IMPROVING THE EFFICIENCY OF FUNCTION MAPPING NEURAL NETWORKS USING HYBRID TRAINING METHODOLOGIES

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2015

By
Jonathon Dilworth
Supervisor: Dr. Richard Neville
School of Computer Science

Contents

Abstract **9**

Declaration **10**

Copyright **11**

Acknowledgements **12**

1 Introduction **13**

1.1 Basic Introduction to the Research Domain 13

1.2 Aims of the Research Project 13

1.3 Research Context 14

1.4 Research Questions 14

1.5 Deliverables 15

1.6 Report Organisation 16

1.7 Concluding Remarks 16

2 Background and Literature Review **17**

2.1 An Introduction to Artificial Intelligence 17

2.2 An Introduction to Machine Learning 18

2.3 An introduction to Evolutionary Computing 18

2.4 Basic Concepts 19

2.4.1 Biological Neural Networks 19

2.4.2 Artificial Neural Networks 20

2.5 Mathematical Models 20

2.5.1 An Artificial Neuron 20

2.5.2 Artificial Neural Networks 22

2.6 Learning Paradigms 23

2.6.1 The Perceptron Learning Rule . . . 23
2.6.2 The Delta Rule . . . 23
2.6.3 Backpropagation and Multi-Layer Nets . . . 24
2.6.4 Evolutionary Algorithms . . . 25
2.7 Challenges to do with ANNs and EAs . . . 26
2.7.1 Problems with Back-propagation (BP) . . . 26
2.7.2 Problems with Evolutionary Algorithms (EAs) . . . 26
2.7.3 Model Design, Implementation and Validation . . . 26
2.8 Concluding Remarks . . . 27

3 Research Experiment Design **28**
3.1 Project Structure . . . 28
3.1.1 Workflow . . . 30
3.1.2 Routines . . . 30
3.1.3 Routine Platform Classification . . . 31
3.1.4 Advantages of Adopting a Generic Workflow Design . . . 31
3.2 Project Directory Structure . . . 32
3.3 Project Analysis Requirement . . . 33
3.4 Generic Experimental Methodology . . . 33
3.5 Experiment One Design . . . 34
3.6 Experiment Two Design . . . 35
3.7 Experiment Three Design . . . 35
3.8 Generic Experiment Functionality . . . 37
3.9 Concluding Remarks . . . 37

4 Experiment Implementation **38**
4.1 Development Background . . . 38
4.1.1 Development Environment . . . 38
4.1.2 Technologies Used in Implementation . . . 39
4.2 Experimental Results Collection . . . 40
4.2.1 Implementation of Training and Testing Platform . . . 40
4.3 Training Data Representation . . . 42
4.4 Representation of Initialisation Parameters . . . 42
4.5 Algorithms: Training and Testing platform . . . 43
4.5.1 Network Initialisation . . . 43
4.5.2 Feed Forward Propagation . . . 43

4.5.3 Training Algorithms . . . 44
4.5.3.1 The Evolutionary Algorithm . . . 44
4.5.3.2 Back-propagation . . . 45
4.5.4 Control Mechanism . . . 46
4.6 Implementation of the Evaluation Platform . . . 46
4.7 Parameter Settings for each Experiment . . . 47
4.7.1 Experiment One . . . 47
4.7.2 Experiment Two . . . 48
4.7.3 Experiment Three . . . 48
4.8 Concluding Remarks . . . 49

5 Critical Analysis and Evaluation **50**
5.1 Evaluation Methods . . . 50
5.1.1 Evaluating via the Results Table . . . 51
5.1.2 Results Visualisation . . . 51
5.2 Experiments . . . 52
5.2.1 Experiment One: Training with an EA . . . 52
5.2.1.1 Results Analysis . . . 52
5.2.2 Experiment Two: Training with BP . . . 53
5.2.2.1 Results Analysis . . . 54
5.2.3 Experiment Three: Training with HYB . . . 54
5.2.3.1 Results Analysis . . . 55
5.3 Evaluation Conclusion . . . 56
5.4 Concluding Remarks . . . 56

6 Conclusion and Future Work **57**
6.1 Development of Ideas . . . 57
6.2 Summary of Achievements . . . 58
6.3 Critical Analysis and Evaluation . . . 58
6.4 Identification of Improvements and Further Work . . . 58
6.4.1 Improvements . . . 58
6.4.2 Further Work . . . 59
6.5 Reflection . . . 59
6.6 Concluding Remarks . . . 60

Bibliography **61**

A Additional Introduction Information **64**
A.1 Research Relevance Table 64
A.2 Theory Research Relevance Tree 65

B Additional Background and Literature **66**
B.1 Definitions of Intelligence and The Turing Test 66
B.2 Machine Learning, Applications and Paradigms 67
B.3 The History of Evolutionary Computing 68
B.4 Neural Networks: The Periphery 68
B.5 The History of ANNs 68
B.6 Further Representation of ANNs 69
B.7 The Perceptron Rule: Further Depth 70
B.8 The Delta Rule: Derivation 70

C Additional Design Details **73**
C.1 The Evaluation Platform: Sub-Routines 73
C.2 Further Reading on Project Analysis 74
C.2.1 Construction of the Training and Testing Platform 74
C.2.2 Implementation of the Learning Algorithms 74
C.2.3 Implementation of the Control Mechanism 75
C.2.4 Construction of the Evaluation Platform 75
C.2.5 Summary 76
C.3 Project Analysis Requirements Table 77
C.4 Generic Experiment Design 78
C.5 Generic Experiment Data Structures and Algorithms 79
C.6 Experiment Functionality Workflow Table 80
C.7 Experiment Platforms Table 81

D Additional Implementation Details **84**
D.1 Generic Experiment Implementation Methodology 84
D.2 Data Structures and Algorithms Overview 84
D.3 Implementation Specifics for Neurons and ANNs 85
D.4 Representation of Training Examples 91
D.4.1 Representation of TEs: Implementation Specifics 93
D.5 Representation of Initialisation Parameters 95
D.6 Individual Data Type 97

D.7 EA: Algorithmic Parameters and Return Values 98
D.8 EA: Behaviour of High Level Functions 99
D.9 BP: Parameters 100
D.10 Experiment One: Parameters Table 101
D.11 Experiment Two: Parameters Table 102
D.12 Experiment Three: Parameters Table 103

E Additional Analysis Details **104**
E.1 Sensitivity Analysis for Experiment Two 104

List of Tables

4.1 Initialisation parameters associated with experiment one. 48
4.2 Initialisation parameters associated with experiment two. 48
4.3 Initialisation parameters associated with experiment three. 49

A.1 Research Relevance Table 64

C.1 Project Analysis Requirement Table 78
C.2 Generic Functionality of Experiments. 80
C.3 Routine description for the Training and Testing Platform 82
C.4 Routine description for the Evaluating Platform 83

D.1 Describes the data attributes that belong to each data structure. 87
D.2 Outlines each function associated with its respective data structure. . . 88
D.3 Outlines each function associated with its respective data structure. . . 91
D.4 Describes the data attributes that are associated with representing an training data. 93
D.5 Outlines each function associated with its respective data structure. . . 94
D.6 Defines and provides a description for each parameter contained within the parameters data structure. 97
D.7 Provides an overview of each parameter accepted by the implemented evolutionary algorithm. 98
D.8 Illustrates the behaviour of each high level function adopted by the implemented evolutionary algorithm. 99
D.9 Provides an overview of each parameter accepted by the implemented back-propagation algorithm. 100
D.10 Initialisation parameters associated with experiment one. 101
D.11 Initialisation parameters associated with experiment two. 102
D.12 Initialisation parameters associated with experiment one. 103

List of Figures

2.1 Depicts and labels a biological neuron. 19
2.2 Illustrates a mathematical model of a biological neuron. 21

3.1 The generic workflow design implementation for the research project. . 29
3.2 Structure of the evaluating platform (#RT04) 31
3.3 Directory structure of the project. 32
3.4 A flowchart that illustrates experiment one. 34
3.5 A flowchart that illustrates experiment two. 35
3.6 A flowchart that illustrates experiment three. 36

4.1 Visual implementation of an ANN as a 2-d array. 41
4.2 Illustrates the method by which any nodes adjacency list may be accessed. 41

5.1 A visual illustration of the result set produced by experiment one. 53
5.2 A visual illustration of the result set produced by experiment two. 54
5.3 A visual illustration of the result set produced by experiment three. . . . 55

A.1 Illustrates the Theory Research Relevance Tree. 65

B.1 An intuitive depiction of an ANN. 69

D.1 Illustrates the generic data structure used for each experiment. 92
D.2 Demonstrates that the training data can be conceptualised using two design matrices. 92

E.1 Illustrates the performance of an artificial neural network as it is being trained, using values for prime varying from 0.05 to 0.5. 104

Abstract

IMPROVING THE EFFICIENCY OF FUNCTION MAPPING NEURAL NETWORKS USING HYBRID TRAINING METHODOLOGIES
Jonathon Dilworth
A report submitted to the University of Manchester
for the degree of Bachelor of Science, 2015

Variations of artificial neural networks are prevalent in modern day technology and their applications range from analysing speech and identifying images [1] to predicting stock market fluctuation [2]. As this area of research continues to grow, it can be safely assumed that the capabilities associated with these technologies will become even more extensive. This report investigates the effects of using multiple training algorithms, specifically an evolutionary algorithm and the back-propagation algorithm, in an intermittent fashion in order to assess the training efficiency compared to adopting either training algorithm independently. Through the investigation of numerous mathematical models and by designing and implementing a platform capable of evaluating the efficiency of the training process, given any particular training algorithm, it was shown that the utilisation of a hybrid training methodology can greatly improve the efficiency of the training process.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank Dr. Richard Neville for providing me with continuous mentoring throughout the project duration. I would also like to thank my family for continuous support throughout my time at university.

Chapter 1

Introduction

Overview

This research project explores numerous methods[1] associated with configuring an artificial neural network (ANN) with the intent of performing an approximate function mapping.

1.1 Basic Introduction to the Research Domain

Artificial neural networks emerged from the connectionist paradigm [3], which provides a biologically inspired model that considers existential phenomena[2] a process emanating from the interconnection of simple units, which form networks . Artificial neural networks are primarily used for classification, regression and clustering type problems [4].

1.2 Aims of the Research Project

The aims associated with the research project embody the following propositions:

1. The acquisition of knowledge associated with artificial neural networks and their related research domains.

[1] Specifically individual training algorithms in addition to hybridised methodologies encompassing these algorithms.

[2] Such as the ability to think and to behave.

2. An inquisition into the research domain (§ 1.3), leading to the formation of either one or more hypothesises (§ 1.4).
3. The exploration of mathematical models (§ 2.5) that translate to a system of data structures and algorithms (§ 4.2.1) which can be used to validate and test any given hypotheses through the critical analysis of a series of experiments (§ 5).
4. To gain experience associated with the appropriate manner in which scientific research is conducted[3] and to reflect upon this experience.

1.3 Research Context

In order to pursue this project an inquisition into multiple research domains is required. These domains are outlined in the research relevance table and the theory research relevance tree, viewable in **Appendices A.1** and **A.2** respectively.

1.4 Research Questions

By reviewing the research domains outlined in § 1.3, the following research questions are presented:

1. How efficient is the process associated with training a function mapping ANN through the utilisation of:
   (a) A genetic algorithm?
   (b) The back-propagation algorithm?
2. 2. Would the implementation of a control mechanism used to switch between a genetic algorithm and the back-propagation algorithm result in a more efficient training methodology?

In order to perform a research project a hypotheses is required, which can be derived from any research questions [5]. Therefore the hypothesis associated with this research project is described below.

[3] This is important because it embodies the production of novel and empirical evidence, associated with a given research domain, that may be used in order to further extend the collective knowledge of our species.

H#1: The implementation of a control mechanism to switch between a genetic algorithm and the back-propagation algorithm results in a more efficient method of training artificial neural networks than using either of the training algorithms independently.

1.5 Deliverables

Numerous deliverables are set as prerequisites to enable the testing and validation of the proposed hypothesis, and are as follows:

1. The implementation and validation of[4]:
   (a) Foundational mathematical models that include:
      i. A threshold logic unit.
      ii. A sigmoidal neuron.
   (b) Foundational training methodologies that include:
      i. The perceptron rule.
      ii. The delta rule.
2. The implementation of the training and testing platform (§ 3.1.3) that enables the generation of an ANN, specified to any dimensionality, in addition to the deployment of any particular training methodology.
3. The implementation of the following training methodologies:
   (a) The back-propagation algorithm.
   (b) A genetic algorithm.
4. The construction of a control mechanism to enable the alternation between different training methodologies.
5. 5. The implementation of an evaluation platform (§ 3.1.3).

These deliverables are further discussed as a set of analysis requirements in § 3.3.

[4] In order to evaluate more complex mathematical models and to construct data structures and algorithms to represent these models, the foundational models that underpin these more complex models must be validated.

1.6 Report Organisation

The structure concerning the remainder of this report is presented as follows:

Chapter 2, Background and Literature Review: provides a comprehensive review surrounding the background and literature associated with artificial neural networks.

Chapter 3, Research Experiment Design: introduces the experimental design techniques that have been implemented in order to carry about an analysis for the research project.

Chapter 4, Experiment Implementation: comprises of details relating to the implementation of each experiment designed in order to perform the project analysis.

Chapter 5, Critical Analysis and Evaluation: provides a detailed, critical analysis of the result sets from each experiment and a supplementary evaluation conclusion.

Chapter 6, Conclusion and Future Work: concludes the entire project and potential future research opportunities.

1.7 Concluding Remarks

This chapter has provided a set of aims for the research project, in addition to the exploration of various related research questions which lead to the formation of a hypothesis. Furthermore, a set of deliverables have been supplied and the organisational structure of the remaining report has been outlined.

Chapter 2

Background and Literature Review

Overview

This chapter provides a review of the literature surrounding artificial neural networks by firstly introducing artificial intelligence, machine learning and evolutionary computing. Subsequently, the basic concepts underpinning the technologies implemented in the project are discussed (§ 2.4). After which, the mathematical models that define these concepts are explored and an analysis of the learning paradigms used to train these models is presented (§ 2.5, § 2.6). Finally, the challenges associated with aspects of the project are discussed and a brief outline of model design, implementation and validation is provided (§ 2.7, § 2.7.3).

2.1 An Introduction to Artificial Intelligence

Artificial intelligence is an academic field of study concerned with developing technologies[1] that attempt to resemble intelligence [6].

In order for a machine to be considered intelligent it must possess the following capabilities [7]:

1. **Natural language processing** to allow the machine to successfully communicate using human linguistics.
2. **Knowledge representation** to store what it knows or any information it receives.

[1] These technologies are usually implemented in the form of classical computing hardware or software.

3. **Automated reasoning** to utilise stored information to answer questions or as premises to justify a conclusion.
4. **Machine learning** to adapt to new situations and to extrapolate patterns.

See **Appendix B.1** for further reading on AI.

2.2 An Introduction to Machine Learning

Machine learning emanates from artificial intelligence[2] as a constituent research area and is concerned with exploring the means by which a machines performance may automatically improve[3] with experience [8]. A machines performance is usually measured by some performance function, which is based off its ability to solve a given problem, such as the recognition of a specified pattern. See **Appendix B.2** for further reading on Machine Learning.

2.3 An introduction to Evolutionary Computing

Evolutionary computing is a branch of artificial intelligence that draws its inspiration from nature, specifically from the process of evolution. It aims to solve optimisation problems[4] by producing candidate solutions[5] according to a stochastic framework, and assessing the quality of those candidate solutions through the utilisation of a fitness function[6]. Candidates classified as having a desirable fitness are used in combination with a stochastic framework to produce the seeds for constructing further candidate solutions [9]; this cycle is repeated until some stopping criterion is met, usually when a good enough solution has been found. See **Appendix B.3** for further reading on the History of Evolutionary Computing.

[2] It can be seen that machine learning is cited as a prerequisite to passing the Turing test, which belongs to the domain of artificial intelligence.

[3] The performance of a computer program is assessed by its ability to solve a given problem.

[4] These types of problems are thought of as an environment, for example an environment could be a system that has an optimal configuration (a neural network), but is currently configured to a sub-optimal setting.

[5] A candidate solution can be thought of as an individual that represents some parameter configuration for the environment.

[6] A fitness function describes how desirable the configuration that a specific individual represents is.

2.4 Basic Concepts

The fundamental concepts that underpin the research project are discussed within this section, which provides an intuitive description for biological neural networks and artificial neural networks.

2.4.1 Biological Neural Networks

The human brain is constructed from approximately 86 billion neurons, that are connected together in order to form networks [10], it takes input data from the peripheral nervous system[7] and is responsible for the capabilities[8] outlined in § 2.1 [11].

