TOWARDS A MULTIPURPOSE NEURAL NETWORK APPROACH TO NOVELTY DETECTION

a thesis submitted to

The University of Kent

in the subject of computer science

for the degree

of doctor of philosophy.

By

Simon J. Haggett

2008

Abstract

Novelty detection, the identification of data that is unusual or different in some way, is relevant in a wide number of real-world scenarios, ranging from identifying unusual weather conditions to detecting evidence of damage in mechanical systems. However, utilising novelty detection approaches in a particular scenario presents significant challenges to the non-expert user. They must first select an appropriate approach from the novelty detection literature for their scenario. Then, suitable values must be determined for any parameters of the chosen approach. These challenges are at best time consuming and at worst prohibitively difficult for the user. Worse still, if no suitable approach can be found from the literature, then the user is left with the impossible task of designing a novelty detector themselves.

In order to make novelty detection more accessible, an approach is required which does not pose the above challenges. This thesis presents such an approach, which aims to automatically construct novelty detectors for specific applications. The approach combines a neural network model, recently proposed to explain a phenomenon observed in the neural pathways of the retina, with an evolutionary algorithm that is capable of simultaneously evolving the structure and weights of a neural network in order to optimise its performance in a particular task.

The proposed approach was evaluated over a number of very different novelty detection tasks. It was found that, in each task, the approach successfully evolved novelty detectors which outperformed a number of existing techniques from the literature. A number of drawbacks with the approach were also identified, and suggestions were given on ways in which these may potentially be overcome.

Acknowledgements

Firstly, I would like to extend my deepest thanks to my supervisor, Dr. Dominique Chu. His support, dedication, encouragement and guidance are what made this thesis possible. I am also grateful to my previous supervisor, Professor Ian Marshall, for his support and guidance during the first half of my doctoral study program.

I would also like to thank the members of my supervisory panel, Dr. John Crawford and Dr. Colin Johnson, for their encouragement over the past three years as well as for their comments on drafts of this thesis.

I am also extremely grateful to Professor Peter Welch and Dr. Fred Barnes for allowing me access to the pi-cluster, without which the experiments conducted in this thesis could not feasibly have been completed within the three years.

Thanks also to my parents Chris and Barbara, and my brother Rob, who have all encouraged me throughout this doctoral degree. I would especially like to thank my partner, Sarah Waterhouse, for her love, support and tolerance over the past three years, especially during the final writing up phase.

This work was partly funded by an EPSRC studentship and partly by a departmental bursary.

Dedication

For Sarah.

Contents

Abstract ii

Acknowledgements iii

Dedication iv

List of Tables x

List of Figures xii

1 Introduction 1

1.1 Data Mining . . . 2

1.1.1 Classification . . . 5

1.1.2 Cluster Analysis . . . 8

1.2 Novelty Detection . . . 10

1.2.1 Novelty Detection and Record Data . . . 12

1.2.2 Novelty Detection and Time Series Data . . . 12

1.3 Challenges of Applying Novelty Detection to a Particular Scenario . . . 15

1.4 Aims and Objectives . . . 16

1.5 Contribution of this Work . . . 17

1.6 Summary . . . 18

1.7 Overview of the Thesis . . . 18

2 Existing Approaches to Novelty Detection 20

2.1 Introduction . . . 20

2.2 Statistical Approaches to Novelty Detection . . . 22

2.2.1 k-Nearest Neighbour . . . 22

2.2.2 Negative Selection . . . 25
2.2.3 Support Vector Machines . . . 27
2.2.4 Kernel-based Online Anomaly Detection . . . 31
2.3 Neural Network Approaches to Novelty Detection . . . 32
2.3.1 Multilayer Perceptron . . . 33
2.3.2 Autoassociative Networks . . . 37
2.3.3 Radial Basis Function Networks . . . 42
2.3.4 Self-Organising Map . . . 45
2.3.5 Adaptive Resonance Theory . . . 49
2.3.6 Evolutionary Neural Networks . . . 51
2.4 Summary . . . 53

3 Techniques Used in this Research **55**
3.1 Introduction . . . 55
3.2 Predictive Coding . . . 56
3.3 Dynamic Predictive Coding . . . 59
3.3.1 Neural Network Model . . . 60
3.3.2 Analysis of the Adapted Network . . . 62
3.4 Dynamic Predictive Coding and Novelty Detection . . . 65
3.4.1 Illustrative Neural Network . . . 65
3.4.2 Visual Environments . . . 66
3.4.3 Sensitivity Analysis . . . 68
3.4.4 Simulation over Visual Environments . . . 69
3.4.5 Analysis of the Receptive Field . . . 71
3.4.6 Defining the Recent Past . . . 73
3.4.7 Evaluating Dynamic Predictive Coding Against the Requirements in Chapter 1 . . . 75
3.5 Evolutionary Neural Networks . . . 77
3.5.1 Evolutionary Algorithms . . . 78
3.5.2 Evolving Neural Networks . . . 81
3.5.3 Competing Conventions . . . 83
3.5.4 Crossover of Networks with Very Different Topologies . . . 85
3.5.5 Protecting Structural Innovations . . . 86

3.6 Neuroevolution of Augmenting Topologies . . . 87
3.6.1 Encoding Strategy . . . 87
3.6.2 Historical Markings . . . 89
3.6.3 Crossover . . . 89
3.6.4 Mutation . . . 93
3.6.5 Speciation and Reproduction . . . 95
3.6.6 Initial Population . . . 98
3.6.7 Discussion . . . 98
3.7 Summary . . . 99

4 The Proposed DPC+NEAT Approach 101

4.1 Introduction . . . 101
4.2 The DPC+NEAT Approach . . . 102
4.2.1 Encoding DPC Networks . . . 104
4.2.2 Hidden Neurons . . . 105
4.2.3 Work Required from the User . . . 107
4.2.4 DPC+NEAT Parameters . . . 108
4.3 Evaluating DPC+NEAT in a Test Scenario . . . 109
4.3.1 Diagonal Environments . . . 110
4.3.2 Performance of the Illustrative Neural Network . . . 113
4.3.3 Optimising Performance . . . 114
4.3.4 Method . . . 117
4.3.5 Results . . . 119
4.3.6 Discussion . . . 123
4.4 Response to Noise . . . 125
4.4.1 Impact of Noise on Original and Evolved Networks . . . 125
4.4.2 Evolving Networks with Noisy Environments . . . 128
4.4.3 Discussion . . . 131
4.5 Novelty Detection with DPC+NEAT . . . 131
4.5.1 Interpreting Network Response . . . 132
4.5.2 Correlation Measure . . . 133
4.5.3 Non-Adaptive Networks . . . 135
4.5.4 Hidden Neurons . . . 135

4.5.5 Initial Population . . . 137
4.6 Summary . . . 137

5 Evaluating DPC+NEAT **139**
5.1 Introduction . . . 139
5.2 Novelty Detection Tasks . . . 141
5.2.1 Novelty Detection over Record Data . . . 142
5.2.2 Novelty Detection over Time Series Data . . . 145
5.3 Comparison Approaches . . . 150
5.4 Measuring Performance . . . 152
5.4.1 Class Skew . . . 153
5.4.2 ROC Analysis . . . 154
5.4.3 Performance Measure . . . 156
5.5 Overfitting . . . 157
5.6 Statistical Comparison . . . 158
5.7 Experimental Setup . . . 162
5.7.1 Normalisation . . . 162
5.7.2 Tasks based on Record Datasets . . . 162
5.7.3 Time Series Tasks . . . 163
5.7.4 Fitness Evaluation . . . 164
5.7.5 Parameters . . . 165
5.8 Results . . . 167
5.8.1 Record Datasets . . . 170
5.8.2 Time Series Tasks . . . 175
5.9 Evaluation . . . 182
5.10 Summary . . . 185

6 Summary and Conclusions **187**
6.1 Summary of the Thesis . . . 187
6.2 Drawbacks of DPC+NEAT . . . 190
6.2.1 Fitness Function Requires Novel Data . . . 190
6.2.2 Customisation of Hidden Neurons . . . 192
6.2.3 Time Required For Evolution . . . 193
6.2.4 DPC+NEAT Parameters . . . 194

6.3 Relating DPC+NEAT to the Aims and Objectives in Chapter 1 194
6.4 Further Directions for Improvement 198
6.5 Conclusions 199

A DPC+NEAT Parameters **202**

Glossary of Key Terms **204**

Bibliography **212**

List of Tables

1.1 An extract of Fisher's classical Iris dataset 3

3.1 A summary of the evaluation of the DPC neural network model against the requirements in chapter 1. 77

3.2 Pseudocode of the general evolutionary algorithm, given by Eiben and Smith . . 78

5.1 The damage cases used by Sohn *et al.* in the Hard Disk model 149

5.2 A confusion matrix for two-class classification and novelty detection problems . . 153

5.3 The application-specific parameters for the four comparison approaches used in this work 165

5.4 The parameters for each comparison approach not found experimentally 166

5.5 The parameters for the comparison approaches that were found experimentally . 167

5.6 The performance achieved by DPC+NEAT for the record datasets 168

5.7 The performances achieved by the comparison approaches for the record datasets 168

5.8 The average mean true positive and true negative rates achieved by the 100 novelty detectors evolved by DPC+NEAT and the mean rates achieved by each comparison candidate for the record datasets 168

5.9 Statistical comparison of the novelty detectors evolved by DPC+NEAT with the comparison approaches for the record datasets 169

5.10 The performance achieved by DPC+NEAT for the first two time series datasets . 176

5.11 The performances achieved by the comparison approaches for the first two time series datasets 176

5.12 The average mean true positive and true negative rates achieved by the 100 novelty detectors evolved by DPC+NEAT and the mean rates achieved by each comparison candidate for the first two time series datasets 176

5.13 Statistical comparison of the novelty detectors evolved by DPC+NEAT with the comparison approaches for the first two time series datasets 177

5.14 The performances achieved by DPC+NEAT and the comparison approaches for the ECG time series dataset 179
5.15 The performances achieved by DPC+NEAT, with non-thresholding hidden neurons, and the comparison approaches for the ECG time series dataset 181
A.1 The parameters used by DPC+NEAT throughout this thesis 202

List of Figures

1.1 A two-dimensional visualisation of the Iris dataset 4
1.2 An example of a time series dataset 5
1.3 The class memberships of the data elements in the Iris dataset 6
1.4 Class boundaries for the Iris dataset 7
1.5 Hyperspherical class boundaries in the Iris dataset 9
1.6 An example time series with the presence of a single novel event 13
2.1 The McCulloch-Pitts model of a neuron 33
2.2 An example multilayer perceptron network 34
2.3 The autoassociative neural network architecture proposed by Rumelhart *et al.* . . 38
2.4 The autoassociative neural network proposed by Kramer 39
3.1 The predictive coding mechanism proposed by Elias 57
3.2 A small portion of a television raster 58
3.3 The "circular" method of predictive coding over image points from a television raster 59
3.4 A series of receptor neurons in a classic centre-surround orientation 60
3.5 A schematic of the DPC neural network model 61
3.6 The illustrative DPC neural network 65
3.7 An example of a 4×4 greyscale image which serves as input to the illustrative neural network 66
3.8 The structure of the four artificial visual environments with perfect correlational relationships 67
3.9 The six artificial visual environments defined by Hosoya *et al.* 68
3.10 The sensitivity graph produced by our implementation of the illustrative neural network model 70
3.11 The state of the network's receptive field when subjected to each visual environment 71

3.12 The effect of varying τ on the illustrative neural network 73
3.13 The effect of varying m on the illustrative neural network 74
3.14 An example of a parse tree used in genetic programming 80
3.15 The matrix-based encoding scheme used by Han and Cho 81
3.16 Two functionally equivalent neural networks with different labels assigned to the nodes in the hidden layer 83
3.17 The genotype representation of two neural networks with identical topologies . . 84
3.18 The offspring neural networks generated by the crossover operator defined by Han and Cho 84
3.19 Two neural networks that illustrate the variable-length genome problem 85
3.20 A neural network with the structure necessary to perform the XOR function, but with unoptimised weights 86
3.21 An example genotype in the representation used by NEAT 88
3.22 The crossover operator implemented by NEAT 90
3.23 The genotype representation used by NEAT of the two neural networks shown in figure 3.16 91
3.24 The crossover operator implemented by NEAT applied to the two neural networks shown in figure 3.16 92
3.25 The add node mutation operator defined by NEAT 94
3.26 The add connection mutation operator defined by NEAT 95
4.1 A block diagram illustrating the DPC+NEAT approach 103
4.2 A DPC neural network with its corresponding genotype 106
4.3 The new diagonal environments to be used in the test scenario 110
4.4 The structure of the diagonal environments 110
4.5 The receptive field of the output neuron of the illustrative neural network model when adapted to the left diagonal environment 112
4.6 The result of applying stimuli from the right diagonal environment to a network adapted to the left diagonal environment 112
4.7 The sensitivity graph produced by the illustrative neural network in the test scenario 114
4.8 The sensitivity graph produced by the best performing network found by DPC+NEAT in the first experiment 119
4.9 The structure of the best performing network found by NEAT in the first experiment 120

4.10 The sensitivity graph produced by the best performing network found by NEAT in the second experiment 122
4.11 The structure of the best performing network found by NEAT in the second experiment 123
4.12 The performance achieved by the original illustrative neural network and the best performing networks from the two previous experiments as increasing levels of noise are applied 126
4.13 The sensitivity graphs produced for $k = 0.4$ by the original illustrative network and the two best performing networks 127
4.14 The effect of varying k on the performances of the best networks evolved by DPC+NEAT for particular values of k 130
4.15 The sensitivity graph produced by the original illustrative neural network when Pearson's correlation coefficient is used in the anti-Hebbian learning rule 134
5.1 A visualisation of first two principal components of Fisher's classical Iris dataset 144
5.2 A visualisation of Fisher's classical Iris dataset along the first and third principal components 144
5.3 Different perspectives of a time series generated by the sliding window technique 147
5.4 A basic Receiver Operating Characteristics (ROC) graph 155
5.5 The structure of the best evolved novelty detector for the Biomedical dataset . . 172
5.6 The structure of the best evolved novelty detector for the Breast Cancer dataset 172
5.7 The structure of the best evolved novelty detector for the Iris-Setosa dataset . . 173
5.8 The structure of the best evolved novelty detector for the Iris-Versicolor dataset . 173
5.9 The structure of the best evolved novelty detector for the Iris-Virginica dataset . 173
5.10 The positions of the data elements in the Iris-Versicolor dataset misclassified by the best evolved novelty detector for that dataset 174
5.11 The structure of the best evolved novelty detector for the Mackey-Glass dataset . 178
5.12 The structure of the best evolved novelty detector for the Hard Disk dataset . . . 179
5.13 The structure of the best evolved novelty detector for the ECG dataset 180
5.14 The structure of the best evolved novelty detector for the ECG dataset with non-thresholding hidden neurons 181

Chapter 1

Introduction

A large number of real-world applications involve the collection and analysis of significantly large amounts of data, of which only a subset is typically of interest. For example, a network of sensors deployed in an outdoor environment to monitor meteorological conditions generates significant quantities of data, describing the various weather conditions of that environment observed over time. This may include attributes such as temperature, wind speed and direction, rainfall, soil moisture levels, and levels of sunlight. The majority of this data will describe what may be considered to be the normal state of this environment, i.e. unremarkable weather patterns which are not unexpected for the current season, and so provide little or no new information. However, unusual events, such as intense storm activity, unusual temperature variations, or strange changes in sunlight levels, *do* provide new information that is likely to be of significant importance. An intense storm may require measures to be put in place to protect life or property. Unusual temperature variations or peculiar patterns in sunlight levels may indicate a strange event in the environment that requires further investigation, or possibly a problem with the sensor nodes themselves. A node may have become obstructed in some way, for example by animals or other objects in the environment, or have developed a fault which requires that it be repaired or replaced. All these events require some form of human intervention.

Alternatively, a series of sensors may be used to monitor the state of a mechanical device, such as a gearbox. These sensors may measure the different vibrations generated by the gearbox at various positions on its casing. Again, most of the data generated by these sensors will reflect the normal operating conditions of the gearbox during its day-to-day use. However, if the gearbox develops a fault, or sustains damage, then the vibrations it produces are likely to

change, and so the fault or damage may be identifiable through subtle changes in the vibration data. Detecting such faults or damage in a timely manner is critical to ensure that the resulting disruption is minimised and to avoid sustaining further damage.

Another real-world application would be monitoring the activity of a computer network. A large network typically generates massive amounts of data, even if only the headers of packets (assuming a packet-based network) are considered alone. Since most of the behaviour exhibited by the network may be considered to be normal, it is again the case that the majority of this data contains no new information. However, it may also be the case that, on occasion, the data may have embedded within it important new information, such as evidence of an intrusion attempt or other security threat. For example, a series of unusual connections to or from a particular machine may indicate attempts to compromise that machine, or that the machine is already under unauthorised control. Alternatively, activity on a single computer system, such as unusual patterns of commands being issued or changes being made to normally static files, may suggest evidence of a security breach. Only by identifying the presence of new information in the data can such important events be detected.

In each of the applications described above, the sheer quantity of the data generated prevents a human from identifying new information by simply examining the data. The problem of extracting useful information, or knowledge, from data has motivated a significant field of computer science research known as *data mining*. This thesis is concerned with one particular problem within this field, *novelty detection*, which we introduce in this chapter. In the next section, we present a brief high-level overview of the data mining field, including two well known areas that are related to novelty detection. We then define novelty detection itself in section 1.2 and in section 1.3 examine the challenges faced by the non-expert user when trying to employ novelty detection in a particular real-world scenario. In section 1.4 we identify the aims of this research, to develop an approach that allows novelty detection to be easily, efficiently and effectively employed in a wide range of applications, and we define a number of objectives that such an approach should fulfil. Finally, we briefly summarise the contribution made by this thesis and give an overview of the remaining chapters.

1.1 Data Mining

Data mining is concerned with extracting, or *mining*, information, or *knowledge*, that is embedded in large amounts of data [54]. More precisely, data mining identifies hidden patterns in

Petal Length (cm)	Petal Width (cm)
...	...
1.4	0.2
1.5	0.2
1.7	0.4
1.6	1.6
4.4	1.4
4.9	1.5
3.7	1.0
5.8	1.6
4.8	1.8
5.9	2.3
...	...

Table 1.1: An extract of Fishers classical Iris dataset [41]. Only two of the four attributes of this dataset are considered here.

the data which can be presented in some way to convey knowledge to the user [54]. Tan *et al.* identify four core tasks within data mining: predictive modelling, cluster analysis, association analysis and novelty detection [113]. Predictive modelling tasks include *classification*, in which the group, or class, to which a given data element belongs is predicted, given the properties of the data element and prior knowledge of the classes, and *regression*, which is concerned with predicting the value of a particular continuous (real-valued) attribute of the data, given the values of other attributes. Cluster analysis attempts to establish groups, or clusters, of closely related elements in the dataset, whilst association analysis aims to discover rules which describe attribute-value conditions that occur frequently in the data [54]. Novelty detection attempts to identify the data that deviates from some model of normality. This task is the focus of this thesis and is presented in greater detail in section 1.2.

The data that we wish to mine may be in one of a number of different forms. Two types of data we will commonly consider in this thesis are *record data* and *time series data*. In a record dataset, each data element, or record, is a single row of a table, which may be stored in a file or a database system. Each column in the table defines an *attribute* of the data, i.e. some particular property or characteristic of each record [113], and the number of attributes defines the *dimensionality* of the data. An extract of a record dataset describing the petal length and width in centimetres of 150 Iris plants[1] [41] is shown in table 1.1. Each record has two attributes, the length and width, and so the dataset is two-dimensional. We may therefore visualise this

[1]This dataset, provided by Fisher [41], is a classical dataset in the fields of classification and pattern recognition. It actually consists of four attributes in total, but for simplicity we consider only two attributes here. We return to this dataset in chapter 5, where we consider all four attributes.

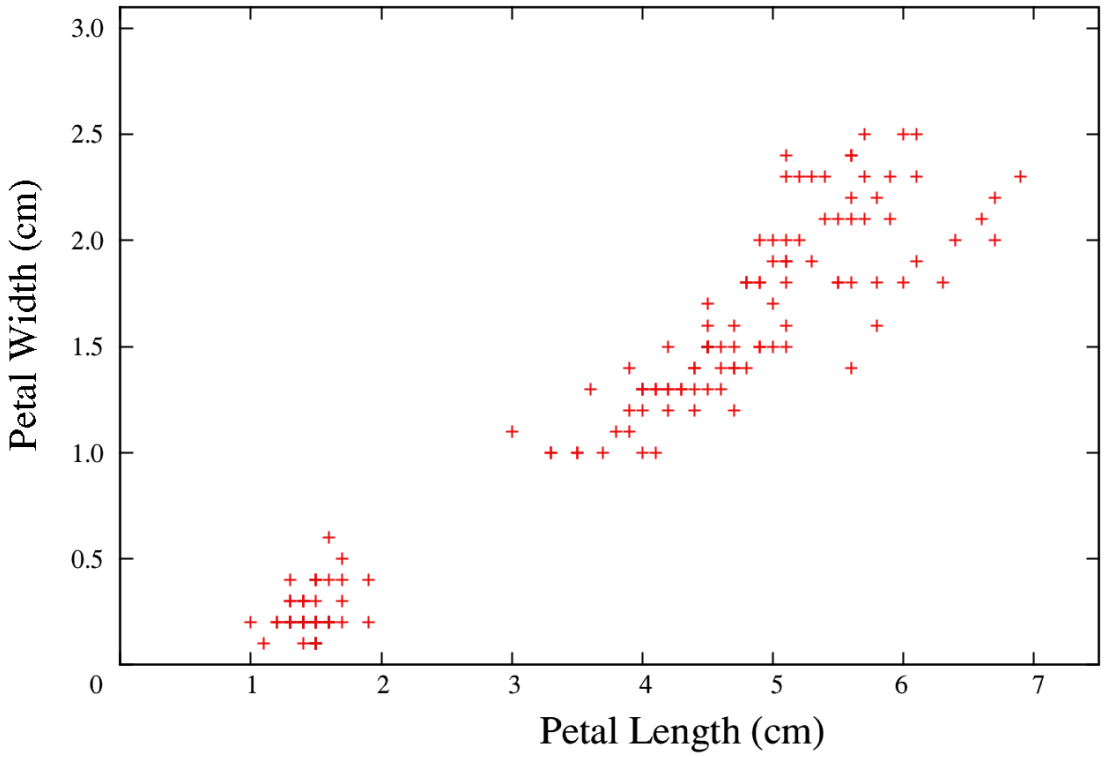


Figure 1.1: A two-dimensional visualisation of the Iris dataset. Again, as in table 1.1, only two of the four attributes are considered here.

dataset as a scatterplot, as shown in figure 1.1. Since each record in table 1.1 corresponds to a single point in figure 1.1, we may also refer to the records as *data points*.

Time series data is different to record data in that it consists of one or more series of time-ordered values. Each series is associated with an attribute, or *variable*, which represents the property being observed over time. In an outdoor sensor network, for example, these variables would represent different properties of the environment, such as temperature, rainfall, soil moisture levels, etc. A time series dataset may be referred to as being *univariate*, consisting of only one variable, or *multivariate*, consisting of two or more variables. An example of a univariate time series dataset is shown in figure 1.2.

The four core tasks discussed at the start of this section each fall into one of two general groups of data mining task [113]: predictive tasks and descriptive tasks. Predictive modelling tasks fall into the category of predictive tasks, as they each attempt to predict an attribute of the data given the other attributes. In classification, this attribute is a class label whilst in regression it is a real value. Novelty detection also falls into this category, since it attempts to predict whether or not previously unseen data represents normality or is novel in some way.

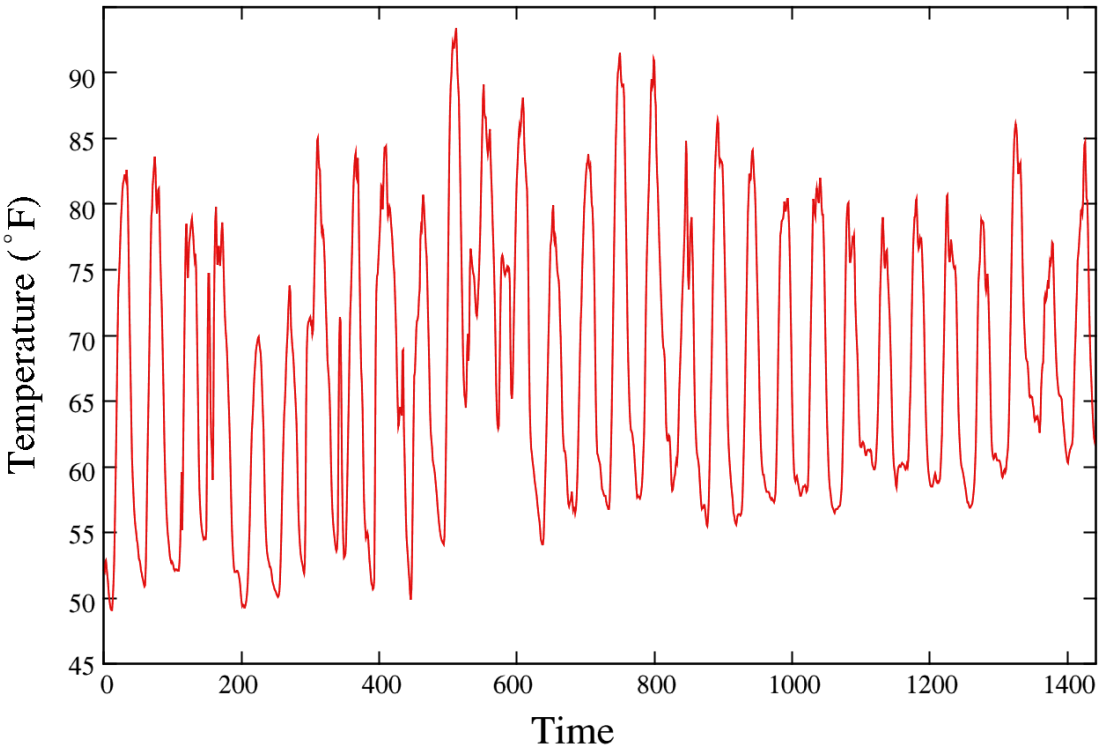


Figure 1.2: An example of a time series dataset. This series describes temperature changes measured by a single sensor node in an indoor environment over a period of time [6].

Cluster analysis and association analysis fall into the category of descriptive tasks, since they both attempt to identify underlying relationships in the data (similarities between elements in the case of cluster analysis, and strongly associated features in the case of association analysis).

The work in this thesis focuses on the task of novelty detection, which we present in section 1.2. Throughout this thesis, we view novelty detection from the perspective of classification, in that a novelty detector classifies data points as either normal or novel. Another data mining task that is related to novelty detection is cluster analysis, which, as we will see in chapter 2, has provided the basis for a number of approaches to novelty detection in the literature. Therefore, before introducing novelty detection in the next section, we first briefly introduce classification and cluster analysis.

1.1.1 Classification

The problem of data classification is possibly one of the most studied data mining tasks [44]. Elements of a dataset can be divided into groups, or *classes*, where the data belonging to each class shares some common properties. For example, given a variety of different coloured objects,

one may naturally group these objects by colour. Then, each object belongs to a class identifying its colour, and all objects in the same class share that colour. The goal of classification is to learn, from a training dataset, a model of each class and subsequently use this information to predict the class of new data elements [54]. Each element in the training set is associated with a *class label*, identifying the class to which it belongs. The process of learning to correctly classify the training set data is referred to as *supervised learning*, since the performance of the classifier over the training set may be repeatedly evaluated and this information used to further inform its learning.

