	0.562	0.271

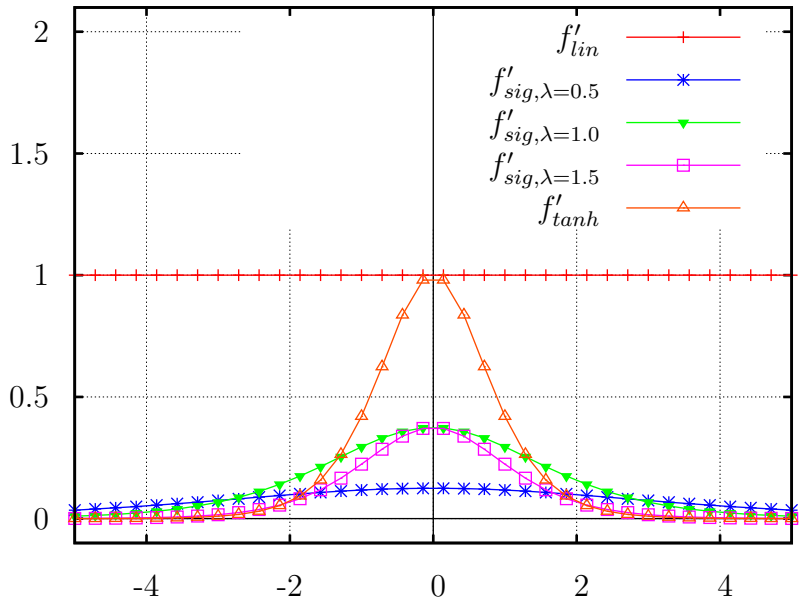


Figure 8.1: Activation function gradients.

Gradient Hypothesis

Chapter 7 hypothesised that the bounds of the activation functions were leading to divergent swarms. However, from the results given in Table 8.1 it can be concluded that, instead of the asymptotic bounds, the *change in magnitude of the gradient, i.e. the steepness of the activation function significantly affects the swarm diversity* when training FFNNs: shallower activation function gradients lead to swarm divergence.

From a theoretical point of view, the influence of the gradient on swarm convergence could be explained as follows: considering a single neuron, with weights defined by particle position $\mathbf{x}$. In minimising the error, a particle seeks to find a position $\mathbf{x}^*$ such that $f_{sig}(\mathbf{net}_x^*, \mathbf{z}_k) = \mathbf{t}_p$ where $\mathbf{t}_p$ is the target output for input pattern $\mathbf{z}_p$. The distance and direction the particle needs to move in the search space is then defined as $\Delta\mathbf{x}^* = \mathbf{x}^* - \mathbf{x}$. Algorithmically, this is achieved through the velocity update. The social and personal best positions approximate $\mathbf{x}^*$ over the course of optimisation: that is, the personal best position, $\mathbf{y}$, approaches the neighbourhood best position, $\hat{\mathbf{y}}_i$, which ideally approaches the solution vector, $\mathbf{x}^*$, and serve as attractors for other particles. The basis of the velocity update equation (Equation (2.10)) is defined by the difference to the attractor positions: $\Delta\mathbf{x}_y = \mathbf{y} - \mathbf{x}$ and $\Delta\mathbf{x}_{\hat{y}} = \hat{\mathbf{y}}_i - \mathbf{x}$.

The gradient of the activation function, by definition, relates any changes in function output, that is, the activations of the neurons in the network, to the change in input, i.e. the weighted sum of the particle position with the input pattern. Additionally, the change in function input is inversely proportional to the gradient. For a neuron, for which $\hat{\mathbf{y}}_i$ defines the neighbourhood best position, consider the distance to the current position: $\Delta\mathbf{x}_{\hat{y}} = \hat{\mathbf{y}}_i - \mathbf{x}$. For an activation function with a steep gradient, the distance $\Delta\mathbf{x}_{\hat{y}}$ is *smaller* compared against the $\Delta\mathbf{x}_{\hat{y}}$ for an activation function with a shallow gradient if the functions have the same output.

This effect is illustrated in Figure 8.2 for a function with a shallow gradient, $f_{sig,\lambda=0.5}$, and a function with a steep gradient, $f_{sig,\lambda=1.5}$. The figure shows that the distance $|\Delta\mathbf{x}^*| = |\mathbf{x}^* - \mathbf{x}|$ required to change the activation from value $f_{sig}(\mathbf{net}_x, \mathbf{z}_p)$ to value $\mathbf{t}_p$ is significantly larger when $\lambda = 0.5$ compared to $\lambda = 1.5$.

The increased distance due to the activation function gradient directly leads to increases in the magnitude of $\Delta\mathbf{x}_y$ and $\Delta\mathbf{x}_{\hat{y}}$ used in the velocity update of particles, which

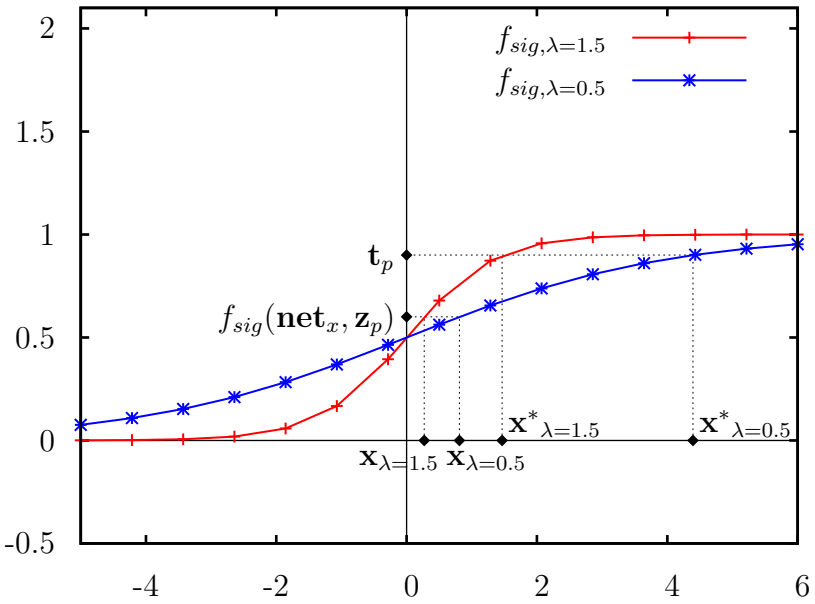


Figure 8.2: Gradient function input and output to activation functions with shallower ($\lambda = 0.5$) and steeper ($\lambda = 1.5$) gradients for a single input pattern.