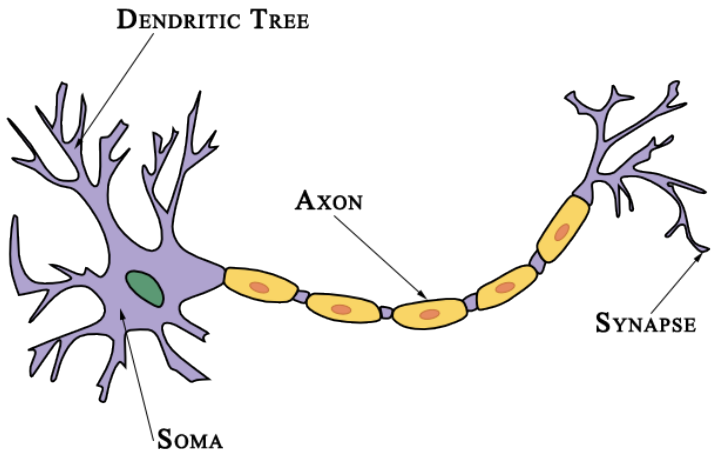

Figure 2.1: Depicts and labels a biological neuron.

A single biological neuron comprises of identifiable components, annotated in figure 2.1, that serve a set of individual purposes [12] which are outlined below:

1. **Synapse:** The component that forms a connection between neurons.
2. **Dendritic tree:** This has multiple receptors on each branch that bond with neurotransmitters[9] at the synaptic cleft[10] and are responsible for propagating an electrochemical stimulation to the soma.

[7] *See Appendix B.4 for further reading.*

[8] The human equivalent of these capabilities.

[9] Neurotransmitters are endogenous chemicals responsible for transmitting signals in the space between neurons; they can inhibit (decrease) or excite (increase) the signal strength too.

[10] The synaptic cleft is the space between the end of an axon and the beginning of a dendrite.

3. **Soma:** The cell body that receives electrical impulses from the dendrites. If enough cumulative excitement from the dendritic tree is present, depolarization[11] occurs, causing a voltage spike across the neural axon.
4. **Axon:** This component is responsible for carrying the electrical signal from the soma to the synapse, where vesicles release neurotransmitters into the synaptic cleft.

2.4.2 Artificial Neural Networks

Artificial neural networks are a computational model based on the principles of biological neural networks. When implemented in a computer these models can learn and be trained using the kind of paradigms that will be discussed in a later section (§ 2.6). See **Appendix B.5** for further reading on the History of ANNs.

2.5 Mathematical Models

The mathematical models that define previously the discussed concepts are presented in this subsection, starting with the mathematical representation of an artificial neuron followed by the representation of an artificial neural network.

2.5.1 An Artificial Neuron

An artificial neuron aims to capture the behaviour of a biological neuron (§ 2.4.1) by mapping it to a mathematical model comprising of a set of variables, in addition to a set of functions.

To consider each component of a biological neuron individually, the dendritic tree is represented as a set of inputs[12] (equation 2.1 : figure 2.2) and since each input is representative of a signal, transmitted by an inhibitory or an excitatory neurotransmitter at the synapse, each input must have an associated weight (equation 2.2 : figure 2.2).

$$X = \{x_1, x_2, x_3, ..., x_n\} \tag{2.1}$$

[11] The cell membrane of a neuron is polarised, meaning that there is an electric difference across the cell membrane.

[12] These inputs are formalised as a vector.

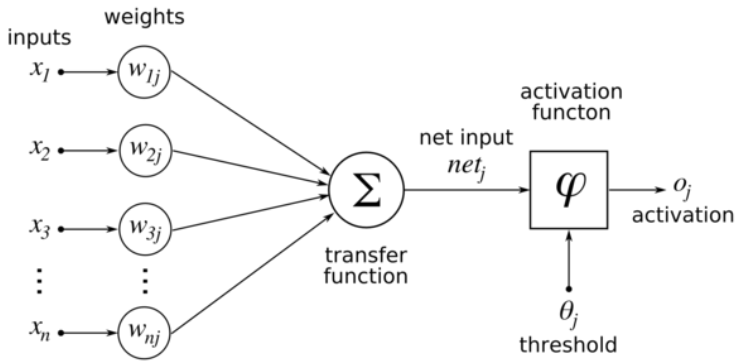

Figure 2.2: Illustrates a mathematical model of a biological neuron.

$$W = \{w_1 j, w_2 j, 2_3 j, ..., w_n j\} \tag{2.2}$$

Cumulative excitement at the soma is defined as the summation of each input multiplied by its associated weight (equation 2.3, figure 2.2 transfer function) [13].

$$net_j = \sum_{i=1}^{n} x_i w_i j \tag{2.3}$$

When the cumulative excitement exceeds the threshold value θ_j the neuron outputs a signal down the axon, which is conceptualised as the activation function of the neural model (equation 4, figure one activation function) [13].

$$\varphi(net_j) = \begin{cases} 1 & if net_j \geq \theta_j \\ 0 & if net_j < \theta_j \end{cases} \tag{2.4}$$

In order to simplify the model, the threshold value θ_j is added as a weight with an associated constant input of minus one, this is known as the bias term (equation 2.5) and subtracts the threshold from the cumulative excitement, resulting in the neuron firing as the excitement surpasses zero (equation 2.6).

$$bias = (-1 \times \theta_j) \tag{2.5}$$

$$\varphi(net_j + bias) = \begin{cases} 1 & if net_j \geq 0 \\ 0 & if net_j < 0 \end{cases} \tag{2.6}$$

The output of the neuron is the result of the activation function (equation 2.7).

$$o_j = \varphi(net_j + bias) \tag{2.7}$$

As of yet, this model only provides a description for a simple threshold activation function, which restricts the model to outputting a one or a zero. By replacing the previously discussed activation function with an alternative function that resembles a threshold, namely a sigmoid function, it is possible to produce an output that ranges somewhere between zero and one[13].

$$\varphi(net_j) = \frac{1}{1 + e^{x \times net_j}} \tag{2.8}$$

In the above equation it is assumed that the bias exists as a weight and an input[14] in their respective sets. The value c provides the ability to morph the sigmoid such that it stretches the curve in order to provide a better fit for functions during training[15].

2.5.2 Artificial Neural Networks

An artificial neural network is composed of numerous artificial neurons which are connected together to form a directed graph (equation 2.9) [14].

$$ANN = (N, C) : n \in N, c \in C \tag{2.9}$$

Every neuron may also be separated into three separate sets of layers, the input layer, the hidden layer and the output layer (equation 2.10) [14].

$$N = N_{in} \cup N_{hid} \cup N_{out} \tag{2.10}$$

The network is structured such that the input layer precedes the first hidden layer, then there may then be any number of hidden layers that precede each other, finally followed by the output layer. Since the research project concerns itself primarily with fully connected feed forward neural networks, a connection exists that routes the output of each node in every layer, to the input of each node in its subsequent layer. See **Appendix**

[13] This becomes incredibly useful when discussing learning algorithms later on in this chapter.

[14] The input for the bias always has an associated value of -1, whereas the threshold is treated as if it were any other weight.

[15] For example c could be assigned a value of 1 / 0.15 to morph the shape of the threshold (sigmoid) so that the network better fits the function described by the training set. This will be explained in more detail later in the report.

B.6 for further reading on Neural Network Representation.

2.6 Learning Paradigms

This section provides an analysis of the learning paradigms which are utilised by the project in order to train[16] a single neuron and a neural network, by firstly introducing the simplest learning algorithm, the perceptron learning rule, followed by a review of the delta rule, back-propagation and finally a brief overview of evolutionary algorithms is supplied.

2.6.1 The Perceptron Learning Rule

The perceptron rule is a learning algorithm used to align the output of a simple neuron[17] by adjusting each of the weights in its weight vector in the following fashion (equation 2.11):

$$w' = w + \Delta w \tag{2.11}$$

The change in weight is defined by the update value[18] below (equation 2.12) [15]:

$$\Delta w = \alpha(t - z)x \tag{2.12}$$

This learning rule is repeatedly applied to the perceptron unit until the actual output is equal to the target output.

2.6.2 The Delta Rule

Through the introduction of an alternative activation function, such as the sigmoid function defined by equation 2.8 in § 2.5.1, a new and important learning paradigm is also introduced. This paradigm is based on the reduction of a cost function, which provides a measure of the difference between the actual output and the target output of the network. By adjusting the weights of various neurons, this cost function can be minimised in order to reduce the overall error from the encompassing network. This requires the application of gradient decent and the utilisation of derivatives, making the choice of an

[16] Training a model consists of aligning the actual output from the model to a desired output from the model, given a specified input.

[17] A perceptron.

[18] Also coined 'the delta value'

appropriate cost function important[19]. The equation for the cost function chosen for the project is defined below (equation 2.13):

$$e^p = \frac{1}{2} \times \sum_{j=1}^{N} (t_j^p - z_j^p)^2 \tag{2.13}$$

Where p represents the training pattern[20] and j represents the node in the case of a single layer net with N nodes. In order to update each weight to minimise the cost function, the delta value (which is fully derived in the **Appendix B.8**) must be calculated for each weight (equation 2.14):

$$\Delta W_{ji} = \alpha \times (t_j^p - z_j) \times \sigma'(net_j) \times x_{ji}^p \tag{2.14}$$

This value is then used to update each weight in the same manner as equation 2.11.

2.6.3 Backpropagation and Multi-Layer Nets

Not all problems can be solved using a single layer of sigmoidal neurons, it becomes necessary to use hidden layers. This gives rise to the conception of multi-layer artificial neural networks, which cannot be trained using the previously outlined techniques. In order to train these networks, a technique called back-propagation, that contains two phases, must be utilised [16]. The first phase, the feedforward phase, is characterised by the following equations (equation 2.15 and equation 2.16):

$$\sum_{j \in N_{hid}}^{N_{hid}} o_j = \varphi(net_j) \tag{2.15}$$

Where $\varphi(net_j)$ is the activation function (equation 2.8) for any hidden node; similarly for the output layer:

$$\sum_{j \in N_{out}}^{N_{out}} z_j = \varphi(net_j) \tag{2.16}$$

After the network output z_j has been produced, the cost function (§ 2.6.2 : equation 2.13) is calculated in order to produce an error value for the network, which enables the

[19] To minimise a function using gradient decent, the function itself must be differentiable, also it would be nice to have a mathematically convenient function, such that our cost function can be nicely differentiated.

[20] A training pattern is an element that belongs to the training set. The element encompasses every input to be presented at the input layer and every output that should be generated given that particular input. We are dealing with more than one input and more than one output, because the input and output layers can contain multiple nodes and are therefore represented as input and output vectors.

back-propagation phase to initialise. Firstly, the delta values for each output and hidden node are calculated.
For each output node:

$$\delta_j = (t_j^p - z_j) \times \sigma'(net_j) \tag{2.17}$$

For each hidden node:

$$\delta_j = \sigma'(net_j) \times \sum_{k \in N_{out}}^{N_o ut} \delta_k w_{kj} \tag{2.18}$$

Subsequently each hidden and output node can be updated according to the following equation:

$$\Delta w_{ji} = \alpha \times \delta_j \times x_{ji}^p \tag{2.19}$$

These equations can be derived similarly to the derivation of the delta rule in **Appendix B.8**

2.6.4 Evolutionary Algorithms

Evolutionary algorithms aim to solve optimisation problems through the maximisation of a fitness function (§ 2.3). Conveniently, desirably configuring the weights for a neural network is classified as an optimisation problem where the fitness function may be defined as the inverse, or the reciprocal of the previously discussed cost function (§ 2.6.2 : equation 2.13). The weights for the network can be encoded in a real-valued manner and classified as an individual, in a pool of individuals initiated using a stochastic framework[21]. The maximisation of the fitness function is then driven by performing a series of operations on these individuals and assessing their fitnesses, these operators are shown below:

1. **Selection:** Evaluate the fitnesses of the population and choose the best individuals for reproduction.
2. **Crossover:** Swap encoded values[22] between individuals selected for reproduction according to a stochastic framework.
3. **Mutate:** Randomly apply a change to one or more encoded values within selected individuals.
4. **Replacement:** Produce a new population by replacing lesser individuals with new individuals form the reproduction process.

[21] In our case, a Gaussian distribution.
[22] weights.

2.7 Challenges to do with ANNs and EAs

This section introduces potential problems associated with back-propagation and evolutionary algorithms.

2.7.1 Problems with Back-propagation (BP)

Learning algorithms that use gradient-based optimisation only perform as well as the landscape characterised by the cost function and activation functions [17], this means that if the configuration of the weights result in a fairly flat hypersurface the performance of the back-propagation algorithm will be poor.

2.7.2 Problems with Evolutionary Algorithms (EAs)

Evolutionary algorithms are prone to prematurely converge before reaching a better optimised solution due to a potential lack of genetic diversity between individuals in the population. This lack of genetic diversity results in the domination of the population by a small number of elite individuals.

2.7.3 Model Design, Implementation and Validation

The model design is characterised by the mathematical models and learning paradigms that have been highlighted throughout this section (§ 2.5 and § 2.6), and is discussed in full within the next chapter. The implementation of these designs is reviewed in detail and referenced throughout chapter four.

Validation is accomplished by producing a collection of prototypes, starting with the simplest, that can be validated independently. The validation of independent prototypes is approached in the research project by monitoring the fluctuation of weights, in addition the output of the model; and then comparing these values to the predicted values that have been already calculated.

2.8 Concluding Remarks

This chapter has presented a review of the literature surrounding the research domain and has provided a brief overview relating to design, implementation and validation.

Chapter 3

Research Experiment Design

Overview

This section introduces the experimental design techniques that have been implemented to carry out an analysis for the research project. Firstly, an introduction to the project structure is presented and the reasoning behind the adopted architecture is discussed. Secondly, the analysis requirements (§ 3.3) for the project are examined, which provides a foundation to formulate numerous experiments. These experiments are then discussed and analysed (§ 3.5 - § 3.7).

3.1 Project Structure

The aim of a research project is to test the research hypotheses (§ 1.4 : H#1) through the evaluation of relevant and controlled experimentation. Such experiments provide the foundational research which enables the analysis of results and the proposal of conclusions that may be used as arguments for any hypotheses [18].

In order to ascertain that the premise is well founded, experimental structure and design is of utmost importance, since a well-constructed experiment ensures a high degree of validity. This is why the adoption of a generic workflow design has been implemented as a framework for this project, where each experiment is structured using the same set of routines, called a workflow, illustrated by figure 3.1.

Figure 3.1 demonstrates that the project is split into a number of experiments, labelled experiment 1, 2, 3, ... , up to experiment N. Each of these experiments can be defined

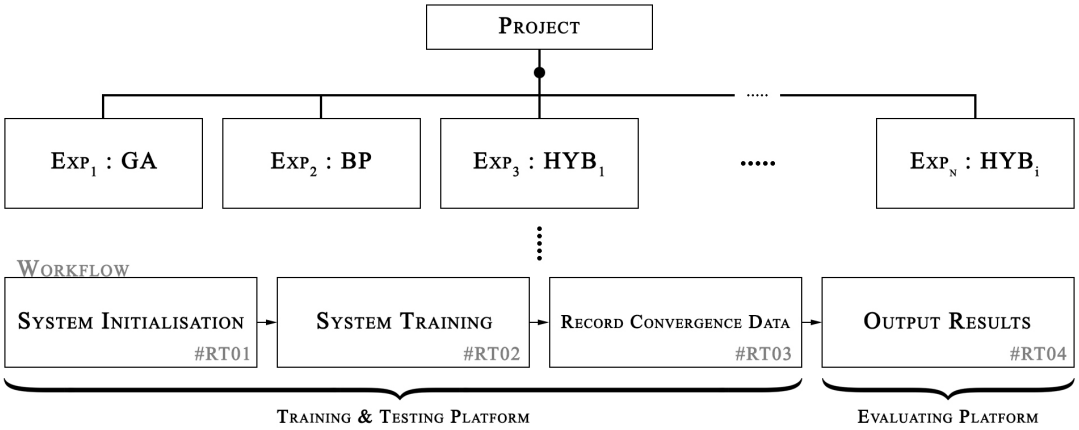

Figure 3.1: The generic workflow design implementation for the research project.

as a sequence of directed routines, labelled #RT01, #RT02, #RT03, #RT03, where the subsequent execution of each routine, in their entirety, is defined as the workflow; which may be used analogously with the term experiment. These routines can be separated into their constituent platforms, which are labelled as the training and testing platform and the evaluation platform.

The only alteration between each experiment is the selection of the training model, denoted by T, implemented by the system training routine, #RT02:

$$T = \{GA, BP, HYB_1, HYB_2, ..., HYB_N\} \tag{3.1}$$

This ensures that each experiment is performed under the same controlled conditions, enabling a high degree of accuracy with regards to statistical observation[1]. This high degree of accuracy is responsible for distinguishing our premise as reliable; however, it must also be recognised that the appropriate use of logic is a prerequisite to the formulation of a well-grounded conclusion.