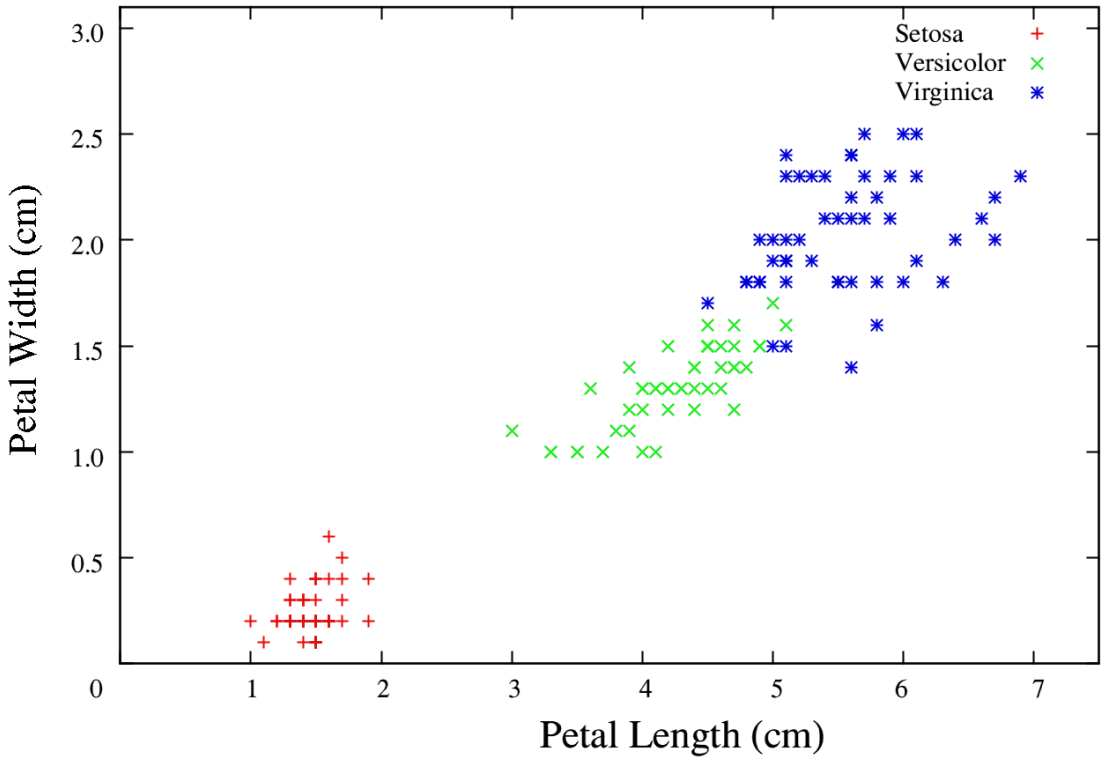
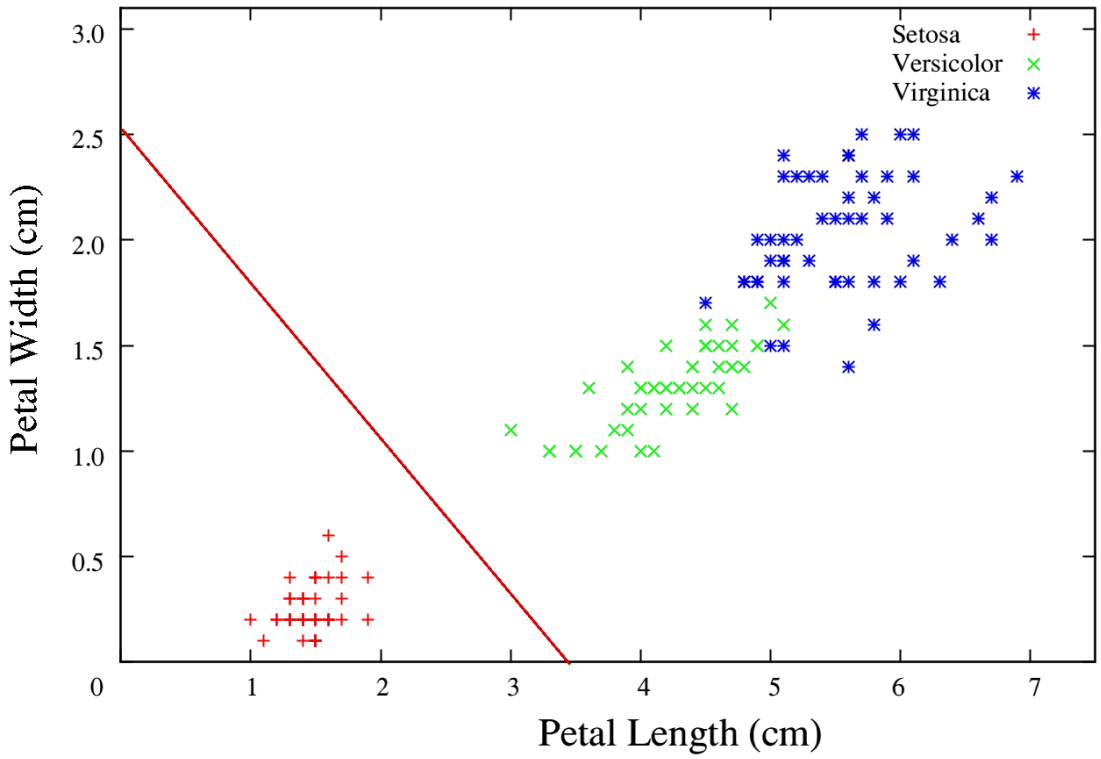
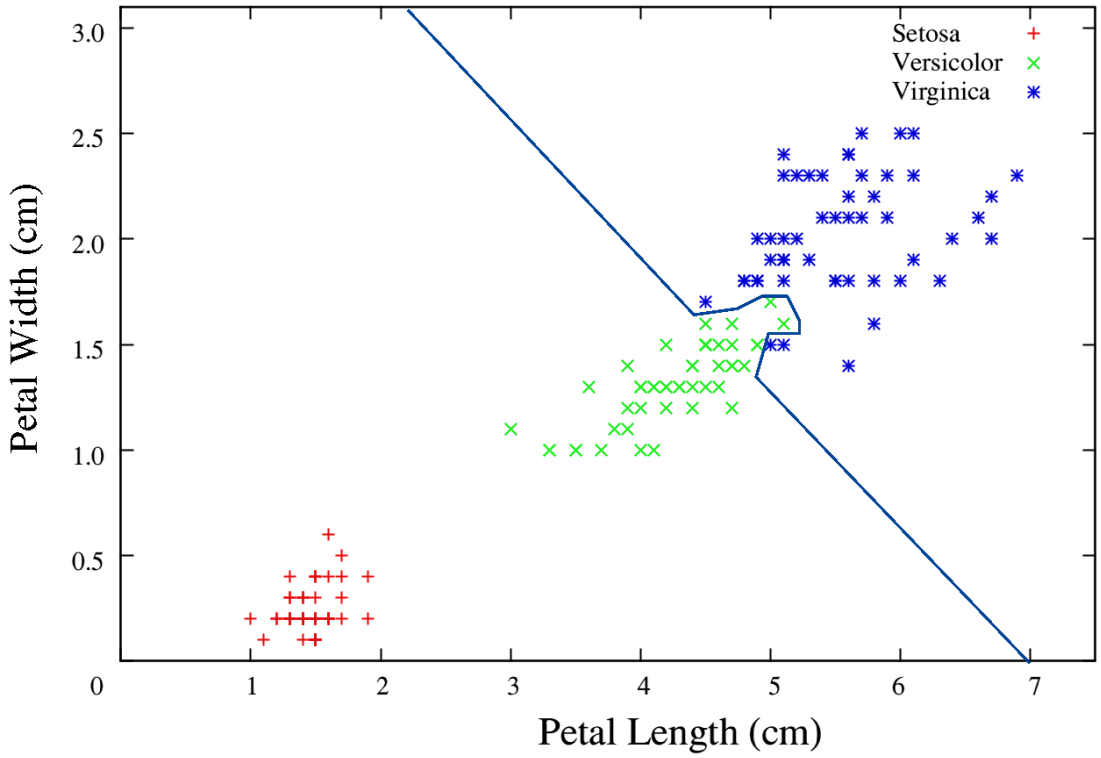


Figure 1.3: The class memberships of the data elements in the Iris dataset.

It is common to look at the classification problem geometrically. The goal of a classifier can be viewed as establishing a series of *class boundaries*, also known as *decision boundaries* or *decision surfaces*, in the geometrical space populated by the data. This space may be referred to as the *input space*, and has the same dimensionality as the data. New data points can then be classified depending on which side of the class boundaries they fall. For example, consider again the Iris dataset presented in figure 1.1. Whilst each record in this dataset corresponds to a different Iris plant, each of these plants is a member of one of three classes: setosa, versicolor or virginica [41], as shown in figure 1.3. One simple kind of class boundary that may be used



(a)



(b)

Figure 1.4: Class boundaries for the Iris dataset, separating (a) the setosa class from the versicolor and virginica classes, and (b) the virginica class from the setosa and versicolor classes.

in this two-dimensional dataset is a line, which is shown to separate the setosa and versicolor classes in figure 1.4(a). However, the separation between the versicolor and virginica classes is clearly nonlinear. A classifier could approximate this separation with a linear class boundary, but this would result in some misclassification of the data. A better approach would be to construct a *nonlinear* class boundary, as shown in figure 1.4(b). The kinds of class boundaries that can be created depend on the particular classifier being used. Whilst the class boundaries may be referred to as lines or curves in two-dimensional space, in higher dimensions these same geometrical concepts are known as hyperplanes and hypersurfaces respectively. That is, when we talk about a hyperplane in n-dimensional space, we are simply referring to the same geometrical concept that is a line in two-dimensional space and a plane in three-dimensional space.

It is important to note that, whilst in the above example we drew the class boundaries whilst considering the entire dataset, in reality a classifier must decide the shape and position of these boundaries based on the training dataset alone. The more representative the training dataset is of the classes, the more likely it is that new data will be accurately labelled by the classifier. However, if the classifier fits the training data too closely, then it may give a poor performance on any new data it is faced with. This well-known problem is referred to as *overfitting* and we consider this in more detail in chapter 5.

1.1.2 Cluster Analysis

Cluster analysis, or clustering, is a descriptive data mining task which finds groups, or clusters, of data within a dataset. Each data point belonging to a particular cluster is, in general, more similar, or related in some way, to the other data points in that cluster than to those in any other [113]. The clusters are constructed based on the information directly available from the data itself.

In many datasets, the notion of a cluster is not well defined since the data may be grouped in a variety of different ways, depending on what is most useful or meaningful in a particular scenario [113]. A cluster may simply be defined as a set of data points in which each data point is closer in some way to every other data point in the cluster than to any data point not in the cluster. Alternatively, cluster membership may be determined by a representative data point known as a prototype, whereby data points belong to the cluster if they are more similar to that cluster's prototype than to any other prototype. Or, data density may instead be used, in which a cluster may be defined as a crowded (dense) region of data points surrounded by an

area of low density. Similarly, collections of clusters, known as *clusterings*, may be organised in different ways [113]. The clusters may be nested, in that a particular cluster may have one or more subclusters associated with it, or unnested. It may be useful to impose the restriction that a particular data point may only belong to a single cluster, or perhaps overlaps between clusters may be permitted. Finally, clusters might be formed over the entire dataset, or just a subset which excludes those observations that are uninteresting in some way.

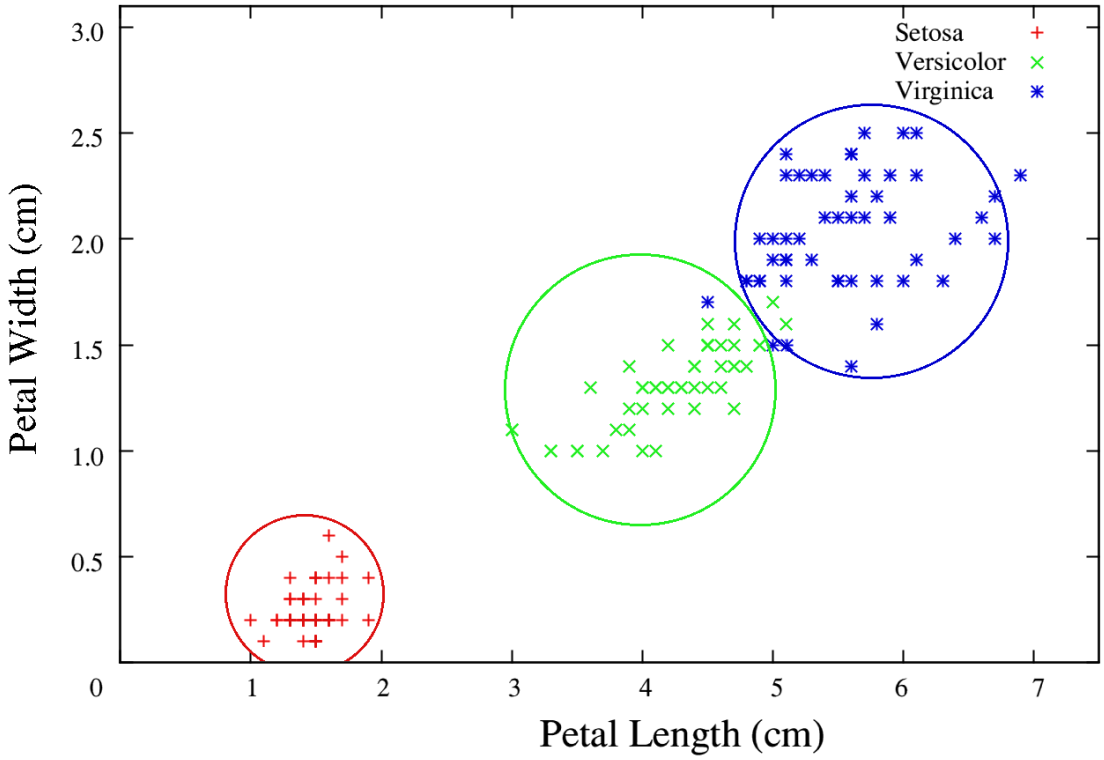


Figure 1.5: Hyperspherical class boundaries in the Iris dataset.

Cluster analysis establishes groupings within datasets, which can subsequently be used to perform classification. The class label of a given data point is defined by the cluster to which that data point belongs [113]. For example, the majority of the data in the Iris dataset shown in figure 1.3 can clearly be grouped into three clusters, with each cluster representing a different class, as shown in figure 1.5. Since the clusters are inferred from the data directly, and no labelled examples are provided, a classifier based on cluster analysis is trained using an *unsupervised learning* process. The kind of class boundaries constructed naturally depends on the kind of clustering technique used. For example, a prototypical clustering approach may construct *hyperspherical* class boundaries around the data, such as those shown in figure 1.5 for the Iris dataset. As with the terms hyperplane and hypersurface, a *hypersphere* is simply the

generalisation of the geometrical concept of a sphere in three-dimensional space to n-dimensional space.

1.2 Novelty Detection

Consider again the example of the outdoor sensor network given at the start of this chapter. In this scenario one could establish a series of classes with each identifying a different weather condition, for example: sunny, cloudy, rain, gales, storm, etc. Examples of these classes could then be used to train a classifier to group new observations. But what about data that indicates some problem with the sensor nodes, such as becoming obstructed by animals or other objects in the environment, or a fault with a particular node itself? One could try to anticipate such occurrences and somehow derive characteristics that may be used to define additional classes on which the classifier may then be trained. But deriving the characteristics for a particular fault condition may be difficult without resorting to inducing that fault by deliberately damaging a sensor node in some way, which is clearly undesirable. Furthermore, how can one hope to anticipate each and every single possible fault that may occur? Clearly, rather than anticipating these occurrences, a better approach would be to identify deviations from known conditions. Is it possible to identify data that is unlike anything we would expect to see, using only our knowledge of the data that we are familiar with? This question motivates the study of novelty detection, which is the focus of this thesis.

Novelty detection is, in its most general form, the detection of data that is unusual or unknown, given some acquired knowledge. That is, novelty is defined in terms of what has been seen before [85]. For example, a single rotten apple in a basket of healthy apples could be considered as novel. Its class (rotten) goes against what we would expect (healthy), given that the vast majority of the apples in the basket are healthy (knowledge). However, if instead roughly half the apples in the basket are rotten then, based on that basket alone, we wouldn't consider a rotten apple to be novel at all. This is because now our knowledge, defined by the basket, is that apples are just as likely to be rotten as they are healthy. But, if we considered this basket as one of a series of otherwise healthy baskets, then we might say that the *basket* is novel. In this case, our knowledge, based solely on the previous healthy baskets, tells us that baskets should contain healthy apples. In the example of the outdoor sensor network given above, an observation would be considered novel if it represented an unusual weather condition or a fault of some kind. However, what constitutes an unusual weather condition may

depend on the timescale upon which the knowledge is based. If this knowledge is limited to 6 hours of measurements, for example, then events such as sunrise and sunset may be considered novel. Alternatively, with a knowledge consisting of 6 years of measurements, then typically only extreme weather conditions and uncommon faults would be highlighted as novel.

As mentioned in section 1.1, novelty detection is most related to the data mining tasks of classification and cluster analysis. A classifier is normally trained to recognise a finite set of known classes, and this training defines its knowledge, i.e. how to identify the class of a particular data point given the assumption that the data point belongs to one of the known classes. However, what if this assumption is false and the data point actually belongs to some unknown class? The classifier then has insufficient knowledge to correctly classify this anomalous data point. Worse still, since the classifier has no way of realising that its knowledge is insufficient, it may still attempt to classify the data point nonetheless, inevitably resulting in misclassification. However, as we will see in chapter 2, some approaches to classification have been proposed that can refuse to classify data points which seem unusual. By refusing to classify such data points, a classifier is then performing a kind of novelty detection in that it recognises that the data points are likely to belong to an unknown class. Cluster analysis is particularly interesting from the perspective of novelty detection, since it enables class boundaries to be constructed that surround data points, as opposed to partitioning the entire space. A data point not belonging to any of the known classes should fall outside the boundary of all clusters, and so can easily be identified as novel. A variety of different approaches to novelty detection based on cluster analysis techniques, such as Kohonen's self-organising map (presented in chapter 2), have been proposed in the literature. However, many other approaches have also been proposed employing techniques from a wide range of different areas within machine learning. In chapter 2, we examine a wide selection of these different approaches.

From the perspective of classification, a data point may be considered novel if it does not belong to a known class. Such data points could also be referred to as *anomalies*, or *outliers*. Hence, many approaches to detecting them are commonly referred to as anomaly detectors or outlier detectors [47, 58]. Whilst the terms anomaly and novelty may be used interchangeably, the same is not necessarily true for outlier. It is possible that an outlier is still a member of a known class [104], meaning that the detection of outliers may not strictly be the detection of true novel or anomalous data points. Despite this, some authors still consider novelty/anomaly detection and outlier detection to refer to the same problem [47]. In this thesis, we are concerned with identifying novel, anomalous, entities alone. Novelty detection is also commonly referred to

as *one-class classification*, in recognition of the fact that a novelty detector is typically required to recognise both normal and novel data when trained on normal data alone. In this way, a novelty detector may be viewed as determining whether a particular data point lies inside or outside of the normal class. However, in some novelty detection tasks a limited amount of novel data is available to train the detector. For example, approaches to fraud detection sometimes use examples of data pertaining to fraudulent activity during the training phase [11]. These kinds of novelty detection tasks are beyond the scope of the work presented in this thesis.

1.2.1 Novelty Detection and Record Data

As mentioned in section 1.1, in the work presented in this thesis we consider novelty detection problems involving two kinds of data: record data and time series data. Many real-world applications store data in record datasets. For example, blood tests taken from a series of patients may be stored as a series of records, where each record corresponds to a different patient and each attribute represents a different measurement. In such a scenario, one could use a novelty detector to identify patients whose blood test data indicates the possible presence of disease. Similarly, measurements taken from breast masses from a number of patients may be stored as a record dataset, and a novelty detector could then be used to identify those masses which may signify the presence of breast cancer.

In a record dataset, no ordering or temporal information exists between the data elements [113]. This means that a novelty detector must decide whether or not a particular data element is novel using only the knowledge it has acquired during a training phase. Each data element normally requires a separate decision, which may be associated with it in the same way as a class label in a classification problem. A novelty detector may be thought of as predicting the class label of each dataset, where the available classes to choose from are “normal” and “novel”.

1.2.2 Novelty Detection and Time Series Data

Novelty detection is also commonly used to detect unusual, or novel, events in time series data. For example, novelty detection is frequently used as an approach to fault detection [32], that is identifying faults or breakdowns in mechanical systems. It has also been used in tasks such as identifying unusual heartbeats from patients with arrhythmia [7, 68], detecting anomalies in computer network traffic [3] and detecting unusual features of an indoor environment explored by a mobile robot [85]. In many cases, a novel event consists of some number of data points

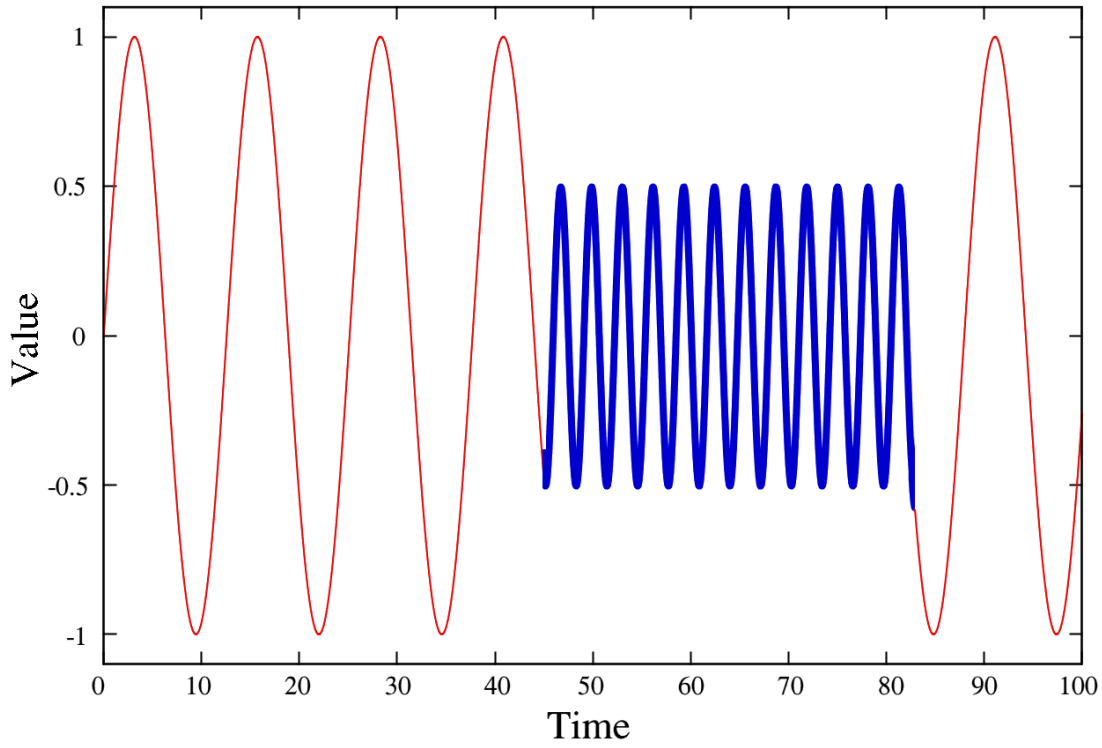


Figure 1.6: An example time series with the presence of a single novel event (highlighted in blue).

which deviate from the normal behaviour of the series. Again, these are also referred to by some authors as anomalies, or outliers [112]. An example of a novel event in a time series is shown in figure 1.6. The highlighted novel points do not fit the normal behaviour of the remaining points in the series. A novelty detector may respond by labelling every one of these points as novel, in the same way it does for record data, or it may highlight the *event* instead, i.e. providing a real-time alarm as early as possible during the onset of the event. The most appropriate way of flagging novel data points depends on the particular scenario being considered.

Change Detection

When dealing with data that has a temporal component, one may naturally be interested in identifying some form of *interesting* change in that data. In some cases, any change might be interesting as it may be unusual or unexpected. On the other hand, data may be undergoing constant change over time. In this case, what constitutes an interesting change would depend on the application.

Some authors equate change detection with abrupt change-point detection in the context

of time series data [30]. Abrupt change-point detection, or simply change-point detection, is concerned with identifying those points at which a *sudden* change occurs in the properties of the data [30]. A change may be persistent, thus yielding a single change-point, or multiple change-points may exist in the series. Change-point detection is also referred to as time series segmentation [128], since the change-points delimit segments of the series in which data has common properties. Since a novel event in a time series, such as that shown in figure 1.6, does exist as a segment, with different properties to the remainder of the series, they can be identified by change-point detection methods. However, depending on the application, a time series segment may not necessarily represent a novel event. For example, a musical composition consists of numerous segments, each corresponding to a different musical note [30]. Whilst a note might represent a novel event (e.g. if it has not been used before, or it differs from the note specified on the written score), the majority of notes naturally do not. In applications where abrupt change is always indicative of novelty, change-point detection may be used as an approach to identify novel events.

Other forms of change detection have also been proposed. For example, Forrest *et al.* consider the problem of detecting changes, possibly caused by viruses, in computer files [43]. In this scenario, the content of the files should remain static and so any change is considered unusual, or novel. Alternatively, Fang *et al.* examine the problem of identifying changes in video sequences taken from the windscreen of a car travelling along a highway [38]. Here, as the authors point out, change is constantly occurring in the video sequence. The kinds of change that are interesting in this scenario are those which could have an impact on driver safety.

Forgetting

Another challenge one is faced with when identifying anomalies in time series data is that the environment from which the time series is generated may change over time. Events which were previously uncommon may become commonplace, and therefore change their status from being novel to being normal. For example, in the time series produced by a sensor measuring outdoor temperature, a period of time in which the temperature is above 25°C may be considered a normal event during the summer months, but novel during the winter. The time series measured by the temperature sensor exhibits seasonal variations, where the normal conditions change over time. In such applications, an important quality of a novelty detector is the ability to "forget" about events witnessed in the distant past, basing its concept of normality on what has been seen "recently". The period of time that defines "recent" again depends on the application.

1.3 Challenges of Applying Novelty Detection to a Particular Scenario

Novelty detection is used in wide range of different real-world problems, such as identifying possible signs of cancer, detecting irregular heart beats, identifying evidence of intrusions in a computer network and detecting faults in mechanical machinery. It is a useful technique in any problem which involves identifying small amounts of unusual, novel, data from large amounts of uninteresting, normal, data. As we will show in chapter 2, many existing approaches to novelty detection tend to focus on one specific problem or group of problems. By doing so, these approaches can be tailored to exploit the unique properties of the chosen problem(s) to achieve a good performance. These approaches also tend to require the definition of a number of parameters, whose values critically impact on performance.

In many real-world scenarios in which novelty detection is required, it is likely that the user, the person wishing to utilise the novelty detector, is not an expert in either novelty detection or the data mining field in general. However, they are likely to have an expert knowledge of their particular problem. For such a user, employing a novelty detector presents significant challenges. First, they must decide on an existing approach which is suitable for their problem. Unless their problem is similar enough to one which has been used to demonstrate the performance of a particular novelty detector in the literature, the user will themselves need to evaluate those approaches they consider to be promising. Once a novelty detector has been selected, the user must decide appropriate values for any parameters that it may have, which is typically a trial-and-error process. These challenges are at best time consuming and at worst prohibitively difficult for the user. It is also possible that their problem is so unusual that no suitable approach exists, in which case the user's only option would be to construct a new specialist novelty detector, a task which requires expert knowledge.

In order to make novelty detection more accessible, an approach is required which does not pose the above challenges. It should be capable of performing novelty detection in a wide range of different scenarios, without requiring expert knowledge from the user. It should also not require a large amount of time from the user in setting it up for a particular novelty detection task, and it should operate efficiently and effectively in that task. But, with the seemingly infinite number of real-world scenarios, involving data with vastly different properties, is this really achievable? To attempt to answer this question, we focus on developing such an approach in this thesis.

1.4 Aims and Objectives

The aims of the work presented in this thesis can be succinctly described by the following sentence:

> *To develop an approach which enables novelty detection to be performed easily, efficiently and effectively in a wide range of different scenarios.*

From these aims, we may derive a number of objectives that the approach must achieve. Firstly, it should be *generally suitable* for use in a wide range of different novelty detection problems. To accomplish this, the approach should minimise the assumptions it makes about the nature of the problems to which it will be applied. For example, a novelty detector designed specifically for use with time series data would naturally be unsuitable for use in problems involving record data. Alternatively, a detector might assume that the underlying distribution of the data is Gaussian. Whilst this may not prevent the detector from being physically applied to a problem in which this assumption is untrue, it would be unlikely to achieve an effective performance. Assuming that the approach is not physically unsuitable for use in a particular novelty detection task, its performance is a natural indicator of its suitability for solving that task.

The general suitability of the approach will also depend on how much it may be *customised* to the novelty detection task at hand. However, whilst this customisability is important, it often has an impact on the amount of *work*, or effort, required by the user for configuration. For example, many novelty detection approaches in the literature may be customised by a series of parameters, whose values critically determine their performance. This significantly increases the effort required by the user, since a time consuming search of the parameter space must be performed. As stated above, the approach we aim to develop should be easy for the user to apply to a particular novelty detection task, and so it should minimise the amount of work required.

Finally, the complexity of the approach should be no more than is required for the particular novelty detection task to which it is to be applied. It should not have properties that are redundant or unnecessary in the context of that task. To some extent, this might be achieved by exploiting the unique properties presented by the data on which the task is based. This constraint prevents us from considering monolithic systems which comprise a multitude of different novelty detection techniques. These systems are undesirable since they would require more resources than an effectively tailored novelty detector, which could prohibit their use in some cases.

To summarise, the approach we aim to develop in this thesis must achieve the following objectives:

- *General Suitability:* The approach should be suitable for and effective in a wide range of different novelty detection tasks.
- *Customisability:* The approach should permit a high degree of customisability.
- *Work:* The effort required from the user to customise the approach for a particular task should be minimal.
- *Complexity:* The complexity of the approach should be justifiable for the particular task.

We use these objectives to systematically evaluate both existing approaches to novelty detection in the literature (chapter 2) and the approach we propose in this thesis (chapters 5 and 6).

1.5 Contribution of this Work

The main contribution of this thesis is the development of a new approach to novelty detection which attempts to satisfy the objectives outlined above. This approach combines *Dynamic Predictive Coding* (DPC), a biologically inspired neural network model proposed by Hosoya *et al.* [63], with Stanley's *Neuroevolution of Augmenting Topologies* (NEAT) method [108], an evolutionary algorithm specifically designed to evolve neural networks of any topology. The proposed approach, DPC+NEAT, aims to satisfy the above objectives by automatically constructing neural network novelty detectors for particular scenarios. By freely designing a suitable network structure, as well as selecting values for the network's parameters, the approach has the potential to produce highly customised detectors that perform effectively, whilst requiring a minimal amount of work from the user. By introducing new network structure slowly through the course of evolution, the approach also has the potential to construct novelty detectors which have a justifiable level of complexity for the task at hand.

In order to gain insight into its potential effectiveness, we first examine the performance of the proposed approach in a test scenario, based on the problem for which the DPC model was originally devised. We examine the effect of allowing DPC+NEAT to choose the values of the parameters of the neural network in addition to its structure, as well as observing the impact of artificial noise on the performance of the approach. We then evaluate DPC+NEAT in a number of different novelty detection tasks, over both record and time series datasets. The performances

of the detectors evolved for each task are compared against that given by a number of existing approaches to novelty detection in the literature. From this evaluation, we assess the success of DPC+NEAT in meeting the aims of this research.

1.6 Summary

We have introduced the data mining task of novelty detection, initially from the perspective of classification, and described a number of different forms of novelty that may occur. In data with no temporal component, novel data points are those which do not fall into any of those classes known *a priori*. However, in data with a temporal dimension, one may be interested in different forms of novelty, such as those points in a time series which indicate some significant change has occurred, or unexpected changes in usually static data.

Novelty detection may be extremely useful in many different real-world problems which involve identifying small amounts of unusual, novel, data from large amounts of uninteresting, normal, data. However, for the non-expert user it may be prohibitively difficult to utilise an approach to novelty detection in their particular scenario. Not only must they identify and evaluate a suitable approach, they also need to configure this approach to enable a good performance. A system that allows novelty detection to be easily, efficiently and effectively applied to a particular problem would not pose such challenges to the user, and thus would make novelty detection more accessible.

In this thesis, we present an approach which aims to allow novelty detection to be easily, efficiently and effectively performed in a wide range of different scenarios. This approach, DPC+NEAT, automatically constructs a neural network novelty detector which is highly customised for the particular novelty detection task at hand. We have discussed four objectives that the proposed approach should accomplish in order to satisfy the aim of this research, and we use these objectives as an evaluation criteria for our proposed approach in chapters 5 and 6.

1.7 Overview of the Thesis

We now provide a brief overview of the remainder of this thesis:

Chapter 2 surveys the current existing approaches to novelty detection in the literature and evaluates each approach against the objectives derived in section 1.4. We establish that none of the approaches fulfils all of these objectives, thus justifying the need for a new approach.