in turn increases the size of the velocity of particles. Large velocities, in the absence of a $\mathbf{V}_{max}$ parameter, are known to cause divergent swarms, as illustrated through extremely large swarm diversities [15]. This is observed in the results: as the steepness of the gradient decreases, the diversity is shown to increase.

Results obtained in Chapter 7 also showed decreased swarm divergence for the linear and hyperbolic tangent activation functions. This is consistent with the gradient hypothesis: both functions have significantly larger gradients than the sigmoid function for the active range as seen in Figure 8.1.

Finally, the hypothesis that shallower gradients cause divergence in swarms when training FFNNs can naturally be extended to the bounds of the function: As particles move closer to the bounds, the steepness of the function decreases further, thereby further increasing distances from particle positions to neighbourhood and personal best positions in the velocity update.

Training and Generalisation Accuracy

The training error results are given in Table 8.3. For most data sets a non-strict decrease in training error can be seen with an increase in the λ value. However, the Friedman test produced a p-value of 0.2227 showing that the decrease in training error is not significant across all data sets.

Since swarm divergence decreases with an increase in λ, the swarm becomes more capable of exploitation with steeper gradients. Therefore, it is expected that the training accuracy of the FFNNs would increase as λ increased, even if the improvement was insignificant. It is possible that larger λ values could significantly improve the training accuracy.

The generalisation results are given in Table 8.4. A Friedman test of the generalisation results yielded a p-value of 0.8983, indicating that the λ parameters and changes in swarm diversity did not have a significant impact on the generalisation accuracy of the FFNN. This is consistent with the findings of earlier chapters, where variation in factors affecting diversity, e.g. the swarm size, was also shown to have little effect on generalisation accuracy.

Summary

The results given here clearly show that the activation function gradient is and should be a significant consideration when using the PSO as a training algorithm for FFNNs. Appropriate gradients that lessen swarm divergence should be investigated on a per problem basis. Optimal gradients for the activation functions do not, however, guarantee convergence and has to be used in conjunction with appropriate choices for other control parameters, specifically $\mathbf{V}_{max}$. It is reasonable to assume that optimal values for $\mathbf{V}_{max}$ and other control parameters will also depend on the activation function gradients, though, such a study is left as future work. Further, the effect of the gradient on other types of FFNNs, such as product-unit neural networks [18], should also be investigated when using PSO as a training algorithm.

Table 8.3: Training error means for FFNNs using static sigmoid functions with indicated λ values.

Data Set	$\lambda = 0.1$	$\lambda = 0.3$	$\lambda = 0.5$	$\lambda = 0.7$	$\lambda = 0.9$	$\lambda = 1.0$	$\lambda = 1.1$	$\lambda = 1.3$	$\lambda = 1.5$	$\lambda = 1.7$	$\lambda = 1.9$
Diabetes	0.14760	0.14561	0.14401	0.14442	0.14421	0.14315	0.14385	0.14331	0.14231	0.14279	0.14230
Glass	0.59377	0.57300	0.57125	0.56360	0.55975	0.54763	0.54923	0.54838	0.54398	0.55149	0.54957
Iris	0.03011	0.02811	0.02968	0.02968	0.02965	0.02917	0.02860	0.02841	0.02980	0.02897	0.03002
WDBC	0.01814	0.01763	0.01724	0.01670	0.01663	0.01679	0.01797	0.01623	0.01785	0.01713	0.01679
Wine	0.00437	0.00371	0.00458	0.00405	0.00375	0.00480	0.00396	0.00427	0.00456	0.00371	0.00476
Henon Map	0.00078	0.00072	0.00069	0.00078	0.00069	0.00059	0.00065	0.00070	0.00067	0.00057	0.00068
Logistic Map	0.00079	0.00080	0.00076	0.00079	0.00074	0.00078	0.00076	0.00078	0.00083	0.00080	0.00077
Mackey-Glass	0.00063	0.00058	0.00060	0.00060	0.00062	0.00062	0.00060	0.00060	0.00059	0.00057	0.00060
Sunspot	136.832	141.074	140.61	138.902	139.750	136.973	138.031	137.991	135.322	135.708	137.323
TS5	0.00214	0.00219	0.00221	0.00223	0.00230	0.00232	0.00230	0.00219	0.00241	0.00228	0.00228
Friedman p-value: 0.2227											

Table 8.4: Generalisation error means for FFNNs using static sigmoid functions with indicated λ values.