[1]This is because we are essentially observing two primary variables, the —E— and the epoch, if we are not changing anything else, then we are allowing ourselves a high degree of accuracy in terms of our results, because there are no external intervening factors that can spoil our result set. This means we have a high degree of internal validity, which means we have a solid premise for our conclusion, which means we have a good scientific argument to support or to oppose our hypotheses.

3.1.1 Workflow

Each experiment is composed of a set of enumerable, directed routines, denoted by RT, called a workflow, W. This workflow is generic and must be executed in an identical manner for each experiment, in general:

$$W = \{RT_{01} \leq RT_{02} \leq ... \leq RT_n\} \tag{3.2}$$

3.1.2 Routines

A routine embodies a set of directed sub-routines, denoted by SR, which when sequentially executed in their entirety warrant the functionality of the routine:

$$RT = \{SR_{01} \leq SR_{02} \leq ... \leq SR_n\} : RT_n \in W \tag{3.3}$$

Each routine is described below:

#RT01: System Initialisation: The pre-processing stage, where the programme imports and employs a set of parameters to determine the initial state of the system. This includes network representation, training data, training methodology and display parameters. The parameters can be imported by:

1. Selecting an existing XML file.
2. Selecting an existing pre-programmed test harness.

#RT02: System Training: Train the model using the employed training methodology by making a set of calls to the function library.

#RT03: Record Convergence Data: As training is undertaken, the total error of the system is adjusting. Measure the rate at which the error is transforming and store it within a suitable data structure.

#RT04: Output Results: Push the recorded convergence data into a format, such that the data is viewable in:

1. A raw text file.
2. An Excel spreadsheet.

3.1.3 Routine Platform Classification

The workflow can be broken up into two principal platforms, each of which embodies a set of routines:

1. **Training and testing platform:** this platform is responsible for executing the set of routines that generate the data for the experiment.
2. **Evaluating platform:** this platform is responsible for executing the output results routine, which is in turn responsible for performing the following set of subroutines depicted by figure 3.2.

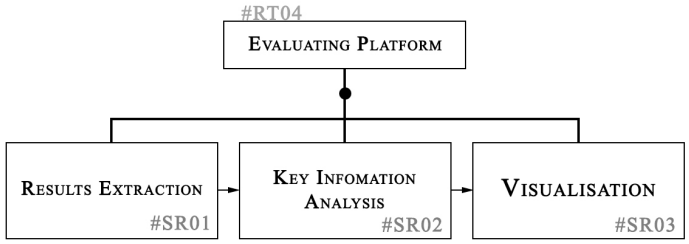

Figure 3.2: Structure of the evaluating platform (#RT04)

Figure 3.2 shows that the evaluating platform, which comprises of one routine, can be split into three separate subroutines, the first of which, #SR01 is the results extraction subroutine, followed by #SR02, the key information analysis subroutine, finally followed by #SR03, the visualisation subroutine. See **Appendix C.1** for further reading on the these sub-routines.

3.1.4 Advantages of Adopting a Generic Workflow Design

There are several advantages associated with adopting the experimental design outlined above, these advantages are listed below:

1. Routines that are shared between multiple experiments can be reused; this equates to a substantial reduction in redundancy of repeated code.
2. Concerns are inherently separated between routines, signifying that any abnormality within a particular domain of the programme is easily distinguished as being associated with a particular routine.

3. The overall project structure is characterised by its generic architecture, this property translates to causality between components, such that any alteration to a shared routine will result in the application of that alteration to all experiments that share the routine.

3.2 Project Directory Structure

Project directory structure is important since it is responsible for separating different concerns into easily identifiable elements, which aids in managing and executing the project.

Since the project is composed of various experiments, there exists one principal directory (Experiments) for the encapsulation of each experiment. The other principal directory (Library) is responsible for containing the constituent components that make up each, illustrated by figure 3.3.

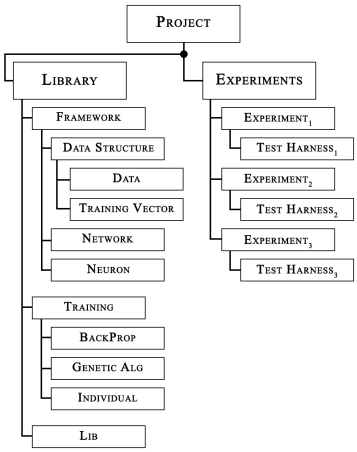

Figure 3.3: Directory structure of the project.

Each experiment has a co-responding test harness, which is essentially the implementation of the generic workflow, W. This test harness performs the experiment by making a number of calls to the library that contains each of the routines.

3.3 Project Analysis Requirement

In order to perform the analysis of various training methodologies associated with the research project, there exists a number of primary requirements that must be met:

1. Construction of the training and testing platform described in the previous section.
2. Implementation of the learning algorithms adopted by this project 1.4.
3. Implementation of a control mechanism responsible for switching between the learning algorithms.
4. Construction of the evaluation platform described in the previous section.

See **Appendix C.2** for further reading on the these requirements.

3.4 Generic Experimental Methodology

In addition to having a project structure and a set of analysis requirements, the project must also possess an experimental methodology. Experimental methodologies are important because they provide a generic framework for every experiment to be structured around, such that replication of results is supported and any confirmation bias is avoided [18].

Assuming that a mathematic model validating the hypotheses has already been established, the generic experimental methodology adopted by this project involves the following [5]:

1. The identification of a problem.
2. The proposed solution to the specified problem.
3. The aims of any proposed experiments.
4. The objectives of any proposed experiments.
5. The definition of the model parameters.
6. Tuning of the models parameters, such that optimal values may be selected (sensitivity analysis).
7. Stated expected experimental outcome.
8. Full documentation production for all experiments undertaken.

See **Appendix C.4** for further reading on the Generic Experiment Design and **Appendix C.5** for Generic Experiment Data Structures.

3.5 Experiment One Design

The first experiment to be undertaken aims to train an artificial neural network by solely utilising an evolutionary algorithm as the training methodology.

The objective of this experiment is to observe the reduction in error as the network is trained and to produce documentation to aid in supporting our hypothesis (**H#1**).

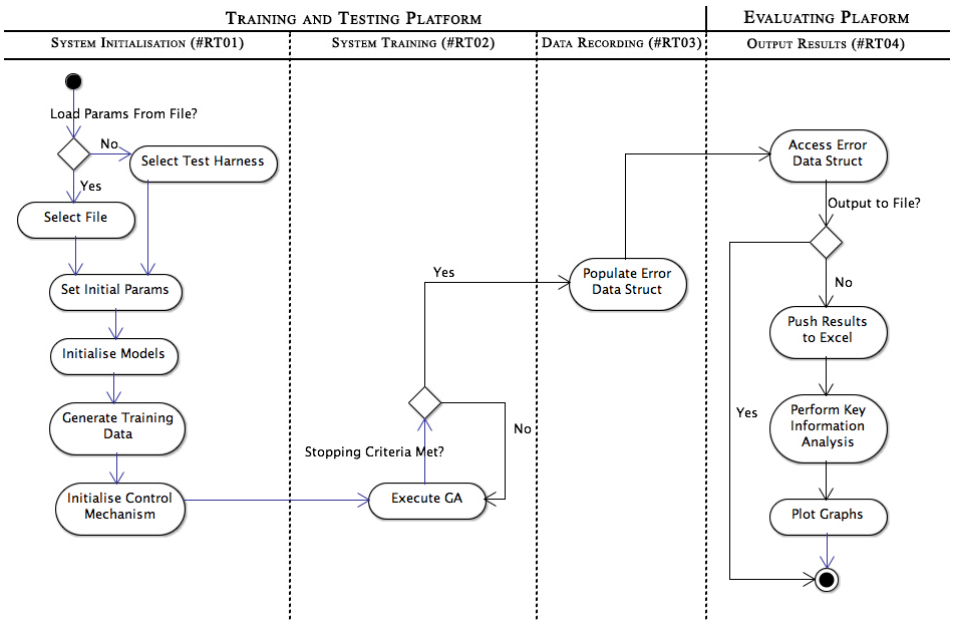

Figure 3.4: A flowchart that illustrates experiment one.

Figure 3.4 describes the workflow carried out by experiment one. The parameters that determine the initial state of the system are programmed into a test harness, alternatively these can be stored in an XML file and loaded in, this is depicted by #RT01, where the workflow has a decision option to select a file or to select a test harness. Once the initial system parameters are loaded, the neural network, training data and control mechanism are initialised (#RT01). The control mechanism is responsible for overseeing the training process, and executes the evolutionary algorithm (#RT02) until the stopping criteria is met. At this point, the systems error data is made available to

the evaluating platform (#RT03), which either dumps the data to a text file, or produces a spreadsheet with some preliminary analysis and graph plots (#RT04).

3.6 Experiment Two Design

The second experiment to be undertaken aims to train the network by solely utilising the back-propagation algorithm as the training methodology with the objective of observing the training efficiency.

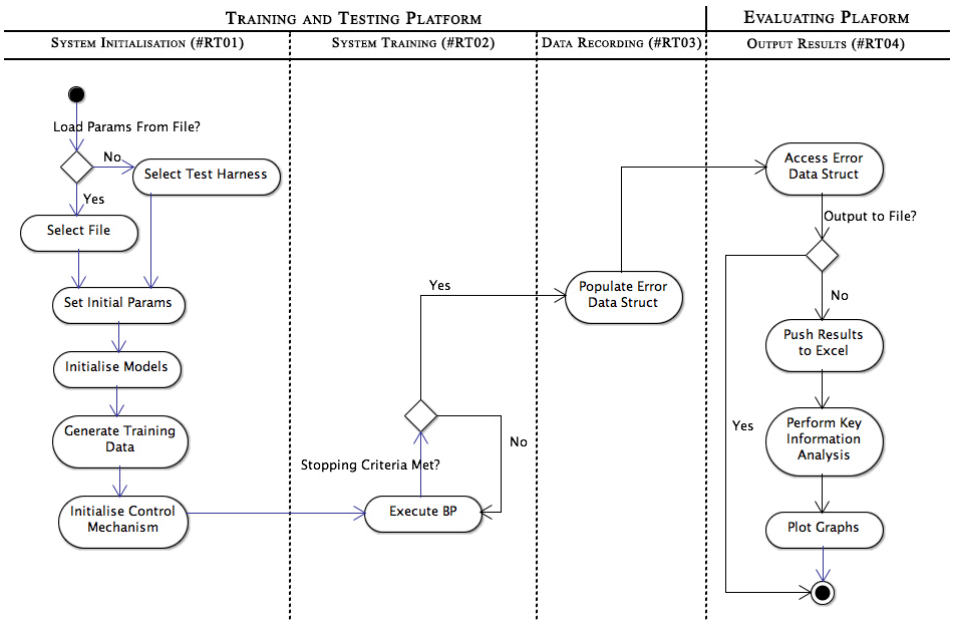

Figure 3.5: A flowchart that illustrates experiment two.

Figure 3.5 illustrates the workflow undertaken for experiment two, which follows a similar structure to that of experiment one (§ 3.5), the only difference is that the primary training algorithm (#RT02) employed by the system is back-propagation, rather than the evolutionary algorithm.

3.7 Experiment Three Design

The third experiment to be undertaken aims to train the network through the utilisation of a hybrid methodology, which uses the control mechanism to switch between an evolutionary algorithm and the back-propagation algorithm. Furthermore, the objective of

this experiment is to observe the training efficiency of this particular methodology and to compare the results to the previous two experiments.

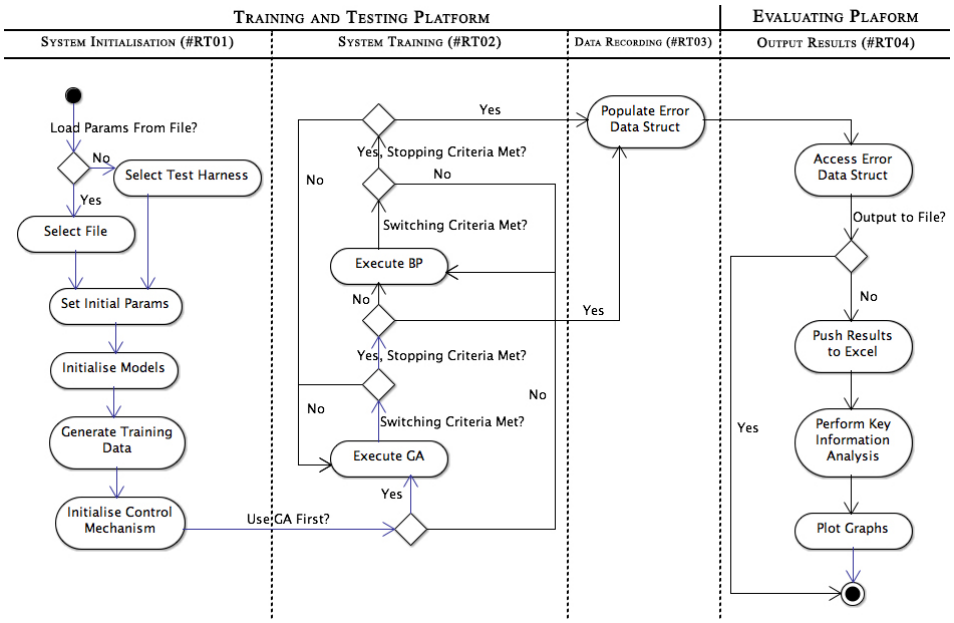

Figure 3.6: A flowchart that illustrates experiment three.

Figure 3.6 illustrates the workflow undertaken for experiment three, which significantly differs in the system training routine (#RT02) when compared to the two previous experiments. This is a symptom of the utilisation of the control mechanism, which is used to switch between the two previously discussed learning algorithms. The figure shows that the mechanism has a number of properties defined by the initial parameters that were set during the system initialisation, the two most notable properties being:

1. Switching Criteria (depicted in #RT02): the number of iterations that one algorithm must run for before switching to the alternative algorithm.
2. Stopping Criteria (depicted in #RT02): the total number of iterations that must be performed before pushing the system into the next state.

Once the stopping criterion has been met, the experiment enters the next routine; and from this state the workflow resumes as has been outlined in the previous experiments (§ 3.5 and § 3.6).

3.8 Generic Experiment Functionality

It is possible to perform experiment one and two using the workflow implemented by experiment three because the control mechanism can be configured to run a single learning algorithm. This is illustrated by the table in **Appendix C.6**. Furthermore, each experiment can be broken up into its constituent routines, which is demonstrated by the table in **Appendix C.7**.

3.9 Concluding Remarks

This chapter has outlined the design to be employed by the project in order to fulfil the analysis requirements raised by the hypotheses. The next chapter will discuss the implementation details associated with these experiment designs.

Chapter 4

Experiment Implementation

Overview

This chapter comprises of details relating to the implementation of each experiment designed in order to perform the project analysis (§ 3.3). Specifically, this chapter starts by introducing the development environment in which each experiment has been constructed. Subsequently, the construction of the data structures and the implementation of algorithms that embody the training and testing platform is discussed (§ 4.2.1). Finally this chapter examines the parameterisation details of each experiment individually (§ 4.7.1, § 4.7.2, § 4.7.3).

4.1 Development Background

The following subsection presents an overview of the development environment, in which the project experiments were constructed in addition to the discussion of the technologies utilised in order to perform the implementation of these experiments.

4.1.1 Development Environment

A development environment is characterised by the set of programming tools[1] and platforms used to construct a particular piece of software [20]. The development environment adopted by this project is distinguished by the following characteristics:

[1] A programming tool can be any piece of software that supports any software development specific task [19], some examples may include integrated development environments, text editors, compilers, linkers and assemblers.

1. Operating system: MS Windows 7 Enterprise, 6.1.7601 SP1.
2. Programming Language: C#.
3. IDE and Tools: MS Visual Studio 2012, MS Excel 2010, MS Excel Interop Library.

The technologies listed above show that the entire project analysis is performed within a Microsoft windows development environment, the reasoning behind the decision to use these tools is discussed in the following subsection.

4.1.2 Technologies Used in Implementation

C# and Excel

C# was chosen to develop the training and testing platform that generates the data for the project analysis, and Excel was leveraged as the platform for performing the analysis itself.

The primary reason for choosing C# as the programming language for developing the training and testing platform was due to the high degree of integration that C# provides with other Microsoft products, specifically, Microsoft Excel. This high degree of integration emanates from a number of libraries developed by Microsoft that enable the seamless flow of data from the source program, which has been written by the developer, into Excel [21]. This seamless flow allows the data generated by our training and testing platform to move smoothly into our evaluating platform.

Furthermore, to consider C#s language constructs, it resembles a particularly similar syntax to that of many other languages[2], this expiates the research process and reduces the learning curve for a developer who is well-versed in other relevant languages.