Chapter 3 describes Hosoya *et al.*'s DPC neural network model [63], which was originally proposed to explain a phenomenon observed in the neural pathways of the retina. However, the model can also be viewed as performing a kind of novelty detection over visual images. One drawback of the model is that it does not define a specific network topology, which must instead be tailored manually to the particular application. Therefore, we also examine algorithms capable of automatically evolving the topology and weights of a neural network. Of these algorithms, we find that Stanley's [108], Neuroevolution of Augmenting Topologies (NEAT) approach overcomes or avoids a number of key problems faced when evolving neural networks. The DPC model and the NEAT method form the basis of our proposed approach to novelty detection presented in this thesis.

Chapter 4 presents our proposed DPC+NEAT approach. We combine Hosoya *et al.*'s DPC model with Stanley's NEAT algorithm to construct a system which is capable of automatically evolving neural network novelty detectors for specific applications. We examine the performance of the system in a test scenario, and propose a series of modifications to allow the system to construct novelty detectors for other kinds of novelty detection tasks.

Chapter 5 evaluates DPC+NEAT according to the objectives given in section 1.4. The system is used to find detectors in a number of very different novelty detection tasks, over both record and time series datasets. We also compare the performance of the detectors with a number of the approaches surveyed in chapter 2.

Chapter 6 concludes this thesis. A summary of the work is presented, and the drawbacks of the proposed DPC+NEAT approach are discussed. These drawbacks define a number of directions in which this work may be extended. In light of the drawbacks identified, we consider whether or not DPC+NEAT successfully meets the aims of this research. We also present other ideas on how the proposed approach may be further improved.

Chapter 2

Existing Approaches to Novelty Detection

In this chapter, we evaluate a range of existing approaches to novelty detection from the literature against the objectives stated in chapter 1. We demonstrate that no existing approach successfully fulfils all four objectives, thus justifying the need for a new approach. We also identify a technique applied to novelty detection that has the most potential for fulfilling these objectives.

2.1 Introduction

As we have described in chapter 1, novelty detection refers to the problem of identifying those data elements that are unusual in some way given some acquired knowledge on the state of normality. This could be viewed as the two-class classification problem of determining whether each data element presented belongs to the *normal* or *novel* class. However, unlike a two-class classifier, which is provided with information about both classes, a novelty detector is usually only given information on the normal class alone. This is due to the fact that, whilst a large quantity of normal data can be easily provided in a given scenario, there tends to be very few examples of the novel class available.

A large variety of different approaches to the problem of novelty detection have been proposed in the literature. These generally fall into one of two groups: *statistical approaches* and *neural network approaches* [83, 84]. Statistical approaches to novelty detection may be further subdivided into *parametric* and *non-parametric* methods. Parametric statistical methods make

strong assumptions about the underlying distribution of a population: typically this assumption is that the distribution is Gaussian. Non-parametric approaches make no assumptions about the distribution and are therefore more flexible [83]. Neural network approaches cover a wide variety of different types of neural network. These include multilayer perceptron networks, autoassociators, radial basis function networks, self-organising maps, adaptive resonance theory networks and evolutionary neural networks.

The approaches that we examine in this chapter use different kinds of learning strategies. Some approaches employ an *offline learning* strategy, in which a model of the normal class is derived during a training phase involving a specially provided training dataset. The learning technique may be supervised or unsupervised in this case, depending on whether or not labels can be assigned to the training data. The quality of the knowledge acquired by the learner may then be assessed through a test phase, in which the learner is evaluated over a test dataset consisting of data not seen during training. Alternatively, some approaches perform *online learning*, where the training phase takes place in-situ using real data taken directly from the novelty detection task at hand. This generally requires an unsupervised learning technique to be used, since the data is not usually labelled. For the same reason, a test phase is generally not feasible either. Naturally, an online learning approach can also be applied to a scenario in which offline learning is required. An approach capable of online learning may also support *adaptive learning*, where it continuously learns about the data and is able to adapt to any changes in the properties of this data. This learning strategy may permit an approach to *forget* about properties of the data that have not been recently observed. Such an ability is especially important in those time series scenarios in which normality is defined in terms of the recent past. An adaptive learning strategy is generally the most flexible since this adaptive capability can normally be easily disabled when not required in a particular scenario and, since it performs online learning, can also be applied to scenarios in which an offline learning strategy is required.

In chapter 1, we described four objectives that an approach which could be used to easily, efficiently and effectively perform novelty detection in a wide range of scenarios should fulfil. The approach should be (1) *generally suitable* for use in a wide range of different novelty detection tasks, (2) permit a high degree of *customisability*, (3) minimise the *work* required from the user in configuring it for use in their particular scenario and (4) have a level of *complexity* that is justifiable for the novelty detection task to which it is being applied. In this chapter, we evaluate a range of approaches to novelty detection from the literature against these objectives. We find that no existing approach is capable of fulfilling all four objectives, therefore demonstrating the

need for a new approach. In the next section, we review non-parametric statistical approaches to novelty detection. In section 2.3, we change our focus to neural network approaches. We conclude this chapter in section 2.4 with a summary of this review.

2.2 Statistical Approaches to Novelty Detection

Statistical approaches to novelty detection tend to model training data based on its statistical properties, and then use this model to determine whether or not elements from the test set appear to belong to the same distribution [83]. They may be subdivided into *parametric* approaches, which assume that the data comes from a family of known distributions, and *non-parametric* approaches, in which the distribution of the data, as well as any associated parameters of this distribution, is inferred from the data itself [83]. Since parametric approaches are only suitable in applications where the assumptions made on the distribution of the data hold, their suitability for different novelty detection applications is clearly limited and so they are not considered here. Instead, we review the non-parametric approaches, which have a much wider applicability.

2.2.1 k-Nearest Neighbour

The first non-parametric approach that we consider is k-nearest neighbour (KNN). This technique classifies data points based on their k nearest neighbours in the input space, with respect to some distance measure (such as Euclidean distance). During the training phase, the KNN technique simply *stores* labelled training data points. All of the computational work required for the classification of testing data points takes place when those data points are presented for classification [90]. When classifying individual data points, a common rule is to assign those data points to the class in which the majority of their k nearest neighbours belongs [90], with each neighbour having an equal influence on the classification. Alternatively, the influence of the neighbours may be weighted according to distance, such that those neighbours which lie closer to the data point being classified are assigned a higher weight than other points lying further away [90].

One way of applying the KNN technique to novelty detection is to assume that a novel data point will lie at a considerable distance from the training data points. This approach is used by Guttormsson *et al.* [52], who compared KNN with three other novelty detection methods for the task of identifying faults in a turbine generator. With the KNN technique, a data point is declared novel if the distance to its closest neighbour exceeds some decision threshold.

Therefore, a hyperspherical decision boundary is established around each training data point, whose radius is the decision threshold. However, in the novelty detection task considered in the study, the authors found that an alternative approach, which constructed a hyperelliptical boundary around the data, gave the best performance. This is because the data in the considered task is spread in an elliptical cloud, causing approaches based on spherical boundaries to classify too large a region of the input space as normal.

Another method of employing KNN for novelty detection was proposed by Tax and Duin [116]. For a particular test data point $\mathbf{x}$, an estimate of its likelihood $p(\mathbf{x})$ may be calculated based on the volume of the sphere, centred on $\mathbf{x}$, containing the k nearest neighbours of $\mathbf{x}$ [33]. For $k = 1$, this is shown to be equivalent to the quotient of two Euclidean distance measures [116]: the distance between $\mathbf{x}$ and its nearest neighbour, $NN(\mathbf{x})$, and the distance between this neighbour $NN(\mathbf{x})$ and its own nearest neighbour $NN(NN(\mathbf{x}))$. If this estimate is below a predefined threshold, then $\mathbf{x}$ is considered to be novel. In a task identifying vibration signals indicating damage in a submersible water pump, the proposed KNN method was shown to outperform three other novelty detection approaches. However, when the dimensionality of the dataset was reduced, the performance of the KNN method deteriorated.

Yang *et al.* [129] applied a nearest neighbour approach to the problem of First Story Detection (FSD), a form of novelty detection used in the field of text mining. FSD refers to the problem of identifying the earliest reports of new events from chronologically ordered documents (such as newswire stories or television broadcasts [129]). Each event may be classified into one of a number of known topics, for example the event *"1996 TWA Flight 800 crash"* may be placed into the topic *"aeroplane accidents"*. Since what constitutes a new event changes over time, any approach to FSD must operate in an adaptive manner, updating its model of the normal class each time a new event occurs. The approach proposed by Yang *et al.* first classifies an incoming event into one of a series of topics. For each topic, an FSD algorithm based on KNN is then used to determine the novelty of the event. In a history of all past events, the nearest neighbour to the incoming event, according to some similarity score, is identified. The similarity between the incoming event and its nearest neighbour is then thresholded, with the incoming event being labelled as novel if these two events are not sufficiently similar. When applied to a benchmark dataset, the method was shown to give a very good performance.

Another approach to novelty detection based on KNN was proposed by Angiulli [5]. For a given input vector, this approach, called Nearest Neighbour Domain Description (NNDD), generates a vector holding the distances between the input vector and its k nearest neighbours.

If this vector falls outside of the hypersphere centred at the origin of $\mathbb{R}^k$ with radius θ, then the input vector is labelled as novel. Angiulli also proposed an extension to NNDD, which tackles an important drawback of the KNN technique: in order to classify a given input vector, KNN must compare that vector with each element of the training set which, in applications with large training sets, becomes computationally expensive. The proposed extension to NNDD, Condensed NNDD (CNNDD), selects a subset of the elements of the training set such that when trained with these elements NNDD is able to correctly classify all elements of the training set as normal. This subset is constructed with only two passes of the training set. The CNNDD approach was shown to outperform three other novelty detection methods, including Tax and Duin's approach discussed above [116], over a number of benchmark datasets.

Evaluation

The KNN technique has been successfully applied to a variety of different novelty detection tasks. It has also been used in an adaptive fashion, being able to continuously update its concept of normality [129]. However, this adaptive variant of KNN was not capable of forgetting, and classification would therefore become increasingly computationally intensive as more data elements are presented. A significant drawback of the KNN technique is that it uses an over-complicated representation of the normal class, consisting of the entire training set. This level of complexity is unjustifiable in the majority of novelty detection tasks, in which a decision boundary would be sufficient to separate the normal and novel classes. However, this drawback has been tackled by condensing the training set to its key elements [5], with promising results. But employing this condensing approach whilst allowing the novelty detector to continuously learn about its environment has not yet been demonstrated.

Another drawback of the nearest neighbour approach is that it is especially sensitive to the *curse of dimensionality* [90]. The distance measure used is based on all attributes of the data, whilst in reality a large number of these attributes may provide no information on whether or not a data instance is normal or novel. In this case, the large number of irrelevant attributes may distort the calculated distance between data instances. To avoid this problem, some form of feature selection or dimensionality reduction technique, such as principal component analysis, may first be applied to the data. For applications involving high-dimensional data, this increases the complexity of the approach. Alternatively, an attribute weighting strategy could be used to highlight those attributes which are more relevant than others. However, finding the optimum weight for an attribute is again a complex problem [101].

2.2.2 Negative Selection

Novelty detection has also been considered in the domain of artificial immune systems (AIS). The biological immune system is capable of identifying virtually any foreign cell or molecule [43]. To accomplish this, it produces T cells which have receptors that can detect foreign proteins, referred to as *non-self*. The receptors are created through a pseudo-random genetic process, and so it is likely that some receptors are produced which bind to the body's own cells, known as *self*. Therefore, T cells undergo a selection process, called *negative selection*, where those T cells with receptors which bind to self are destroyed leaving only those T cells which identify non-self, which are then released.

Forrest *et al.* [43] first proposed an approach based on negative selection to perform change detection over computer files, in order to detect computer viruses. In this approach, the data to be protected is represented as a string in some alphabet, which is referred to as the self string. This self string is then divided into equal size segments to produce the collection of self strings S. A series of random strings are then generated and tested against each element of S. If a random string matches any element in S (i.e. matches self), it is rejected. Otherwise, it is deemed to match non-self and so is placed in a non-self detector collection R. Two strings are considered to match if they have r contiguous matches at the same locations. Once the collection of detectors has been constructed, the collection S is then monitored by continuously matching strings in S with detectors in R. If a match occurs, then the corresponding element in S has changed in some way. Using a binary string representation, the approach was shown to perform well at detecting changes made to compiled program files.

González *et al.* [47, 48] generalised the negative selection algorithm to real space. In this approach, which they called real-valued negative selection (RVNS), a series of self samples, represented as n-dimensional vectors, are used as input. The area of the input space in which these samples lie is referred to as the *self space*, whilst the remaining areas of the input space constitute the *non-self space*. RVNS then proceeds by first generating a random collection of detectors, also n-dimensional vectors, and then evolving these detectors so that they do not match any of the self samples. A match occurs if a self sample falls within a specified radius r of a detector, indicating that the detector overlaps the self space. If a detector matches a self sample, then it is moved in a direction away from the self space. Each time a match occurs for a particular detector, the age of that detector is increased. Detectors whose age is greater than some threshold are subsequently replaced with new random detectors. The authors combined

the RVNS algorithm with a supervised learning classification algorithm to yield a hybrid novelty detection mechanism. RVNS was used to generate non-self samples to simulate novel data, which allowed the classification algorithm to establish a boundary separating normal and novel data without using any real examples of anomalies. There is no restriction on the kind of classification algorithm that can be used, provided that it can accept real-valued input vectors. The hybrid approach was demonstrated, using a multilayer perceptron as the classifier, on four different benchmark datasets (both time series and record-based) and compared with a negative selection algorithm based on a binary string representation as well an approach based on Kohonen's self-organising map [47]. The hybrid approach performed well on all four datasets, giving a similar performance to the self-organising map approach. The binary-valued negative selection algorithm, however, failed to produce satisfactory results for two of the datasets.

Ji [66] modified the RVNS algorithm to allow detectors to have different sizes. This approach, V-detector, assigns an independent radius to each detector which permits the best coverage possible given the detector's location in the input space. A collection of detectors is constructed sequentially as follows. A new detector is generated randomly, as in RVNS. If this detector does not fall into a space covered by any existing detectors in the collection, then a radius is assigned to the new detector such that it does not overlap any surrounding self samples, and the new detector is then added to the collection of detectors. This collection continues to expand with the construction of new detectors until one of three conditions are reached: (1) the estimated coverage of the non-self space achieved by the detectors reaches a sufficient level (defined by parameter c_0), (2) the maximum number of permissible detectors (again a controllable parameter) is reached, or (3) too much of the input space is covered by the self region. The estimate of the coverage of the non-self space achieved by the detectors can be derived by simple point estimation, i.e. basing the estimate on the coverage of m sample points, or by more robust hypothesis testing techniques [66]. V-detector was compared against two other AIS-based algorithms for the task of novelty detection over the well-known Iris and Biomedical datasets (which we present in chapter 5). For the Iris dataset, V-detector was shown to give a comparable performance but with far fewer detectors. For the Biomedical dataset, V-detector was shown to achieve a lower detection rate than the other approaches, but also a lower false alarm rate. In addition, V-detector was also shown to achieve a good performance in a novelty detection task based on time series data.

Stibor *et al.* [111] presented an alternative approach to novelty detection based on another concept taken from the biological immune system known as *positive selection*. Here, detectors

are constructed to match elements of the *self* space, rather than non-self. The radius used globally for all detectors is determined by repeatedly evaluating the ability of the constructed detectors to correctly classify the data from the training set. The positive-selection algorithm was compared against both V-detector and Schölkopf *et al.*'s one-class support vector machine [99], presented in the next section, over the Iris and Biomedical datasets. In all but one of the experiments, where the performances of V-detector and positive selection were comparable, the positive-selection approach was shown to outperform V-detector. The positive selection approach also outperformed Schölkopf *et al.*'s support vector machine approach in all cases.

Evaluation

Like KNN, negative selection has been demonstrated over a range of different novelty detection applications, from detecting changes in computer files [43] to identifying anomalies in time series data [66]. The most versatile approach described here is Ji's V-detector [66]. It is also feasible to implement an adaptive learning strategy in a negative selection algorithm, by replacing those detectors which have come to match the self set with new randomly generated detectors after a certain period of time. However, this approach still requires a number of parameters to be set, which constitutes a significant amount of work. Another drawback of the negative selection approach is that it assumes that decision boundaries can be constructed by a series of hyperspheres in the input space. This could lead to an overcomplicated representation in scenarios where such decision boundaries could more easily be represented by another geometrical construct, such as a hyperplane.

2.2.3 Support Vector Machines

Support vector machines (SVMs) are kernel-based classifiers which have attracted a lot of recent attention in the neural network community [57]. The goal of SVM is, given a training dataset, to identify the *optimal decision boundary*, or *optimal hyperplane*, which best separates two classes in that dataset. The optimal hyperplane is the one which maximises the *margin* between itself and each class, where this margin is defined as the distance from the hyperplane to the closest data points from each class. Those data points used to define the margin of the optimal hyperplane are called the *support vectors*. However, a technique which relies on establishing a single hyperplane in the input space to separate two classes is restricted to linearly separable classification problems, whilst the majority of real-world classification problems are nonlinear.

To overcome this limitation, SVM first nonlinearly maps the data from the input space to some very high-dimensional abstract space known as a *feature space*. This is motivated by Cover's theorem on the separability of patterns, which states, in qualitative terms, that a complex classification problem nonlinearly transformed to a high-dimensional space is more likely to be linearly separable than in the original low-dimensional space [26, 57]. The optimal hyperplane is then constructed in the feature space, which corresponds to a nonlinear decision boundary in the input space. But a transformation of data to some high-dimensional feature space is likely to be computationally intensive, and it is unclear as to how high the dimensionality of that feature space should be. However, it can be shown that the only operation that must be performed in the feature space is the calculation of the inner products of vectors in that space which correspond to vectors in the input space. Furthermore, this inner product calculation can be substituted by an *inner product kernel*, which is a function defined in terms of vectors from the input space [57]. This substitution is commonly referred to as the *kernel trick* [79, 99]. For a more detailed description of SVMs, the reader is referred to [57] and [120].

Tax and Duin [114, 115] proposed a new novelty detection mechanism based on SVM. In their approach, which they called support vector domain description (SVDD), the objective is not to locate an optimal hyperplane separating two classes of data, but instead to identify the hypersphere with smallest radius that encompasses most of the data points in a given training dataset. To avoid the most outlying points of the training set from exerting strong influence over this hypersphere, such outliers are permitted to lie outside the hypersphere. Those test data points which lie outside the hypersphere are classified as novel. Since data is not typically spherically distributed, SVDD, like SVM, first maps data points to a feature space. Again, this is performed using an inner product kernel, which the authors choose to be a Gaussian kernel. This kernel has a free parameter σ which decides its width. The larger the parameter σ, the less support vectors are used and the more sphere-like the decision boundary in the input space. SVDD was compared against four other novelty detection methods [115], including a KNN approach. Over four novelty detection benchmark datasets, SVDD was shown to perform comparably, and often better than, the other methods.

Schölkoph et al. [99] proposed an alternative SVM approach for novelty detection. In this approach, instead of attempting to find the optimal hyperplane which separates data points from two classes, the hyperplane which best separates the data points from the origin of the feature space is sought. The best hyperplane is one which (a) separates all training points from the origin and (b) maximises the distance between itself and the origin. In the training

phase, all data points are assumed to belong to the normal class. In the test phase, each new point presented is mapped to the feature space and classified depending on which side of the hyperplane it lies. Data points are considered to be novel if they lie on the opposite side of the hyperplane to the training data. Because this approach classifies data points as either belonging to the normal class or not, it is referred to as one-class SVM [79], or simply 1-SVM [75]. One drawback of this approach, however, is that it makes the assumption that the origin of the feature space falls inside an area covered by the novel class [100].

Campbell and Bennett [13] proposed an approach which avoids this drawback of 1-SVM. In this approach, instead of repelling a hyperplane away from the origin of the feature space, the hyperplane is attracted onto the training data points [13]. This corresponds to a surface in the input space that wraps around the data; which, according to the authors, can then be easily found using linear programming techniques. In order to cope with noisy data, outliers may again be ignored when constructing the surface. After the training phase is complete, all data points lying outside the surface are considered as novel. The approach was demonstrated over the Biomedical dataset, as well as on data taken from a condition monitoring task, where it was shown to perform well.

Ma and Perkins [78] proposed a framework for adaptive novelty detection with confidence over time series data. The framework defines novelty to be an event in the time series consisting of a certain number n of unexpected data points, termed surprises. The number of surprises n required for an event to be considered as novel is chosen such that the probability of the n surprises occurring in a normal segment of the time series is below some predefined confidence level. The authors demonstrate the framework using a technique called support vector regression (SVR) to model the time series. SVR applies the SVM method to the problem of establishing nonlinear regression lines through the data [105]. The hyperplane established in the feature space corresponds directly to this regression line in the input space. The authors had previously combined SVR with an incremental learning technique for SVM, proposed by Cauwenberghs and Poggio [22], to allow it to adapt to changes in the normal conditions of the time series [80]. This modified version of SVR is used in the currently presented study [78]. The regression function constructed by SVR defines the model of normality, and deviations from this model constitute surprises. The approach was demonstrated over both synthetic and real-world univariate time series datasets. It was shown to work well over three sine-based synthetic time series, correctly identifying an artificially generated anomaly in each series with no false positives. The real-world time series used is the well-known Santa Fe Institute Competition dataset A [123]. Here,

two anomalies were highlighted by the detector, which were validated through visual inspection by the authors.

Desobry *et al.* [30] proposed an online abrupt change-point detection algorithm, based on 1-SVM. This approach considers, for a given time step t, two descriptor subsets: the immediate past subset $\mathbf{x}_{t,1} = \{x_i\}_{i=t-m_1,..t-1}$ and the immediate future subset $\mathbf{x}_{t,2} = \{x_i\}_{i=t,...,t+m_2-1}$, where x_i is the data point from the time series at time step i and m_1 and m_2 define the number of data points in the subsets. The approach is based on the premise that, in the absence of an abrupt change, the location of $\mathbf{x}_{t,1}$ and $\mathbf{x}_{t,2}$ in the input space is the same. A *dissimilarity measure* for $\mathbf{x}_{t,1}$ and $\mathbf{x}_{t,2}$, derived from the hyperplanes generated by two 1-SVM classifiers, is defined in the feature space but computed in the input space using the kernel trick. For each time step t, the first 1-SVM classifier is trained on the descriptor subset $\mathbf{x}_{t,1}$, whilst the second classifier is trained on the subset $\mathbf{x}_{t,2}$. The dissimilarity measure is computed for $\mathbf{x}_{t,1}$ and $\mathbf{x}_{t,2}$ and thresholded to determine the existence of a change-point. Instead of the computationally inefficient approach of completely retraining the two classifiers from scratch on each time step, the incremental learning method proposed by Cauwenberghs and Poggio [22] may be employed to update the classifiers. Whilst the authors have not demonstrated this technique in a novelty detection task, abrupt change-point detection can potentially be used to perform novelty detection in scenarios where such abrupt changes are unexpected.

Evaluation

SVM is another popular approach to novelty detection. The original SVM model has been extended to allow both one-class classification [99] and adaptive learning [22]. Approaches based on SVM have been demonstrated in novelty detection tasks based on time series [78] and record datasets [99]. However, the success of an SVM-based approach in a particular application depends primarily on how the data is preprocessed as well as the selection and tuning of an appropriate kernel [119]. Deciding on the appropriate preprocessing method and kernel to use in a particular application requires expert knowledge, which may potentially prevent a user from being able to employ such an approach in their novelty detection task.

2.2.4 Kernel-based Online Anomaly Detection

The final statistical approach to novelty detection we consider is Ahmed *et al.*'s Kernel-based Online Anomaly Detection (KOAD) approach [3], which is based on Engel *et al.*'s Kernel Recursive Least Squares (KRLS) algorithm [37]. This is related to SVMs in that it again exploits the kernel trick to transform the data to some high-dimensional feature space. However, to the best of our knowledge KOAD appears to be a unique approach to novelty detection which does not directly fall into the groups of approaches considered above. Therefore, we present this approach in isolation here.

KOAD is motivated by the assumption that, when transformed to some high-dimensional feature space, normal data should cluster and so is likely to be able to be described by a *dictionary* of a relatively small number of data elements [3]. As with SVM, this high-dimensional feature space is considered only indirectly by using the kernel trick. Each data element presented to the algorithm is compared with this dictionary by evaluating the distance between the element and the cluster defined by the dictionary in the feature space. If this distance exceeds a predefined threshold, ν_2, then a "Red1" alarm is raised by KOAD and the data element is labelled as novel. If this distance is below a second threshold, ν_1, then the data element is deemed to be sufficiently similar to the model of normality represented by the dictionary, and so is labelled as normal. However, if the distance lies between ν_1 and ν_2, then the data element is regarded as representing *unusual* behaviour. It may signify an anomaly, but it could also indicate a change in the model of normality. To resolve this, the data element is re-evaluated after l subsequent time steps. An "Orange" alarm is also raised to signal the occurrence of the unusual event. In the re-evaluation, the data element is compared with those data elements seen by KOAD during the l time steps. If the data element is deemed to be similar enough to a significant number of the l subsequent elements, then it is considered to represent normal behaviour. Otherwise, the data element is considered to be novel and a "Red2" alarm is raised. The dictionary used by KOAD is continuously updated. Data elements which raise an "Orange" alarm but fail to raise a "Red2" alarm (i.e. unusual yet normal elements) are added to the dictionary, since they represent the limits of the dictionary's knowledge of the current model of normality. Elements in the dictionary are periodically re-evaluated against the L data elements seen most recently. Those elements in the dictionary that no longer appear to represent normality are subsequently deleted.

The authors compared KOAD with two other novelty detection algorithms on an anomaly

detection task over a multivariate time series dataset describing network traffic behaviour [3]. Before proceeding, a number of anomalies in the data were manually identified by the authors. In the experiment, KOAD was shown to give a detection performance comparable to the other approaches, but was quicker at identifying anomalies. KOAD was also applied to the very different novelty detection task of identifying unusual events from images taken from six cameras monitoring a road network [4]. Again, KOAD was shown to give a good performance.

Evaluation

KOAD has been shown to successfully identify novel events in two very different scenarios, indicating the possible versatility of the approach. However, as previously mentioned, KOAD is only suitable for novelty detection tasks over time series datasets. The algorithm also requires a number of parameters to be set by the user, with the threshold parameters ν_1 and ν_2 having a critical impact on performance. Currently, there is no automatic way of deriving these thresholds, but such an automatic method is currently the subject of further research by the authors [3, 4].

2.3 Neural Network Approaches to Novelty Detection

We now shift our focus to those approaches to novelty detection based on artificial neural networks. An artificial neural network, or simply *neural network* [57], is a collection of processing units, *neurons*, connected in some manner with directed links known as *synapses* with some strength or *weight*. They are motivated by biological neural systems such as the human brain, which is capable of performing tasks such as pattern recognition, perception and motor control many times faster than a digital computer [57]. A neural network encodes *knowledge*, acquired from its environment through the process of learning, into the weights of its synapses [57]. In addition to supervised and unsupervised learning, neural networks can also be trained using *reinforcement learning*, where, through continuous interaction with an environment, the learner identifies behaviour which yields some reward from a trainer or critic [57, 90].

Many different kinds of neural network have been discussed in the literature. We first look at the application of one of the most widely used types, the multilayer perceptron [84], to novelty detection. We then proceed to examine autoassociative networks, radial-basis function networks, self-organising maps, and networks based on Grossberg's theory of adaptive resonance. Finally, we examine approaches which combine neural networks with evolutionary search algorithms to

allow the automated construction of novelty detectors for particular tasks.

2.3.1 Multilayer Perceptron

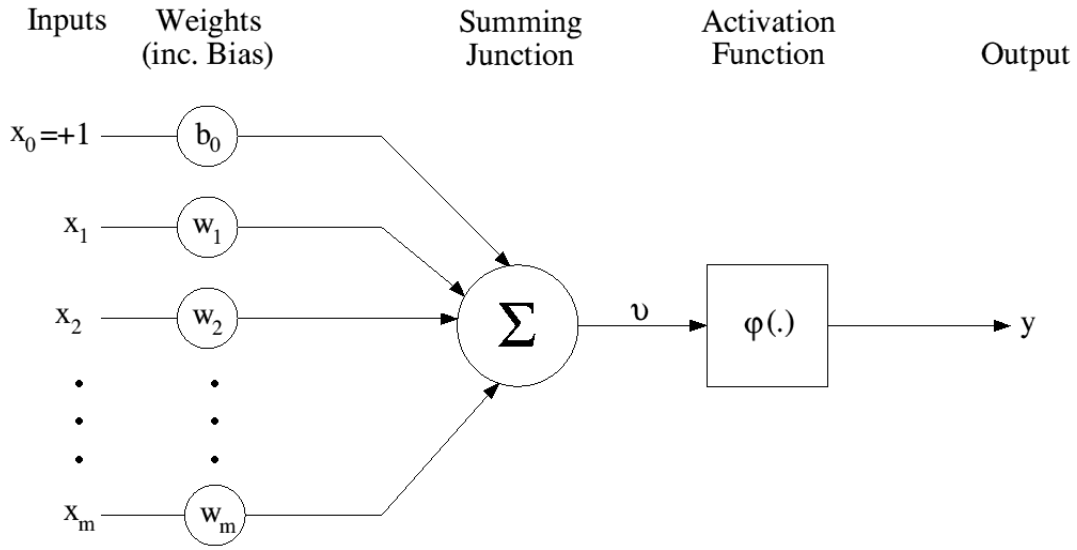


Figure 2.1: The McCulloch-Pitts model of a neuron. This model is used by Rosenblatt to implement the single-layer perceptron [57].

The perceptron, originally proposed by Rosenblatt [96], is the simplest form of neural network capable of pattern classification [57]. It consists of a McCulloch-Pitts model of a neuron, shown in figure 2.1, with adjustable synaptic weights and bias whose values are determined by a supervised learning approach. When used for classification, this *single-layer perceptron* is able to establish a linear decision boundary (i.e. a hyperplane) in the input space, thus limiting it to two-class classification problems in which the classes are linearly separable. However, as we mentioned in section 2.2.3, most classification problems require nonlinear decision boundaries to be constructed. This motivates an extension to this technique, known as the *multilayer perceptron*.