Data Set	$\lambda = 0.1$	$\lambda = 0.3$	$\lambda = 0.5$	$\lambda = 0.7$	$\lambda = 0.9$	$\lambda = 1.0$	$\lambda = 1.1$	$\lambda = 1.3$	$\lambda = 1.5$	$\lambda = 1.7$	$\lambda = 1.9$
Diabetes	0.16065	0.15920	0.15978	0.16087	0.15958	0.15836	0.15893	0.15972	0.15992	0.15995	0.16019
Glass	0.91096	0.88109	0.90121	0.88126	0.86613	0.87905	0.88944	0.88882	0.88846	0.88018	0.85932
Iris	0.04735	0.03739	0.03649	0.03668	0.03905	0.03709	0.03678	0.03724	0.03851	0.03599	0.03741
WDBC	0.02936	0.02869	0.02806	0.02852	0.03011	0.02917	0.02925	0.02846	0.03061	0.02838	0.02798
Wine	0.04293	0.04667	0.04781	0.04400	0.04456	0.04828	0.04239	0.04876	0.04765	0.05511	0.04786
Henon Map	0.00935	0.01586	0.01382	0.01091	0.00905	0.01067	0.00916	0.01140	0.00943	0.01116	0.00776
Logistic Map	0.00129	0.00146	0.00128	0.00144	0.00135	0.00139	0.00144	0.00134	0.00140	0.00134	0.00136
Mackey-Glass	0.00132	0.00127	0.00137	0.00128	0.00125	0.00124	0.00131	0.00124	0.00128	0.00133	0.00131
Sunspot	235.892	224.827	239.413	232.798	225.713	236.703	226.920	223.639	227.579	222.131	229.918
TS5	0.00428	0.00455	0.00441	0.00432	0.00432	0.00452	0.00417	0.00442	0.00442	0.00442	0.00447
Friedman p-value: 0.8983											

8.4.2 Adaptive Activation Functions

The results for the adaptive activation function experiment are given in this section. As described in Section 8.3.4, the PSO *lambda-gamma* training technique was applied to both scaled and unscaled data. Both result sets are given in this section and compared with PSO training of standard FFNNs on scaled data.

Training Accuracy

The training error results are given in Table 8.5. On scaled data sets the PSO-LG algorithm performed comparable to the PSO algorithm in spite of the much greater complexity and dimensionality of the problem; for some data sets a lower training error was obtained by the PSO-LG algorithm. For all data sets, the algorithm obtained larger training errors when the data was *unscaled*. The result of the Friedman test, p = 0.007447, shows that a significant difference exists between the three approaches.

Table 8.5: Training error results for PSO *lambda-gamma* trained neural networks for scaled and unscaled data sets.

	PSO-LG *scaled*		**PSO-LG** *unscaled*		**PSO** *scaled*	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	0.14454	0.00512	0.19412	0.01041	0.14335	0.00618
Glass	0.56890	0.09440	0.81191	0.34170	0.55411	0.07292
Iris	0.01983	0.00761	0.02584	0.00721	0.01606	0.00521
WDBC	0.01505	0.00429	0.03992	0.01649	0.01674	0.00365
Wine	0.00049	0.00099	0.02951	0.01406	0.00108	0.00067
Henon Map	0.00703	0.02247	0.01870	0.04045	0.00067	0.00052
Logistic Map	0.00080	0.00021	0.00089	0.00020	0.00075	0.00015
Mackey-Glass	0.00063	0.00015	0.00070	0.00018	0.00060	0.00011
Sunspot	136.93798	16.41584	202.37518	164.09695	142.29804	16.38328
TS5	0.00227	0.00028	0.00223	0.00017	0.00233	0.00041
Friedman p-value: 0.007447						

Wilcoxon rank sum tests, shown in Table 8.6, indicated that training on *unscaled* data yielded significantly worse training errors compared with either PSO-LG or standard

PSO training on scaled data. No significant difference in training performance between the PSO-LG and standard PSO training was found on scaled data.

On the much harder problem of unscaled data, the PSO-LG still performed well. The algorithm was capable of successfully training the FFNN, but could not achieve the same degree of accuracy as with scaled data. However, no attempt was made to optimise the parameters of the algorithm for this task, nor was the algorithm allowed more execution time on the harder problem, and it is possible that training accuracy may be further improved if the networks are trained for longer.

Table 8.6: Wilcoxon rank sum test p-values for training error results of PSO-LG trained networks.

Algorithm	**PSO-LG** *scaled*	**PSO** *scaled*
PSO *scaled*	1.0000	-
PSO-LG *unscaled*	0.0081	00081

Generalisation Accuracy

Table 8.7 presents the generalisation results for the PSO-LG trained networks, again comparing against standard FFNNs on scaled data. Results similar to the training error are seen: the PSO-LG algorithm achieved similar accuracy to the PSO algorithm on *scaled* data. On *unscaled* data, the PSO-LG obtained higher generalisation errors than either of the other approaches. A Friedman test revealed that a significant difference exists between the three result sets, with a p-value of 0.003346.

The pairwise Wilcoxon rank sum tests, summarised in Table 8.8, indicate no significant difference in generalisation performance exists between the standard PSO and PSO-LG algorithms on scaled data, but significantly worse generalisation resulted from training on unscaled data.

Although the increased generalisation error could also, at least partially, be prescribed to the increased difficulty of the problem, better insight is gained when looking at the generalisation factor results given in Table 8.9. All the training methods obtained generalisation factors larger than one on all the data sets, regardless of scaling, indicative

Table 8.7: Generalisation error results for PSO *lambda-gamma* trained neural networks for scaled and unscaled data sets.

	PSO-LG *scaled*		**PSO-LG** *unscaled*		**PSO** *scaled*	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	0.16316	0.01151	0.2083	0.01136	0.15875	0.00928
Glass	0.88728	0.22642	1.02902	0.42767	0.85734	0.22522
Iris	0.03311	0.01456	0.03242	0.01257	0.0287	0.01291
WDBC	0.0326	0.01021	0.04979	0.01854	0.02777	0.00689
Wine	0.04229	0.0284	0.07557	0.03486	0.03299	0.01603
Henon Map	0.02973	0.05732	0.03755	0.06567	0.00603	0.00675
Logistic Map	0.00142	0.00062	0.00151	0.00076	0.00133	0.00054
Mackey-Glass	0.00138	0.00058	0.00153	0.00088	0.00131	0.00054
Sunspot	254.97422	51.77698	383.26526	186.5984	225.50895	42.37241
TS5	0.00449	0.00114	0.00438	0.00098	0.00453	0.00116
Friedman p-value: 0.003346						

of overfitting. However, the result of a Friedman test yielded a significant p-value of 0.005517, indicating that the overfitting differed significantly between training methods.