IDE and Operating System

The choice of integrated development environment and operating system were consequential of the choice of programming language and evaluation tools.

To elaborate, C# has been developed by Microsoft and runs primarily on the Microsoft

[2] C# closely resembles other languages, specifically: Java, C and C++ [22]

.NET framework [3] [24]; therefore developing on Windows was the most sensible choice, since the code can be easily executed natively and there is plenty of documentation offered by Microsoft relating to C# and the .NET framework.

There are various IDEs that can be used to write C#, however since Visual Studio is also developed by Microsoft, it offers the same high degree of integration that has been previously discussed.

4.2 Experimental Results Collection

This remainder of this chapter examines the implementation details surrounding the data structures and algorithms employed to construct the training and testing platform which enables the execution of each experiment. This is responsible for providing the ability to **perform experimental results collection** and allows for the project analysis. In addition, the construction of the evaluation platform (§ 4.6) is discussed.

Further reading regarding the generic implementation methodology, in addition to an overview of data structures and algorithms is available in **Appendices D.1** and **D.2**, respectively.

4.2.1 Implementing Models for the Embodiment of the Training and Testing Platform

Each of the models outlined in chapter two (§ 2.5) that map the principles of biological neurons to mathematics, can be implemented as a series of data structures that training algorithms (§ 2.6) can use. These implementations embody the training and testing platform outlined in chapter three (§ 3.1).

The model of a sigmoidal neuron is implemented by mapping its associated data components and functions to a C# class, as is demonstrated by the table viewable in **Appendix D.3**.

To construct the network, an additional class must be created that possess a container

[3] C# can also be executed on an open source implementation of Microsofts .NET framework called Mono [23]

to provide a means of accessing each neuron in the network, in addition to its data attributes. This has been implemented as a two dimensional array, where the first dimension represents the layer index and the second dimension represents the node index, as is described by snippet 4.1 and is depicted by figure 4.1.

$$neuralNetwork[layerIndex][nodeIndex] \tag{4.1}$$

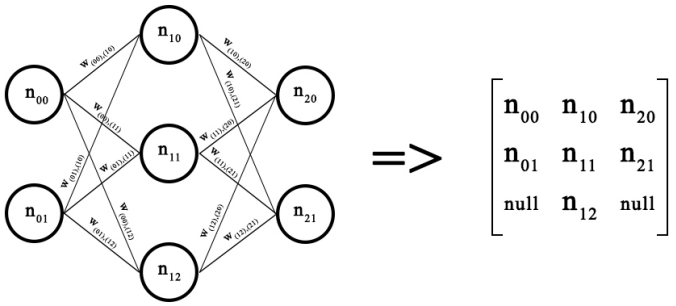

Figure 4.1: Visual implementation of an ANN as a 2-d array.

In order to form connections between each neuron in the network, an additional attribute called an adjacency list [25] is added to each neuron. This list describes every connection from the source node[4] to any target nodes and is accessible through the data structure as described by snippet 4.2:

$$neuralNetwork[layerIndex][nodeIndex].getAdjacencyList() \tag{4.2}$$

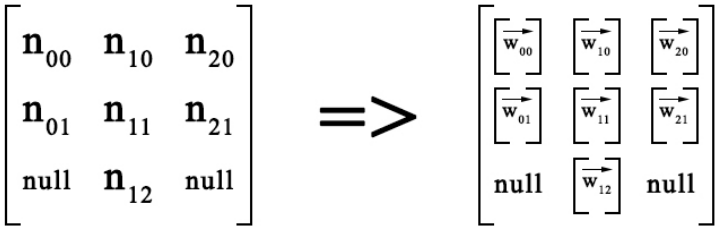

Figure 4.2: Illustrates the method by which any nodes adjacency list may be accessed.

Figure 4.2 demonstrates that by utilising a 2-dimensional array, depicted on the left side of the figure, it is possible to reference each node and to obtain its adjacency list.

[4] the nodes in the graph are the neurons.

This method is also utilised to access each neurons respective weight vector.

In addition to containing each node in the network, each of the functions associated with the network model must be implemented as part of the class in order to provide the essential functionality, such as initialising the network, initialising an input vector and performing a feed forward propagation through the network. Specific details are provided in **Appendix D.3**.

4.3 Training Data Representation

In order to calibrate an instantiated artificial neural network (the principal network model) through the utilisation of previously discussed training methodologies (§ 3.5, § 3.6, § 3.7), such that the output provides a good approximation to a specified function, a data structure must exist that provides the ability to contain a number of training examples[5]. This data structure is implemented as an array containing a series of objects, representative of training examples, that point to a desired output given some input; and can be conceptualised as a design matrix [26]. Further details are explained in **Appendix D.4**.

4.4 Representation of Initialisation Parameters

Each experiment has a number of initialisation parameters that define the state of the system before any training methodology has been executed. These parameters are either selected by choosing a pre-defined test harness, or are loaded from an external XML file and are stored in a data structure that provides convenient access to each parameter value. This data structure is simply a container for each parameter, such that the set of parameters may be collectively and easily passed between components in the system. Further implementation details are available to view in the **appendix D.5**.

[5] Training examples represent the specified function mapping.

4.5 Algorithms: Training and Testing platform

The execution of each experiment requires the implementation of numerous algorithms. These algorithms modify the data structures described in § 4.2.1 to provide the functionality required to produce a set of results for each experiment (§ 3.5, § 3.6, § 3.7). This subsection outlines the algorithms implemented into order to execute the workflow as described in chapter three (§ 3.1).

4.5.1 Network Initialisation

Before any training algorithms can be executed, the system must first be initialised through the use of the system initialisation routine (§ 3.1.2). The primary sub-routine[6] which initialises the principal network structure, the algorithmics of this sub-routine are outlined below.

Algorithm 1: Network Initialisation

Data: numLayers, numInputs, numHidden, numOuput
Result: void

```
1 AddLayer(numInputs);
2 for layer 1 to numLayers do
3 |   AddLayer(numHidden);
4 end
5 AddLayer(numOuput);
6 InitWeightsAndInputs;
7 GenerateFullConnectivity;
```

Algorithm 1 accepts a number of initialisation parameters which are used to specify the dimensionality of the network and makes multiple references to data structure functions that are outlined in the **Appendix D**.

4.5.2 Feed Forward Propagation

As a prerequisite for training, a performance metric that expresses some form of distance between the actual output of the system and the desired output of the system must exist (§ 2.13). In order to produce a value this from this metric, the network must

[6]The primary sub-routine provides the majority of the functionality that characterises the behaviour of the routine its encapsulated within.

be capable of producing an output using a 'feed-forward' function (§ 2.6.3).

Algorithm 2: Feed Forward Propagation

```
Data: [] inputVector, layers, network
Result: void
SetInputVector(inputVector);
foreach layer in (layers - 1) do
    foreach node in layer do
        sourceNode := network[layer][node];
        foreach target in sourceNode.adjacencyList do
            target.inputs[sourceNode.index] := sourceNode.outputTransfer;
        end
    end
end
```

4.5.3 Training Algorithms

To bring the output of an initialised network into alignment with its target function, numerous training algorithms based on the learning paradigms (2.6) were implemented. This subsection focuses on the particular training algorithms implemented.

4.5.3.1 The Evolutionary Algorithm

Algorithm 3: Evolutionary Algorithm

```
Data: maxIndividuals, numCycles, network, trainingData
Result: Individual
population = initIdividuals(maxIndividuals);
for cycle := 0 to numCycles do
    deemUnfit(population);
    evaluateFitness(population, network, trainingData);
    retainElitestIndividuals(population, 0.05);
    survivalLottery(population);
    selectionOnNonSurvivors(population);
    population = produceNextGeneration(population, network, trainingDatat);
end
return population.fittestIndividual;
```

Algorithm 3 outlines the high level functionality that characterises the evolutionary algorithm utilised in the project training methodologies, the designs of which are highlighted in chapter three (§ 3.5 - § 3.7). For a further explanation of the 'Individual' data-type in addition to algorithmic parameter definitions and return values, see **Appendices D.6** and **D.7**.

4.5.3.2 Back-propagation

Algorithm 4: Back-propagation

```
   Data: network, trainingData, learningRate
   Result: void
 1 inputVectors := getInputVectors;
 2 foreach input in inputVectors do
 3     network.propagationForward(input);
 4     targets := trainingData.getTargets(input);
 5     foreach node in network.outputNodes do
 6         node.calculatePartialOutputNode;
 7     end
 8     foreach layer in network.hiddenLayers do
 9         foreach node in layer do
10             node.calculatePartial;
11         end
12     end
13     foreach node in network.outputNodes do
14         node.calculateDelta;
15         node.updateWeights;
16     end
17     foreach layer in network.hiddenLayers do
18         foreach node in layer do
19             node.calculateDelta;
20             node.updateWeights;
21         end
22     end
23 end
```

Algorithm 4 presents a high level interpretation[7] of the back-propagation algorithm, implemented to train the network using a generalised version of the delta rule § 2.6.2. The previously discussed semantics (§ 2.6.3) for this algorithm hold. For a description of the parameters and a review of the high level functions used in this algorithm see **Appendices D.9** and **D.3** respectively.

4.5.4 Control Mechanism

The implemented control mechanism[8] provides the capability to switch between the two primary learning algorithms outlined thus far. This control mechanism accepts a set of configuration variables specified by the parameters data structure during the system initialisation routine. These configuration variables dictate the behaviour of the control mechanism with regards to the initial training algorithm to be executed, in addition to the starting and stopping criteria.

4.6 Implementation of the Evaluation Platform

In order to perform the project analysis, the efficiency of any adopted training methodology utilised during any experiment must be recorded as it is executed. After each iteration, the system error[9] is calculated and logged in an array; this provides the error data that is then imported into the evaluation platform.

The evaluation platform (§ 3.1.3) provides a means of analysing the change in error data as our system is trained. As part of each experiment, the principal network model is re-instantiated and re-trained numerous times to provide multiple sets of results for the purpose of experimental reliability. As every new set of error data is pushed to the evaluation platform, the platform extracts meaningful information by calculating the best, worst and average case scenarios with regards to the training efficiency. Subsequently, the platform provides a plot of these results sets, this allows for a visual comparison to be drawn between different experiments.

The platform was implemented as an excel spreadsheet and the error data was pushed

[7] some parameters for functions have been discarded in order to try and convey the conceptual underpinning of this algorithm.

[8] The design of which has been outlined in chapter three (§ 3.7).

[9] generated by the absolute value of the error function.

to the platform through the use of the Microsoft Excel Interop Library. The control mechanism is responsible for utilising the display initialisation parameters[10] in order to control the flow of data to the spreadsheet and to enable the plotting of graphs.

4.7 Parameter Settings for each Experiment

The physical manifestation of each experiment design described in chapter three (§ 3.5 - § 3.7) utilises the implemented data structures and algorithms that have been outlined throughout this chapter. Since each experiment can be individually executed using the same workflow (§ 3.1.1), the only changes that are made between each experiment are to the set of parameters fed to the system initialisation routine.

The following subsections reintroduce each experiment and provide supplementary parameters to explain how the aims and objectives of each experiment translate to their associated model parameters. Since the majority of the parameters remain constant between each experiment, the subsections will only document the change in parameters between each experiment[11].

4.7.1 Experiment One

Experiment one aims to train an artificial neural network solely through the utilisation of the implemented genetic algorithm, with the objective of observing the efficiency of training using this training method.

The experiment is performed by instructing the control mechanism to initialise the genetic algorithm first, to make a specified number of cycles and then to terminate after these cycles have been ran.

[10] from the parameters data structure.

[11] A full set of experiment parameters can be viewed in Appendices - **D.10, D.11, D.12**

Experiment One Controller Parameters	
Parameter Name	Parameter Value
Initial Training Method	GA
Total Cycles	350
Cycles between each Switch	350

Table 4.1: Initialisation parameters associated with experiment one.

4.7.2 Experiment Two

Experiment two aims to train an artificial neural network through the utilisation of back-propagation, with the objective of observing the efficiency of training using solely this training algorithm.

Experiment Two Controller Parameters	
Parameter Name	Parameter Value
Initial Training Method	BP
Total Cycles	350
Cycles between each Switch	350

Table 4.2: Initialisation parameters associated with experiment two.

4.7.3 Experiment Three

The final experiment aims to train an artificial neural network through the utilisation of the hybrid training methodology, as outlined in chapter three (§ 3.7). The objective of this experiment is to observe the performance of using this training methodology and to then compare the results from this experiment with the results from the previous two experiments, in order to provide evidence to support the hypotheses (§ 1.4).

The experiment is performed by instructing the control mechanism to initialise the genetic algorithm first, and to run this learning algorithm for a specified number of cycles in an attempt to try and provide a good estimation as to the configuration of the network.

Subsequently the control mechanism is instructed to fine tune the result by switching to back-propagation for a specified number of cycles. This pattern is repeated until the stopping criterion is met, the instance at which the cumulative number of cycles surpasses a predefined amount. This is demonstrated in the parameters table as follows.

Experiment Three Controller Parameters	
Parameter Name	Parameter Value
Initial Training Method	GA
Total Cycles	350
Cycles between each Switch	10

Table 4.3: Initialisation parameters associated with experiment three.

4.8 Concluding Remarks

This chapter has discussed the implementation details regarding data structures and algorithms employed by the generic experiment workflow (§ 3.1) used to embody the training and testing platform and to perform each experiment. Furthermore, the implementation details surrounding the evaluation platform were discussed and an exploration of the parameterisation of each experiment has been outlined.

Chapter 5

Critical Analysis and Evaluation

Overview

This chapter provides a detailed, critical analysis of the result sets from each experiment and supplies a conclusion directly related to the aggregation of each analysis. It does this by firstly introducing the two primary evaluation techniques used to compare and contrast the result sets generated by each experiment. Subsequently, each experiment is reintroduced and the results from these experiments are reviewed, analysed and evaluated in order to produce an evaluation conclusion (§ 5.3).

5.1 Evaluation Methods

The evaluation methods adopted by the research project aim to test the efficiency of each training methodology (§ 3.5, § 3.6, § 3.7) and involves the analysis of the results[1] produced by each of the experiments.

Firstly, a table is constructed that comprises of the error[2] values and identifies the minimum, mean and maximum error values for a set of repeated experiments. Subsequently, a visual representation of this table is produced in the form of a graph to provide a clear and digestible portrayal of the result sets from each experiment.

[1] The results describe the change in error of the instantiated artificial neural network between subsequent iterations of the adopted training methodology for that particular experiment.

[2] The error is generated through the use of a performance function, as outlined in chapter two (§ 2.13).

5.1.1 Evaluating via the Results Table

The results table is structured such that each column is representative of one individual instance of the experiment and since the experiment is repeated numerous times, there are numerous columns. Each row represents a training iteration[3] at which the employed training methodology has been executed. Three additional columns are added to the end of the table to provide key information; these are the minimum error, the mean error and the maximum error, calculated from the entire set of experiments.

Performing any evaluation through the use of a results table is not the most intuitive approach, but it is the most comprehensive. To elaborate, the results table provides the ability to check the exact error value at any point during any experiment. This proves useful when attempting to validate the behaviour of an algorithm in the instance where a visual representation may appear ambiguous[4]. In addition, the results table provides an effortless method of immediately checking the exact error value at the end of any given experiment.

5.1.2 Results Visualisation

The generation of a visual representation portraying the result set is accomplished through the application of the previously discussed tables. The three additional columns that provide the minimum mean and maximum error values taken from the set of repeated experiments are plotted, the x-axis identifies the training iteration[5] and the y-axis identifies the error modulus[6], such that any point on the graph is representative of the error at any particular training iteration. By connecting these points together and observing how the error values change between subsequent iterations, an evaluation on the algorithmic efficiency of the training methodology employed by the experiment that generated the visualised result set can be formulated.

Furthermore, the algorithmic efficiency of experiments that employ different training

[3] The term training iteration is analogous to epoch.

[4] For example: when the change in error is so small that its difficult to determine if there is in fact any change occurring through the use of a large scale error plot.

[5] This axis is labelled as Epoch.

[6] This is the absolute value of the error.

methodologies can be compared; therefore it can be proved, to some degree of certainty, that one methodology is more efficient than another.

5.2 Experiments

The aim of each experiment is to minimise the error that a specified ANN produces, with the objective of observing how this error changes after a certain number of training iterations. After having observed each of these experiments, an analysis can be performed which will distinguish the most efficient training methodology.

5.2.1 Experiment One: Training with an EA

To consider the experimental methodology explored in chapter three (§ 3.4), in succession to the discussion of model parameters, an expected experimental outcome must be stated prior to performing the experiment. The expected outcome of the experiment anticipates that firstly, a sharp decrease in error will occur during the first two or three iterations[7], subsequently, a number of smaller reductions in error should occur as the algorithm mutates and crosses over configuration values from different individuals in the population. The error will eventually level out, as it is anticipated that the algorithm will prematurely converge to a solution, due to a significant reduction in genetic diversity of the population[8].