In a multilayer perceptron (MLP), such as that shown in figure 2.2, layers of perceptrons are connected by synapses [57]. The first and last layers are known as the input and output layers respectively. The input layer contains nodes which perform no computation, and as such is not usually included when specifying the number of layers. Intermediate layers are called hidden layers. In an MLP, each perceptron has a nonlinear activation function, such as the *logistical function*:

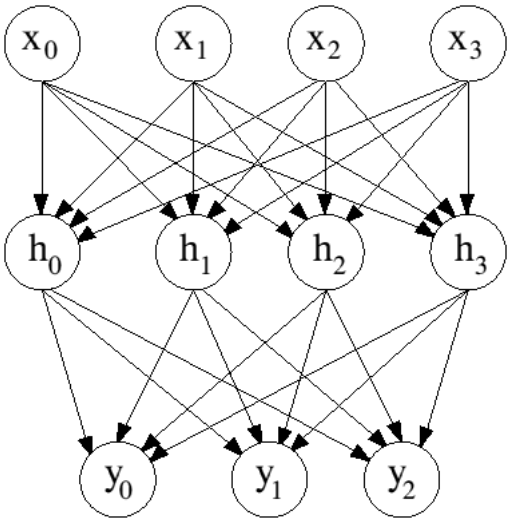


Figure 2.2: An example multilayer perceptron network.

$$y = \frac{1}{1 + exp(-v)} \tag{2.1}$$

where v is the weighted (by synaptic weights) sum of all inputs received by a neuron and y is the output of this neuron. The nonlinearity of the function is important since, without it, the input-output relation of the network would be reduced to that of the single perceptron [57]. Different degrees of connectivity may also exist in an MLP. For example, each neuron in each input and hidden layer may be connected to every neuron in the next layer, as in the network shown in figure 2.2. Such a network is said to be *fully connected*. Alternatively, each neuron may be connected only to some of the neurons in the next layer. In this case, the network is said to be *partially connected*.

As mentioned above, a single-layer perceptron is capable of constructing a linear decision boundary in the input space to classify patterns from two classes. Because an MLP consists of multiple perceptrons, it is capable of constructing several separating hyperplanes, with each perceptron being responsible for constructing its own hyperplane [57]. This allows classification problems with multiple classes to be tackled, as well as problems in which the classes are nonlinearly separable. The supervised learning algorithm most popularly used for training MLPs is the well-known error back-propagation algorithm.

For multiple class classification problems, each neuron in the output layer is often used to represent one class [62]. When presented with a training pattern belonging to the ith class, the desired response from the network is a high-magnitude output from the ith neuron in the

output layer and a low-magnitude output from all other neurons. A simple decision rule known as *winner-takes-all*, in which the output neuron with the highest magnitude output is identified as the winner, can then be used to decide the class selected by the network. However, like many classifiers, a problem with MLPs is their tendency to incorrectly classify patterns which do not belong to any of the known classes [121]. Therefore, approaches have been proposed that enable an MLP network to recognise data that doesn't belong to any known class, or, in other words, detect novel data.

One way of avoiding this misclassification problem is to train the network to reject novel patterns, where examples of novel patterns are approximated by random patterns [121]. However, this is unreliable since it assumes that the random patterns are suitably similar to actual novel patterns. Instead, Vasconcelos *et al.* [121] proposed an alternative approach: when an input pattern is presented to the trained network, the response of the networks output neurons are evaluated to determine if the output of the winning unit exceeds all other outputs by more than some confidence level. If this occurs, then the network accepts that input as belonging to the class represented by the winning output neuron. Otherwise, the input is considered to be novel and rejected from classification. When rejection occurs, a control mechanism then examines the difference between the output of the winning and second-best neurons in the output layer. If this difference is greater than some threshold, the input is assumed to be very different from what was seen before and the network is retrained to reinforce this rejection. During this retraining, the rejected pattern is associated with the output vector $[0, 0, ..., 0]$.

Singh and Markou [104] used a similar approach in applying novelty detection to natural scene analysis. However, in this approach, random patterns are used during the training phase to approximate examples of the novel class, with the patterns generated in such a way that they surround the known classes in the input space. The task of the MLP network in the scenario considered by the authors is to categorise image feature samples as either *known* or *unknown*. Those samples classified by the network as unknown could be novel patterns belonging to unknown classes or, alternatively, might be outliers of a known class. This is determined by performing clustering on all unknown samples, and comparing each cluster to the input distribution of each known class. Those clusters which are separable from these input distributions are taken to represent unknown classes. Those patterns belonging to these unknown classes are then used to adaptively retrain the MLP network. For each unknown class, a new output neuron is introduced to the neural network, along with a number of new hidden neurons. The novel patterns are then used to incrementally train the network to correctly classify data from

the new classes. The authors demonstrated the approach in a video recognition scenario, where the task was to identify novel objects, such as a briefcase or a piece of clothing, positioned in an outdoor environment from a video sequence recorded in that environment. The approach was shown to give a good performance in this task, outperforming three other novelty detection methods.

Cordella *et al.* [25] proposed an alternative approach to that given by Vasconcelos *et al.*. For a particular application, a performance measure is first defined as a function of the recognition, reject and misclassification rates obtained by the network. Since different applications have different tolerances of these rates, this measure is naturally application-specific. The MLP is then trained as normal and tested on a set of samples without being given a reject option. That is, the network must attempt to classify each element of the sample. This is used to derive two reject threshold values, r_1 and r_2, which are then used to implement two classification rules. The first rule, winner-takes-all with reject option (WR-Rule), is used for each data instance applied to the network. This performs precisely like winner-takes-all except that the output of the winning neuron is also thresholded, using r_1. If the output is below this threshold, then the data instance is rejected from classification. If rejection does not occur, then the second classification rule, winner-takes-all with reject-on-difference (WD-Rule), is applied. The WD-Rule examines the difference between the output of the winning neuron and that of the second-best neuron in the output layer. If this difference is less than the second reject threshold r_2, the data instance is rejected. Data instances not rejected by either classification rule are assigned the class represented by the winning neuron.

Evaluation

The approaches based on MLP presented here are primarily focused on classification, implementing novelty detection to allow data from unknown classes to be rejected and so reducing misclassification error. They also rely on a supervised learning technique, meaning that they have the drawback of requiring examples from the novel class. However, this drawback can be overcome by either substituting these examples with random data, which is generated in such a way so as to fall outside the normal class [104], or by reinforcing the rejection of novel data encountered after the training phase [104, 121]. Whilst MLP networks are typically trained offline, both the approach by Vasconcelos *et al.* and that by Singh and Markou allow retraining of the network to occur after the initial training phase. The retraining method used by Singh and Markou is more advanced since it allows the network to learn new classes. However, it does

not permit the forgetting of known classes.

The key advantage of MLP is the level of customisability it offers. This is the most flexible kind of neural network considered in this chapter. MLP networks can assume a wide variety of different structures and connectivity patterns, and therefore can be used to implement many different functions. Whilst the networks considered in this section limit themselves to a layered structure, in section 2.3.6 we describe an approach that uses MLP networks that do not adhere to this type of structure. Different learning rules and activation functions could also potentially be used in an MLP, extending their applicability to tasks requiring adaptive learning capabilities. Therefore, this kind of neural network fulfils two of the objectives stated in chapter 1: it is highly customisable which, in turn, makes it likely that MLP networks can be constructed for a wide variety of different novelty detection tasks.

However, this important advantage of the MLP neural network is associated with its greatest disadvantage. The problem of designing a suitable neural network structure for a particular task is non-trivial, and usually requires the skills of an expert in the field of neural networks [130]. In addition to this, suitable parameters must be chosen for the activation function and the particular learning rule employed. These significant drawbacks prevent MLP from fulfilling our third objective of minimising the work required from the user in configuring the approach for their particular novelty detection task.

One way to avoid this "curse of customisability" is to use an approach in which the structure of the network is partially predefined. We now explore this direction by evaluating four such neural network approaches which have been commonly used to construct novelty detectors.

2.3.2 Autoassociative Networks

An autoassociative neural network, also known as an *autoassociator*, is one whose function is to reconstruct inputs applied to the network at the output layer [73]. A variety of different topologies have been used to implement autoassociative networks. For example, the network may be a fully connected single-layer network [39], which may be trained with a Hebbian learning rule [39]. Alternatively, one or more hidden layers may exist, where at least one of the hidden layers, commonly referred to as a *bottleneck layer*, has fewer neurons than either the input or output layers [73, 97]. Such networks are sometimes referred to as autoassociative neural network encoders [118], or simply *autoencoders*, since at the bottleneck layer they construct some encoding of the input vector which is then decoded to give the reconstruction at the

output layer.

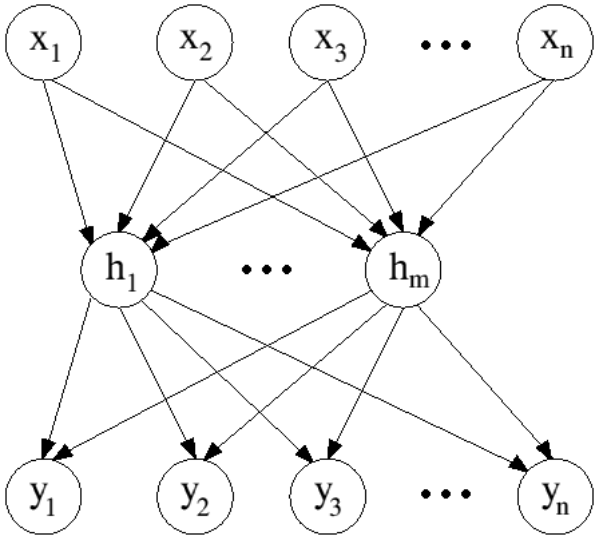


Figure 2.3: The autoassociative neural network proposed by Rumelhart *et al.* [97]. This was shown to effectively solve Ackley *et al.*'s encoder problem [2].

An autoassociative network with a single hidden layer, illustrated in figure 2.3, was shown by Rumelhart *et al.* [97] to effectively solve the *encoder problem*, originally posed by Ackley *et al.* [2]. The encoder problem is to find a way of allowing two visible units, V_1 and V_2, to communicate a message of size n through a hidden unit H of size m, where $m < n$ [2]. Since H has a smaller size than V_1 and V_2, it serves as a bottleneck between the two visible units, hence the name *bottleneck layer*. The network is trained by minimising the sum-of-squares error function [97]:

$$E = \frac{1}{2} \sum_i \sum_j (x_{ij} - y_{ij})^2 \tag{2.2}$$

where x_{ij} is the jth component of input pattern i and y_{ij} is the jth component of the output pattern produced when pattern i is applied to the network. The minimisation of this function can be viewed as a kind of unsupervised learning task, since no independent target data is provided. Others refer to this as a form of *self-supervised* learning [73]. If the hidden neurons have linear activation functions, then, at the global minimum of the error function, the network performs a dimensionality reduction operation called *principal components analysis (PCA)*. This technique finds an m-dimensional representation of the original n-dimensional pattern, where $m \leq n$, such that the original pattern can be reconstructed from this representation with minimal error [57]. The m-dimensional representation is a linear transformation of the original pattern. For a

detailed discussion of PCA, the reader is referred to [9, 57].

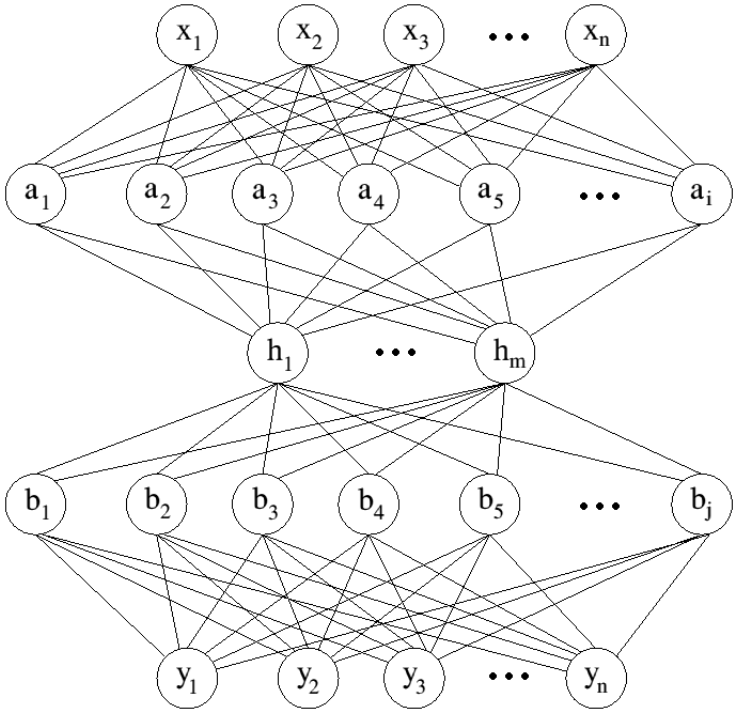


Figure 2.4: The autoassociative neural network proposed by Kramer [73], which is capable of performing nonlinear principal components analysis.

Kramer [73] proposed an autoassociative network, shown in figure 2.4, which is capable of performing nonlinear PCA (NLPCA). Unlike PCA, NLPCA finds a nonlinear transformation of a n-dimensional pattern to m-dimensional space. This allows NLPCA to find and eliminate nonlinear correlations in the data. The autoassociative network used here has three hidden layers: a *mapping layer*, which maps the original pattern to m-dimensional space, the familiar *bottleneck layer* with m neurons, and a *demapping layer*, which reconstructs the original pattern. The neurons in the mapping and demapping layers have logistical activation functions, whilst the neurons in the bottleneck and output layers may use either linear or logistical activation functions.

Since an autoassociative network is trained to reproduce known patterns, it can be used to determine whether or not a given pattern is known, or similar to known patterns, or if it is in fact novel. This allows an autoassociative network to be used as a novelty detector. If the error in the reconstruction of a particular pattern is above some threshold, then that pattern may be considered to be novel. This approach is used by Japkowicz *et al.* [65], who apply the method of novelty detection to a two-class classification problem. An autoassociative network, using the

architecture given by Rumelhart *et al.*, is trained on examples from one class and then used to label data instances from both classes. The threshold applied to the reconstruction error to determine the class to which a data instance belongs is decided during the training phase. For noiseless data, the threshold can be determined with instances from one class only. For noisy data, instances from both classes are required.

A similar approach is used by Sohn *et al.* [106]. They use the error between reconstructed and actual patterns, measured in terms of Euclidean distance, to define a novelty index. Here, the network architecture proposed by Kramer is used. A threshold value may again be established during the training phase. The approach is tested on a simplified numerical model of a computer hard disk, which we use to test our proposed approach in chapter 5. After training on data derived from the model in a normal state, the network is tested using both normal data and data generated after simulating damage in the model. Four different damage scenarios are investigated, where in each scenario two of the models variables are perturbed in some manner. In three of the four damage scenarios, the novelty index generated by the autoassociative network successfully highlighted those data points pertaining to the simulated damage state. A similar approach is used by Thompson *et al.* [118] and Martinelli *et al.* [86] for the tasks of identifying anomalies in network hub CPU usage data and observations from electric power system substations respectively. Thompson *et al.* use the network architecture given by Rumelhart *et al.*, whilst Martinelli *et al.* use Kramer's architecture.

Another very different kind of autoassociative network is the Hopfield network [39]. This network has a single layer of neurons, which are interconnected by *feedback* connections but with self connections (i.e. a connection between a neuron and itself) disallowed. During a *storage phase*, a series of patterns are presented to a Hopfield network which it stores by modifying its synaptic weights, using a Hebbian learning rule. In this phase, the network associates each pattern with itself. After the storage phase, the network can be placed into a *retrieval phase*. First, a partial or corrupted input pattern is placed on the neurons of the network. The network then iteratively updates the states of the neurons until convergence (no state change between successive iterations) is reached. If the partial or corrupted pattern references a known pattern, i.e. a pattern stored in the network, then that known pattern is reconstituted on the neurons. Otherwise, a spurious pattern is returned.

In order to assess the convergence of the network to stable state over time, Hopfield defined an energy function [61] which decreases to a local minimum as the network stabilises during the training phase. Bogacz *et al.* showed that this energy function yields a higher value for an

unknown pattern than for a pattern stored in the network [10], thus allowing the network to be used as a novelty detector. For a prospective pattern, the energy function is calculated and the obtained value thresholded to produce a binary decision. The energy function is defined in terms of the prospective pattern and the networks weights, meaning that the novelty of the pattern can be evaluated in fixed time [28]. Crook and Hayes [28] used a Hopfield network to continuously learn and detect changes in an environment explored by a mobile robot. Information about the environment was provided in the form of video sequences taken from a camera mounted on the robot. Each time an input pattern was classed as novel, it was learnt by the Hopfield network. Similar patterns were subsequently classed as normal. Whilst this approach allows the network to learn about, and update itself to changes to, the environment; it does not allow forgetting. Therefore, if a change to an environment is reversed, and subsequently reintroduced at a later time, this reintroduction would not be detected by the network.

Evaluation

Autoassociator networks have a remarkable capability of autonomously learning the underlying characteristics of normal data presented to the network [118]. This knowledge can be used to identify data that does not conform to the learned model of normality. A measure of novelty is calculated based on the difference between observed and reconstructed vectors, which, depending on the required degree of granularity, may be used to define a scalar novelty index [65, 86, 106, 118]. Whilst an autoassociator network has a structure with fewer free parameters than an MLP network, the user must still decide the size of any hidden layers. For example, if too few nodes are used in the mapping and demapping layers present in the architecture proposed by Kramer, then the accuracy of the reconstruction vector may be low because of the limited representational capacity of the network [73]. If too many nodes are used on the other hand, then the network may be prone to overfitting. The best number of neurons to have in these layers can be determined experimentally. In either Kramer's or Rumelhart *et al.*'s architecture, the size of the bottleneck layer corresponds to the number of factors to be derived from the data. If this is not known *a priori*, then the size of the bottleneck layer must also be determined experimentally.

Of the autoassociative approaches to novelty detection reviewed here, only the Hopfield network has been shown to be capable of adaptive learning [28]. However, a Hopfield network can only store a finite number of patterns reliably [39], and is not capable of *forgetting* stored patterns [85]. This means that, with enough novel patterns, the network will eventually reach its

storage capacity, after which the classification error rate will increase [28]. In their experiments, the Hopfield network used by Crook and Hayes could theoretically store a maximum of $122,093$ patterns with a maximum error rate of 1% [28]. The lack of a forgetting mechanism also limits the networks applicability to scenarios where novelty is based on only recently acquired knowledge.

2.3.3 Radial Basis Function Networks

We next consider the *Radial Basis Function Neural Network* (RBFNN). In these types of network, the hidden neurons have a *radial basis function* (RBF) as their activation function, which depends on the distance between the input vector $\mathbf{x}$ and a prototype vector $\mathbf{x}^i$. RBFs were originally used as a technique for performing exact interpolation of a set of data points [9]. Interpolation is a kind of curve-fitting technique, where the curve, or surface in some space, passes through the set of data points. The exact interpolation problem consists of finding some mapping function $h_k(\mathbf{x})$ which exactly maps a series of n prototype vectors $\mathbf{x}^i$ onto corresponding target vectors $\mathbf{t}^i$, that is, for every i [9]:

$$h_k(\mathbf{x}^i) = \mathbf{t}_k^i \tag{2.3}$$

where $\mathbf{t}_k^i$ represents the kth component of the ith target vector. Thus, $h_k(\mathbf{x})$ defines a surface which exactly passes through each target vector. The RBF technique of finding $h_k(\mathbf{x})$ consists of expressing the function as a series of RBFs $\phi\left(||\mathbf{x} - \mathbf{x}^i||\right)$ [57], where $||\mathbf{x} - \mathbf{x}^i||$ describes the distance, normally taken to be Euclidean, between the vectors $\mathbf{x}$ and $\mathbf{x}^i$. The number of RBFs corresponds to the number n of prototype vectors X^i (input vectors for which the corresponding target vector is known) with the vector $\mathbf{x}^i$ defining the *centre* of the ith RBF. The vector $\mathbf{x}$ represents an input vector for which the target vector is to be approximated using the interpolation method. The mapping function $h_k(\mathbf{x})$ is expressed in terms of the RBFs as [9]:

$$h_k(\mathbf{x}) = \sum_i w_{ki}\phi\left(||\mathbf{x} - \mathbf{x}^i||\right) \tag{2.4}$$

where the weight coefficients w_{ki} may then be derived, given the basis functions and the target values $\mathbf{t}_k^i$.

The RBFNN approximates the RBF technique to find a multidimensional surface which

approximately passes through a series of training vectors, and using this surface to derive approximate target vectors for test data [57]. Instead of constructing a mapping function with one RBF for each of the n training vectors, m RBFs are used, with m, typically much less than n, determined by the complexity of the mapping to be represented [9]. Instead of using the training vectors themselves, the centres of the RBFs are determined by a training phase, as well as any other parameters required by each individual RBF. In common with other neural network approaches, an RBFNN may also feature a bias parameter.

In its most basic form, the RBFNN consists of a single hidden layer [57], where each neuron uses a radial-basis function as its activation function. At the output layer of the network, the neurons are linear, that is they output a linear summation of their inputs. Training of an RBFNN takes place in two phases [9]. First, the training set is used to determine the parameters of the basis functions (including the centres of the functions), governed by the weights between the input and hidden layers. Second, the weights of the network between the hidden and output layers are found, typically by minimising a sum-of-squares error function.

Li *et al.* [76] use an RBFNN to perform classification in the context of condition monitoring and fault diagnosis. The output layer contains one output neuron for each known class of data, in the same manner as a multilayer perceptron classifier. These classes may represent normality, or known fault conditions. Each output neuron is also thresholded to determine whether an input vector is truly from a known class, representing known normal or fault conditions, or from an unknown novel class. In experiments over two datasets, each artificially generated from a different mathematical model, the introduction of this threshold was shown to reduce the misclassification error of the RBFNN, as it rejected novel data from classification.

One drawback of the RBFNN is that the user must decide how many neurons are required in the hidden layer of the network for their particular scenario. If the hidden layer has too few neurons, then insufficient learning will result, whilst too many neurons causes overfitting to occur [127]. Wu and Chow [127] overcome this drawback by allowing the number of hidden neurons to be decided by a mechanism, which they call a cell-splitting grid (CSG), based on Kohonen's self-organising map (presented in the next section). The CSG is a growable variant of the self-organising map, which introduces neurons into the map using a "cell-splitting" mechanism. When a neuron has been activated a certain number of times, that neuron is deleted and four new neurons are introduced within its vicinity in the map. This method allows the distribution of the neurons in the map to preserve the data distribution in the input space. After the CSG has been trained, the number of neurons in the map defines the size of the RBFNN's hidden

layer. Furthermore, the weight vectors associated with each neuron in the map define the centres of the RBFs represented by the hidden neurons. Wu and Chow used this approach to detect both mechanical and electrical faults in a three-phase machine system. Once trained, with the hidden neurons and their corresponding RBF centres determined by the CSG method, the RBFNN demonstrated a very high detection accuracy for both kinds of fault. Furthermore, the network was able to correctly determine the extent of each fault.

Another algorithm for automatically determining the number of hidden neurons in an RBF network was given by Berthold and Diamond [8]. This approach, Dynamic Decay Adjustment (DDA), enforces a separation between the output of the best matching hidden neuron for an input vector and all other neurons in the hidden layer. The separation is defined by two threshold parameters, θ^+ and θ^-, whose values are user-defined. The best matching neuron is required to output a value greater than θ^+, whilst all other neurons must output a value less than θ^-. During the training phase, if no hidden neuron outputs a value greater than θ^+ for an input pattern $\mathbf{x}$, then a new hidden neuron is introduced, centred on $\mathbf{x}$. Each hidden neuron uses a Gaussian activation function, with a width parameter σ_i, and connects to exactly one output neuron. For every training vector applied to the network, the width parameters of all hidden neurons are adjusted to ensure the continued enforced separation for each new input vector.

Oliveira *et al.* [92] used an RBF network with the DDA approach described above, which they called RBF-DDA, to perform novelty detection over short-length time series. In their approach, the sliding window preprocessing technique is used to obtain input vectors from the time series. This technique, which we discuss further in chapter 5, moves a window of n data points over the time series, with the segment of the time series covered by the window at a particular time step constituting the input vector for that time step. These vectors are then used to form a training and test set. One problem is that the original time series is only likely to contain examples from the normal class, whilst in order to test the RBF-DDA approach, novel data is required. To overcome this problem, artificial novel input vectors are generated, which deviate from the original normal patterns by some threshold. Artificial data is also generated to increase the number of available examples from the normal class. When tested on a number of different time series, this approach was shown to allow a good performance from RBF-DDA. However, despite claims from Berthold and Diamond that the DDA thresholds θ^+ and θ^- did not have a critical impact on performance [8], Oliveira *et al.* found that the value of θ^- had a considerable impact [93]. This motivated the authors to modify their proposed approach such that an optimal value for θ^-, as well as an optimal size for the sliding window, could be decided during the training

phase. This modification was shown to significantly improve the performance of RBF-DDA in time series novelty detection tasks.

Evaluation

RBFNNs have been shown to give a good performance over various novelty detection tasks. Whilst an RBFNN uses a standard predefined network architecture, one drawback is that the number of neurons in the hidden layer must be determined by the user. However, this is overcome by approaches such as DDA [8] or CSG [127] which automatically choose a suitable number of neurons based on the training data. DDA also determines suitable values for the width parameters of the hidden neuron activation functions. In section 2.3.6, approaches are also presented which use an evolutionary algorithm to automatically configure an RBFNN.

A drawback of all the RBFNN approaches presented here is that they are not capable of performing adaptive learning, and so are unsuitable for use in tasks which involve dynamic environments. Furthermore, the DDA approach restricts the user to a Gaussian radial-basis function, which may not be suitable in all scenarios. The training phase used in CSG is complex since it employs an approach based on Kohonen's self-organising map to decide the number of hidden neurons appropriate for the network. In the next section, we examine approaches which use the self-organising map directly as a method of novelty detection.

2.3.4 Self-Organising Map

Kohonen's Self-Organising Map (SOM) [70] is an unsupervised learning neural network. This network usually consists of a one- or two-dimensional grid of neurons, the map field, in which each neuron has associated with it a model of some observation. An input vector is projected onto the map, where it triggers one or more neurons. The neuron whose model best matches that input vector becomes the winning neuron. The models held by the winning neuron and its neighbours are then moved closer to the input vector. This means that similar inputs in the future are likely to activate nodes within this neighbourhood. In this way, the network demonstrates self-organisation. The SOM is a kind of clustering technique, as it groups similar data elements into different regions of the map field.

The SOM approach works as follows. Let an input vector $\mathbf{x}(t)$ at time t be denoted by:

$$\mathbf{x}(t) = [x_1(t), x_2(t), ..., x_n(t)]^T \tag{2.5}$$

where n is the dimensionality of $\mathbf{x}(t)$. Let each neuron i in a map field consisting of N neurons contain a model $\mathbf{m}_i$, which is a vector with the same dimension n as $\mathbf{x}(t)$. Then, $\mathbf{x}(t)$ is compared with each $\mathbf{m}_i$ to discover that $\mathbf{m}_i$ which best matches $\mathbf{x}(t)$. The corresponding neuron i is then considered to be the winning neuron. To find the best matching model, one calculates the distance between each model $\mathbf{m}_i$ and the input $\mathbf{x}(t)$. Whilst the most appropriate distance measure may depend on the specific application, a commonly used measure is that of Euclidean distance [85]. Then, the best matching neuron c is that whose model has the smallest Euclidean distance to the input [71]:

$$c = argmin_i||\mathbf{x}(t) - \mathbf{m}_i|| \tag{2.6}$$

Both the winning neuron c and those neurons within c's *neighbourhood* then have their models modified such that they better represent the input $\mathbf{x}(t)$. This neighbourhood is described by a neighbourhood function h_{ci} whose maximal value is given for the winning neuron [71]. The value of h_{ci} decreases with increasing distance of neuron i from the winning neuron c in the map, and may also change with time. Therefore, h_{ci} decides the *size* of the neighbourhood. A variety of different mathematical functions may be assigned to h_{ci}, provided that the neighbourhood they define is symmetric about the winning neuron and that their amplitude decreases with increasing distance from this neuron [57]. For each neuron i in the map, the model $\mathbf{m}_i$ at time $t+1$ is updated using the following learning rule [57]:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \eta(t)h_{ci}(t)(\mathbf{x}(t) - \mathbf{x}_i(t)) \tag{2.7}$$

where $\eta(t)$ is a time-varying learning rate parameter and $h_{ci}(t)$ is the value given by the neighbourhood function for neuron i at time t, which is 0 if i does not lie in the neighbourhood of winning neuron c. The neighbourhood function may be defined such that the neighbourhood size decreases with time [57]. Similarly, one may use a decay function to derive a decaying learning rate. For example, $\eta(t)$ might be defined as [85]:

$$\eta(t) = e^{-Kt/\tau} \tag{2.8}$$

for some constants K and τ.

Initialisation of the values of each model $\mathbf{m}_i$ can be performed using randomly chosen values from the input vector space. However, to allow the learning process to become smoother and

faster, Kohonen recommended that the initial values be chosen as a regular array of vectors on the hyperplane spanned by the two largest principal components of input data [71].

One SOM-based approach to novelty detection was proposed by Theofilou *et al.* [117]. They replaced the original learning rule given in equation 2.7 with an identical but inverse alternative, which they referred to as a long-term depression rule. For a given training input pattern, rather than the probability of the winning node firing for similar input patterns being increased, it is instead decreased by this new rule. This results in a higher response from the network when input patterns unlike those seen in training are applied, thus allowing the network to function as a novelty detector. However, if the environment in which the network is functioning changes, it will fail to correctly identify novel inputs until it has been retrained. This limitation also applies to the original SOM.

In order to allow SOM to operate in dynamic environments, Marsland [85] proposed a variation which exploits a commonly observed biological phenomenon known as *habituation*. Habituation refers to a reversible decrement in the response of some entity to a stimulus that has been seen repeatedly without ill effect [85]. In this Habituating SOM (HSOM), each neuron in the map is connected, via an habituating synapse, to an output neuron. The neuron selected as the winning neuron then fires along this synapse to the output neuron. As it does so, the strength of the habituating synapse decreases, or habituates. Thus, a frequently winning neuron will be able to induce a lower level of activation in the output neuron. This allows novelty detection in that the output neuron of the HSOM will give a high level of activation only for those neurons in the map field that have been activated infrequently. The output value given by the HSOM for a particular input vector quantifies the novelty of that vector. That is, instead of simply labelling the vector as normal or novel, the HSOM is able to provide a more detailed description, using labels such as "normal", "familiar", "unusual" and "novel". The habituable synapses used by the network are also able to forget, or *dishabituate*. That is, if a neuron in the map which was previously frequently chosen as the winning neuron no longer fires, then that neuron's habituable synapse begins to recover its strength. This allows the HSOM to dynamically adapt to the changing novelty of perceptions. Marsland demonstrated the HSOM in the robot inspection task of identifying novel features present in the corridors of a building [85]. Whilst it was shown to give a good performance, the approach had one critical limitation. Because the number of neurons in the map field of a SOM are fixed, it is possible for saturation to occur in the HSOM. That is, a novel perception presented to the network is misclassified as normal, because all neurons in the map have been used to represent other normal perceptions.