The PSO-LG algorithm on unscaled data obtained the lowest generalisation factor per data set, indicating the least severe overfitting. The results of the pairwise Wilcoxon tests, given in Table 8.10, show that the standard PSO achieved insignificantly larger generalisation factors compared with training on unscaled data.

Significantly worse overfitting occurred for the PSO-LG training method on scaled data, especially on the Iris, Wine and Henon Map data sets.

From a theoretical perspective, the λ and γ parameters are additional free parameters in the FFNN model, similar to network weights. Although this makes the network a more powerful approximator, the additional free parameters can also be used to overfit on the training data given enough training time to do so, similar to when a network has too many hidden units

From the training results it is clear that the PSO-LG algorithm is very capable of training FFNNs augmented with λ and γ parameters. However, from the results given

Table 8.8: Wilcoxon rank sum test p-values for generalisation error results of PSO-LG trained networks.

Algorithm	**PSO-LG** *scaled*	**PSO** *scaled*
PSO *scaled*	1.0000	-
PSO-LG *unscaled*	0.0038	0.011

Table 8.9: Generalisation factor results for PSO *lambda-gamma* trained neural networks for scaled and unscaled data sets.

	PSO-LG *scaled*		**PSO-LG** *unscaled*		**PSO** *scaled*	
Data Set	μ	σ	μ	σ	μ	σ
Diabetes	1.13188	0.10941	1.07556	0.07519	1.11162	0.10724
Glass	1.6656	0.73292	1.34103	0.50048	1.62497	0.65538
Iris	5.20826	16.79141	1.35296	0.66201	2.24302	1.72054
WDBC	2.4965	1.49538	1.33351	0.45612	1.75793	0.66297
Wine	1638.20816	5150.65671	3.19061	1.88684	56.56108	71.60725
Henon Map	11.28004	17.16726	8.47548	9.1472	11.63785	12.39576
Logistic Map	2.03988	1.22738	1.88118	1.16858	1.94669	1.07973
Mackey-Glass	2.49672	1.54969	2.44093	1.59815	2.36546	1.24339
Sunspot	1.91989	0.59257	2.17027	0.75092	1.63221	0.48084
TS5	2.00486	0.5568	1.98663	0.5166	1.97498	0.5298
Friedman p-value: 0.005517						

Table 8.10: Wilcoxon rank sum test p-values for generalisation factor results of PSO-LG trained networks.

Algorithm	**PSO-LG** *scaled*	**PSO** *scaled*
PSO *scaled*	0.056	-
PSO-LG *unscaled*	0.029	0.196

here, it seems that the PSO-LG algorithm is indeed using the increased capability of the FFNN to more accurately model the training data, at the cost of generalisation when training on scaled data.

However, the same degree of overfitting was not witnessed on unscaled data. Training on unscaled data is however a much harder problem for the network to approximate. With unscaled data the additional parameters, especially γ, have to be used to adapt to the data set, specifically, the bounds of the target outputs. The more powerful lambda-gamma augmented network can be seen as more appropriate to the problem of unscaled data, and therefore overfitting is reduced.

Summary

The PSO-LG algorithm has been shown as a capable training algorithm for the *lambda-gamma* parametrised FFNNs, with performance that does not differ significantly to that of PSO trained FFNNs on scaled data. The PSO-LG was also successful in training FFNNs on unscaled data at the cost of accuracy. There remains, however, a number of aspects that may still improve performance: optimisation of algorithm parameters, the use of other activation functions, more sophisticated initialisation strategies of the λ and γ parameters and the handling of constraints imposed onto the parameters, to name but a few. These topics are left as future work.

8.5 Summary

This chapter presented two experiments investigating adaptation of the activation function of FFNNs and the effect thereof on the FFNN and PSO as training algorithm.

Using fixed adaptations to the λ parameter of the sigmoid function, Section 8.4.1 further investigated the hypothesis that the asymptotic bounds of the activation functions lead to divergence of the particle swarm as stated in Chapter 7. However, the results showed that the activation function *gradient* has a significant effect on swarm convergence. This effect is more pronounced at the bounds of the function where the gradient is shallowest. A theoretical analysis of the effect was given.

Self-adapting activation functions, in the form of *lambda-gamma* adjusted sigmoid

functions, were investigated in Section 8.4.2. The PSO Lambda-Gamma algorithm for the training of *lambda-gamma* parametrised FFNNs was described. The PSO-LG algorithm was then compared with PSO training of standard FFNNs on both *scaled* and *unscaled* data. The PSO-LG algorithm was shown to be a capable training algorithm on both *scaled* and *unscaled* data sets. Significant overfitting was shown to occur with the PSO-LG algorithm on scaled data. However, on unscaled data, the overfitting was reduced. The dissertation is concluded in the next chapter.

Chapter 9

Conclusions

This chapter presents a summary of the work done in this thesis including all important findings and derivations. A number of possible future research avenues are also discussed.

Section 9.1 summarises the conclusions reached throughout the study and Section 9.2 provides topics for future research.

9.1 Summary of Conclusions

This section summarises the major findings made during the pursuit of the primary objective of this thesis: an empirical analysis of the effect of the PSO algorithm and its parameters on FFNN accuracy and overfitting.

NNs and in particular FFNNs were discussed with background being provided into their training. The phenomenon of overfitting was reviewed along with existing methods that exist to attempt to prevent overfitting during training. A review of the PSO algorithm, its components and parameters was presented. A discussion of existing literature on PSO FFNN training was also given.

An empirical comparison, in terms of FFNN accuracy, of two widely publicised PSO algorithms, the *lbest* PSO and the GCPSO, was given. A discussion of overfitting that occurred for both PSO algorithms was given and it was shown that overfitting occurred early in the optimisation process, no later than 200 iterations with a swarm size of 25. Empirical analysis showed no significant difference in terms of FFNN performance

and overfitting between the PSO and GCPSO trained FFNNs. Analysis of the diversity results also showed severe swarm divergence occurring during optimisation of the FFNNs.