5.2.1.1 Results Analysis

Figure 5.1 provides a visual representation of the result set generated by experiment one. In all cases[9] there is a sudden reduction in error between the first and second iteration[10], followed by a number of smaller reductions in error, shown between the first and fiftieth epoch. Subsequently, it would appear that the algorithm has prematurely

[7] A significant decrease in error usually occurs during the first couple of iterations due to the configuration of the neural weights at initialisation. The configuration of these weights may result in a highly undesirable output since the network is completely untrained and is at its furthest away from its optimum configuration.

[8] This significant reduction in genetic diversity is due to a small number of dominant individuals that eventually take over the entire pool.

[9] The worst case is considered to be the maximum error, the best case is the minimum error and the average case is the mean error.

[10] It is validated that the reduction in error is always between the first and second iteration by referring to the results table and checking the first and second row of results.

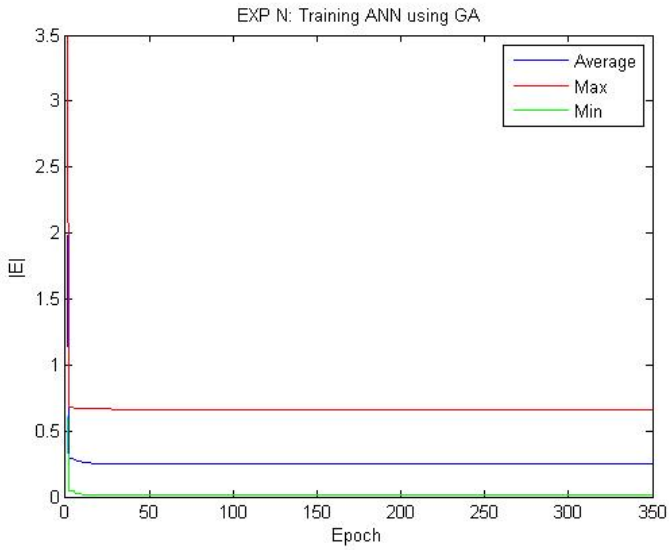

Figure 5.1: A visual illustration of the result set produced by experiment one.

converged to a solution. However, to consider the set of values from the results table, it can be proved that in some experiments there is at least one additional reduction in error after the fiftieth epoch[11]. It could therefore be stated that the algorithm simply needs to continue running for a larger number of iterations to increase the probability of a desirable mutation or crossover; which in turn would subsequently lead to further reduction in error of the system.

5.2.2 Experiment Two: Training with BP

The parameterisation of the back-propagation algorithm requires a sensitivity analysis, which is fully documented in **Appendix E.1**.

The expected outcome of the experiment constitutes of a gradual reduction in error, the rate at which this occurs is predicted to be at its greatest initially and will reduce as the number of iterations proceeds. A point will then be reached where the algorithm has either a significantly reduced error reduction rate, or the algorithm will stabilise completely at a local minimum[12].

[11] These reductions are not visible in figure one because on average they are undetectable and have not occurred in the worst or best cases.

[12] Back-propagation relies on derivatives, therefore, if local optima exist, where the gradient is equal to zero, the algorithm may land at this point and get stuck.

5.2.2.1 Results Analysis

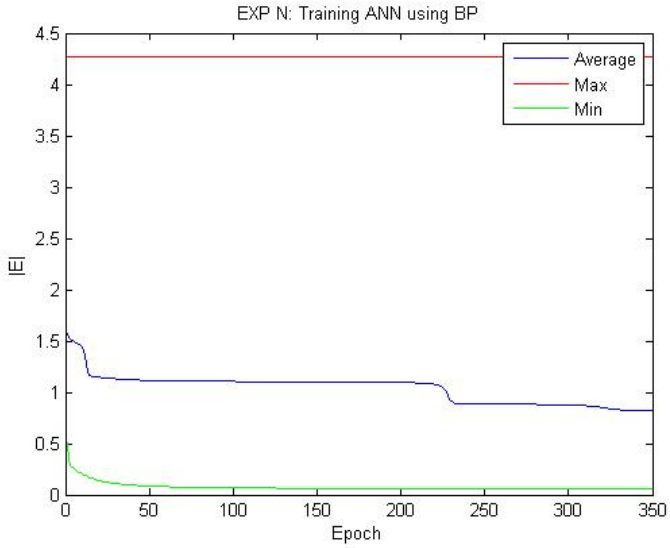

Figure 5.2: A visual illustration of the result set produced by experiment two.

Figure 5.2 portrays the change in error between each epoch of the backpropagation algorithm adopted for experiment two. In the best case the algorithm performs exactly as expected. However, it appears that in the worst case, the neural network has been initialised to a configuration that translates to an extremely low rate of error reduction[13]. The average case falls somewhere in between, as would be expected.

5.2.3 Experiment Three: Training with HYB

As a consequence of the control mechanism designed for this experiment (§ 3.7), the results generated by the experiment are expected to show an expedited rate of error reduction, which tends towards zero much faster, in all cases. This prediction is based off of the previously observed behaviour of the two constituent training algorithms used in this methodology. To further elucidate, the genetic algorithm provides a means of sharply reducing the error and then stabilising at some point, this provides a well-informed estimate with regards to an initial starting position. From this point, backpropagation can fine tune the configuration of the system and if it gets stuck, the genetic algorithm can resume control to pull backpropagation out of the local optimum its arrived at.

[13]By consulting the results table, we can validate that the error is reducing very slowly in the worst case.

5.2.3.1 Results Analysis

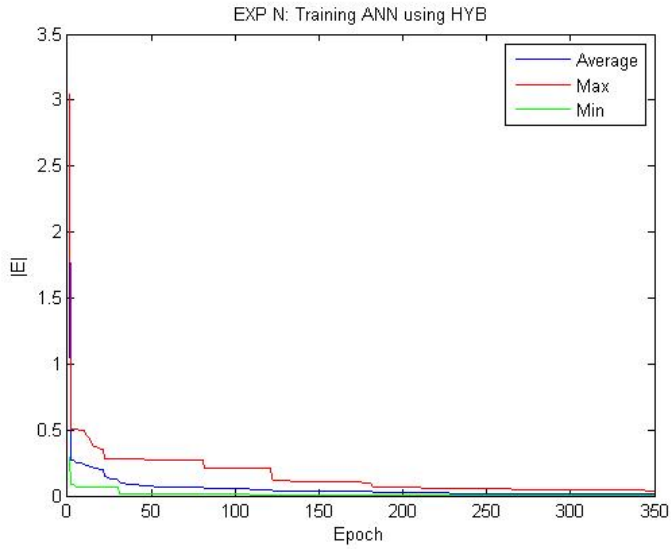

Figure 5.3: A visual illustration of the result set produced by experiment three.

Figure 5.3 demonstrates the result set generated through the utilisation of the previously outlined hybrid training methodology. Firstly, it can be identified that the training methodology is behaving exactly as expected. All cases appear to be consistently[14] making steps towards zero, and in instances where it would appear that one algorithm is stuck, there are distinct regions of the graph[15] that provide an indication of a switching of control from one algorithm to the alternate algorithm, which results in a further reduction of error. Secondly, it should be noted that after 200 cycles, in the worst case, the error is approaching the same region as the average case and the best case; and by the time the algorithm has ran for 350 cycles, the worst case is within 2.3% of the average case[16].

[14]The use of the word consistently is used to point out that there is a consistent pattern of reduction in error as oppose to a reduction in error on each iteration.

[15]These distinct regions of the graph are identified by a horizontal line, followed by a sharp drop, such as the one between 50 and 100 epochs (in the worst case).

[16]There is a difference of 0.023258 between the worst and average case by 350 cycles.

5.3 Evaluation Conclusion

To consider the results from each experiment individually, it can be stated with a fairly high degree of certainty[17] that utilising a genetic algorithm to train an artificial neural network can, in some instances, be relatively efficient[18], however on average, it does not perform to a satisfactory level[19]. This may also be stated regarding the back-propagation algorithm, in the best case, it performs relatively well, however the worst and average cases are considerably poor. Whereas the hybrid methodology adopted by experiment three proves to be substantially more efficient in all cases when compared to the result sets of the previous two experiments.

In conclusion, training an artificial neural network using a hybrid training methodology, such as the one designed, implemented and executed in experiment three, is considerably more efficient than training using either constituent algorithm, which proves the initial hypothesis stated in § 1.4.

5.4 Concluding Remarks

This chapter has provided an introduction to the evaluation techniques utilised in the project, in addition to a critical analysis of the result sets from each experiment using these techniques; and has presented a conclusion based on this analysis, which supports the initial project hypothesis. The final chapter will conclude this project as a whole and provide an insight into potential future avenues.

[17] This degree of certainty relates to the repetition of each experiment.
[18] Relative to the other methods outlined throughout the project.
[19] Ideally, on average, the error of the network should be reduced below 1% within 350 cycles.

Chapter 6

Conclusion and Future Work

Overview

This chapter concludes the project by firstly introducing the development process that lead to the project itself, followed by a summary of achievements outlining what was accomplished through the adoption of the project. After which a critical analysis and evaluation of the project is presented (§ 6.3), subsequently followed by identifications of improvements and further work that could potentially be pursued (§ 6.4). Finally this chapter ends by reflecting on the experience gain through the undertaking of this project (§ 6.5).

6.1 Development of Ideas

Initially, the project was orientated towards producing solely an evolutionary algorithm to train the weights in an artificial neural network; however, as early neural network prototypes were reviewed and implemented, the back-propagation algorithm posed as an interesting development proposition. Implementing both the back-propagation algorithm and an evolutionary algorithm would provide the opportunity to compare the algorithmic efficiency of each. After having implemented and identified problems associated with both algorithms (§ 2.7), the development of a control mechanism to switch between the two algorithms seemed to be the natural next step, this development was undertaken and proved successful.

6.2 Summary of Achievements

As a result of undertaking this project a substantial amount of background research was undertaken, resulting in numerous research questions through the development of ideas (§ 6.1). These research questions were considered and a hypothesis was chosen, then the development of a system, in addition to the adoption of a scientific method (§ 3.4), was used to test and prove the hypothesis (§ 1.4).

6.3 Critical Analysis and Evaluation

To evaluate, the designs employed by each experiment established a reliable framework in which all experiments could be performed reliably, therefore the confidence in the result sets produced by the experiments can be considered to be high. However, the implementation of the applied training algorithms was fairly basic, since adaptations of these algorithms exist[1] that could potentially yield a more efficient performance.

Although more efficient variations of the training algorithms exist, the implemented functionality was sufficient to prove the initial hypothesis.

6.4 Identification of Improvements and Further Work

This section introduces possible improvements to the current project, in addition to potential future research work that could be undertaken.

6.4.1 Improvements

The data structures and algorithms designed (§ 3) and implemented (§ 3) in order to perform the project analysis were burdened with too much complexity. For instance, the implementation of a weight matrix rather than an independent weight vector for each neuron may have been a more ideal approach, because the project concerned itself with relatively small networks and this would have simplified the architecture of the system, thereby potentially saving time.

[1] These are cited in the following section.

The performance of the evolutionary algorithm implemented in experiment one could have yeilded more desirable results with the addition of further mechanisms such as simulated annealing [27]. In addition to improvements to the genetic algorithm, the implementation of momentum for the back-propagation would have aided the algorithm in escaping from local minima [14], thereby increasing the efficiency of the training algorithm.

6.4.2 Further Work

Hybrid systems have proven useful for accelerating the learning process of multi-layer nets, however new technologies are emerging that supersede these types of systems. For instance, extreme learning machines that utilise a single layer of hidden nodes with analytically determined output weights perform much faster [28]. Furthermore, deep learning networks, such as Jeff Hawkins Hierarchical Temporal Memory System [29], have been receiving a large amount of attention recently. Based on this, it would appear that further research into this area of study resides within the domain of deep learning and alternative, more efficient methods of machine learning.

6.5 Reflection

Reflection is important because it provides the ability to clarify meaning through the exploration of issues of concern, consequently resulting in an adaptation of conceptual perspective [30].

There is an overwhelming sense of progression as the author reflects on his experience. Initially the author was unfamiliar with the chosen research domain, which is one reason why he found the project so challenging, but was intrigued by the proposition of drawing inspiration from natural phenomena to accomplish a given problem. The most challenging problem was the conceptual understanding and implementation of the back-propagation algorithm, however it proved to be extremely rewarding when accomplished, which took a number of weeks and pushed the author to the very limit of what he thought he was capable of achieving. As a result of this, the author feels as though he has experienced a perspective change: The proposition of a well paid job in a potentially uninspiring and unfulfilling industry has been rendered bleak, whilst the proposal for further education is deemed most desirable. Preferably, the author would

be most excited by the proposition of another undergraduate degree, possibly a joint honours in mathematics and physics, but whilst this remains to be financially infeasible, a well deserved break is sure to be undertaken, followed by the possibility of returning to university to pursue a masters degree, or a PhD.

6.6 Concluding Remarks

This chapter provided an adequate conclusion to the project, in addition to the exploration of potential further work and a reflection on the authors experience.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Microsoft Research*, 2015.

[2] K. T. B. V. B. M. S. Maes, "A hybrid machine learning system for stock market forecasting," *World Academy of Science*, 2008.

[3] J. Elman, *Rethinking Innateness: Connectionist Perspective on Development (Neural Networks Connectionist Modelling)(Neural Network Modelling Connectionism)*. MIT-Press, 1998.

[4] S. R. and P. Norvig, *'What is AI?', in Artificial Intelligence A Modern Approach*. Prentice Hall, 2009.

[5] R. Neville, "Hypothesis testing process, hypothesis-experiment-evaluation loop," -, 2013.

[6] S. R. and P. Norvig, *'Introduction', in Artificial Intelligence A Modern Approach*. Prentice Hall, 2009.

[7] S. R. and P. Norvig, *'Acting humanly: The Turing Test approach', in Artificial Inteligence A Modern Approach*. Prentice Hall, 2009.

[8] S. R. and P. Norvig, *'Preface', in Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[9] A. and. J.E.Smith, *'Chapter One', in Introduciton to Evolutionary Computing*. Springer Science Business Media, 2003.

[10] F. A. Azevedo, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *The Journal of Comparative Neurology*, 2009.

[11] A. and. J.E.Smith, *'Chapter Two, Biological Neural Networks', in A Brief Introduction to Neural Networks*. Unknown Publisher, 2007.

[12] V. Contributors, *Fundamentals of Neuroscience*. Academic Press, 2008.

[13] S. Marsland, *'McCulloch and Pitts Neurons' in Machine Learning, An Algorithmic Perspective*. CRC Press, 2009.

[14] C. Borgelt, "Introduction to neural networks." http://www.borgelt.net/slides/nn.pdf, 2015.

[15] K. Gurney, *'Training TLUs: The Perceptron Rule' in An Introduction to Neural Networks*. UCL Press, 1997.

[16] K. Gurney, *'Multilayer nets and backpropagation' in An Introduction to Neural Networks*. UCL Press, 1997.

[17] Y. Lecuns, "Yann lecun's answers from the reddit ama." http://fastml.com/yann-lecuns-answers-from-the-reddit-ama/, 2015.

[18] W. Trochim and J, *The Research Methods Knowledge Base*. Atomic Dog, 2006.

[19] K. T. B. V. B. M. S. Maes, "Credit card fraud detection using bayesian and neural networks," *Vrije Universiteit Brussel - Department of Computer Science Computational Modelling Lab (COMO), Brussel*, 1993.

[20] M. Rouse, "What is a development environment?." http://searchsoftwarequality.techtarget.com/definition/development-environment, 2007.

[21] Microsoft, "Microsoft.office.interop.excel namespace." https://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel%28v=office.15%29.aspx, 2013.

[22] T. A. University, "Psc 211 data structures implementations." http://faculty.cs.tamu.edu/welch/teaching/211.s03/lnotes1.pdf, 2013.

[23] M. Project, "Home — mono." http://www.mono-project.com/, 2015.

[24] Microsoft, "Introduction to the c language and the .net framework." https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx, 2013.

[25] geeksforgeeks, "Graph and its representations." http://www.geeksforgeeks.org/graph-and-its-representations/, 2015.

[26] UCLA, "Lecture involving design matrices." http://www.stat.ucla.edu/~dinov/courses_students.dir/07/Winter/PN284.dir/NITP_PN_M284_GLM3.pdf, 2015.

[27] M. O. C. Ware, "Lecture 13: Genetic algorithms." http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/, 2011.

[28] Q.-Y. Zhu, C.-K. Siew, and G.-B. Huang, "Extreme learning machine: Theory and applications," *Selected Papers from the 7th Brazilian Symposium on Neural Networks*, 2006.