To overcome this limitation, Marsland [85] proposed the Grow When Required (GWR) neural network. This kind of SOM starts with two neurons in the map field. New nodes are introduced when the existing nodes are deemed not to satisfactorily classify particular input patterns applied to the network. Therefore, saturation can no longer occur since the size of the map field is no longer fixed. As with the HSOM, GWR provides an output value which quantifies the novelty of an input vector. In addition to the robot inspection tasks for which the HSOM was used, Marsland demonstrated GWR over a number of novelty detection tasks [85], such as detection of abnormal blood samples, fault detection over data obtained from the testing of ball-bearing cages and novelty detection over visual data.

Vieira Neto and Nehmzow [122] examined the performance of GWR with an incremental-learning variant of PCA for the task of novelty detection over visual images recorded by a mobile robot. Rather than considering an entire image frame, patches of the image were instead selected from the visual input by an attention model. These patches were then normalised and passed to the novelty detector for evaluation. In experiments where the robot explored an indoor arena, both GWR and the incremental PCA method were shown to successfully highlight two novel objects, an orange football and a grey box, that had been placed in the arena after training.

Evaluation

The structure of the SOM neural network is again well defined and has few free parameters. The dimensionality of the input and model vectors is application-specific, and can easily be determined by the user. Whilst in the original SOM the size of the map field had to be determined by the user, Marsland's GWR approach allows this map field to automatically and continuously adjust its size as required by the input data. This also potentially allows the network to maintain a structural complexity that is justifiable for the novelty detection task at hand. Since GWR also permits adaptive learning, it can be used in scenarios involving dynamic environments. This approach has also been demonstrated in a variety of different novelty detection tasks [85, 122], suggesting a high degree of versatility.

However, GWR also requires a number of parameters to be set, which increases the work required from the user in a new application. Whilst it has been suggested that only one parameter, the activity threshold, is important to customise for the particular application [29], Marsland repeatedly specified different values for a range of GWR parameters in various experiments [85]. Furthermore, since GWR is designed to provide a value indicating the degree of novelty of a particular input [85], a threshold parameter must be defined in order to translate this into the

binary output *normal* or *novel* in scenarios where such a decision is required.

There are some scenarios in which a SOM-based approach may not be suitable. For example, Keogh *et al.* [67, 69] argued that clustering of time series subsequences, obtained from a sliding window, gives results that are meaningless, since it can be shown that very similar clusters are formed for completely independent datasets. The authors demonstrated this finding using a range of clustering techniques, including SOM, over a number of different time series datasets. The finding was also independently verified by a number of researchers (details are given in [67]). Therefore, since SOM is a clustering technique, one would expect meaningless results from a SOM-based approach when used on time series data that has been preprocessed with the sliding window technique.

2.3.5 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) was first introduced by Grossberg as a theory of human cognitive information processing [50], but has since given rise to a series of real-time neural network models for unsupervised category learning and pattern recognition [20]. An ART network is another kind of self-organised clustering mechanism that associates inputs with learned categories. When an input is applied, the network "makes a hypothesis" as to which category that input belongs. That hypothesis is then tested by comparing the prototypical input for that category with the input applied to the network. If this prototype is "close enough" to the input, then a state of *resonance* is said to occur. This resonance persists long enough for learning to take place, allowing the network to refine its representation of the chosen category. If the prototype does not suitably match the input, then the network searches for a new category which achieves a better match. Should no such category be found, then learning of a new category begins.

Three models of ART networks originally developed by Carpenter and Grossberg include ART 1 [15], which can learn to categorise binary input patterns in a stable manner, ART 2 [14], which can learn to categorise either binary or analogue input patterns and ART 3, which is capable of parallel search of distributed recognition codes in a multilevel network hierarchy [16]. Carpenter *et al.* also proposed ART 2-A [18], which retains the behaviour of ART 2 while achieving an increase in computational efficiency of two to three orders of magnitude over the ART 2 system, and Fuzzy ART [19], which generalises ART 1 to analogue input patterns as well as binary input patterns [20]. The ART 1 model has also given rise to a network architecture,

called ARTMAP, which is a supervised learning system capable of self-organising stable categorical mappings between m-dimensional input vectors and n-dimensional output vectors [20]. The architecture consists of two ART 1 modules, connected by an associative learning network. One module associates input vectors with categories, whilst the other associates output vectors with categories. The two categorical representations generated by the ART modules are then associated by the autoassociative map in a supervised manner. The output vector represents the class of the input vector. Thus, for a given familiar input vector, ARTMAP learns to predict the class to which that input belongs. For example, this is demonstrated by using ARTMAP to distinguish between edible and poisonous mushrooms [20]. Fuzzy ART has been incorporated into ARTMAP to give Fuzzy ARTMAP, which is capable of learning mappings between analogue or binary input and output vectors [17]. In Fuzzy ARTMAP, the ART 1 modules ART_a and ART_b are replaced with Fuzzy ART modules. This enables the ARTMAP system to learn to classify inputs by a fuzzy set of features.

ART networks have been used in novelty detection problems. Liao *et al.* [77] proposed an adaptive framework for novelty detection with unsupervised adaptive neural networks. In this framework, the neural network is used to cluster the data, with each cluster being assigned a label of "normal", "uncertain" or "anomalous". When each input vector is presented, it is compared with the existing clusters that have been established. If the input vector fails to sufficiently match any of these clusters, or is the first to be presented, then a new cluster is constructed for the input vector. This new cluster is initially labelled as "uncertain". If, over a certain period of time, the new cluster grows beyond some predefined threshold, it is then relabelled as "normal", as are all of its members. Otherwise, the cluster is destroyed and its members are labelled as "anomalous", or novel. The framework was demonstrated using Fuzzy ART, as well as an alternative neural network approach, in two network intrusion detection tasks. A comparison was also made against Schölkopf *et al.'s* non-adaptive 1-SVM approach. When used with both neural networks, the framework was shown to give a good performance, outperforming 1-SVM in both tasks.

ARTMAP and Fuzzy ARTMAP have also been used for novelty detection. Carpenter *et al.* [21] proposed an extension of Fuzzy ARTMAP, ARTMAP-FD, to familiarity discrimination tasks. In ARTMAP-FD, a familiarity measure is defined for each input applied to the network. If the familiarity of an input is greater than a predefined threshold t, ARTMAP-FD proceeds to predict the class of that input. Otherwise, the input is assumed to belong to an unfamiliar class and no prediction is made. ARTMAP-FD was demonstrated on a familiarity discrimination task

over simulated radar range profile data by Carpenter *et al.* [21] and on real radar pulse data by Granger *et al.* [49]. It was shown that the threshold t is sensitive to noise, and so the optimal choice of this parameter is important in the success of applications. Different strategies on the setting of this parameter are discussed by Granger *et al.* [49].

Evaluation

ART networks have been shown to perform well in some novelty detection tasks, and are again capable of adaptive learning. However, one drawback is their complexity. For each input pattern, a number of processes must take place in order to identify a suitable category. This complexity is further increased in ARTMAP, fuzzy ARTMAP and ARTMAP-FD, since these approaches comprise two ART modules. Also, since ART performs a kind of clustering, they, for the same reasons detailed for the SOM-based approaches, may not be suitable for some time series novelty detection tasks.

2.3.6 Evolutionary Neural Networks

So far, we have considered neural network approaches to novelty detection which avoid the "curse of customisability" drawback of the MLP approach by providing a partially predefined network architecture. However, we have found that imposing restrictions on network structure has an impact on the suitability of the approach in certain novelty detection tasks. These approaches also require a number of parameters to be set, which still constitutes a significant amount of work for the user.

An alternative way of avoiding this curse is to *evolve* a neural network for a particular novelty detection task. An evolutionary algorithm can be used to discover suitable network structure and weights which enable some measure of performance, or *fitness*, to be optimised. Neural networks that have their structure and/or weights determined in this manner are referred to as evolutionary artificial neural networks [130], or simply evolutionary neural networks (ENNs). We consider ENNs in more detail in chapter 3.

The flexibility of ENNs, in that they can evolve neural networks to perform virtually any function, has allowed them to be used in a range of different applications, including novelty detection. Samanta [98] presented an ENN-based approach to gear fault detection. In this approach, the evolutionary algorithm is used to determine the size of the single hidden layer of an MLP network, as well as the choice of features (derived from raw measurements) to be used

as inputs to this network. But whilst this removes some of the design choices from the user, the neural network is still only suitable for problems in which a single hidden layer is sufficient. The enforcement of a layer structure is also in itself restrictive. Similarly, Hofmann and Sick [59] used an evolutionary algorithm to optimise the performance of an RBFNN in an intrusion detection task. As in Samanta's work, the genetic algorithm determines the elements of the input to be processed by the network, as well as the number of hidden neurons. In addition to this, the choice of basis function used in the RBFNN and the number of training epochs are also left to the genetic algorithm. This method of optimising an RBFNN with a genetic algorithm was shown to give a significantly better performance, compared to manual optimisation. But again, the approach is restricted to novelty detection tasks in which an RBFNN is suitable.

Alternatively, Han and Cho [55] proposed an evolutionary neural network approach in which no restriction is imposed on network structure. Here, MLP networks are evolved which are not limited to the classical layer structure. Instead, new neurons and connections are introduced to the network in an ad-hoc manner. Each network has a maximum number of input, hidden and output neurons, and mutation operations introduce connections between them. When the representation used by the evolutionary algorithm is translated into an actual neural network, only those neurons with one or more connections are included. This approach was shown to be effective in the intrusion detection task of identifying anomalies in system-call sequences. But, as we will discuss in chapter 3, this method suffers from a well-known representational problem known as competing conventions, which can result in damaged offspring being created.

Evaluation

Evolutionary neural networks provide an alternative way of tackling the problem of novelty detection. Instead of designing a specific novelty detector, approaches based on ENN are able to customise the structure and parameters of the detector to allow it to perform well in a particular novelty detection task. However, current approaches vary in the amount of customisability that they support. The first two approaches considered here allow only certain properties, such as the number of neurons in the single hidden layer of a network, to be evolved. This limits the number of novelty detection tasks to which these approaches can be applied. On the other hand, the third approach, proposed by Han and Cho, permits a much greater degree of flexibility in the kinds of networks that it can evolve. However, this still imposes a limit on the maximum number of neurons that can exist in any evolved network. Also, as we will show in chapter 4, their approach suffers from a weakness in its encoding which has now been addressed by

the research community. Finally, the three approaches considered here have focused solely on evolving non-adaptive neural networks, which would be unsuitable for use in dynamic novelty detection tasks.

2.4 Summary

In this chapter, we have reviewed a number of different approaches to novelty detection. We began by considering non-parametric statistical approaches based on k-nearest neighbour (KNN), negative selection and support vector machine (SVM). These each exhibit various limitations which prevent them from fulfilling all of the objectives stated in chapter 1. For example, approaches based on KNN form an overcomplicated representation of the normal class, consisting of all the data elements from the training dataset. Whilst an approach has been recently proposed by Angiulli [5] to overcome this drawback, this is not capable of adaptive learning. Negative selection approaches require a number of parameters to be set by the user, and may again construct overcomplicated representations of decision boundaries between the normal and novel classes in some cases. Approaches based on SVM have been shown to be particularly sensitive to the way in which the data is preprocessed, as well as to the choice of the kernel function, meaning that they generally require a significant amount of work and expert knowledge from the user to configure. We also considered a recent statistical approach proposed by Ahmed *et al.* [3] which does not fall into the above groups. This approach, the Kernel-based Online Anomaly Detection algorithm, is only suitable for use over time series datasets and also requires a number of parameters to be configured.

We next turned our attention to novelty detection approaches based on neural networks. We first considered the multilayer perceptron (MLP), which, whilst focused primarily on classification tasks, has been shown to effectively detect novel data elements. This was also the most customisable form of neural network considered in this chapter, and as such could potentially be used in a wide variety of different novelty detection tasks. However, this advantage of the MLP network is associated with its greatest disadvantage. The task of designing a suitable network structure for a particular scenario, and determining suitable values for the parameters of the chosen learning rule and activation function, is likely to be prohibitively difficult for the user.

One way of avoiding this "curse of customisability" is to use a neural network with a partially predefined structure. We examined four such kinds of neural network: autoassociator networks, radial-basis function networks, self-organising maps and adaptive resonance theory networks.

However, we found that their restricted structure affected their suitability to different novelty detection tasks. For example, both self-organising maps and adaptive resonance theory networks are clustering approaches, which makes them potentially unsuitable in some tasks based on time series data [67, 69]. Both the autoassociator (excluding the Hopfield architecture) and radial-basis function neural networks considered here were incapable of performing adaptive learning, making them unsuitable for use in dynamic environments. Whilst the Hopfield network is capable of adaptive learning, it is unable to forget about input patterns it has previously stored and can potentially become saturated. We also found that most of these approaches required a significant amount of work from the user in their configuration, such as determining suitable values for a number of application-specific parameters.

Instead of restricting the structure of the neural network, an alternative approach is to allow the structure to be automatically tailored to the novelty detection task at hand. A popular way of accomplishing this is to use an evolutionary algorithm to evolve the structure and weights of the network [130]. These approaches, known as evolutionary neural networks, have also been used to evolve novelty detectors. However, in many cases, the evolutionary algorithm is restricted to determining key properties of a network whose structure has again been partly predefined. Yet, in an approach to novelty detection proposed by Han and Cho [55], an evolutionary algorithm was used to evolve the complete structure of a neural network, as well as determining suitable weights. However, the approach did place a limit on the maximum number of neurons a network could have and, as we will discuss in chapter 3, suffers from a significant drawback which has previously been tackled in the research community.

Despite the drawbacks of Han and Cho's approach, we believe that an approach based on evolutionary neural networks can be developed which fulfils our objectives detailed in chapter 1. This becomes the focus of the remainder of this thesis. In the next chapter, we first consider a biologically-inspired neural network model which makes few assumptions about the properties of the data and so could potentially be used in a wide range of different novelty detection tasks. We then explore the topic of evolutionary neural networks in greater detail, and identify a recently proposed approach which addresses many of the challenges faced when evolving both neural network structure and weights. This approach, combined with the neural network model, forms the basis for our proposed novelty detection method, DPC+NEAT, which we present in chapter 4.

Chapter 3

Techniques Used in this Research

In the first part of this chapter, we introduce an adaptive neural network model, Dynamic Predictive Coding (DPC), which has been recently proposed by Hosoya *et al.* as a model of visual information processing in the retina [63]. We describe an implementation of this model and evaluate its suitability as a novelty detector. Then, in the second part of the chapter we examine approaches to constructing neural networks with evolutionary algorithms and consider how this might be used to overcome a significant drawback of the DPC model. The approaches considered in this chapter form the basis of a new approach to novelty detection, which we propose in chapter 4.

3.1 Introduction

The retina is responsible for providing the brain with information about the visual scene. An optical image which falls onto the retina is translated, by a series of retinal circuits, into a neural image which can then be processed by the brain [110]. In vertebrate retinae, this transformation from optical to neural image involves three stages. First, photons falling onto the retina are converted to signals by receptor neurons in a process known as phototransduction. These signals are then passed to a layer of "bipolar" neurons via chemical synapses. Finally, these signals are transmitted to output neurons known as "ganglion" cells. Between the receptor and bipolar layers, there are laterally connecting neurons known as horizontal cells. Between bipolar and ganglion cells, similarly laterally connecting neurons called amacrine cells are present. Both horizontal and amacrine cells modify in some way the transmission of signals across the synaptic layers.

One feature of time-varying images taken from a natural scene is that they possess substantial spatiotemporal correlations [31]. One may therefore exploit these correlations to obtain an efficient encoding of such images. It has been proposed that such an encoding strategy, *predictive coding*, is employed by the retina [107]. Here, neural circuits predict the intensity of a particular image point using intensities of points in the surrounding neighbourhood as well as exploiting correlations in the temporal domain. What is transmitted to the brain is then a signal describing the *difference* between predicted and observed intensities.

However, whilst predictive coding allows an efficient encoding of the average visual scene [63, 107], animals spend considerable amounts of time in environments which yield visual stimuli with markedly different statistical properties [63]. For example, in visual stimuli taken from a forest environment, two points which are vertically separated are likely to be highly correlated, whilst two horizontally separated points are not. This reflects the strong vertical structure of the environment, defined by the trees. In order to maintain efficiency in a range of different visual environments, an encoder would adapt its strategy to exploit the statistics of each environment. In a series of biological experiments, Hosoya *et al.* found that the retina is indeed capable of dynamically modifying its encoding strategy as necessary to maintain efficiency. They referred to this phenomenon as *Dynamic Predictive Coding* (DPC).

The DPC mechanism was formalised by Hosoya *et al.* as a simple feedforward neural network model with adaptive, or *plastic*, synapses governed by an anti-Hebbian learning rule [63]. In the first part of this chapter, we present this model and examine the operation of an illustrative DPC neural network, given by Hosoya *et al.*, over a series of artificial visual environments.

One drawback of the DPC model is that, like the multilayer perceptron discussed in the previous chapter, it does not define a specific network topology. In the second part of this chapter, we examine evolutionary algorithms which are capable of searching for a suitable network topology for a particular problem. This lays the foundation for our proposed approach, combining DPC with such an evolutionary algorithm, which we present in chapter 4.

3.2 Predictive Coding

Predictive coding was first proposed by Elias [35, 36] as a mechanism for transmitting messages, defined as a time series of magnitudes, from a message source to a message sink. The basic procedure is illustrated in figure 3.1 [36]. For the current element of a particular message, a prediction is formulated at the transmitter based on the statistics of the previous elements of

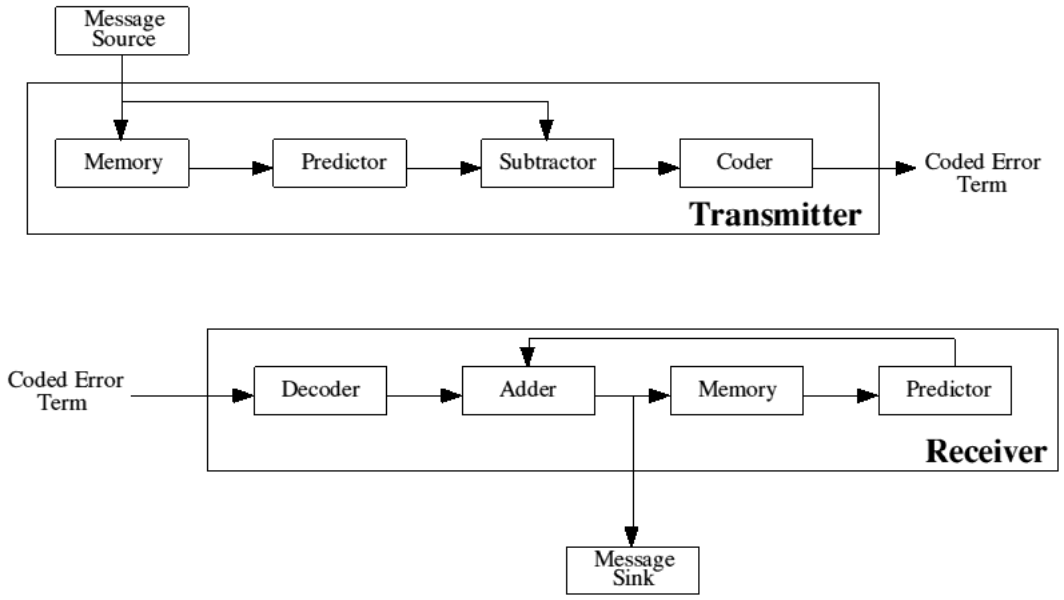


Figure 3.1: The predictive coding mechanism proposed by Elias [36].

that message held in a memory. This prediction is then subtracted from the current element, yielding an error term, or *difference signal*, which is subsequently transmitted. The receiver also generates a prediction in the same manner, but then adds this prediction to the received difference signal to reconstruct the original message element. The reconstructed message element is then passed to the message sink, as well as being stored in the memory of the receiver for future predictions. Since the message element can be reconstructed from the difference signal, this conveys just as much information as the actual message element but in a more efficient representation.

A prediction of the message element at the current time step may be formulated as a linear combination of the previous message elements [36]. The prediction p_i for the message element i over n previous message elements m_j may be defined as:

$$p_i = \sum_{j=1}^{n} a_j m_{i-j} \tag{3.1}$$

This linear predictor p_i may therefore be defined by the coefficients a_j. The best predictor is that which minimises the error (difference signal) between the predicted and observed message elements.

Harrison [56] investigated using predictive coding to obtain an efficient encoding of television pictures. The signal being transmitted is thought of as a series of samples uniformly spaced

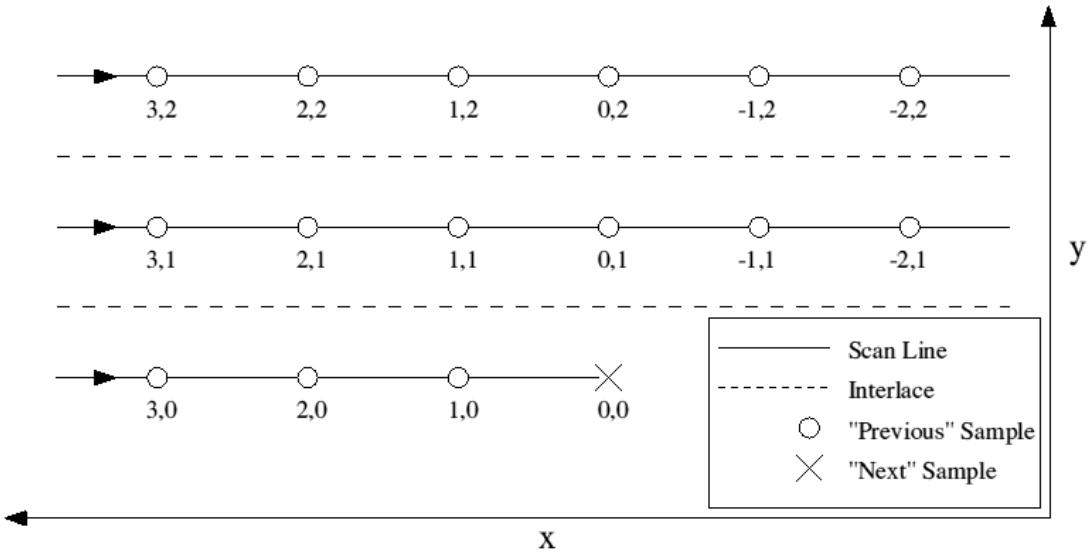


Figure 3.2: A small portion of a television raster, illustrating the geometrical location of single samples (image points) in the vicinity of the "next" signal sample [56].

across each scanline at Nyquist intervals, as shown in figure 3.2. Then, a prediction of the "next" signal sample, considered to be the next image point in the picture, is formulated using "previous" signal samples, which are image points lying to the left of the "next" sample on the current scanline as well as the image points on those lines above the current scanline. This approach therefore allows spatial correlations, commonly found in images, to be exploited. A number of different methods of predictive coding in this domain were discussed, differing only in their selection of exactly which "previous" signal samples to use when predicting the "next" sample. For example, the "previous value" method bases the prediction solely on the immediately preceding sample, whilst the "circular" method uses "previous" signal samples spatially positioned in a circle above the "next" signal sample, as shown in figure 3.3.

Srinivasan *et al.* [107] used the idea of predictive coding, as presented by Harrison, to explain the lateral and temporal inhibition commonly observed in retinal circuits. They proposed that this lateral inhibition exploits correlations between neighbouring image points in a visual scene in order to minimise the range of inputs presented to a retinal neuron. Intensity values at surrounding image points are used to generate a prediction of the intensity at a central point and this prediction is then subtracted from the actual observed intensity. The output of the neuron is then a representation of that image point in terms of prediction error. This is achieved with neurons having a classic centre-surround orientation of their receptive field, which defines the inputs to which they are sensitive, as illustrated in figure 3.4. The neuron receives an

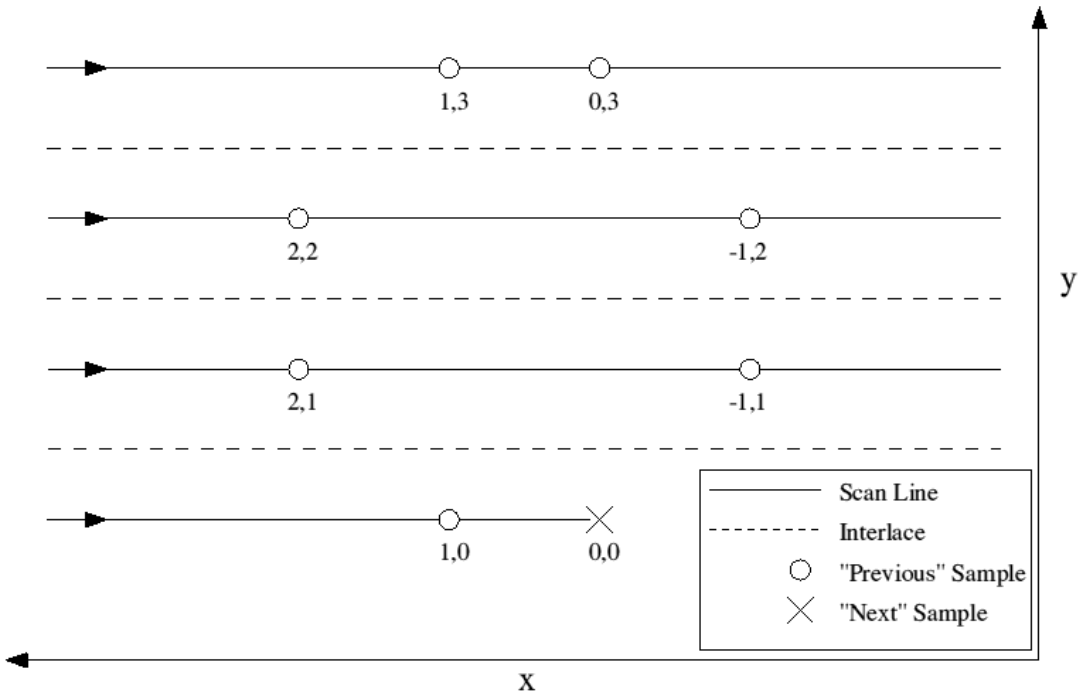


Figure 3.3: The "circular" method of predictive coding over image points from a television raster [56]. The "next" signal sample is predicted using the "previous" single samples indicated in the figure.

excitatory, or positive, input from the image point at the centre (shown shaded in figure 3.4), and inhibitory (negative) input from the image points in the surround. Each image point in the surround is scaled by a inhibitory coefficient, whose magnitude reflects the degree of the correlation between the surround image point and that at the centre. The net effect is that the scaled intensities of the surround image points simultaneously construct and subtract from the neuron a prediction of the centre image point. Predictions may also be formulated based on temporal correlations, by using self-inhibition to subtract previous correlating intensities from the current image point.

3.3 Dynamic Predictive Coding

As pointed out by Srinivasan *et al.*, their predictive coding scheme only considers removing redundancy expected in the average visual scene [107]. Hosoya *et al.* [63] instead argued that the retinal circuits dynamically adapt so as to learn and exploit the correlational structure present in the precise visual scene to which the retina is exposed. They tested the response of the ganglion cells of retinae from salamander and rabbit when stimulated with a series of

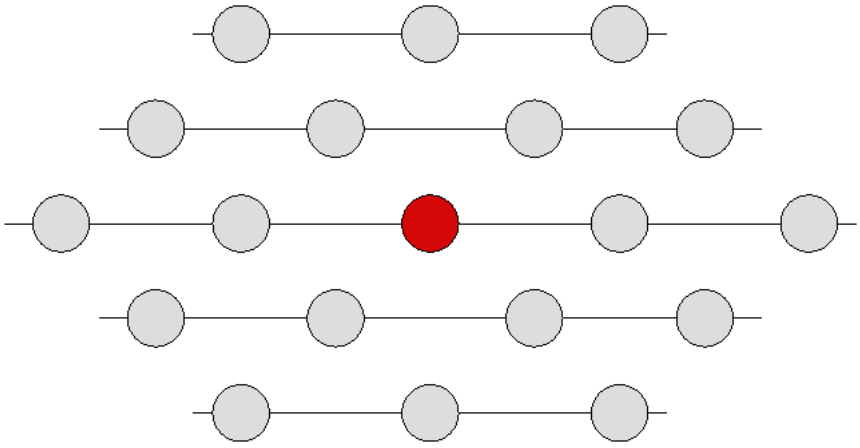


Figure 3.4: A series of receptor neurons in a classic centre-surround orientation [107].

artificially generated flickering greyscale scenes, or *visual environments*. As adaptation to one environment took place, the response of many ganglion cells were found to become less sensitive to the adapting environment and more sensitive to alternative environments. The form of predictive coding performed here was termed *Dynamic Predictive Coding* (DPC). In order to demonstrate how this was achieved, Hosoya *et al.* proposed a neural network model which uses synaptic plasticity to dynamically learn the correlational structure of visual stimuli. In this section, we examine this neural network model in detail.

3.3.1 Neural Network Model

The neural network model given by Hosoya *et al.* [63] represents the retinal circuit from bipolar cells to ganglion cells. Figure 3.5 shows a schematic of this model. Input nodes x_j, representing bipolar cells, connect to output neurons y_i, representing ganglion cells, via fixed synapses, which maintain a constant synaptic weight, and/or plastic synapses, whose weight is continuously changing. The fixed synapse from input x_j to output y_i is denoted by b_{ij}, and the plastic synapse between these neurons is denoted by a_{ij}. According to Hosoya *et al.*, each plastic synapse connects an amacrine and ganglion cell, with the amacrine cell receiving its input from the bipolar cell across a fixed synapse. However, in the mathematical definition of the model, this is simplified by having plastic synapses connect input and output neurons directly. We use this simplification in our presentation of the model.