The $\mathbf{V}_{max}$ parameter and its effect on FFNN training and the observed swarm divergence was analysed. It was found that a lack of the $\mathbf{V}_{max}$ parameter was not the cause of divergence, but that swarm convergence may be aided with the use of a $\mathbf{V}_{max} < 1$ for the data sets used in this study. It was further shown that a smaller $\mathbf{V}_{max}$ value benefited the training accuracy of the networks, likely due to the decreased swarm divergence. Generalisation accuracy and overfitting was mostly unaffected by the choice of $\mathbf{V}_{max}$ parameters.

An analysis of the *inertia* and *social and cognitive acceleration coefficients* was given. Beneficial and adverse ranges were identified for each parameter that improved FFNN accuracy. It was shown that, although specific parameter values could be identified that resulted in both good FFNN training and generalisation accuracy and reduced overfitting, that these values were specific to each data set. Additionally, specific parameter interactions were identified that should be avoided when training FFNN due to the adverse effect on accuracy.

It was also observed that the stochastic elements of the PSO algorithm also had a significant effect on FFNN training, with certain samples performing significantly different from each other under the same conditions and parameters.

The swarm size and its effect was investigated. It was found that, under the condition of a constant number of objective function evaluations, larger swarm sizes resulted in significantly decreased training accuracy while generalisation accuracy and overfitting were unaffected. Swarm divergence shown in previous chapters was shown to worsen as swarm size increased and it was concluded that optimal values for $\mathbf{V}_{max}$ are specific to the swarm size used.

Due to the divergence shown to occur during FFNN training for all parameters tested, it was hypothesised that the FFNN activation function, which determines the optimisation search space, might be the indirect cause of the divergence. An analysis was given of three activation functions: the sigmoid, hyperbolic tangent and linear functions. It was found that the non-linear functions outperformed the linear function in terms of training accuracy. No significant difference was found in terms of generalisation accuracy due to

overfitting that was shown to occur for the non-linear functions.

It was further shown that swarm divergence only occurred for the sigmoid and hyperbolic tangent functions and it was hypothesised that this is due to the asymptotic bounds of the functions. This hypothesis was tested using adaptations to the sigmoid function that changed the nature of the function. It was found that instead of the asymptotic bounds, the *gradient* of the function significantly affects swarm convergence during FFNN training. Steeper gradients was shown to lead to convergent swarms and lower training errors on some data sets.

Finally, due to the findings that specific activation function gradients improved FFNN training, an investigation was made into self-adaptive activation functions using *lambda-gamma* adjusted sigmoid functions. The PSO Lambda-Gamma (PSO-LG) algorithm was given. The PGO-LG algorithm was shown to overfit significantly on *scaled* data sets, however, the algorithm was also shown to be capable of training FFNNs on completely *unscaled* data with reduced training accuracy but improved overfitting.

9.2 Future Work

This section presents and discusses a number of possible future research topics.

Constrained Optimisation

An alternative to adaptive activation functions is to add constraints to the optimisation problem in order to avoid regions outside the active domain of the activation function. A number of techniques have been developed to enhance the PSO to handle constrained optimisation problems [24, 48, 76, 109, 110]. In particular, the method described by Venter and Sobieszczanski-Sobieski aims to repair infeasible particles by resetting the particles' velocities to $\mathbf{0}$ and modifying the velocity update equation for those particle to exclude the momentum term while the position remains infeasible. In this way, infeasible particles are drawn back to a feasible region. This approach seems especially well suited to the bounded activation function problem.

Investigating Swarm Size

The findings in this study have shown that the optimal value of the $\mathbf{V}_{max}$ parameter is specific to the swarm size used. Investigation is required into the correlation between $\mathbf{V}_{max}$ and swarm size. Further, it was shown that larger swarms performed poorly against smaller swarms under the restriction of constant computational effort. Investigation should be done on the use of larger swarms at an increase of computational effort and whether this approach improves training and worsens overfitting.

Other Particle Swarm Optimisers

Although a comparison was made to the GCPSO, the effect on FFNN training of other PSO algorithms may also be investigated. Cooperative PSO algorithms, which have been successfully used to train FFNNs [103, 104], is one such example. Another notable candidate is the Constriction PSO, a PSO model which includes a *constriction coefficient* as an alternative to the use of a $\mathbf{V}_{max}$ parameter [14, 15].

Other Population Based Algorithms

Some aspects of the PSO, notably the size of the population and the restriction of candidate solutions also apply to other population and evolutionary based algorithms for NN training. Investigation is required into whether the findings of this study, primarily the swarm size and the stochastic nature of the algorithm extends to FFNN training by other algorithms.

Extension of Adaptive Neural Networks

The PSO Lambda-Gamma algorithm has shown promise as a training algorithm in this study on both scaled and unscaled data sets. Additional investigation is required on potentially improving the performance of the PSO-LG algorithm: optimisation of the control parameters, alternative initialisation strategies for *lamba* and *gamma* parameters and constrained optimisation are but a few possibilities.

Other Neural Networks

Overfitting and accuracy analysis of PSO training of other NNs, for example product unit NNs [52, 53, 105] or convolution NNs [66], require further investigation.

Bibliography

[1] A. Asuncion and D. J. Newman. UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007. Accessed 25/5/2010 13:18 SAST.

[2] R. Battiti. First-and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

[3] L. M. Belue, K. W. Bauer, and Others. Determining input features for multilayer perceptrons. *Neurocomputing*, 7(2):111–121, 1995.

[4] D. P. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming: An overview. In *AIChE Symposium Series*, pages 92–96. Citeseer, 2002.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[8] S. Bös. Optimal Weight Decay in a Perceptron. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 551 – 556. Springer-Verlag, London, UK, 1996.