[29] J. Hawkins, *"Artificial Intelligence' in On Intelligence*. Owl Books, 2005.

[30] W. Fales, "Reflective learning key to learning from experience," *Journal of Humanistic Psychology*, 1983.

[31] T. I. A. Griffin, "Turing test breakthrough as supercomputer becomes first to convince us it's human." `http://www.independent.co.uk/life-style/gadgets-and-tech/computer-becomes-first-to-pass-turing-test-in-artificial-intelligence-mile` `html`, 2014.

[32] B. K. Oakes, "No, a computer did not just pass the turing test." `http://www.buzzfeed.com/kellyoakes/no-a-computer-did-not-just-pass-the-turing-test#.wye6aL797`, 2014.

[33] P. P. SZOLOVITS, P. R. S. PATIL, M., and W. B. SCHWARTZ, *Artificial Intelligence in Medical Diagnosis*. Ann Intern Med, 1998.

[34] S. R. and P. Norvig, *'Machine Translation', in Artificial Intelligence A Modern Approach*. Prentice Hall, 2009.

[35] J. Hawkins, *On Intelligence*. Owl Books, 2005.

[36] J. Carbonell, *'Paradigms for Machine Learning', in Machine Learning: Paradigms and Methods*. MIT / Elsevier, 1992.

[37] C. Stanford University, "Stanford encyclopedia of philosophy: Alan turing." `http://stanford.library.usyd.edu.au/entries/turing/`, 2015.

[38] Landahl, "A statistical consequence of the logical calcuulus of nervous nets," *Bulletin of Mathematical Biophysics*, 1943.

[39] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psycholoigical Review*, 1958.

[40] S. Marsland, *'The Perceptron' in Machine Learning, An Algorithmic Perspective*. CRC Press, 2009.

[41] M. Olazaran, "A sociological study of the official history of the perceptrons controversy," *Social Studies of Science*, 1996.

[42] P. Werbos, "The roots of backpropagation: from ordered derivatives to neural networks and political forecasting," *John Wiley Sons*, 1994.

[43] K. Gurney, *'Chapter 5: The Delta Rule' in An Introduction to Neural Networks*. UCL Press, 1997.

Appendix A

Additional Introduction Information

A.1 Research Relevance Table

Research Relevance Table					
#	Title	c#	NNs	EAs	Relevance
1	K. Gurney, An Introduction to Neural Networks. UCL Press, 1997.		✔		3
2	Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach. Prentice Hall, 2002.		✔		2
3	Jeff Hawkins, On Intelligence. Times Books, 2004.		✔		1
4	J Carbonell, Machine Learning: Paradigms and Methods. MIT Press, 1990.		✔		2
5	A.E. Eiben and J. E. Smith, Introduction to Evolutionary Computing. Springer, 2008.			✔	3
6	Stephen Marsland, Machine Learning: An Algorithmic Perspective. CRC, 2009.		✔		2
7	Griffiths, Programming C# 5.0. OReilly Media, 2012.	✔			2

Table A.1: Research Relevance Table

A.2 Theory Research Relevance Tree

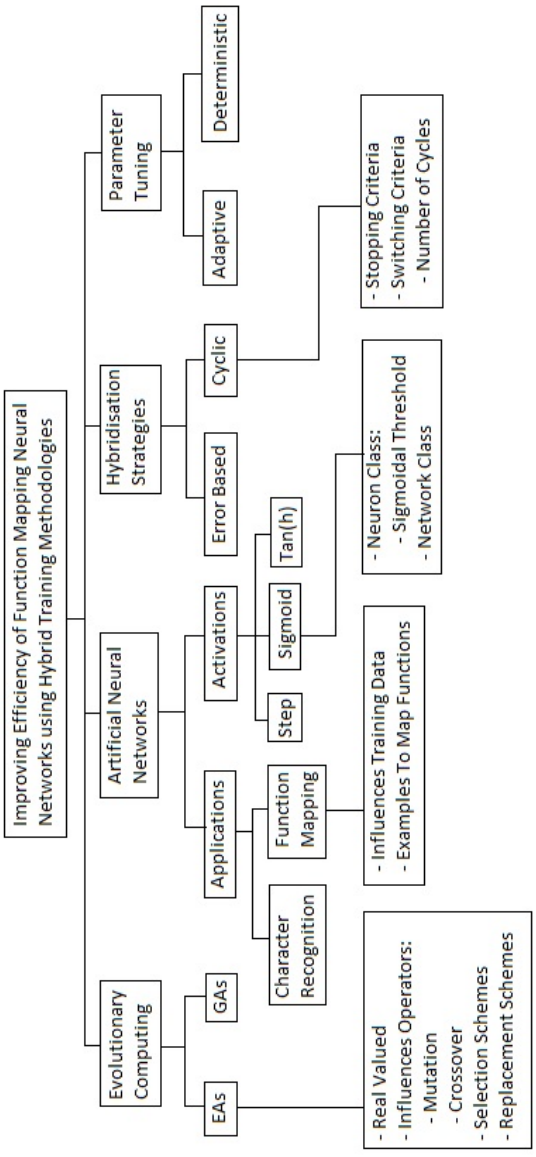

Figure A.1: Illustrates the Theory Research Relevance Tree.

Appendix B

Additional Background and Literature

B.1 Definitions of Intelligence and The Turing Test

Precise definitions of intelligence often vary, for example, the ability to think and to behave in a human manner have both been individually recognised by scholars as a resemblance of intelligence. However, the argument has been put forward that although human beings may be intelligent, artificial intelligence is focused on producing rational[1] thought or behaviour, rather than mimicking human thought or behaviour [4].

Despite the variation in definition, one method that is routinely cited in order to ascertain whether a machine[2] is intelligent or not is famously known as the Turing test. For a machine to pass the Turing test, a human interrogator, after having proposed numerous written questions and having received a response, must not be able to differentiate whether the response came from a human or from a machine [7].

There have been claims that computer programs capable of passing the Turing test exist [31], but these claims are widely in dispute [32], as is the Turing test itself as a means of truly testing intelligence [29].

[1] This isnt to say that humans are inherently irrational, but that humans may behave in an irrational manner on occasion and that behaviour is not a desirable feature of artificial intelligence.

[2] A machine can be thought of as being analogous to a technology or a computer program.

Today, artificial intelligence technologies[3] are prevalent in various industries and provide solutions to common problems such as diagnosing illnesses [33], fraud detection [19] and language translations [34].

B.2 Machine Learning, Applications and Paradigms

Machine learning as a research area overlaps with various concepts from other fields such as statistics, information theory, computation complexity, biology, cognitive science, philosophy and control theory [8].

Applications of machine learning are being used commercially in numerous industries across the world and vary from predicting energy consumption for warehouses[4] [35] to providing systems for stock market forecasting [2].

The field of machine learning can be subdivided into multiple sets of learning paradigms. For instance, learning algorithms which are supervised[5] fall into the inductive paradigm, whereas unsupervised algorithms may belong to the analytical paradigm. Whilst there are methods common to both that may be used in order to solve classification and regression problems, the types of methods can vary significantly. Inductive methods are based on inducing a generalised concept from a sequence of instances of that concept, whereas analytical methods are based on a rich underlying domain theory. Additionally, there exists the connectionist paradigm which encompasses numerous variations of connectionist learning systems, known as artificial neural networks. These machine learning models draw their inspiration from nature by mimicking biological processes that occur in the human brain. This particular paradigm exhibits convenient functional similarities between discriminant learning in inductive systems, in addition to the evolutionary paradigm [36].

[3] Technologies that provide similar capabilities to those identified as prerequisites for passing the Turing test.

[4] If you can predict the energy consumption for a warehouse, you can utilise demand response in order to minimise overheads, which provides the company with a competitive advantage.

[5] Meaning that they are provided with an amount of labelled training data, this training data is essentially instances of similar problems that allow the algorithm to establish a strong ability to identify similar instances of problems.

B.3 The History of Evolutionary Computing

The concept of evolutionary computing was proposed as early as 1948, by Alan Turing and the expression he phrased as genetical or evolutionary search [37]. Research into this area saw further development in the sixties, where three different implementations with the same conceptual underpinning were developed in different countries. These different implementations were eventually all coined under the same name, evolutionary computing [9].

B.4 Neural Networks: The Periphery

The information processing system that humans possess can be separated into two primary components, the central nervous system, which is where information received by the senses is stored and managed, and the peripheral nervous system, whose purpose is to connect the central nervous system to the remainder of the body [11]. The central nervous system can be broken down into two further subdivisions, the spinal cord and the brain.

B.5 The History of ANNs

The earliest computational model of a neuron dates to 1943 when McCulloch and Pitts proposed the threshold logic unit [38], which paved the way a continuation of research into this area of artificial intelligence and lead to the conception of the perceptron in 1958 by Frank Rosenblatt [39]. The perceptron is a linear classifier, since it comprises a set of input nodes connected to threshold logic units using weighted connections [40] and was initially thought to be the next step towards creating genuine artificial intelligence [41]. It was soon discovered that the perceptron could only classify a small number of patterns and research in this area greatly diminished until the discovery of backpropagation[6] by Paul Werbos, which illustrated that artificial neural networks could be trained to solve a larger number of more complex problems, such as the exclusive-or problem [42].

[6] Back-propagation is an algorithm that relies on the backward propagation of error based on the derivative of the activation function.

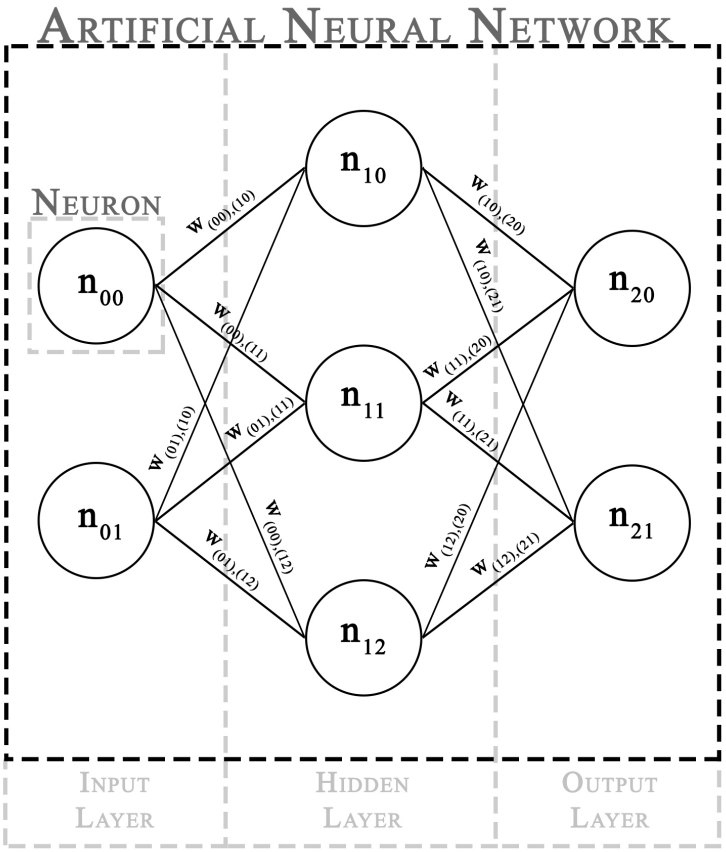

Figure B.1: An intuitive depiction of an ANN.

B.6 Further Representation of ANNs

The above figure provides an illustration that maps the mathematics outlined in chapter two to a visual representation of the model. The entire network is first divided into its constituent layers, which are labelled underneath the network boundary. Each layer contains some number of neurons, indexed in subscript by the layer number, followed by the node number:

$$n_{(layernumber)(nodenumer)} \tag{B.1}$$

The weights for the system are represented in the following fashion:

$$(source, target) \in C \tag{B.2}$$

B.7 The Perceptron Rule: Further Depth

A perceptrons ability to perform some classification is determined by its associated weight vectors and threshold values, since they are the only attributes that the output relies on. It is therefore necessary to adjust these values to bring the functionality of the perceptron into alignment with the desired functionality. This alignment is accomplished through an iterative process, known as training, which updates the weights and thresholds[7] through the presentation of examples from the training set[8].

B.8 The Delta Rule: Derivation

In order to update each weight to minimise the cost function, a calculation must be performed to establish the contribution a particular weight, in a particular node, has made to the error. To establish this contribution, let us first consider the cost function for any given node, defined by the following equation[9]:

$$e_j^p = \frac{1}{2}(t_j^p - z_j^p)^2 \tag{B.3}$$

The error contribution made by a particular weight is defined by observing the change in error with respect to the change in a particular weight, which is defined by the following differential equation:

$$\frac{\delta e_j^p}{\delta w_{ji}} = \frac{\delta e_j^p}{\delta z_j} \times \frac{\delta z_j}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} \tag{B.4}$$

The above equation represents the change in error for a particular node j, for a pattern p, with respect to a weight that belongs to that node i. In order to break down the equation, consider that the error is measured by the cost function:

$$e^p = \frac{1}{2} \times \sum_{j=1}^{N} (t_j^p - z_j^p)^2 \tag{B.5}$$

[7] The thresholds can also be treated as a weights.

[8] Training data sets define multiple instances of a type of problem in the forms of labelled data

[9] It should be noted that this is a convenient cost function for a two reasons in particular: any negative values for (t_j^p - z_j^p) are squared, so they become non-negative and the in front of the equation makes for mathematical convenience as is about to be show.

Which depends on the weight through the activation z_j:

$$z_j \equiv \varphi(net_j) = \frac{1}{1+e^{x \times net_j}} \tag{B.6}$$

Additionally, the activation depends on the summation of the inputs and the weights net_j:

$$net_j = \sum_{i=1}^{n} x_i w_{ij} \tag{B.7}$$

Which in turn is partially reliant on the weight w_{ji}. The partial derivate of the error with respect to the activation is defined as follows:

$$\frac{\delta e_j^p}{\delta zj} = (t_j^p - z_j) \tag{B.8}$$

The partial derivate of the activation with respect to the summation of weights and inputs[10]:

$$\frac{\delta z_j}{\delta net_j} = z_j(1-z_j) \tag{B.9}$$

The partial of the summation of weights and inputs with respect to the weight in question:

$$\frac{\delta net_j}{\delta w_{ji}} = x_{ji} \tag{B.10}$$

When each component of the differential equation B.4 is calculated and substituted in, the following equation is produced:

$$\frac{\delta e^p}{\delta w_{ji}} = (t_j^p - z_j) \times z_j(1-z_j) \times x_{ji}^p \tag{B.11}$$

The weight update should be proportional to the contribution that particular weight has made to the error, such that:

$$\Delta w_{ji} = -\alpha \times \frac{\delta e_j^p}{\delta w_{ji}} \tag{B.12}$$

[10]It should be noted that this is the derivative of the sigmoid function used to characterise the threshold.

The alpha term in the above equation defines the step size and the negativity of this term ensures gradient decent rather than ascent. By substituting equation B.11 into equation B.14, and by encapsulating the negativity into the alpha term, we are given the following equation:

$$\Delta w_{ji} = \alpha \times (t_j^p) \times z_j(1 - z_j) \times x_{ji}^p \tag{B.13}$$

Conventionally, the derivative of the activation function (the sigmoid function) is denoted by a lower case sigma, followed by an apostrophe as shown [43]:

$$\frac{\delta z_j}{\delta net_j} = \sigma' net_j \tag{B.14}$$

This changes the weight update to the following:

$$\Delta W_{ji} = \alpha \times (t_j^p - z_j) \times \sigma'(net_j) \times x_{ji}^p \tag{B.15}$$

Generally, in the case of a single layer net, with N nodes and a set of weights for each node w_{ji} with a co-responding training set P:

$$\Delta W_{ji} = \alpha \times (t_j^p - z_j) \times \sigma'(net_j) \times x_{ji}^p \tag{B.16}$$

Appendix C

Additional Design Details

C.1 The Evaluation Platform: Sub-Routines

The evaluating platforms subroutines comprise of the following:

SR01, Results Extraction: takes the data generated by the training and testing platform and pushes it into either:

1. A raw text file; if this is the case, the routine exits here and the remaining subroutines must be executed manually.
2. An Excel spreadsheet.

SR02, Key Information Analysis: Excel analyses the extracted results data and produces associated key information that can be used to describe the performance of the employed training methodology.

SR03, Visualisation: numerous preliminary graphs are automatically rendered which display plots representing the lower and upper bound values, in addition to the average values. These are used as aids to compare different experiments.