In this neural network model, the output y_i of the ith output neuron is given by:

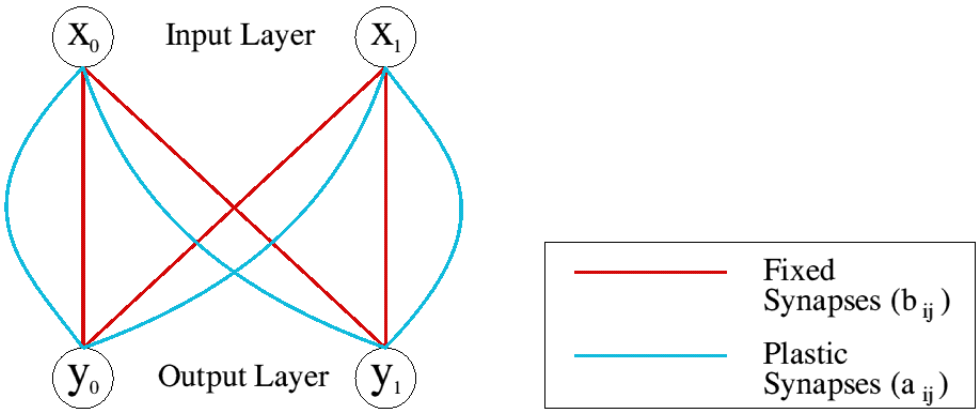


Figure 3.5: A schematic of the DPC neural network model proposed by [63].

$$y_i = \sum_j (b_{ij} + a_{ij}) x_j \tag{3.2}$$

That is, the output neuron simply outputs the linear summation of its inputs, transmitted across both plastic and fixed synaptic connections a_{ij} and b_{ij}. At each output neuron, the goal of the network is as follows. Given a series of intensities transmitted across one or more fixed synapses, the network first attempts to formulate a prediction of the sum of this series, based on those intensities transmitted across the plastic synapses. This is then subtracted from the output neuron, whose response then describes the prediction error. The network formulates the prediction through modulation of the plastic synapses, based on the correlational structure of the input stimulus. The synapses are modified according to the anti-Hebbian learning rule:

$$\frac{da_{ij}}{dt} = \frac{-a_{ij} - \beta \langle y_i x_j \rangle}{\tau} \quad \tau, \beta > 0 \tag{3.3}$$

where β and τ are parameters of the model and $\langle y_i x_j \rangle$ denotes the time-averaged value of $y_i x_j$ [94], which is dependent on the correlation between y_i and x_j [63]. The parameter β controls the networks sensitivity to the correlation between neurons y_i and x_j, whilst τ controls the rate of change permitted by equation 3.3. According to Hosoya *et al.*, $\langle y_i x_j \rangle$ is based on recent values of x_j and y_i [63]. We therefore construct an estimate of $\langle y_i x_j \rangle$ using the m time steps up to and including the current time step t, given by:

$$\langle y_i x_j \rangle = \frac{1}{m} \sum_{k=t-m+1}^{t} y_i(k)\, x_j(k), \; m > 1 \tag{3.4}$$

where $y_i(k)$ and $x_j(k)$ are the values of y_i and x_j respectively at time k. So that, in addition to

the activity at the current time step, previous activity is considered in this estimate, we impose the constraint $m > 1$.

The anti-Hebbian learning rule causes a negative adjustment in plastic synapses when the activity at the presynaptic and postsynaptic neurons is correlated and a positive adjustment when the activity is anti-correlated. The plastic synapses adapt so as to collectively construct a prediction of the inputs received over the fixed synapses by the output neuron and simultaneously subtract this prediction from the output neuron. The exact behaviour of the learning rule can be customised using the parameters β, τ and m.

3.3.2 Analysis of the Adapted Network

The net effect of the adapted network is to suppress the correlated, and therefore redundant, information in an input vector $\mathbf{x}$ in order to derive the difference signal, which may then be transmitted. This was shown by Hosoya *et al.* through a mathematical analysis of the model in an adapted state ([63], supplementary methods), which we present here.

This analysis considers equations 3.2 and 3.3 in matrix notation, which give the following:

$$\mathbf{y} = (\mathbf{B} + \mathbf{A})\mathbf{x} = \mathbf{R}\mathbf{x} \tag{3.5}$$

$$\begin{aligned} \frac{d}{dt}\mathbf{A} &= \frac{1}{\tau}\left(-\mathbf{A} - \beta\left\langle \mathbf{y}\mathbf{x}^T \right\rangle\right) && (3.6)\\ &= -\frac{1}{\tau}(\mathbf{A} + \beta(\mathbf{A} + \mathbf{B})\mathbf{C}) && (3.7) \end{aligned}$$

where $\mathbf{B}$ represents the matrix of fixed synaptic weights $[b_{ij}]$ in the network and $\mathbf{A}$ represents the matrix of plastic synaptic weights $[a_{ij}]$). $\mathbf{C}$ denotes the covariance matrix of the input stimulus $\mathbf{x}$, defined as:

$$\begin{aligned} \mathbf{C} &= \left\langle \mathbf{x}\mathbf{x}^T \right\rangle && (3.8)\\ &= E(\mathbf{x}\mathbf{x}^T) && (3.9) \end{aligned}$$

where $E(.)$ denotes the expectation operator. The sum of the matrices $\mathbf{B}$ and $\mathbf{A}$ define the response matrix $\mathbf{R}$ of the network, and $\mathbf{R}\mathbf{x}$ thus defines the response of the network to the input

vector $\mathbf{x}$. By rewriting equation 3.7 in terms of $\mathbf{R}$, the change in the networks response matrix may be defined as:

$$\frac{d}{dt}\mathbf{R} = -\frac{1}{\tau}\left((\mathbf{R} - \mathbf{B}) + \beta\mathbf{R}\mathbf{C}\right) \tag{3.10}$$

After adaptation is complete, $\frac{d}{dt}\mathbf{R} = 0$, and so:

$$\begin{aligned} -\frac{1}{\tau}\left((\mathbf{R} - \mathbf{B}) + \beta\mathbf{R}\mathbf{C}\right) &= 0 && (3.11)\\ \mathbf{R} - \mathbf{B} + \beta\mathbf{R}\mathbf{C} &= 0 && (3.12)\\ \mathbf{R}\left(1 + \beta\mathbf{C}\right) &= \mathbf{B} && (3.13)\\ \mathbf{R} &= \mathbf{B}(1 + \beta\mathbf{C})^{-1} && (3.14) \end{aligned}$$

To examine the effect of the response matrix $\mathbf{R}$ on input vector $\mathbf{x}$ after adaptation is complete, we may examine $\mathbf{R}$ in the *eigenbasis* of the covariance matrix $\mathbf{C}$. A basis $\mathbf{L}$ in vector space V is a series of vectors $\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_n$, which may be used in some linear combination to express any vector in that space. They define the directions of the axes of a vector space, along which all vectors in that space can be described. A series of vectors can only define a basis if they (i) are *linearly independent*, i.e. each vector gives a direction not described by any other, and (ii) *span* the vector space, that is any vector in V may be expressed as a linear combination of the series [51]. The *eigenbasis* of a matrix is a basis in vector space defined by the set of eigenvectors of that matrix. If an $n \times n$ matrix has n distinct eigenvalues, it therefore has n linearly independent eigenvectors and as such the matrix has an eigenbasis in the vector space [12]. Moreover, it can be shown that any symmetric $n \times n$ matrix has n orthonormal eigenvectors and therefore has an eigenbasis [51].

The covariance matrix $\mathbf{C}$ is symmetric, since $\mathbf{x}\mathbf{x}^T = \mathbf{x}^T\mathbf{x}$ (and so $E(\mathbf{x}\mathbf{x}^T) = E(\mathbf{x}^T\mathbf{x})$), which means that it will always have an eigenbasis. The representation $\mathbf{C}_E$ of $\mathbf{C}$ in its eigenbasis may be derived as [12]:

$$\begin{aligned} \mathbf{C}_E &= \mathbf{S}^{-1}\mathbf{C}\mathbf{S} && (3.15)\\ &= \mathbf{S}^T\mathbf{C}\mathbf{S} && (3.16) \end{aligned}$$

where $\mathbf{S}$ is an $n \times n$ matrix where the ith column is the ith orthonormal eigenvector λ_i of $\mathbf{C}$, thus making $\mathbf{S}$ orthogonal [51] (which, in turn, means that $\mathbf{S}^{-1} = \mathbf{S}^T$ [12]). Since $\mathbf{S}$ is an orthogonal matrix, $\mathbf{C}_E$ is a diagonal matrix [51], with the ith element on the diagonal taking the value of the ith eigenvalue of $\mathbf{C}$, denoted here by c_i. Therefore, in the eigenbasis of $\mathbf{C}$, the response matrix $\mathbf{R}$ becomes:

$$\mathbf{R} = \mathbf{B} \begin{pmatrix} \frac{1}{1+\beta c_1} & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & \frac{1}{1+\beta c_n} \end{pmatrix} \tag{3.17}$$

and $\mathbf{Rx}$ is:

$$\mathbf{Rx} = \mathbf{B} \begin{bmatrix} \frac{x_1}{1+\beta c_1} \\ \frac{x_2}{1+\beta c_2} \\ ... \\ \frac{x_n}{1+\beta c_n} \end{bmatrix} \tag{3.18}$$

where, in the eigenbasis of $\mathbf{C}$, B and $\mathbf{x}$ are defined in terms of the eigenvectors of $\mathbf{C}$.

The eigenvectors of $\mathbf{C}$ are called the *principal components* of $\mathbf{x}$ [9]. These define orthogonal "lines of fit" through the input vectors upon which the covariance matrix $\mathbf{C}$ is derived. The ith principal component represents the direction with most variance that is orthogonal to the previous $i-1$ components. Therefore, each principal component represents the direction of a correlational relationship between the elements of $\mathbf{x}$, whose strength is indicated by the variance along that direction. The first principal component defines the orthogonal regression line (i.e. the "line of best-fit") through the data [64].

Equation 3.18 shows that, in its adapted state, the network suppresses the input vector $\mathbf{x}$ along the directions given by the principal components by a factor of $\frac{1}{1+\beta c_i}$, where c_i is the eigenvalue corresponding to the ith principal component. As the variance in each direction is reduced, the correlation between the elements of $\mathbf{x}$ along each direction is suppressed. Because the suppression factor is a function of the eigenvalue of each principal component, those components with higher variance, representing higher degrees of correlation in $\mathbf{x}$, are suppressed to a greater extent. When the correlational structure of $\mathbf{x}$ changes, this is reflected in the covariance matrix $\mathbf{C}$. This in turn induces a new period of learning, which continues until equation 3.17 is satisfied once again.

3.4 Dynamic Predictive Coding and Novelty Detection

We now explore a simulation (originally reported by Hosoya *et al.* [63]) of an illustrative implementation of the DPC neural network model. In this simulation, a number of visual environments, similar to those used in the biological experiments reported in [63], are applied to the illustrative neural network. The network is shown to adapt to each environment in such a way that it is able to differentiate between stimuli from that environment and stimuli from the other environments. This behaviour is observed by periodically measuring the sensitivity of the network to each environment throughout the simulation. This ability to differentiate between the environments constitutes a form of novelty detection. The network defines normality in terms of what has been seen recently, and we show how this concept of "recent" can be customised by tuning the model parameters τ and m. We also evaluate the DPC model against the objectives we established in chapter 1 and identify a significant drawback of the approach.

3.4.1 Illustrative Neural Network

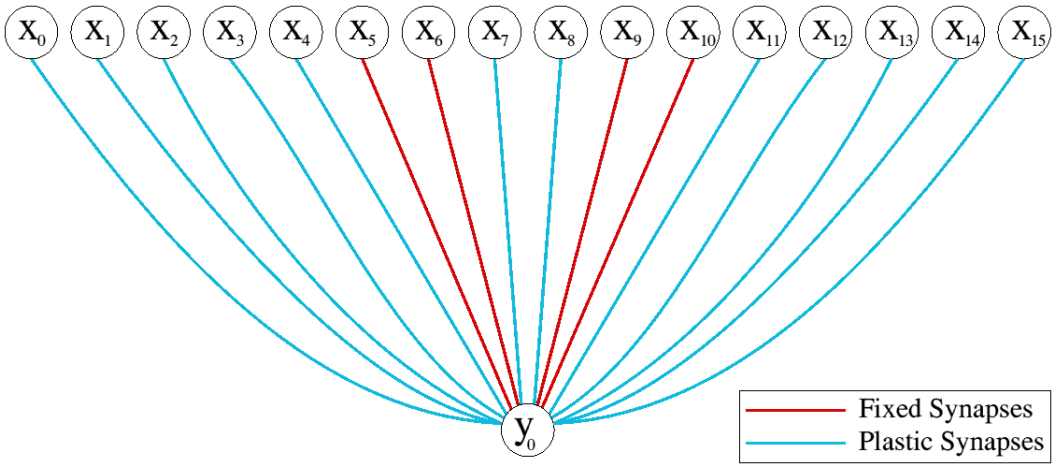


Figure 3.6: The illustrative DPC neural network given by Hosoya *et al.* [63].

In order to demonstrate their DPC model, the single layer illustrative neural network shown in figure 3.6 was given by Hosoya *et al.*. [63]. Consider a 4 × 4 pixel greyscale image, shown in figure 3.7, where each pixel provides a single input to the network. The network aims to predict the sum of the centre 2 × 2 pixels, outlined in figure 3.7, given the correlational relationships between all pixels, and output the difference between the predicted and observed sums. The network has a single output neuron y, which all pixels are connected to by plastic synapses. In addition, the 2 × 2 centre pixels are also connected to the output neuron via fixed synapses

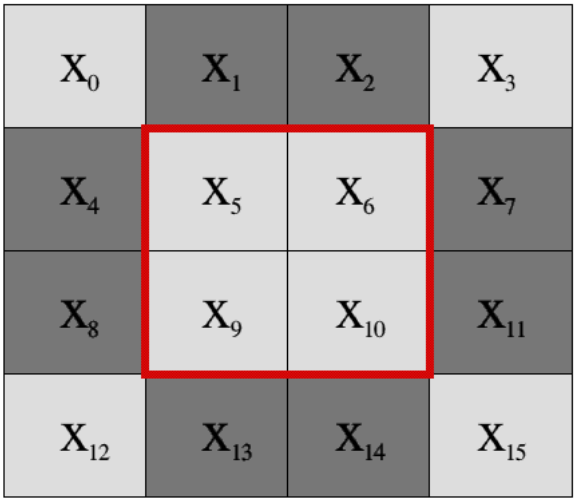


Figure 3.7: An example of a 4 × 4 greyscale image which serves as input to the illustrative neural network. The goal of the network is to suppress the sum of the intensities of the centre 2 × 2 pixels, outlined here in red.

with weight 1.

The receptive field of the output neuron has the classic centre-surround organisation widely observed in retinal neurons [107]. In the network's initial state, each plastic synapse has a weight of zero, meaning that the output neuron is initially driven by the centre 2 × 2 pixels only. Over time, the plastic synapses have their weights updated by the anti-Hebbian learning rule given in equation 3.3, such that a prediction of the sum of the 2 × 2 pixels is formed and subtracted from the observed sum. This effectively updates the receptive field so that the response from the centre is suppressed by the surround. Such a function was first observed in centre-surround receptive fields of real retinal neurons by Kuffler [74].

3.4.2 Visual Environments

Hosoya *et al.* [63] demonstrated the operation of this illustrative neural network over six artificially generated visual environments. These are akin to the visual stimuli used to test adaptation to changing spatial correlations in the biological experiments performed by the authors [63]. Four of these environments have perfect correlational structure, whilst a fifth environment defines a flickering greyscale image with *no* correlational structure. Finally, a sixth environment represents the absence of stimulus.

The four environments with perfect correlational structure represent a flickering uniform field (*Uniform*), a flickering checkerboard pattern (*Checker*) and flickering vertical and horizontal bars (*Vertical* and *Horizontal*). The fifth environment with no correlational structure is called

Random. In this environment, each pixel is assigned an independent random intensity. The environment representing the absence of stimulus, labelled *None*, is a steady grey screen [63].

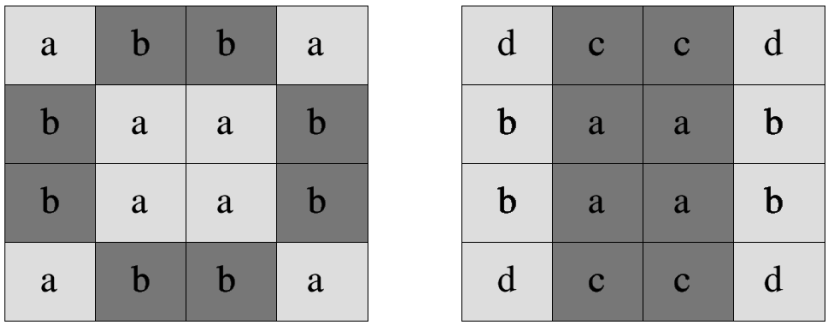


Figure 3.8: The structure of the four artificial visual environments with perfect correlational relationships. *Left:* The structure of the Uniform and Checker environments. *Right:* The structure of the Vertical and Horizontal environments.

For each environment (apart from None), pixel intensity values are drawn from a standard Gaussian distribution. For the environments with perfect correlational relationships between pixels, the intensities of all pixels are defined by a single random variable $n_u(t)$. This variable has its intensity value updated every u time steps with another value independently drawn from the Gaussian distribution. Let $a(t)$, $b(t)$, $c(t)$ and $d(t)$ be four new random variables. Each pixel is then assigned the value of one of these variables, as shown in figure 3.8. The values taken by each random variable depend on the particular visual environment. The two variables $a(t)$ and $b(t)$ are defined for Uniform as [63]:

$$a(t) = b(t) = n_u(t) \tag{3.19}$$

and for Checker as:

$$a(t) = -b(t) = n_u(t) \tag{3.20}$$

For Vertical and Horizontal, all four random variables are used. For Vertical, the variables take the values [63]:

$$a(t) = -b(t) = c(t) = -d(t) = n_u(t) \tag{3.21}$$

and for Horizontal [63]:

$$a(t) = b(t) = -c(t) = -d(t) = n_u(t) \tag{3.22}$$

By assigning $a(t) = c(t)$ in Vertical, one obtains a vertical bar with a width of two pixels, passing directly through the centre of the environment. Conversely, by assigning $a(t) = b(t)$ in Horizontal, a similar bar is obtained but with a 90 degree rotation about the centre of the image.

The final environment, None, was defined to be a steady grey screen [63]. Since pixel intensities are drawn from a standard Gaussian distribution, grey is defined as having an intensity value of zero. Furthermore, a zero-vector is unable to induce adaptation in the network, meaning that the network will remain at or return to an unadapted state (i.e. all plastic synaptic weights being equal to zero).

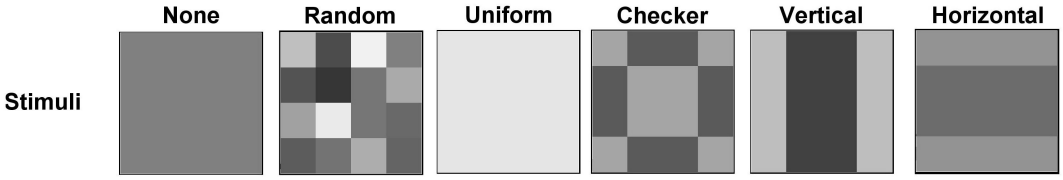


Figure 3.9: The six artificial visual environments defined by Hosoya *et al.* [63].

The artificial visual environments defined above are illustrated in figure 3.9. The goal of DPC is to learn the correlational structure of a series of inputs and use this information to minimise the error of predicted future intensities. Therefore, of the six environments shown in figure 3.9, a DPC neural network would only be able to adapt to the four flickering environments with correlational structure. As mentioned before, the environment Random has no correlational structure and the environment None is incapable of inducing adaptation. For brevity, we will refer to the environments to which a DPC neural network can adapt (Uniform, Checker, Vertical and Horizontal) as *adaptable environments*.

3.4.3 Sensitivity Analysis

When adapted to a particular environment A, the illustrative neural network should learn the correlational structure of that environment and use this information to minimise the magnitude of its response, i.e. the difference signal. Therefore, large changes in stimuli from environment A should yield small changes in the networks response or, in other words, the network should have a reduced sensitivity to environment A. However, when shown stimuli from a different

environment B, the prediction formulated by the network should have a large error. This in turn results in a response from the network with a larger magnitude, which causes the network to have a high level of sensitivity to this environment. When the network is permitted to adapt to environment B, its sensitivity to this environment should decrease, whilst its sensitivity to environment A increases.

To analyse the performance of the network, Hosoya *et al.* [63] observed how the sensitivity of the output neuron to the four adaptable environments varied as all six environments were shown in turn to the network. Sensitivity of the output neuron y to a given environment E is defined as the square root of the time-averaged variance of y taken over stimuli from environment E [63]:

$$S_E = \sqrt{\langle var(y) \rangle_E} \tag{3.23}$$

Sensitivity to a given environment E is monitored throughout the simulation as follows. At regular intervals during the simulation, the networks plastic synaptic weights are frozen. Stimuli from environment E is then applied to the network and the variance of the response of the output neuron is sampled. The variances obtained in multiple intervals are then used to calculate the networks sensitivity to environment E as defined in equation 3.23. The sensitivity values obtained for each environment through the duration of the simulation can then be plotted, as we will show in the next section. In their work, Hosoya *et al.* presented S_E as a normalised value in the range $[0, 1]$ [63]. Maximum sensitivity occurs for stimuli from a particular environment when no prediction is attempted, and so is therefore the variance of the sum of the centre pixels.

3.4.4 Simulation over Visual Environments

Hosoya *et al.* showed that the illustrative neural network was able to dynamically adapt to each of the four adaptable environments, giving a low response to stimuli from the environment to which it had adapted and a high response to stimuli from the other three environments [63]. To verify this result, we constructed a simulation of the illustrative network and repeated this experiment. In our simulation, each visual environment was shown to the network for 2,500 time steps, with the intensity of each pixel being updated after every time step ($u = 1$). Sensitivity values were averaged over 100 time steps, with each variance based on 100 samples of the network output. The sensitivity graph we obtained from this simulation is shown in figure 3.10. This is similar to that shown by Hosoya *et al.* (figure 5(d) in [63]). The receptive fields of the output

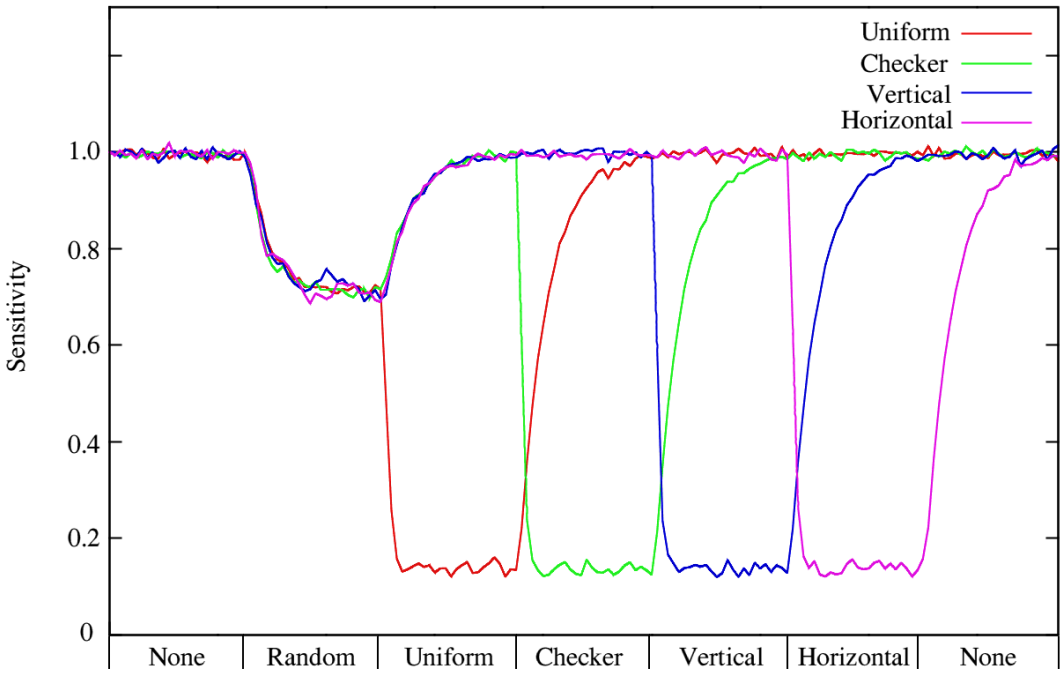


Figure 3.10: The sensitivity graph produced by our implementation of the illustrative neural network given in [63]. Each curve plots the varying sensitivity of the network to one of the four adaptable environments throughout the simulation.

neuron at the various stages of the simulation are visualised in figure 3.11. Again, these agree with those shown by Hosoya *et al.* [63]. The parameters we used in our implementation were $\beta = 0.4$, $\tau = 500$ and $m = 2$.

We now examine each stage of the simulation in more detail. The first environment shown was None. Since stimuli from this environment cannot induce adaptation, the network remained sensitive to all four adaptable environments as shown in figure 3.10. The corresponding receptive field in figure 3.11 confirmed that the plastic synapses remained in an unadapted state with a weight value of zero. When Random was applied, the sensitivity of the network to the adaptable environments fell by approximately 0.25. The corresponding receptive field showed a slight attenuation in its centre, which appears to be the result of chance short-term correlations between the independent pixels in the image. The next four environments applied to the network were the adaptable environments themselves. From figure 3.10, we can see that as the network adapted to each environment, its sensitivity to that environment fell significantly. As shown in figure 3.11, the receptive field was modified to reflect the corresponding correlational structure of each adaptable environment. Those pixels which *correlated* with the centre pixels

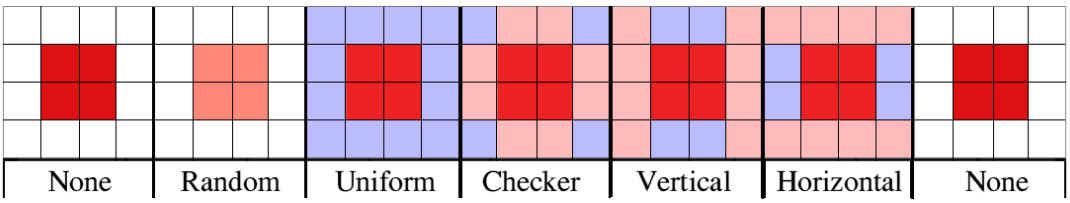


Figure 3.11: The state of the network's receptive field when subjected to each visual environment. The label underneath each receptive field identifies the corresponding visual environment. Each receptive field is shown as a grid, where each grid square corresponds to a pixel x_j in the 4×4 greyscale image. The colours used represent the net synaptic connectivity $(a_{ij} + b_{ij})$ to each pixel x_j [63]; red is positive, blue is negative and white indicates a value of zero.

were scaled by negative synaptic weights, whilst the pixels which *anticorrelated* were scaled by positive synaptic weights. When adapted to each of the four adaptable environments, the network remained sensitive to the other three environments. When the environment being shown to the network changed to another adaptable environment, for example from Uniform to Checker, the network began to learn the correlational structure of the new environment, whilst forgetting the structure of the previous environment. This corresponded to a decrease in sensitivity to the new environment and a simultaneous increase in sensitivity to the previous environment. With the final application of the environment None, the network forgot the correlational structure of the previous environment (Horizontal) and gradually returned to its unadapted state. This was confirmed by the corresponding receptive field shown in figure 3.11.

When adapted to a particular visual environment, the illustrative network demonstrated its ability to differentiate between stimuli from that environment and stimuli from other visual environments. For example, when adapted to the Uniform environment, the network demonstrated a low sensitivity to stimuli from Uniform, and a high sensitivity to stimuli from the Checker, Vertical and Horizontal environments. By doing this, the network performed novelty detection, separating stimuli from the learned *normal* class from stimuli belonging to a series of *novel* classes. Here, each novel class is a visual environment to which the network is not currently adapted. Environments that the network has previously learned, but have not been seen in the recent past, are also considered novel, since they have been forgotten by the network.

3.4.5 Analysis of the Receptive Field

We now examine the properties of the receptive field when the network is in an adapted state. When the network adapts to an environment, the plastic synapses change so as to identify the correlational structure of that environment and use this information to formulate a prediction

of the sum of the centre pixels. This prediction is simultaneously subtracted from the output neuron, yielding the difference signal. For example, consider an environment, *Env1*, where each of the centre pixels has an intensity p. The pixels in the periphery may have an intensity of either p or $-p$. A pixel which correlates with the centre pixels, thus having an intensity p, is scaled by a negative plastic synaptic weight, yielding $-ap$. Conversely, a pixel with intensity $-p$ is scaled by a positive plastic synaptic weight, yielding $a(-p) = -ap$. Therefore, a proportion of the magnitude of each correlating and anticorrelating periphery pixel is subtracted from the output neuron, with the sign of the corresponding plastic synaptic weight adjusting for the direction of the correlation. Since each pixel has the same level of correlation, but a different direction, the sum $\mathbf{Ax}$ of inputs carried across the plastic synapses may be defined as:

$$\mathbf{Ax} = -16ap \tag{3.24}$$

where a is the synaptic weight value of each plastic synapse. This results in suppression of the sum of the centre pixels for Env1 and a smaller output from the network, which in turn results in a lower sensitivity to this environment. If the plastic synaptic weights are then frozen, and a different adaptable environment *Env2* is applied to the network, then the sum $\mathbf{Ax}$ will change according to the different correlational relationships in the new environment. Comparing the two environments Env1 and Env2, s pixels will be found to retain the same correlational relationship in both environments, and d pixels will have a different correlational relationship. The magnitudes of the s pixels will be subtracted from the output neuron as before, i.e. $-sap$. However, the change in sign from the d pixels will result in a change in sign of the scaled result, therefore resulting in the magnitudes of these pixels being *added* to the output neuron, yielding dap. The sum of inputs carried across the plastic synapses is then:

$$\mathbf{Ax} = dap - sap \tag{3.25}$$

Therefore, if $d = s$ then $\mathbf{Ax} = 0$ as the inhibition and excitation resulting from the plastic synapses cancels out. This causes the network to yield an output value of $\mathbf{Bx}$, the sum of the centre pixels. Comparing each of the four adaptable environments with each other, one can observe that between any two environments exactly eight pixels on the periphery change their correlational relationship with the centre pixels. This leaves four pixels in the periphery maintaining the same correlational relationship, as well as the centre pixels themselves. This allows the network to suppress the environment to which it has adapted, whilst remaining

sensitive to the other adaptable environments.