[9] R. Brits, A. P. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning*, pages 692–696. Citeseer, 2002.

[10] X. Cai, N. Zhang, G. K. Venayagamoorthy, and D. C. Wunsch. Time series prediction with recurrent neural networks using a hybrid PSO-EA algorithm. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 2, pages 1647—-1652 vol.2, 2004.

[11] M. Carvalho and T. B. Ludermir. An analysis of PSO hybrid algorithms for feed-forward neural networks training. In *Ninth Brazilian Symposium on Neural Networks*, pages 6–11, 2006.

[12] M. Carvalho and T. B. Ludermir. Particle swarm optimization of feed-forward neural networks with weight decay. In *Sixth International Conference on Hybrid Intelligent Systems*, page 5, 2006.

[13] Christopher W Cleghorn and Andries P Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm Intelligence*, 8(1):35–59, 2014.

[14] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 3. IEEE, 1999.

[15] M. Clerc and J. Kennedy. The Particle Swarm Explosion , Stability , and Convergence in a Multidimensional Complex Space. *Convergence*, 6(1):58–73, 2002.

[16] A. K. Connect, A. Krogh, and J. A. Hertz. A Simple Weight Decay Can Improve Generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1992.

[17] G. Dreyfus. *Neural networks: methodology and applications*. Springer, 2005.

[18] R. Durbin and D. E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142, 1989.

[19] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[20] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 7, pages 84–88, 2000.

[21] R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 81–86. Piscataway, NJ, USA: IEEE, 2001.

[22] R. C. Eberhart, P. K. Simpson, and R. W. Dobbins. *Computational Intelligence PC Tools*. Academic Press Professional, 1996.

[23] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks–a review. *Pattern Recognition*, 35(10):2279–2301, 2002.

[24] A. El-Gallad, M. El-Hawary, A. Sallam, and A. Kalas. Enhancing the particle swarm optimizer via proper parameters selection. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 792–797. IEEE, 2002.

[25] A. P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[26] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, second edition, 2007.

[27] A. P. Engelbrecht. Particle swarm optimization: Velocity initialization. In *Proceedings of the Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[28] A. P. Engelbrecht, I. Cloete, J. Geldenhuys, and J. M. Zurada. Automatic scaling using gamma learning for feedforward neural networks. In *Proceedings of the International Workshop on Artificial Neural Networks*, pages 374–381. Springer-Verlag, 1995.

[29] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, 2:532, 1990.

[30] H. Fan. A modification to particle swarm optimization algorithm. *Engineering Computations*, 19(8):970, 2002.

[31] L. J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming*. 1999.

[32] N. Franken. Visual exploration of algorithm parameter space. In *Proceedings of the Congress on Evolutionary Computation*, pages 389–398. IEEE, 2009.

[33] N. Franken and A. P. Engelbrecht. Comparing PSO structures to learn the game of checkers from zero knowledge. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 234—-241, 2003.

[34] B. Fritzke. Incremental learning of local linear mappings. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 95, pages 217–222. Citeseer, 1995.

[35] J. E. Gentle. *Random number generation and Monte Carlo methods*. Springer Verlag, 2003.

[36] F. Girosi, M. Jones, and T. Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7:219–269, 1995.

[37] E. A. Grimaldi, F. Grimaccia, M. Mussetta, and R. E. Zich. PSO as an effective learning algorithm for neural network applications. In *Proceedings of the 3rd International Conference on Computational Electromagnetics and Its Applications*, pages 557–560, 2004.

[38] S. J. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in Neural Information Processing Systems*, 1:177–185, 1989.

[39] M. H. Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[40] D. M. Hawkins. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, December 2003.

[41] S. Haykin. *Neural Networks: A Comprehensive Foundation, 1st edition.* 1994.

[42] T. Hendtlass and M. Randall. A survey of ant colony and particle swarm meta-heuristics and their application to discrete optimization problems. In *Proceedings of the Inaugural Workshop on Artificial Life*, pages 15–25, 2001.

[43] G. Hinton. Learning translation invariant recognition in a massively parallel networks. In *PARLE Parallel Architectures and Languages Europe*, pages 1–13. Springer, 1987.

[44] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4(1):61–66, 1991.

[45] E. Hjelmå s and B. K. Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236–274, 2001.

[46] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[47] K. Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[48] X. Hu and R. C. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proceedings of the sixth world multiconference on systemics, cybernetics and informatics*, volume 5, pages 203–206. Citeseer, 2002.

[49] C. M. Huang and F. L. Wang. An RBF network with OLS and EPSO algorithms for real-time power dispatch. *IEEE Transactions on Power Systems*, 22(1):96–104, 2007.

[50] H. Hüning. A node splitting algorithm that reduces the number of connections in a hamming distance classifying network. In *Proceedings of new trends in neural computation: International Workshop on Artificial Neural Networks*, page 102. Springer, 1993.

[51] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proceedings of the First IEEE Conference on Visualization*, pages 361–378, 1990.

[52] A. Ismail and A. P. Engelbrecht. Global optimization algorithms for training product unit neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 132–137 vol.1, 2000.

[53] A. Ismail and A. P. Engelbrecht. Pruning product unit neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 257–262, 2002.

[54] S. Joe and F. Y. Kuo. Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 29(1):49–57, 2003.

[55] C. Juang. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics.*, 34(2):997–1006, 2004.

[56] C. Juang and Y. Liou. On the hybrid of genetic algorithm and particle swarm optimization for evolving recurrent neural network. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 3, pages 2285—-2289 vol.3, 2004.

[57] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[58] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, 1995.

[59] J. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[60] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1671–1676, 2002.

[61] T. Krink, J. S. VesterstrOm, and J. Riget. Particle swarm optimisation with spatial particle extension. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1474–1479. IEEE, 2002.