C.2 Further Reading on Project Analysis

C.2.1 Construction of the Training and Testing Platform

The first primary requirement involves developing the underlying platform on which the learning algorithms and training methodologies will be executed. The functionality associated with this platform embodies a number of secondary requirements that must be met:

1. The platform must be able to generate a fully connected, feed forward neural network, specified to any dimensionality. Dimensionality in this context refers to the number of hidden layers and the number of nodes in each layer.
2. The platform must be capable of producing a set of training data according to the initial parameters specified by the user.
3. The nodes in each layer must be customisable such that different activation functions and activation function parameters can be adjusted using the initial parameters specified by the user. Learning algorithms (or training models) must be deployable by the platform, such that network training can be undertaken and neural weights can be modified.
4. The platform must be capable of recording and storing data related to the error of the system.

C.2.2 Implementation of the Learning Algorithms

The second primary requirement encompasses the implementation of the two principal learning algorithms adopted by the research project:

1. An evolutionary algorithm.
2. The back-propagation algorithm.

In order to ensure the functionality of these algorithms and in addition to the ability to perform the project analysis; there exist a number of secondary requirements that must be satisfied:

1. An iteration performed by either algorithm must not result in an increase in overall error of the system.

2. Each algorithm must be parameterised, such that the user has the ability to alter the conditions under which each algorithm is executed.
3. In addition to these secondary requirements, it is advantageous to perform a sensitivity analysis for each algorithm. This sensitivity analysis varies specified parameters of a selected algorithm and measures the performance under these varied conditions. Once observed, an educated decision to select a particular value for a particular parameter allows each algorithm to be tuned, which is responsible for optimising its performance.

C.2.3 Implementation of the Control Mechanism

The third primary requirement is the implementation of the control mechanism that is responsible for switching between the two principal learning algorithms. The secondary requirements incorporate a number of properties that the control mechanism must satisfy to support the integrity of the project analysis, these properties are listed below:

1. The ability to initialise any adopted learning algorithm in accordance with the parameters specified by the test harness must be ensured by the control mechanism.
2. The control mechanism must be able to iteratively execute any of the principal learning algorithms adopted by this project.
3. The capability of switching between different learning algorithms according to some criteria specified by the test harness must be present.
4. The control mechanism must be capable of retaining the state of the system (neural weights) during each switch between any adopted learning algorithms.

C.2.4 Construction of the Evaluation Platform

The final primary requirement embodies the construction of the evaluation platform. The evaluation platform is integral to the analysis process, since it provides an outlet for the visual representations of the data generated by the training process; its functionality is therefore pivotal to the project as a whole. This functionality is characterised by the following set of secondary requirements:

1. The evaluation platform must have access to and have the ability to withdraw error data from the data structure that is responsible for logging the system error.

2. The evaluation platform must have access to Excel in order to push any analysis data directly to a spreadsheet.
3. The ability to perform calculations on any error data must be ensured by the evaluation platform.
4. The platform must be able to produce a set of preliminary graphs that describe the performance of the adopted training methodology.

C.2.5 Summary

To conclude, this project aims to analyse the efficiency of training fully connected feed forward neural networks using two principal learning algorithms, in conjunction with a control mechanism, used to switch between each algorithm. It does this in order to collect evidence, via an analytical procedure, to test the initial hypotheses. The embodiment of the project depends on a number of requirements that have been outlined throughout this chapter, which provides a foundational basis for the implementation of the project.

For a table that conveys the requirements outlined in this section see **Appendix C.3**.

C.3 Project Analysis Requirements Table

Project Analysis Requirement	
Primary Requirement	Secondary Requirement
Construction of Training and Testing Platform	1. The platform must be able to generate fully connected feed forward neural networks, specified to any dimensionality. 2. Training data generation must be driven by initial parameters specified by the user. 3. Node activation function must be customisable. 4. Learning algorithms must be deployable on top of the platform.
Implementation of Training Algorithms	Training algorithms to be implemented: 1. Genetic Algorithm. 2. Backpropagation Algorithm. Algorithmic requirements: 1. Iterations must not result in an increase in error. 2. Parameters must be modifiable by initial values specified by the user. In addition, a sensitivity analysis by varying the modifiable parameters of each algorithm would provide learning optimisation.

Implementation of Control Mechanism	1. Must be able to initialise learning algorithms in accordance with initial parameters specified by user. 2. Must be capable of executing any adopted learning algorithm. 3. Capability to switch between algorithms whilst retaining system state must be present.
Construction of Evaluation Platform	1. Must be capable of withdrawing error data from training and testing platform. 2. Must be able to push data into a text file or Excel. 3. The ability to perform calculations on error data in order to produce useful system performance information must be present. 4. The ability to produce preliminary visualisations must be available.

Table C.1: Project Analysis Requirement Table

Table C.1 concisely demonstrates the research projects analysis requirements in an easily digestible fashion. The leftmost column references each of the primary requirements (§ 3.3), whereas the right most column discusses each of the secondary requirements that embody their associated primary requirement.

C.4 Generic Experiment Design

Each experiment to be undertaken aims to generate an artificial neural network based on a set of specified network parameters describing network representation; and then train the network by utilising a specified training methodology. The adjustable network

parameters have previously been tuned by a sensitivity analysis. During the course of each experiment, the error of the network will be recorded as the neural weights are modified and the evaluation platform will be employed as a means of outputting or visualising the training methodologies performance.

C.5 Generic Experiment Data Structures and Algorithms

This section briefly outlines the design concepts on which each of the data structures utilised by the generic experiment workflow (§3.1.1) are implemented.

Each experiment initialises a number of models during the system initialisation routine (§3.1.2), these models provide the underlying structure on which training algorithms can be deployed on during the system training routine (§3.1.2). Since each experiment is performed on the same platforms, the data structures and algorithms utilised to build and train the underlying model can remain consistent between experiments.

In addition, each experiment possesses some number (dictated by the initialisation parameters) of training examples, which describe a mathematical function. Each training example fundamentally comprises of two principal vectors, stored as arrays, the first of which defines the input values for the network; the second of which defines the expected output given that particular input.

These training examples are themselves held within an array; an array within an array can be conceptualised as a matrix, this form of data representation is known as a design matrix [26], the implementation of this concept in the project is described in further detail in chapter four.

C.6 Experiment Functionality Workflow Table

The following table describes the overall functionality of the experiments listed above, as a whole, but only considers the entry and end points such that the experiments can be thought of as a black box.

Experiment One Workflow Functionality	
Input	Set of parameters describing the initial state of the system: 1. Network Representation. 2. Training Data. 3. GA and BP Parameters. 4. Control Mechanism Parameters. 5. Display Parameters.
Output	Outputs either: 1. Text File 2. Spreadsheet containing key information analysis and preliminary plots.

Table C.2: Generic Functionality of Experiments.

Table C.2 provides an intuitive, generic representation of each experiment as a function that accepts a number of specified input parameters, in addition to a set of outputs, the descriptions of which are found in the rightmost column of the table.

C.7 Experiment Platforms Table

<table>
<tr><td colspan="3">Training and Testing Platform</td></tr>
<tr><td>Routine (#RT-N)</td><td colspan="2">Sub-Routines</td></tr>
<tr><td>System Initialisation (#RT01)</td><td colspan="2">1. Load initial parameters into the programme, either from a file or using a pre-programmed test harness. These parameters are outlines in table 2.
2. Set initial parameters, such that the system is ready to generate the models required for the experiment.
3. Initialise the neural network model, specified to the dimensionality given by the set of initial parameters.
4. Produce a set a training data for the model, using the initial parameters to define:
(a) Mathematical function.
(b) Number of training examples.
5. Initialise the control mechanism and hand over the programme control flow to this object.</td></tr>
<tr><td rowspan="2">System Training (#RT02)</td><td>Exp1</td><td>Initialise and execute genetic algorithm until the stopping criteria is met.</td></tr>
<tr><td>Exp2</td><td>Initialise and execute the back-propagation algorithm until the stopping criteria is met.</td></tr>
</table>

	Exp3	1. Inspect the set of initial parameters in order to deduce which algorithm is to be initialised first. 2. Iteratively execute this algorithm until the switching criteria is met. 3. Perform a check to determine if the stopping criterion has been met; if so branch to the next routine, if not, switch algorithm. Note: The stopping criteria will only be met once the switching criterion is also met.
Data Recording (#RT03)	Make system error data available to the output results routine on the next platform.	

Table C.3: Routine description for the Training and Testing Platform

Evaluating Platform	
Routine	Sub-Routines
Output Results (#RT04)	1. Access the system error data from the data structure provided by the previous platforms routine. 2. Inspect the set of initial parameters in order to deduce whether the data should be dumped to a file or pushed into excel. (a) If the data is to be dumped to a file, perform data dump and exit programme. 3. Push system error data to excel. 4. Perform key information analysis to generate a set of lower and upper bound values, in addition to a set of average values. 5. Use the key information generated by the previous sub-routine to plot preliminary graphs that describe the performance of the system under the employed training methodology.

Table C.4: Routine description for the Evaluating Platform

Table C.3 and table C.4 are broken down into two main columns; the leftmost provides an identifier that links the description, found in the rightmost column, to a routine that has been previously outlined in (§ 3.1). In addition, the table is further divided into two principal sections which provide the details relating to the training and testing platform and the evaluating platform respectively. The first principal section is denoted by the horizontal row at the top of the table and the second is denoted by the horizontal row found towards the bottom of the table. These are captioned Training and Testing Platform and Evaluating Platform respectively.

Appendix D

Additional Implementation Details

D.1 Generic Experiment Implementation Methodology

Since each experiment is performed on the same platform, the data structures and algorithms employed in order to generate a set of experiment results for the project analysis can remain consistent between each experiment. This is an intrinsic property of the adopted project structure (§ 3.1).

D.2 Data Structures and Algorithms Overview

There exist a number of constituent mathematical models that describe the behaviour of each component[1] within the project, when aggregated together these components fulfil the requirements demanded by the project analysis (§ 3.3). To elaborate, the implementation of each component enables the execution of the workflow (§ 3.1.1) and allows for the physical manifestation of each experiment, such that a set of results may be produced and analysed in order to extract meaningful information.

In order to implement each component, the mathematical models that express them must (§ 2.5) be examined and translated into a set of data structures and algorithms.

The purpose of each data structure is to provide a means of representing the current state of a particular system component by enabling the storing and accessing of data

[1] A data component can be thought of as an individual project class, which may or may not be a data structure itself, but may attribute towards the implementation of a data structure or any particular project model.

associated with that component[2] [22]. To satisfy these characteristics, every data structure possesses numerous functions that provide the ability to deposit, withdraw or modify data associated with that particular structure.

Algorithms are implemented on top of data structures, such that they provide some additional functionality that utilises the data structure.

D.3 Implementation Specifics for Neurons and ANNs

The implementation details for the data structures associated with the neuron model and the artificial network model[3] are outlined throughout this appendix and are demonstrated in the following set of tables and figures.

Principal Network Model		
Data Structure	Data Attribute	Description
Network $\{2\}$	[][] neuralNetwork	A multi-dimensional array that contains a reference to every neuron in the network. This array is indexed by passing the layer identifier[4] and the neuron identifier[5] as the first and second index of the array respectively.
Neuron $\{1\}$	[] inputs (double)	A multi-dimensional array that contains a reference to every neuron in the network. This array is indexed by passing the layer identifier1 and the neuron An array that contains the value for each input to this neuron[6].

[2]Similarly, to consider the aggregation of all data structures is to consider the state of the system in its entirety.

[3]Analoges to the the principal network model.

[4]The layer identifier is an index that ranges between 0 and the number of layers 1.

[5]The neuron identifier is an index is assigned to each node in any particular layer and ranges between 0 and the number of nodes in that layer1.

[6]If this neuron belongs to a hidden layer or an output layer, the inputs translate to the outputs from the previous layer.

	[] weights (double)	An array that contains the value of each weight for each corresponding input.
	output (double)	The output value of the node (calculated during a function call to the node[7]) is stored in this variable.
	outputTransferFunction (enum)	An enumerated type where each enumeration describes a different transfer function, these functions are described below: 1. Input: only set for input nodes and defines the activation transfer function as an identity function. 2. Sigmoid: set for hidden nodes and output nodes, this defines the activation transfer function as the following[8]: $f(x,\gamma) = \frac{1}{1+e^{\frac{-cx}{\gamma}}}$ (D.1)
	prime (double)	The γvalue to be passed into the activation output transfer function as a parameter.
	layerID (int)	A value that identifies the layer which this neuron belongs to.
	nodeID (int)	A value that identifies the position of this neuron in any given layer.
	isInputNode (boolean)	A true of false value which indicates whether or not this neuron belongs to the input layer of the network.

[7]This is explained in further detail in the subsequent part of this section.

[8](x and $\gamma$are function parameters, whereas c is some constant)

	[] adjacencyList (Neuron)	An array that contains a reference to each of the neuron objects adjacent to this neuron object.
	partial (double)	During back-propagation, the value calculated as the partial derivative of this nodes activation transfer function is stored here.
	[] delta (double)	During back-propagation, the amount by which each weight should be altered during any iteration is stored here in this array.

Table D.1: Describes the data attributes that belong to each data structure.

Table D.1 demonstrates the constituent data structures that characterise the principal network model (an ANN). The table contains three columns, the first of which defines the data structures name and can be linked to a figure by the reference supplied in curly braces to the right of each structure name. The second column provides a definition for each data attribute[9] encapsulated by its associated data structure. These definitions provide an indication as to whether that component is an array, indicated by a prefixed set of square braces, or a multi-dimensional array, indicated by prefixed multiple sets of square braces[10]. The word directly after the prefix provides the name of the component and the suffix, in brackets, defines the data type. The final column in the table provides a description for each data attribute which outlines the attributes purpose, in addition to any further information that may be associated with that attribute.

Each data structure provides a number of functions to store, access and modify each data attribute, these functions are provided in the table below with their respective parameters.

[9] A data attribute, in this context, would be any variable stored in a data structure that attributes to the state of the system and may be or may not be used in order to perform a translation from the current state to another state.

[10] The number of prefixed sets of square braces indicates the number of dimensions represented by that array.

Principal Network Model			
Data Structure	Function	Parameters	Return Type
Network {2}	initNetwork	#ofLayers (int) #ofInputNodes(int) #ofHiddenNodes(int) #ofOutputNodes(int)	void
	addLayer	#ofNodesInLayer(int)	void
	generateFullConectivity	-	void
	setInputVector	[] input (double)	void
	initWeights	-	void
	propagateForward	[] input (double)	void
	generateOutputVector	-	[]double
	Getters & Setters	*	*
Neuron{1}	productSummation	-	double
	outputTransfer	-	double
	addConnection	adjacentNode(Neuron)	void
	calculatePartialOutputNode	target(double)	double
	calculatePartialHiddenNode	-	double
	calculateDelta	learningRate(double)	[]double
	updateWeights	-	void
	Getters & Setters	*	*

Table D.2: Outlines each function associated with its respective data structure.

Table D.2 provides definitions for each function associated with its respective data structure. The first column provides the name of the data structure and can be linked to its associated figure by the number in the curly braces to the right of the name. The second column defines the name of each function that the structure provides. The third column indicates the parameters that each function accepts (these are described in the same fashion as the data attributes in table 1) and the final column provides the return type.

Principal Network Model		
Data Structure	Function	Description
Network $\{2\}$	initNetwork	Generates a fully connected feed-forward neural network with a specified number of layers, input nodes, hidden nodes and output nodes, as defined by the parameters. This is accomplished by: 1. Incrementally iterating from zero up until the value that has been passed in which defines the total number of layers. 2. Making a call to addLayer, passing the specified number of nodes for that particular layer as the parameter. 3. Making a call to initWeights. 4. Making a call to generateFullConectivity.
	addLayer	Adds an additional layer to the network containing the number of nodes defined by the parameter that has been passed in.
	generateFullConectivity	Iterates through every layer, connecting every node in each layer with every node in the next layer.
	setInputVector	Clamps a set of input values represented as an array to the input layer.
	initWeights	Randomly initialises each weight in the neural network according to a Gaussian distribution.

	propagateForward	Clamps an input vector to the input layer and iterates through each layer, activating every neuron in that layer, such that each neuron produces an output which can be fed into the next layer of neurons. When this reaches the output layer, the function exits.
	generateOutputVector	Takes the output values from the output nodes and returns them in the form of an array.
	Getters & Setters	Functions are provided to retrieve and modify every data attribute illustrated in table D.1.
Neuron{1}	productSummation	Multiplies the input vector by the weight vector and returns the summated result.
	outputTransfer	Passes the result form productSummation through the activation transfer function and returns the result.
	addConnection	Adds a reference to another neuron to its adjacency list.
	calculatePartialOutputNode	Calculates the resultant value of the partial derivative of the activation transfer function assigns this value to the partial data attribute and subsequently returns this value, based on the assumption that this node is an output node.

	calculatePartialHiddenNode	Calculates the resultant value of the partial derivative of the activation transfer function assigns this value to the partial data attribute and subsequently returns this value, based on the assumption that this node is a hidden node.
	calculateDelta	Calculates the change in weight for each value in the weight vector, stores this as an array, assigns these delta values to the delta data component $\{\}$ and then returns the array.
	updateWeights	Modifies each value in the weight vector by adding the values from the delta data component to their respective weight values.
	Getters & Setters	Functions are provided to retrieve and modify every data attribute illustrated in D.2.