3.4.6 Defining the Recent Past

As has been demonstrated, the DPC neural network model bases its concept of normality on the recent past. For example, when adapted to Uniform the illustrative neural network finds stimuli from the Checker environment to be novel. However, as it continues to be exposed to the Checker environment, it begins to adapt, eventually forgetting about Uniform and becoming familiar with Checker. The period of time required for the network to learn and forget provides the definition of "recent".

The ability of the DPC model to learn and forget is controlled by the parameters τ and m, which were described in section 3.3.1. Specifically, τ controls the rate of change of the plastic synapses, whilst m defines the period over which the correlation between each input node x_j and the output node y_i is measured. We now demonstrate how varying these parameters can affect the definition of "recent" for the illustrative neural network.

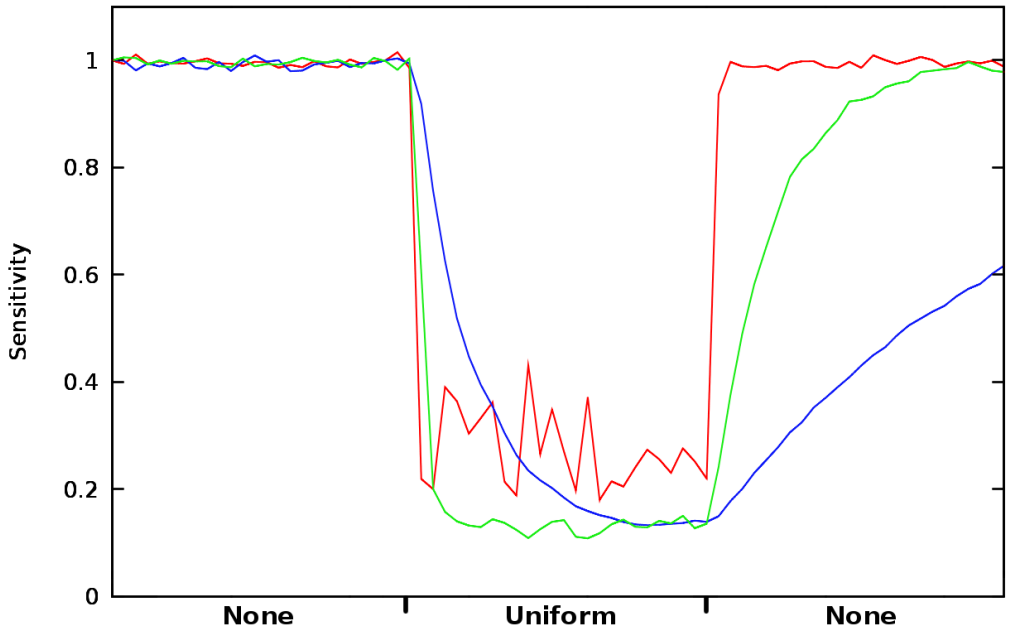


Figure 3.12: The effect of varying τ on the illustrative neural network. Curves are plotted for $\tau = 10$ (red), $\tau = 500$ (green) and $\tau = 3000$ (blue). The remaining parameters β and m are set to 0.4 and 2 respectively.

We first consider the effect of varying τ on the network. The sensitivity curves generated for $\tau = 10$, $\tau = 500$ and $\tau = 3000$ are shown in figure 3.12. When τ is small, the network is able to

both adapt to and forget about the Uniform environment very quickly, since the rate of change permitted in the weights of the plastic synapses is larger. However, the larger changes permitted in each a_{ij} also cause increased fluctuations in sensitivity when the network is adapted. The spikes in sensitivity correspond to spikes in the plastic synaptic weight values. Increasing τ results in the adaptation and decay periods also increasing. With a large τ, the network neither fully adapts to the Uniform stimulus nor completely forgets before the end of the simulation.

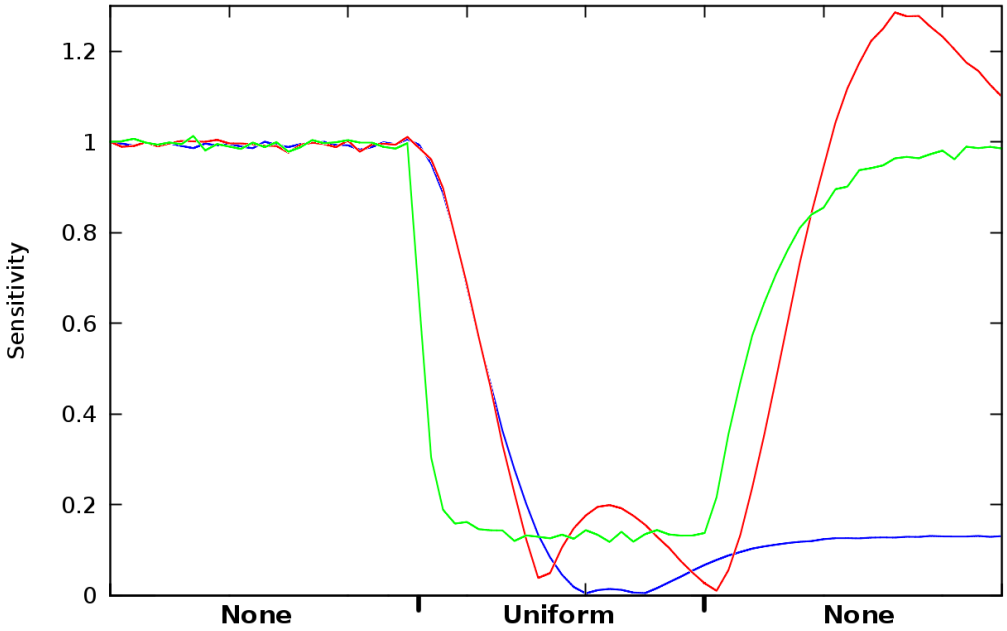


Figure 3.13: The effect of varying m on the illustrative neural network. Curves are plotted for $m = 2$ (green), $m = 2500$ (red) and $m = 5000$ (blue). The remaining parameters β and τ are set to 0.4 and 500 respectively.

Figure 3.13 shows the sensitivity curves generated for $m = 2$, $m = 2500$ and $m = 5000$. For $m = 2500$ and $m = 5000$, the network remains insensitive to Uniform for a longer period of time after stimuli from this environment has been withdrawn. Indeed, for $m = 5000$ the network's sensitivity to Uniform fails to recover during the simulation. Similarly, when the Uniform environment is first applied, the network does not immediately begin to adapt. This is because the network remembers the period of no activity present during the first application of None for longer.

From the simulations above, we can observe that varying τ and m have different effects on the network. As τ is increased, the period of time required by the network to learn and forget also increases. This is because τ controls the rate of change of the plastic synaptic

weights, and therefore can be viewed as controlling the memory of these synapses. However, regardless of the value of τ, the time points in the simulation at which the network begins its learning and forgetting phases remain approximately constant. This is not the case when m is varied. Increasing m delays the time point at which the network begins both its learning and forgetting phases. Like τ, this parameter also has an impact on the length of these phases. This occurs because the learning rule remembers previous stimuli for a greater period of time. For example, for $m = 5000$, stimuli from the Uniform environment continues to have an effect on the correlation term of the learning rule for the remainder of the simulation, thus shifting the time point at which the forgetting phase is due to occur beyond the final step of the simulation. Therefore, m can be viewed as controlling the memory of the correlation term of the learning rule.

Since the DPC model has two kinds of memory, i.e. the plastic synaptic weights and the correlation term of the learning rule, its definition of what constitutes the recent past combines two different concepts of "recent". At the synaptic level, "recent" is the period of time in which previous knowledge encoded into the synaptic weights may persist. At the level of the learning rule, "recent" is the period of time over which the correlation between two neurons should be observed. Since both concepts can be customised, DPC neural networks are able to use the most appropriate definition of "recent" for the particular novelty detection task at hand.

3.4.7 Evaluating Dynamic Predictive Coding Against the Requirements in Chapter 1

In chapter 1, we established a series of objectives that we believe an approach to novelty detection should fulfil: the approach should be *generally suitable* for and effective in a wide range of different novelty detection tasks, it should have a high degree of *customisability*, it should require a minimal amount of *work* from the user in its configuration for a particular task, and it should have a justifiable level of *complexity* for that task. We now evaluate the DPC model against these criteria.

The ability of the illustrative neural network to differentiate between stimuli from the environment to which it has adapted and stimuli from the remaining adaptable environments clearly demonstrates the potential of DPC as an approach to novelty detection. Here, the environments are akin to classes seen in classification. The network successfully separates the normal class from a series of novel classes, which can be observed by examining the sensitivity of the network

to the various environments. Another promising feature of the illustrative neural network is its simplicity. No activation function is used in the output neuron, instead the network's function is given by synaptic plasticity based on a simple anti-Hebbian learning rule. The parameters β, τ and m can be used to customise the behaviour of this rule. The parameter β controls the network's sensitivity to the observable correlational relationships within the environment, whilst τ and m both determine the length of time which constitutes the recent past.

However, the general suitability of DPC for other novelty detection tasks is as of yet unknown. The model makes the assumption that the data being processed will exhibit a correlational structure. Whilst it is well-known that such structure commonly exists in images, this is not true of other forms of data. However, it is not unreasonable to assume that attributes within a dataset used in some novelty detection task bear some relationship to one another, defining some correlational structure in the dataset. Other approaches to novelty detection reviewed in chapter 2 have also exploited correlation, such as autoassociator networks [73, 97]. However, the anti-Hebbian learning rule is only capable of exploiting *linear* correlations in data. To make this approach suitable for use in applications where data has nonlinear correlations, the learning rule would need to be modified. Furthermore, the problems that the illustrative neural network model has been applied to currently are limited in that they involve stimuli which has a regular structure, exhibiting a high degree of symmetry, and with perfect correlational relationships between all elements. In chapters 4 and 5, we address this by exploring the ability of DPC networks to process data which lack some or all of these properties.

The DPC model allows neural networks to be constructed with a high degree of customisability. The model defines the network to be feedforward, as well as featuring both fixed connections and those modified by the anti-Hebbian learning rule. In chapter 4, we expand the DPC model to also include hidden neurons. Unlike many of the neural network approaches reviewed in chapter 2, a DPC neural network is not tightly restricted to a particular kind of topological structure. This also means that the complexity of a DPC neural network can also be minimised for a particular application. Parameters in the anti-Hebbian learning rule add another layer of customisability.

However, in common with the multilayer perceptron approach discussed in chapter 2, the lack of a predefined structure is also the biggest drawback of this approach. Designing a suitable structure is an experimental process, which is prohibitively difficult for the non-expert. Suitable values must also be determined for the anti-Hebbian learning rule which, again, can be a time consuming process. Therefore, the work required from the user in applying DPC to a particular

novelty detection problem is significant, meaning that such an approach would fail to satisfy our third objective mentioned above.

Criterion	Degree of Success
General Suitability	*Unknown:* Requires further investigation
Customisability	*Good:* Not tightly restricted to any particular topology; learning parameters are adjustable
Work	*Poor:* User is required to determine a suitable topology for particular application
Complexity	*Good:* Can potentially be minimised for particular application

Table 3.1: A summary of the evaluation of the DPC neural network model against the requirements in chapter 1.

A summary of this evaluation is shown in table 3.1. The key drawback of using the DPC neural network model is the work required from the user in constructing a suitable neural network for a particular application. As we identified in chapter 2, one approach that has been used to overcome this drawback is to search for a suitable topology using an evolutionary algorithm. In the second part of this chapter, we explore this approach and identify a promising algorithm from the literature that could be used to evolve DPC neural networks. This sets the scene for the novelty detection approach that we propose in chapter 4. In the next section, we introduce evolutionary neural networks, approaches which employ evolutionary algorithms to determine the structure and/or the weights of a neural network to maximise some measure of performance, and identify the common problems that can arise in evolving the structure and weights of a network. In section 3.6, we explore an algorithm, Stanley's Neuroevolution of Augmenting Topologies [108], which allows the evolution of both network structure and weights, whilst overcoming the problems detailed in section 3.5.

3.5 Evolutionary Neural Networks

An evolutionary neural network (ENN) uses evolution as another method of adaptation in addition to learning [130]. An evolutionary algorithm may be used to determine connection weights, suitable network architecture or appropriate learning rules [130]. In this section, we first introduce evolutionary algorithms and then examine how these are applied to neural networks. We are primarily interested in evolutionary approaches which are capable of simultaneously searching for near-optimal network topology and connection weights. Such approaches may be referred to as topology and weight evolutionary artificial neural networks (TWEANNs) [108].

```
1.  BEGIN
2.      INITIALISE population with random candidate solutions;
3.      EVALUATE each candidate;
4.      REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
5.          SELECT parents;
6.          RECOMBINE pairs of parents;
7.          MUTATE the resulting offspring;
8.          EVALUATE new candidates;
9.          SELECT individuals for the next generation;
10.     OD
11. END
```

Table 3.2: Pseudocode of the general evolutionary algorithm, given by Eiben and Smith [34].

We identify three important issues that arise with TWEANNs and, in the next section, present an approach proposed by Stanley [108] which tries to overcome these issues.

3.5.1 Evolutionary Algorithms

An evolutionary algorithm is a search algorithm that employs Darwinian evolution principles to find a solution which optimises some measure of performance or *fitness*. A population of "individuals", representing candidate solutions, is evolved over a series of generations until either a candidate solution of sufficient quality is found or some previously set computational limit is reached [34]. Each individual is represented as a *genotype*, which contains all the information necessary to construct a realisation of that individual. This realisation, i.e. the actual solution it represents, is referred to as a *phenotype* (the terms genotype and phenotype are borrowed from the field of molecular genetics [34]). Evolution takes place by allowing individuals of high fitness to have a higher probability of reproducing, yielding offspring for the next generation. Individuals "compete" with one another, based on their fitness, to be selected as parents during the reproduction phase, for inclusion of their offspring in the next generation. The fitness measure, normally derived by a *fitness function*, may be seen as applying an environmental pressure to the population, which causes natural selection (survival of the fittest) to occur.

A pseudocode representation of the general structure of an evolutionary algorithm is shown in table 3.2 [34]. Each member of the initial population is typically generated randomly in some way. Lines 4-10 of the algorithm in table 3.2 iteratively evolve the population. The termination condition may be that a certain level of fitness must be achieved by one or more individuals in the population, or that evolution should continue only for a predetermined number of generations. Lines 3 and 8 determine the fitness of each candidate of the population, normally by using some

predefined fitness function. The selection of individuals on lines 5 and 9 is performed according to the results of this fitness evaluation: fitter individuals are preferred over less-fit individuals.

Lines 6 and 7 of table 3.2 illustrate the reproduction phase of an evolutionary algorithm. This phase normally involves two operations: recombination, also known as *crossover*, and *mutation* [34]. The crossover operator generates one or two offspring genotypes by merging, or crossing over, information from two parent genotypes. This operator mimics sexual reproduction, observed in the majority of higher biological organisms on Earth [34]. The mutation operator is a stochastic unary operator which, when applied to a genotype, makes some (typically random) modification to that genotype. The precise definition of both the crossover and mutation operators are dependent on the actual representation being used for the genotype.

Genetic Algorithms

The most widely known kind of evolutionary algorithm is the *genetic algorithm* [34], originally proposed by Holland [60]. A population of genotypes, of some representation, are evolved according to a fitness criterion, using crossover and mutation operators to facilitate reproduction. There is no single definition of the genetic algorithm [34, 89]. The precise implementation depends critically on the chosen representation, since this dictates the design of the crossover and mutation operators. Following Holland's work, many genetic algorithms use a binary encoding (i.e. a string of binary digits, known as a bitstring), and much of the existing theory on genetic algorithms assumes this representation [89]. However, many problems are not naturally expressed as bitstrings [89], and so other forms of representation, such as a decimal encoding, are also used.

Other Kinds of Evolutionary Algorithm

Another type of evolutionary algorithm is Rechenberg and Schwefel's *evolution strategies* approach. Evolution strategies are typically employed for continuous parameter optimisation and have a strong emphasis on mutation during the reproduction phase [34]. The goal of evolution strategies is to find an n-dimensional vector which minimises a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The genotypical representation of each individual in the population consists of an n-dimensional vector X, which is to be optimised, and a series of parameters of the mutation operator. These mutation parameters are evolved in addition to the elements of X, thus making the approach

self-adaptive[1]. Crossover is also supported, in which two parents, chosen at random from the population, produce a single offspring. At the end of each generation, the fitness of each individual, including offspring, is calculated for selection. Depending on the selection mechanism being used, members of the next generation are selected (based on fitness), either from the offspring only or from all individuals, to carry over to the next generation.

Evolutionary algorithms have also been considered for the task of constructing computer programs to carry out certain tasks. One such approach is *evolutionary programming*, originally proposed by Fogel *et al.* [42]. Evolutionary programming was initially used to evolve a population of predictors, where each predictor was a finite state machine (FSM) [34]. However, more recently the approach has been employed in the optimisation of real-valued parameter vectors, bearing similarities to evolution strategies.

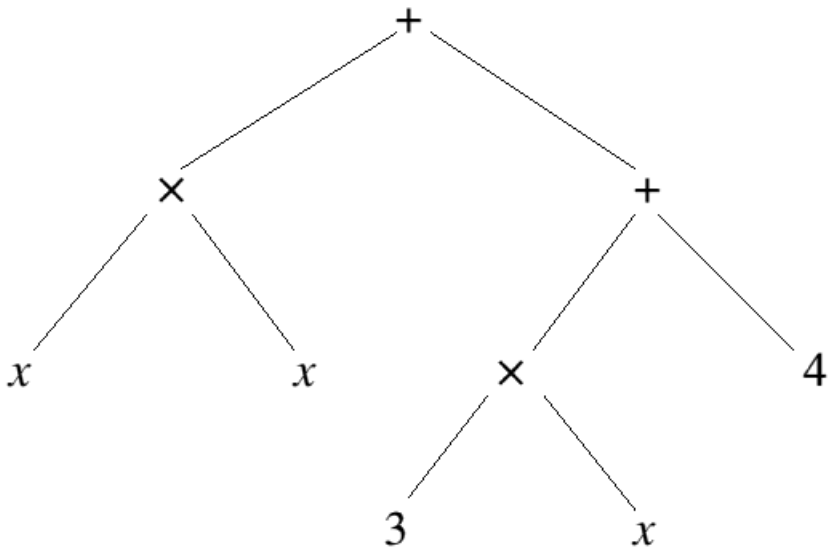


Figure 3.14: An example of a parse tree used in genetic programming. This parse tree represents the function $f(x) = x^2 + 3x + 4$.

Another approach to the evolution of computer programs is Koza's *genetic programming* method [72]. In this approach, the genotype of each individual has a parse tree representation, which may be translated into a program in some computer programming language (originally Lisp [72]) to yield the corresponding phenotype representation. Each node in the parse tree is either a function, defining some operation, or a terminal, defining a variable or constant value. Each branch extending from a function node connects one argument to that function. An example parse tree for the function x^2+3x+4 is shown in figure 3.14. The functions and terminals

[1]Self-adaptivity, in general, refers to the ability of an evolutionary algorithm to vary its own parameters during search [34]

that may appear in the parse tree are limited to the elements of two predefined sets: the function set and the terminal set [89], which are provided *a priori* by the user. The genetic programming algorithm then proceeds to evolve a population of programs, using crossover and mutation operators defined over the parse tree representation. Through the mutation operators, new structure (i.e. new function and terminal nodes) may be added, whilst the crossover operation allows subtrees within two programs to be swapped over [89]. The fitness of an individual is evaluated by running the program, given by that individuals phenotype representation, over a series of "fitness cases" for which the input-output relationship is known [90].

3.5.2 Evolving Neural Networks

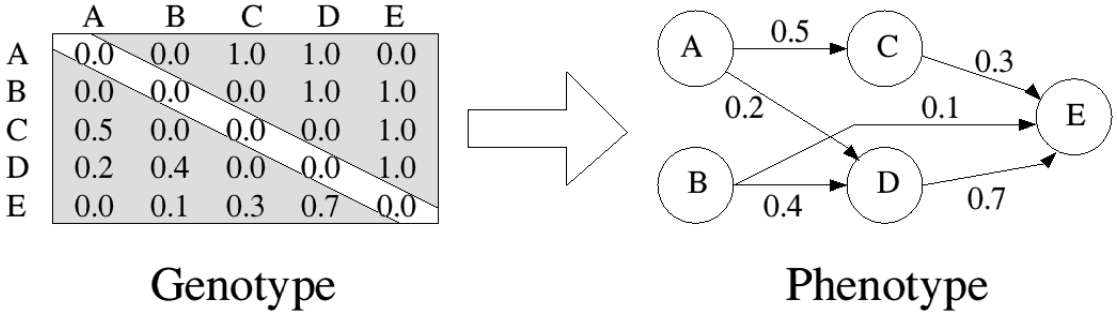


Figure 3.15: The matrix-based encoding scheme used by Han and Cho [55]. In this example, the representation is limited to expressing networks with a maximum of 5 neurons.

When designing an evolutionary algorithm to evolve a neural network, the first challenge that must be tackled is finding a suitable genotypical encoding of the neural network. Encoding methods generally fall into two schemes: *direct encoding*, in which all information about the network is held in the chromosome, and *indirect encoding*, where only the most important information is stored [130]. Once an appropriate encoding method has been determined, the key reproduction operators, crossover and mutation, must then be defined. Finally, appropriate methods for selecting parents from the population as well as deciding those individuals that should be carried over to the next generation must also be determined. The design of an appropriate fitness function is separate from the design of the evolutionary algorithm, since it is dependent on the particular application to which the evolutionary algorithm is to be applied.

We now consider the design of two of the ENN-based approaches to novelty detection identified in chapter 2. In the approach proposed by Samanta [98], an indirect encoding method is used to evolve multilayer perceptron (MLP) networks with a single hidden layer. In this approach, the genotype encoding of a neural network is a vector of length $n + 1$. The first n elements of

this vector define the n inputs, chosen from a set of possible inputs, that should be processed by the network. The final element defines the number of neurons that the hidden layer should contain. The crossover operator used in this approach is defined as a linear combination of two parent genotypes, whilst the mutation operator randomly selects an element of the genotype vector and replaces this element with a new random number. Individuals of the population to be used to produce successive generations are selected by using *rank-based selection*. First, the population is ranked, based on fitness, and then each individual is assigned a probability, according to their rank, of being selected. Selection is then performed randomly, according to these assigned probabilities.

The more versatile approach proposed by Han and Cho [55] uses a direct encoding scheme. The genotype of each network has a matrix representation, shown in figure 3.15 for a network with 5 nodes. The matrix has dimensions $N \times N$, where N is the maximum number of nodes in the neural network. The upper-right triangle of the matrix defines the synaptic connections that exist in the network, where 1 indicates the presence of a connection between two nodes and 0 indicates no connection. The lower-left triangle contains the weight values corresponding to each connection. When translating this representation to an actual neural network, redundant structure, such as hidden neurons with no connections, and structural elements unsuitable for a network using the back propagation learning method, such as connections between input neurons or between output neurons, are inhibited. In this approach, the crossover operator is defined over two parent neural networks as follows. One hidden node is randomly selected as a pivot point. Then, the connections to and from that hidden node in the two parents are exchanged, along with the corresponding weight information, to give two new offspring networks. The mutation operator used in this approach either adds a new connection to a network or deletes an existing connection as follows. Two nodes are selected from the network at random. If these nodes are unconnected, then a new connection is created between them with a random weight. Otherwise, if a connection already exists then this is deleted from the network. As with Samanta's approach [98], rank-based selection is again used to determine those individuals who should contribute to future generations.

Of the two approaches considered above, we are most interested in that proposed by Han and Cho. Since this approach evolves both the structure and the weights of neural networks, it represents a kind of TWEANN. This approach is also appealing because it uses a direct encoding method, which allows all aspects of the networks design to be determined through the evolutionary search. This is important as it maximises the degree to which the evolutionary

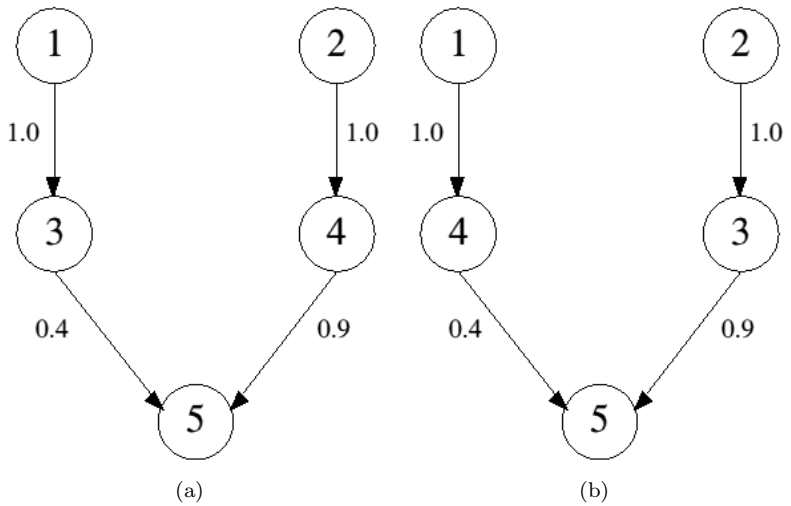


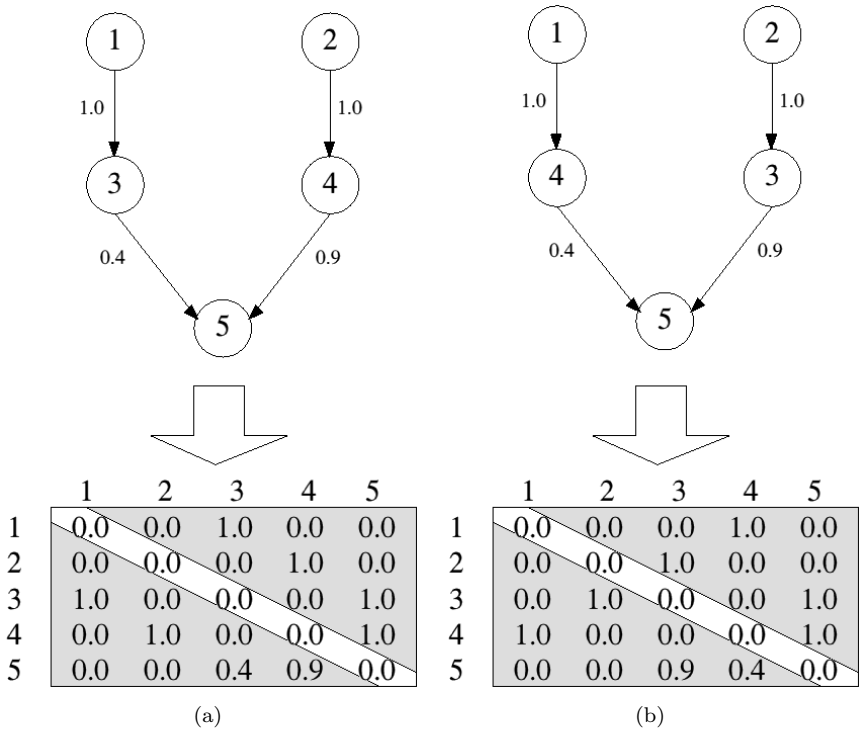
Figure 3.16: Two functionally equivalent neural networks with different labels assigned to the nodes in the hidden layer.

algorithm may customise a network for a particular application. However, this approach has the drawback of requiring the user to specify the maximum number of hidden neurons that an evolved network may have. It also fails to address some significant problems that have long been associated with evolving neural networks using direct encoding methods. We now examine three such problems in detail.

3.5.3 Competing Conventions

Depending on the particular encoding method employed, it may be possible for neural networks with functionally equivalent topologies to be represented by a number of different genotypes [130]. For example, consider the neural networks shown in figures 3.16(a) and 3.16(b). These networks have identical topologies but different labels have been assigned to the hidden neurons. Assuming each hidden neuron is identical, then the labels assigned to the neurons are arbitrary [95], and neurons can be relabelled without causing any change in the functionality of the network. With the encoding method used by Han and Cho [55], two different representations are produced for these networks, as shown in figures 3.17(a) and 3.17(b). This problem is known as the *competing conventions* problem [124], or the permutation problem [95].

When the crossover operator is applied to different representations of the same network, it becomes very inefficient and ineffective in producing good offspring [130]. This can be demonstrated by applying the crossover operator defined in Han and Cho's approach [55] to the networks in figures 3.16(a) and 3.16(b). This crossover operation yields the offspring networks



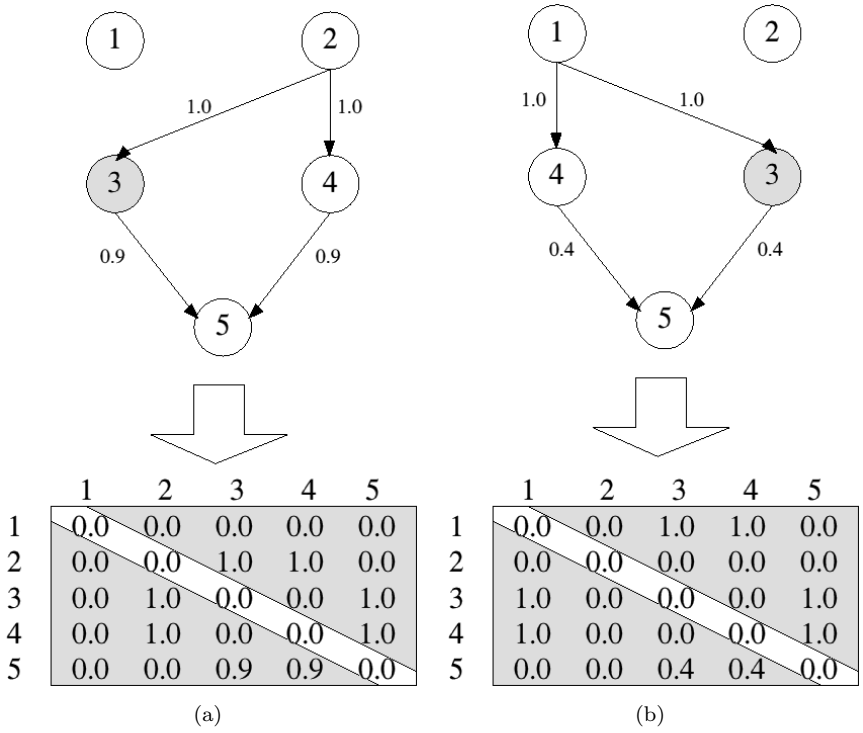
	1	2	3	4	5
1	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	1.0
4	0.0	1.0	0.0	0.0	1.0
5	0.0	0.0	0.4	0.9	0.0

(a)

	1	2	3	4	5
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0
3	0.0	1.0	0.0	0.0	1.0
4	1.0	0.0	0.0	0.0	1.0
5	0.0	0.0	0.9	0.4	0.0

(b)

Figure 3.17: The genotype representation of the two neural networks shown in figure 3.16, using the matrix encoding method employed by Han and Cho [55].