[62] I. Kusçu and C. Thornton. Design of artificial neural networks using genetic algorithms: Review and prospect. *Cognitive and Computing Sciences, University of Sussex*, 1994.

[63] T. Y. Kwok and D. Y. Yeung. Constructive feedforward neural networks for regression problems: A survey. *Department of Computer Science, Hong Kong University of Science and Technology, Technical Report*, 1995.

[64] S. Lawrence and C. L. Giles. Overfitting and Neural Networks: Conjugate Gradient and Backpropagation. In *International Joint Conference on Neural Networks*, volume 1, page 1114, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[65] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[66] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361, 1995.

[67] L. Leerink, C. L. Giles, B. G. Horne, and M. A. Jabri. Learning with product units. *Advances in Neural Information Processing Systems*, pages 537–544, 1995.

[68] C. L. Lin, S. T. Hsieh, T. Y. Sun, and C. C. Liu. Cluster distance factor searching by particle swarm optimization for self-growing radial basis function neural network. In *International Joint Conference on Neural Networks*, pages 4825–4830, 2006.

[69] H. B. Liu, Y. Y. Tang, J. Meng, and Y. Ji. Neural networks learning using vbest model particle swarm optimisation. In *Proceedings of International Conference on Machine Learning and Cybernetics*, volume 5, 2004.

[70] K. M. Malan and A. P. Engelbrecht. Algorithm comparisons and the significance of population size. In *Proceedings of the Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 914–920, 2008.

[71] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

[72] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural network training. *In Proceedings of the International Joint Conference on Neural Networks*, pages 1895–1899, 2002.

[73] S. Naka, T. Genji, T. Yura, and Y. Fukuyama. Practical distribution state estimation using hybrid particle swarmoptimization. In *IEEE Power Engineering Society Winter Meeting*, volume 2, 2001.

[74] National Geophysical Data Center. NGDC/WDC STP, Boulder-Sunspot Number Data via FTP from NGDC. http://www.ngdc.noaa.gov/stp/SOLAR/ftpsunspotnumber.html, November 2009. Accessed 4/11/2009 18:06 SAST.

[75] O. Olorunda and A. P. Engelbrecht. Measuring exploration/exploitation in particle swarms using swarm diversity. In *Proceedings of the Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1128–1134, 2008.

[76] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies–Theory and Application: New Trends in Intelligent Technologies*, 76:214–220, 2002.

[77] D. W. Patterson. *Artificial Neural Networks: Theory and Applications*. Prentice-Hall Series in Advanced Communications. Prentice Hall, 1996.

[78] E. S. Peer, F. van den Bergh, and A. P. Engelbrecht. Using neighbourhoods with the guaranteed convergence PSO. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 235–242, 2003.

[79] J. Platt. A Resource-Allocating Network for Function Interpolation. *Neural Computation*, 3(2):213–225, June 1991.

[80] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008.

[81] L. Prechelt. Adaptive parameter pruning in neural networks. *International Computer Science Institute, Technical Report*, 1995.

[82] A. Ratnaweera, S. K. Halgamuge, and H. Watson. Particle swarm optimization with self-adaptive acceleration coefficients. In *Proceedings of the First International Conference on Fuzzy Systems and Knowledge Discovery*, pages 264–268, 2003.

[83] R. Reed. Pruning algorithms: a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[84] Z. Ren and Y. San. Designing for rbf networks based on particle swarm optimization and regularized orthogonal least squares. In *The Sixth World Congress on Intelligent Control and Automation*, volume 1, 2006.

[85] H. W. Ressom, Y. Zhang, J. Xuan, Y. Wang, and R. Clarke. Inferring network interactions using recurrent neural networks and swarm intelligence. In *Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 1, page 4241, 2006.

[86] M. Riedmiller. Supervised learning in multilayer perceptrons-from backpropagation to adaptive learning techniques. *Computer Standards and Interfaces*, 16, 1994.

[87] J. Riget and J. S. Vesterstrø m. A diversity-guided particle swarm optimizer-the ARPSO. *Department of Computer Science, University of Aarhus, Aarhus, Denmark, Techical Report*, 2:2002, 2002.

[88] A. Röbel. The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks. Technical report, Technische Universität Berlin, 1994.

[89] F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

[90] J. Salerno. Using the particle swarm optimization technique to train a recurrent neural model. *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence*, pages 45–49, 1997.

[91] R. Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278, 1996.

[92] W. S. Sarle. Stopped Training and Other Remedies for Overfitting. In *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pages 352–360, 1995.

[93] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 69–73, May 1998.

[94] J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991.

[95] J. A. Snyman. A new and dynamic method for unconstrained minimization. *Applied mathematical modelling*, 6(6):449–462, 1982.

[96] J. A. Snyman. An improved version of the original leap-frog dynamic method for unconstrained minimization: LFOP1 (b). *Applied mathematical modelling*, 7(3):216–218, 1983.

[97] I. M. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[98] I. M. Sobol and Y. L. Levitan. The production of points uniformly distributed in a multidimensional cube. *Preprint IPM Akad. Nauk SSSR*, (40), 1976.

[99] J. M. Steppe, K. W. Bauer Jr., and S. K. Rogers. Integrated feature architecture selection. *IEEE Transactions on Neural Networks*, 7(4):1007–1014, 1996.

[100] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, 1999.

[101] H. H. Thodberg. Improving generalization of neural networks through pruning. *International Journal of Neural Systems*, 1(4):317–326, 1991.

[102] I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.

[103] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[104] F. van den Bergh and A. P. Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *South African Computer Journal*, 26:84–90, 2000.

[105] F. van den Bergh and A. P. Engelbrecht. Training product unit networks using cooperative particle swarm optimisers. In *International Joint Conference on Neural Networks*, volume 1, pages 126–131, 2001.

[106] F. van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimiser. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, page 6 pp. vol.3, 2002.