Table D.3: Outlines each function associated with its respective data structure.

Table D.3 follows the same layout as table D.1, except the second column represents functions associated with the data structure that have been outlined in table D.2 and not data attributes.

D.4 Representation of Training Examples

Figure D.1 provides a visual representation of the data structure used to contain any specified function mapping for the purpose of supervised training. The specified function mapping is represented by a number of training examples, n, contained within an array, portrayed by the square brackets on the left hand side of the figure. Each training example is an object which itself contains, and provides access to, two arrays; the first

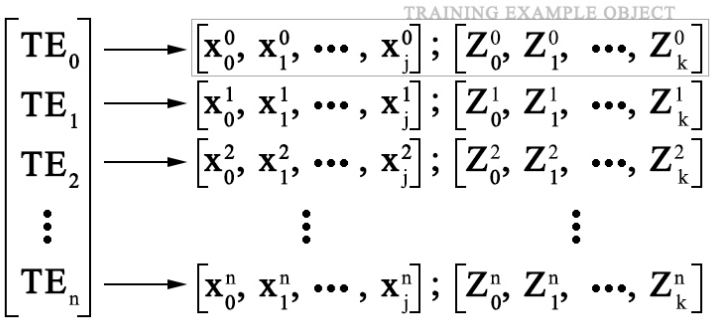

Figure D.1: Illustrates the generic data structure used for each experiment.

of which defines the inputs that should be presented to the network (the input vector, denoted by x) and the second of which defines the desired outputs, given those inputs (output vector, denoted by z). The subscript on each input and desired output defines which node that particular value belongs to in the input and output layers respectively, where j represents the number of nodes in the input layer and k represents the number of nodes in the output layer. Similarly, each input and output attribute has a superscript value which denotes the training example that it belongs to; this enables the conceptualisation of this structure in the form of two independent design matrices [26] as is illustrated in the figure below.

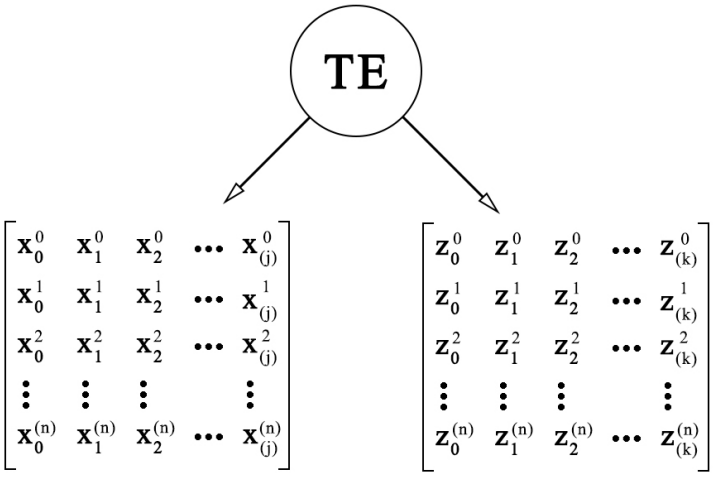

Figure D.2: Demonstrates that the training data can be conceptualised using two design matrices.

Figure D.2 shows that the set of training examples is made up of two primary matrices, the first of which, presented on the left hand side of the figure, defines the set of inputs

that may be presented to the network, where each row represents a single training example[11] and each entry in that row is related to an associated input node[12]. It is also of importance to note that the number of rows, n, represents the number of training examples in the training set. Similarly, for every set of inputs there exists a set of desired outputs, defined by the matrix on the right hand side of the figure, the representation of which follows the same fashion as the input matrix.

D.4.1 Representation of TEs: Implementation Specifics

The implementation details for the training example data structure is outlined in the following set of tables.

Training Data Representation		
Data Structure	Data Attribute	Description
Training Example	[] Inputs (double)	The set of input values for this particular training example.
	[] outputs (double)	The set of desired output values, given the set of input values for this particular training example.
Data	[] examples (TrainingExample)	The entire set of training examples that describe some particular function mapping.
	functionType (enum)	The functionType that this variable is set to describes the function mapping that the set of training examples defines.

Table D.4: Describes the data attributes that are associated with representing an training data.

[11] The training example that each matrix entry is associated with is denoted by the superscript value of that matrix entry.

[12] The input node that each matrix entry is associated with is denoted by the subscript value of that matrix entry.

Table D.4 provides a description for every data attribute associated with the training data representation, in the same style as adopted by table D.1. The enumerated type functionType has the following enumerated values:

1. UNDEF: undefined function.
2. INV: f(x) = -x.
3. SQUARE: f(x) = x^2.
4. TWOX: f(x) = 2x.
5. COSXSINY: f(x,y) = sin(x) * cos(y).

Training Data Representation			
Data Structure	Function	Parameters	Return Type
Training Example	Getters & Setters	*	*
Data	DynamicallyGenerate	functionType(enum) #ofTrainingVectors(int) #ofInputsPerFuncParams(int)	void
	Getters & Setters	*	*

Table D.5: Outlines each function associated with its respective data structure.

Table D.5 provides definitions for each function associated with its respective data structure in the same format as Table 2 (refer to the description of table 2). The Getters Setters functions are provided to retrieve and modify every data attribute contained within table 4; whilst the DynamicallyGenerate function provides a means of generating a number (#ofTrainingVectors) of training examples, that map some function (functionType), such that the entire set of training examples that are generated provide an approximation to the function that has been selected to be mapped. The #ofInputsPerFuncParam parameter is used to define the number of input nodes that should be used to clamp each functionType parameter to. For example, if functionType is set to COSXSINY and #ofInputsPerFuncParam is set to 2, there should be four input nodes, and the first two would be assigned the value x, whilst the second two would be assigned the value y.

The Data data structure can then be used during supervised training to clamp input vectors to the input layer and to compare the actual output of the network with the desired output of the network.

D.5 Representation of Initialisation Parameters

Initialisation Parameters contained within Parameters Data Structure		
Parameter Category	Parameter[13]	Description
Representation	layers (int)	Total number of layers to be present in the artificial neural network.
	inputs (int)	Number of nodes in the input layer.
	hidden (int)	Number of nodes in the hidden layer.
	output(int)	Number of nodes in the output layer.
Training Data	functionType (enum)	Mathematical function that the entire training set must approximate.
	numberOfExamples (int)	Number of training examples that should make up the training set.
	inputsPerParameter (int)	Number of input nodes per function parameter (functionType).
Genetic Algorithm	populationSize (int)	Size of the initial pool of individuals.
	maxMutations (int)	Maximum number of mutations per generation.
	maxCrossovers (int)	Maximum number of crossovers per generation.
	probOfMutation (double)	Probability of mutation at every instance in which a new individual is produced.
	probOfCrossover (double)	Probability of crossover at every instance in which a new individual is produced.

[13] data attribute.

Back Propagation	learningRate (double)	Step size that is utilised during gradient decent[14], as the error is minimised.
	sigmaPrime (double)	γvalue to be passed into the activation output transfer function as a parameter.
Controller	initTrainingMethod (enum)	Initial training algorithm to use, enumerated values are: 1. GA. 2. BP.
	totalCycles (int)	Total number of cycles until the experiment terminates.
	switchCylces (int)	Number of cycles before switching to alternate learning algorithm.
	additionalParams (bool)	Enable or disable the customisation of probOfMutation and probOfCrossover. Having this value set as false will set probOfMutation to its default: 0.25 and probOfCrossover to its default: 1.
	numberOfExperiments (int)	How many times the experiment should be repeated.
Displayer	initRow (int)	Initial position that the row pointer[15] is pointing at when dumping data into Excel.
	initCol (int)	Initial position that the column pointer is pointing at when dumping data into Excel.
	graphLeft (int)	Margin on the left side of the page when generating graphs in Excel.

[14] If the step size is too big, the system will oscillate, if the step size is too small, backpropagation can take a long time to converge. This will be explained in further detail in the algorithms part of this section.

[15] The row pointer instructs the program which row the current piece of data should be outputted to.

	graphTop (int)	Margin at the top of the page when generating graphs in Excel.
	graphWidth (int)	Width of the graphs to be generated in Excel.
	graphHeight (int)	Height of the graphs to be generated in Excel.

Table D.6: Defines and provides a description for each parameter contained within the parameters data structure.

Table D.6 illustrates each of the parameters contained within the parameters data structure that may be passed to the initialisation routine during the execution of any given experiment. The first column provides a category which each set of parameters belongs to[16], the second column defines the parameter data type in a similar fashion to previous tables encountered in this section and the final column outlines functionality of each parameter. The 'Representation and Training data categories relate to concepts that have been discussed previously in this section, however the remaining categories are to be explained in further detail in the next set of sections.

D.6 Individual Data Type

The 'Individual' data type provides the ability to encode the real valued system parameters[17] into an object, such that each evolutionary operator 2.6.4 can be performed on any individual.

[16]These categories translate to levels within the XML document that may be parsed.
[17]This equates to the weight vectors for the entire principal network model.

D.7 EA: Algorithmic Parameters and Return Values

Evolutionary Algorithm Parameters	
Parameter Name	Description
maxIndividuals	Number of individuals to be initialised in the population pool.
numCylces	Amount of times the genetic operators are to be applied to the population.
Network	An instantiated artificial neural network.
trainingData	The set of training data for this experiment.

Table D.7: Provides an overview of each parameter accepted by the implemented evolutionary algorithm.

The leftmost column of table 7 provides the parameter name, which can be linked to the parameters listed in algorithm 3; the rightmost column provides information related to the use of each parameter.

In addition, algorithm 3 returns a value, the data type of which is Individual. This allows the algorithm to neatly package and passes back the set of neural weight associated with the fittest individual to the initial function call.

D.8 EA: Behaviour of High Level Functions

Evolutionary Algorithm Functions	
High Level Function	Description
initIndividuals	Initialises a pool of individuals to the size of the specified parameter. The set of weights encoded within each individual are initialised using the same Gaussian distribution selector that initWeights uses to initialise the network model.
deemUnfit	All individuals in the pool are set as being unable to survive into the next generation.
evaluateFitness	The fitness of each individual is generated by evaluating the error metric for the network and taking its inverse.
retainElitest	Some percentage (described by the second parameter passed to this function) of the fittest individuals are selected to survive into the next generation.
survivalLottery	50% of random individuals are selected to survive into the next generation for the purpose of diversity.
selectionOnNonSurvivors	Any member of the population that is set to not survive is entered into a tournament selection.
produceNextGeneration	A new generation of individuals is produced based on the selected individuals who are fit for survival into the next generation.

Table D.8: Illustrates the behaviour of each high level function adopted by the implemented evolutionary algorithm.

D.9 BP: Parameters

Back-propagation Algorithm Parameters	
Parameter Name	Description
network	An instantiated artificial neural network.
trainingData	The set of training data for this experiment.
learningRate	The size of the step taken every time the algorithm iterates.

Table D.9: Provides an overview of each parameter accepted by the implemented back-propagation algorithm.

Since the implemented back-propagation algorithm is working with an instantiated artificial neural network, numerous variables that may be required to execute the training algorithm can be accessed through this network object, rather than passed via parameters.

D.10 Experiment One: Parameters Table

Parameter Category	Parameter Name	Parameter Value
Network Representation	Number of Layers	3
	Number of Input Nodes	2
	Number of Hidden Nodes	2
	Number of Output Nodes	1
Training Data	Training Function	x^2
	Number of Training Examples	10
	Number of Inputs per Function	2
Genetic Algorithm	Population Size	[100]
	Initial Population	Random, Gaussian
	Maximum Mutations	1.0
	Maximum Crossovers	1.0
	Probability of Mutation	0.25
	Probability of Crossover	0.75
	Selection Scheme	Tournament Selection (TS)
	Replacement Scheme	Generational Elitist
Backpropagation	Learning rate	-0.1
	Activation Function	sigma-prime
	Sigma Prime Value	0.15
Controller	Initial Training Method	GA
	Total Cycles	350
	Cycles between each Switch	350
	Use Additional Parameters	false
	Number of Experiments	10
Displayer	Initial Row Pointer	1
	Initial Column Pointer	1
	Left Margin	10
	Top Margin	80
	Graph Width	300
	Graph Height	250

Table D.10: Initialisation parameters associated with experiment one.

D.11 Experiment Two: Parameters Table

Parameter Category	Parameter Name	Parameter Value
Network Representation	Number of Layers	3
	Number of Input Nodes	2
	Number of Hidden Nodes	2
	Number of Output Nodes	1
Training Data	Training Function	x^2
	Number of Training Examples	10
	Number of Inputs per Function	2
Genetic Algorithm	Population Size	[100]
	Initial Population	Random, Gaussian
	Maximum Mutations	1.0
	Maximum Crossovers	1.0
	Probability of Mutation	0.25
	Probability of Crossover	0.75
	Selection Scheme	Tournament Selection (TS)
	Replacement Scheme	Generational Elitist
Backpropagation	Learning rate	-0.1
	Activation Function	sigma-prime
	Sigma Prime Value	0.15
Controller	Initial Training Method	BP
	Total Cycles	350
	Cycles between each Switch	350
	Use Additional Parameters	false
	Number of Experiments	10
Displayer	Initial Row Pointer	1
	Initial Column Pointer	1
	Left Margin	10
	Top Margin	80
	Graph Width	300
	Graph Height	250

Table D.11: Initialisation parameters associated with experiment two.

D.12 Experiment Three: Parameters Table

Parameter Category	Parameter Name	Parameter Value
Network Representation	Number of Layers	3
	Number of Input Nodes	2
	Number of Hidden Nodes	2
	Number of Output Nodes	1
Training Data	Training Function	x^2
	Number of Training Examples	10
	Number of Inputs per Function	2
Genetic Algorithm	Population Size	[100]
	Initial Population	Random, Gaussian
	Maximum Mutations	1.0
	Maximum Crossovers	1.0
	Probability of Mutation	0.25
	Probability of Crossover	0.75
	Selection Scheme	Tournament Selection (TS)
	Replacement Scheme	Generational Elitist
Backpropagation	Learning rate	-0.1
	Activation Function	sigma-prime
	Sigma Prime Value	0.15
Controller	Initial Training Method	GA
	Total Cycles	350
	Cycles between each Switch	10
	Use Additional Parameters	false
	Number of Experiments	10
Displayer	Initial Row Pointer	1
	Initial Column Pointer	1
	Left Margin	10
	Top Margin	80
	Graph Width	300
	Graph Height	250

Table D.12: Initialisation parameters associated with experiment one.

Appendix E

Additional Analysis Details

E.1 Sensitivity Analysis for Experiment Two

The parameterisation of the backpropagation algorithm requires a sensitivity analysis in order to calibrate the prime value1 for the activation function. After performing this analysis an educated decision was made to set the value to 0.15, since this provided the most desirable performance, as is outlined in the following graph.

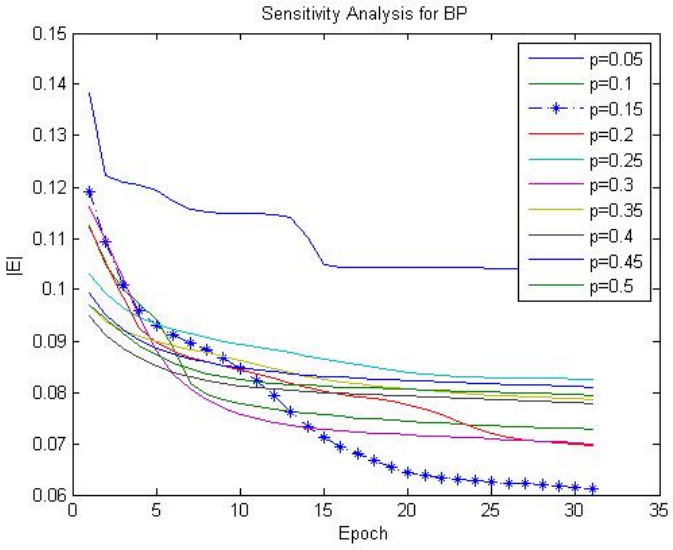

Figure E.1: Illustrates the performance of an artificial neural network as it is being trained, using values for prime varying from 0.05 to 0.5.

The sensitivity analysis provided above plots an average error value[1] (x-axis) of an artificial neural network, per training iteration (y-axis), as it is trained using back-propagation. The multiple plots that are visible represent different configured values of prime ranging from 0.05 to 0.5, as is depicted in the legend to the right of the figure. After thirty iterations, it is shown that setting the value of prime to 0.15 provides the best performance, in the average case.

[1] The average error was taken over a set of 10 experiments.