[107] F. van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[108] A. B. van Wyk and A. P. Engelbrecht. Lambda-gamma learning with feedforward neural networks using particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 1–8, 2011.

[109] G. Venter and J. Sobieszczanski-Sobieski. Particle swarm optimization. *AIAA journal*, 41(8):1583–1589, 2003.

[110] G. Venter and J. Sobieszczanski-Sobieski. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. *Structural and Multidisciplinary Optimization*, 26(1-2):121–131, 2004.

[111] E. J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990.

[112] P. J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Unpublished doctoral dissertation, Harvard University.*, page 33, 1974.

[113] L. F. A. Wessels and E. Barnard. Avoiding false local minima by proper initialization of connections. *IEEE Transactions on Neural Networks*, 3(6):899–905, 1992.

[114] D. White and P. Ligomenides. GANNet: A genetic algorithm for optimizing topology and weights in neural network design. In *New Trends in Neural Computation*, pages 322–327. Springer, 1993.

[115] P. M. Williams. Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, 7(1), 1995.

[116] C. Zhang, H. Shao, and Y. Li. A new evolved artificial neural network and its application. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation*, volume 2, 2000.

[117] C. Zhang, H. Shao, and Y. Li. Particle swarm optimisation for evolving artificial neural network. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2487—-2490 vol.4, 2000.

[118] J. Zhang, T. Lok, and M. R. Lyu. A hybrid particle swarm optimization-backpropagation algorithm for feedforward neural network training. *Applied Mathematics and Computation*, 185(2):1026–1037, February 2007.

[119] F. Zhao, Z. Ren, D. Yu, and Y. Yang. Application of An Improved Particle Swarm Optimization Algorithm for Neural Network Training. In *International Conference on Neural Networks and Brain*, volume 3, 2005.

[120] J. M. Zurada. *Introduction to Artificial Neural Systems*. West Engineering Series. PWS Publishing Company, 1992.

[121] J. M. Zurada. Lambda learning rule for feedforward neural networks. In *IEEE International Conference on Neural Networks*, pages 1808–1811, 1993.

Appendix A

Acronyms

A list of acronyms used throughout the text is given in this appendix.

ANN	Artificial Neural Network
BP	Backpropagation
CG	Conjugate Gradient
CI	Computational Intelligence
EA	Evolutionary Algorithm
EP	Evolutionary Programming
FFNN	Feedforward Neural Network
GA	Genetic Algorithm
GCPSO	Guaranteed Convergence Particle Swarm Optimisation
GD	Gradient Descent Backpropagation Training Algorithm
LDS	Low-Discrepancy Sequence
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error

NN Neural Network

PRNG pseudo-random number generator

PSO Particle Swarm Optimisation

Appendix B

Symbols

This appendix lists the mathematical symbols used throughout this thesis accompanied by their definitions. Each section lists the symbols used per chapter, with only newly introduced symbols for the chapter being shown. Symbols in bold text indicate vectors.

B.1 Chapter 2: Background

D	A data set.
D_T	A training data set.
D_G	A generalisation data set.
p	A data set pattern.
$\mathbf{t}_p$	Pattern target vector.
$\mathbf{z}$	Neural network input layer.
I	Neural network input layer size.
J	Neural network hidden layer size.
K	Neural network output layer size.
$\mathbf{y}_p$	Hidden layer activation vector for pattern p.
$\mathbf{o}_p$	Output layer activation vector for pattern p.
K	Neural network output layer size.
P	Data set pattern size.

P_T	Training data set pattern size.
P_G	Generalisation data set pattern size.
$\mathbf{t}_p$	Target vector for pattern p.
$f(net)$	Neuron activation function.
$f_{sig}(net)$	Sigmoid activation function.
net	Input signal for a neuron activation function.
λ	Gradient parameter for the sigmoid function.
$\mathcal{E}$	Neural network error.
$\mathcal{E}_T$	Neural network training error.
$\mathcal{E}_G$	Neural network generalisation error.
$\mathbf{w}$	Neuron weight vector.
$fanin$	Number of incoming weights for a neuron.
ρ_F	Röbel generalisation factor.
MSE_T	Mean squared training error.
MSE_G	Mean squared generalisation error.
Ω	Regularisation penalty term.
GL_α	Generalisation loss criterion.
$\mathbf{x}$	Particle position.
$\mathbf{v}$	Particle velocity.
$\mathbf{y}$	Particle personal best position.
$\hat{\mathbf{y}}$	Particle neighbourhood best position.
ω	PSO inertia parameter.
c_1	PSO cognitive acceleration coefficient.
c_2	PSO social acceleration coefficient.
r	Stochastic variable for particle velocity calculation.
n_s	PSO swarm size.
$\mathcal{D}_{avg}$	Average distance around swarm centre measurement.
ρ	GCPSO hypercube search size control parameter.

s_c	GCPSO hypercube search success count.
f_c	GCPSO hypercube search failure count.

B.2 Chapter 3: Comparing PSO and GCPSO

n	PSO search space dimensionality.
E_T	Training error measurement.
E_G	Generalisation error measurement.

B.3 Chapter 7: Considering Activation Functions

γ	Sigmoid range parameter.
$f_{tanh}(net)$	Hyperbolic tangent activation function.
$f_{lin}(net)$	Linear activation function.

B.4 Chapter 8: Adaptive Activation Functions

$\mathbf{x}^*$	Search space global optimum.

Appendix C

Derived Publications

A list of derived publications is given in this appendix.

- Andrich B van Wyk and A. P. Engelbrecht. Overfitting by PSO trained feedforward neural networks. In Evolutionary Computation (CEC), 2010 IEEE Congress on (pp. 18). doi:10.1109/CEC.2010.5586333
- Andrich B van Wyk and A. P. Engelbrecht. Lambda-gamma learning with feedforward neural networks using particle swarm optimization. In Swarm Intelligence (SIS), 2011 IEEE Symposium on, pages 18, 2011.