	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	1.0	0.0
3	0.0	1.0	0.0	0.0	1.0
4	0.0	1.0	0.0	0.0	1.0
5	0.0	0.0	0.9	0.9	0.0

(a)

	1	2	3	4	5
1	0.0	0.0	1.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	1.0
4	1.0	0.0	0.0	0.0	1.0
5	0.0	0.0	0.4	0.4	0.0

(b)

Figure 3.18: The offspring neural networks generated by the crossover operator, as defined by Han and Cho [55], when applied to the networks shown in figure 3.16.

shown in figure 3.18. These networks are considered damaged because they both (i) omit information common to both parents and (ii) duplicate information [124]. The first network omits the entire pathway from the first input node to the output neuron. This is not simply a case of the input node becoming disconnected; the weight information of the pathway is also lost. The second network shows the same loss of information, but this time for the pathway from the second input node to the output neuron. The first network also duplicates the pathway from the second input node to the output neuron, and the second network duplicates the pathway from the first input node.

3.5.4 Crossover of Networks with Very Different Topologies

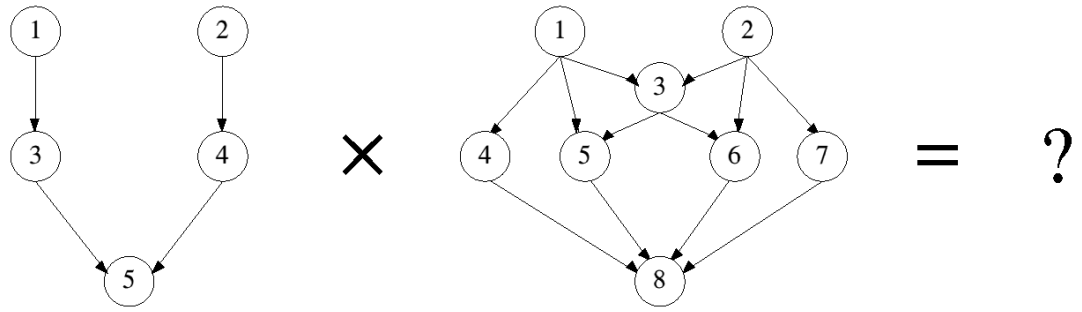


Figure 3.19: Two neural networks that illustrate the variable-length genome problem. The topologies of these networks are so different that it is unclear as to how an effective crossover operator should be defined [108].

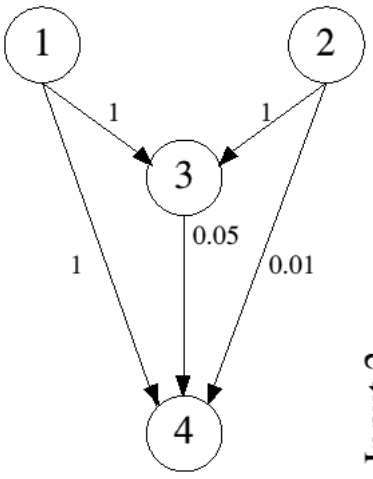
In a TWEANN, the problem of crossing over individuals in a population is further exacerbated by the potential differences in topology between those individuals. For example, how should a crossover operator combine the two networks shown in figure 3.19? Another problem concerns topologically different networks that implement similar functionalities. If two such parent networks perform the same function, can crossover be performed such that this function is inherited by the offspring networks? This problem has been referred to by Stanley as the "variable-length genome problem" [108].

Because of the great difficulties of defining a meaningful crossover operator over genotypes representing networks with very different topologies, some researchers have avoided using the crossover operator altogether [130], instead relying on mutation operators alone. However, Han and Cho [55] continue to use a crossover operator which is susceptible to the problems mentioned above. In our work, we choose an evolutionary approach which uses a crossover operator that addresses both of these problems.

This problem could also potentially be addressed by using Estimation of Distribution Algorithms (EDAs) [91]. These are evolutionary algorithms which do not use either crossover or mutation operators to generate offspring. Instead, offspring are produced by first constructing a probability distribution model of promising solutions, based on a selection of individuals from the current generation, and then sampling from this model. However, whilst EDAs have been used to train neural networks [82], they have not to our knowledge been used to evolve network structure.

3.5.5 Protecting Structural Innovations

As new structure is introduced to a neural network, it tends to result in an initial degradation in the fitness of that network [108]. The introduction of new structure may require optimisation of new weights, corresponding to the new structure, as well as existing weights in the network. This optimisation may lead to an overall improvement in network performance in the long-term. However, in many TWEANNs, all individuals in a population are forced to continuously compete with one another. Therefore, networks with new structural innovations which could prove valuable in the long run may be prematurely discarded because of the initial impact of the new structure on network fitness. The effect of this is at best to prolong the search, and at worst to prevent a successful outcome. The approach proposed by Han and Cho is again susceptible to this problem.



Input 2 \ Input 1	0	1
0	0	1
1	0	1

Figure 3.20: A neural network with the structure necessary to perform the XOR function [57, 108] but with unoptimised weight values. Without further optimisation, this network gives a poor classification performance, as shown in the corresponding truth table.

As an elementary example, consider the problem of finding a neural network capable of

performing the XOR function. A network without any hidden neurons cannot possibly implement XOR, since it is only capable of establishing a single hyperplane in the input space [57]. Therefore, a population consisting entirely of networks with no hidden neurons will never solve the problem. The closest approximation such a network could achieve is to provide the correct output for three of the four possible input patterns. A simple fitness measure may reward such networks with a fitness of 75 out of 100. The network shown in figure 3.20 on the other hand has the minimum required additional structure to allow it to completely solve the problem [57, 108] and so achieve a fitness score of 100. However, with the unoptimised weight values shown in the figure, this network gives the correct response in only 2 out of 4 cases (as shown by the truth table in figure 3.20), thus only achieving a fitness score of 50. Therefore, it is possible that the structural innovation given by this network would be prematurely discarded, which at best would result in prolonging the evolutionary search. Note that, for simplicity, we have considered the introduction of all required structure in one operation. In practise, the process is more incremental, but the same problem still arises.

Ideally, one would like to protect in some way new structural innovations from being discarded until they have been given time to mature. As we will see in the next section, an effective approach is to *speciate* the population, grouping networks with similar structure and having networks competing directly with others inside these groups instead of with all other members of the population as a whole.

3.6 Neuroevolution of Augmenting Topologies

Stanley recently proposed a novel approach to neuroevolution [108], Neuroevolution of Augmenting Topologies (NEAT), which attempts to overcome the problems mentioned in sections 3.5.3-3.5.5. In this approach, a direct encoding strategy is used which does not impose limits on the maximum size of neural network solutions, unlike the approach used by Han and Cho [55]. Historical markings are used to track structural elements of different networks which are compatible with each other. Speciation is employed to protect new structural innovations. In this section, we discuss the NEAT algorithm in detail.

3.6.1 Encoding Strategy

A genotype in NEAT represents a single neural network and consists of a series of node genes, representing individual neurons in the network, and connection genes, representing individual

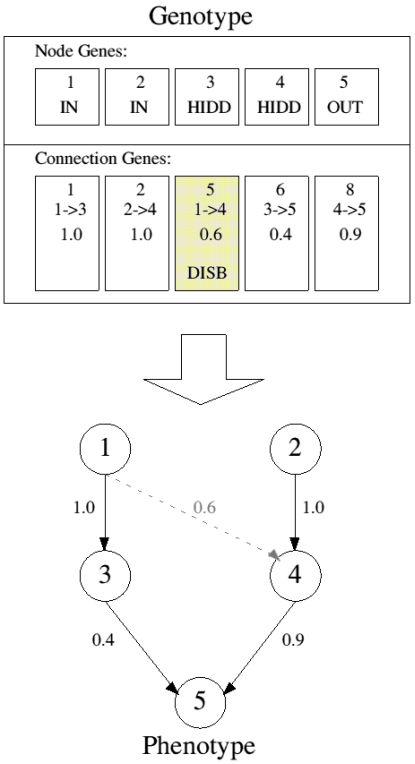


Figure 3.21: An example genotype in the representation used by NEAT with its corresponding phenotype neural network.

synaptic connections. A node gene contains a unique identification number for the corresponding node as well as identifying the type (input, hidden, output) of the neuron. A connection gene holds information identifying the nodes connected, the connection weight (or synaptic strength), whether or not the connection is enabled, as well as an *innovation number* whose purpose we explain shortly. A phenotype, the realisation of a neural network described by a genotype, is constructed from the information in all node genes and all *enabled* connection genes. Figure 3.21 illustrates a genotype and its corresponding phenotype.

The number of neurons and synapses present in a particular neural network is unbounded in NEAT. By simply adding node and connection genes to a genotype, new structure can be realised in the phenotype. However, as the structure of the network is increased, so is the dimensionality of the weight space. Since a high-dimensional weight space is harder to search, NEAT follows the philosophy of adding new structure slowly, so as to allow sufficient time to explore the current weight space first. This also means that NEAT tends to evolve solutions with a low level of complexity.

3.6.2 Historical Markings

In order to track compatible structural elements between a population of networks with different topologies, NEAT employs a system of historical markings. The innovation number of a particular gene tracks the structural innovation represented by that gene. For example, in figure 3.21, the connection from node 1 to node 4 is considered as a structural innovation which is different from that of the connection from node 2 to node 4. Therefore, the two genes are assigned different innovation numbers.

When a new connection is introduced to a particular network, a search is performed on a list of innovations to determine whether or not that connection has been created in another network in the current generation. Two connections match if they represent a link between the same two neurons in the network, that is two neurons with the same identification number. If the connection has not been created before in the current generation, it is assigned a new innovation number. The reference to historical markings is used by Stanley to highlight the fact that genes with the same historical origin, and thus representing the same structure, can be identified from the innovation numbers.

3.6.3 Crossover

The introduction of historical markings allows NEAT to solve a significant problem faced by TWEANNs: the design of a meaningful crossover operator between networks of different topologies. By using the innovation numbers, it is now trivial to identify compatible structure between two networks that can be swapped over. For two parents, the crossover operator yields a single offspring. All connection genes of both parents (including disabled genes) are aligned as shown in figure 3.22. The connection genes of both parent genotypes which have the same innovation number are called *matching* genes. Those genes of one parent without a match in the second parent are labelled as *disjoint* or *excess* genes. A disjoint gene is a non-matching gene in one parent, whose innovation number falls in the range of innovation numbers covered by the genes in the second parent. An excess gene has an innovation number falling outside this range.

The crossover operation then proceeds as follows. The matching genes are either selected randomly from each parent or averaged to give a single offspring gene. All disjoint and excess genes are inherited only from the fitter parent (or from either parent randomly in the event of equal fitness). If a gene inherited by the offspring is disabled in either parent, then it has a preset chance of remaining disabled in the offspring. Therefore, genes once disabled may, by

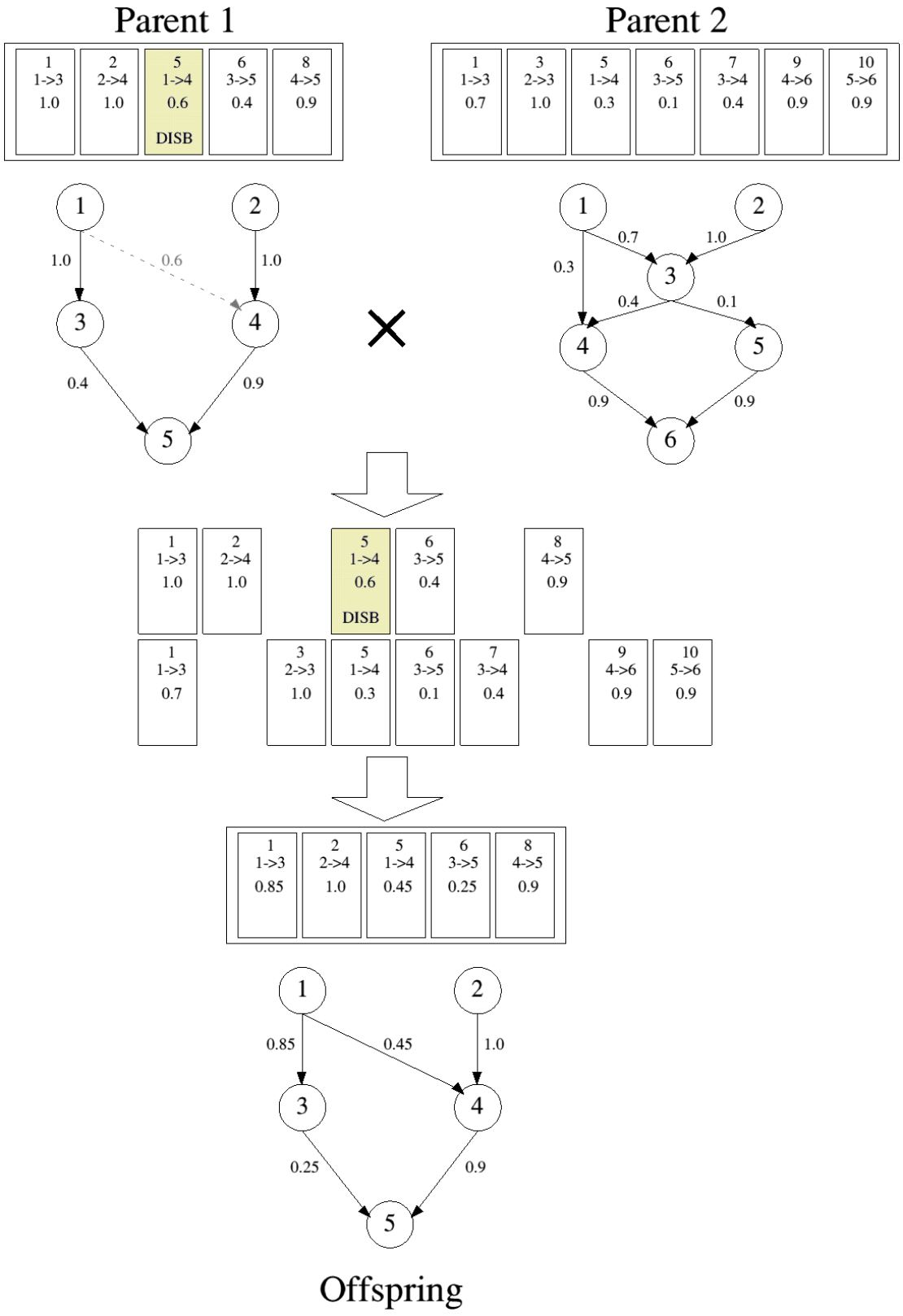


Figure 3.22: The crossover operator implemented by NEAT.

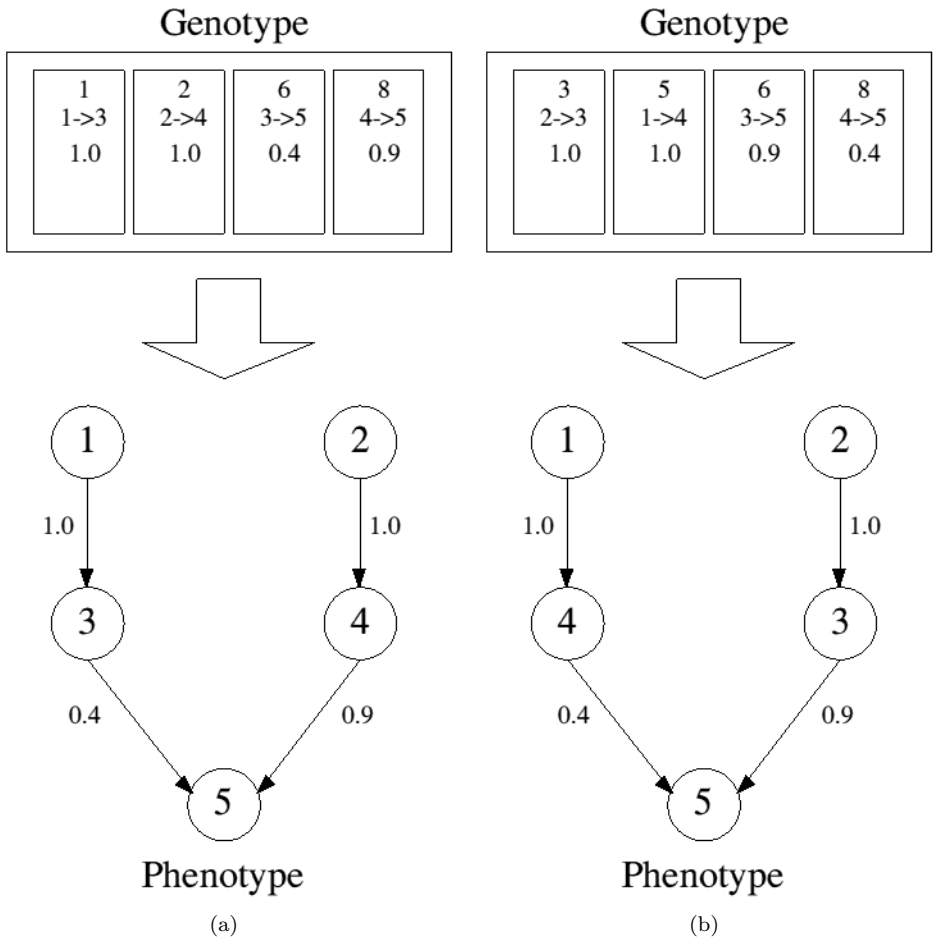


Figure 3.23: The genotype representation used by NEAT of the two neural networks shown in figure 3.16.

chance, be used again in the future. Finally, all node genes from either parent referenced by the connection genes in the offspring are automatically copied to the offspring genotype.

The crossover operation for two parent genotypes Parent1 and Parent2, where it is assumed that Parent1 is fitter than Parent2, is shown in figure 3.22. For clarity, only the connection genes are considered. As can be seen, this operator allows a non-damaged offspring to be constructed from two networks with different topologies. If the crossover operator is applied to two networks of *completely* different topologies (i.e. no matching genes), then the outcome is an offspring which is simply a copy of the more fit parent. However, by speciating the population NEAT prevents crossovers between two such different genotypes from occurring [108].

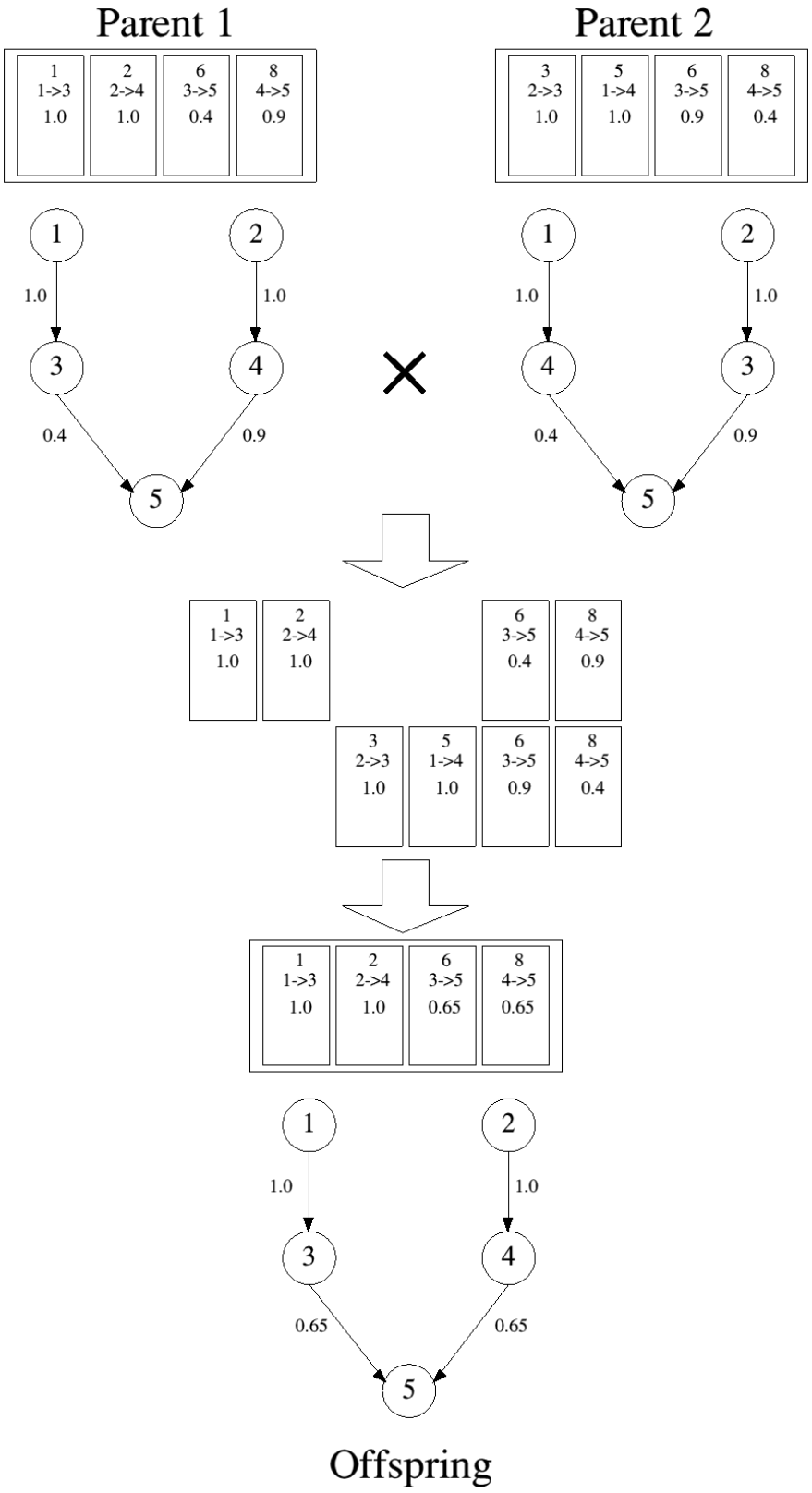


Figure 3.24: The crossover operator implemented by NEAT applied to the two neural networks shown in figure 3.16.

Competing Conventions and the Variable Length Genome Problem

Whilst NEAT's crossover operator allows the successful recombination of two networks with different topologies, how does this help tackle the competing conventions problem? Consider again the two networks shown in figure 3.16. Figure 3.23 shows the NEAT representation of each network and figure 3.24 illustrates the crossover operation between these networks. As shown in figure 3.24, NEAT first lines up the genes in the two genotypes to identify both identical and different topological structure between the two networks. In both networks each hidden neuron is connected to the output neuron, and so NEAT identifies the corresponding connection genes as representing the same structure. The genes in each network corresponding to the connections between the input nodes and hidden neurons, however, are identified by NEAT as representing different structure, and are classed as excess or disjoint genes (depending on their historical markings). Assuming the excess and disjoint genes are inherited from the first parent, and that weights of matching genes are averaged, the offspring network shown in figure 3.24 is produced. Unlike the offspring produced by Han and Cho's crossover operator [55], this network preserves the pathway from each input neuron to the output neuron, but with the weights of both connections between hidden and output neurons set to 0.65.

Notice from figure 3.23 that the two parent networks still have different genotype representations. Therefore, the competing conventions problem is not solved by NEAT, but instead it is prevented from having a negative impact on the crossover operation. The net effect of the crossover operation demonstrated in figure 3.24 is to create a genotype which is a copy of the first parent, but with a mutation applied to the weights between the hidden and output neurons.

NEAT also addresses the variable length genome problem discussed earlier. For two networks with similar, yet different topologies, NEAT produces an offspring which inherits the structure both parents have in common, as well as the additional structure from the more fit parent. If both parent networks perform a similar function, then intuitively the offspring network also has the potential to realise this function. In this way, NEAT can be seen as solving the variable length genome problem. For two completely different networks, NEAT disallows crossover through speciation. In this case, NEAT can be viewed as *avoiding* the problem.

3.6.4 Mutation

In NEAT, three different mutation operators are defined: add node, add connection and perturb weights. The add node and add connection mutations allow new structure to be introduced.

In the add node mutation, an existing connection in the network is effectively split by the new neuron. The add connection mutation creates a new link between two previously unconnected nodes. The perturb weights mutation enables exploration of the weight space of a particular network. Each mutation operator is controlled by a rate parameter. Since NEAT's goal is to explore the weight space often, and introduce new structure only gradually, the perturb weight rate parameter tends to be set to a high value whilst the add node and add connection rate parameters are set to a low value.

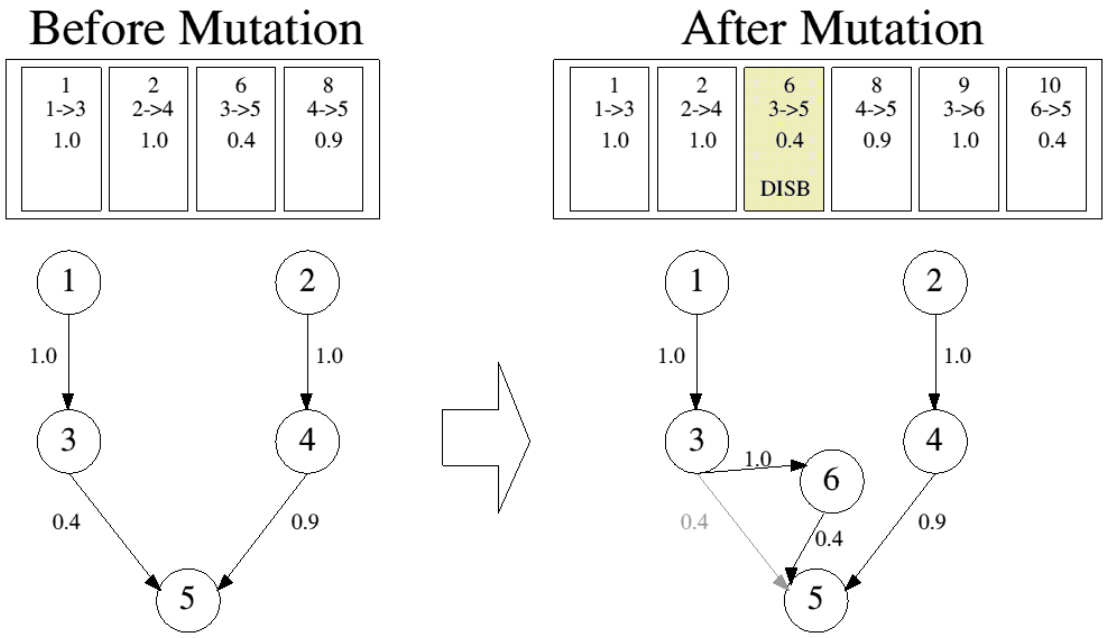


Figure 3.25: The add node mutation operator defined by NEAT.

The add node mutation operator introduces a new neuron into the network as shown in figure 3.25. An existing connection is chosen at random in the genotype and disabled. Then, a new node gene is introduced, as well as two connection genes which establish a link between that neuron and the neurons previously linked by the disabled connection in the network. The new connection leading to the new neuron is assigned a weight of 1, whilst the new connection leading from this neuron inherits the weight of the disabled connection. This operation has the effect of splitting the chosen connection with the new neuron, immediately incorporating the neuron into the network structure. The add connection mutation operator introduces a new connection into the network between two previously unconnected neurons, chosen at random. The new connection is assigned a random weight value. This mutation is illustrated in figure 3.26. Finally, the perturb weights mutation operator randomly adjusts the connection weights

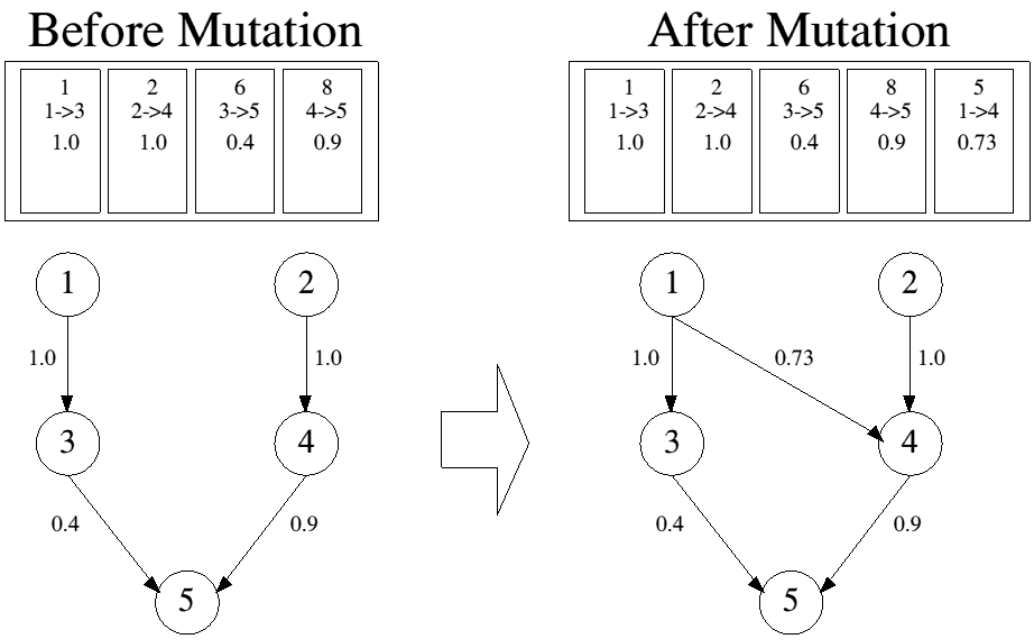


Figure 3.26: The add connection mutation operator defined by NEAT.

in a network. Each genotype has a preset chance, defined by the perturb weights rate parameter, of having its connection weights perturbed. If a genotype is to have its weights mutated, then each weight is either adjusted or replaced by some random value.

3.6.5 Speciation and Reproduction

As previously discussed, the introduction of new structure tends to have an initial negative impact on the fitness of a neural network, despite the fact that, in the long-term, the new structure could allow the network to achieve a much better fitness. In order to protect new structural innovations, NEAT *speciates* the population, such that similar networks are grouped together into a particular species. If a particular network is too different to belong in any existing species, a new species is created for that network.

However, for speciation to work, some measure of similarity, or compatibility, between two networks must be defined. The problem of identifying which parts of two networks are compatible, necessary to allow the design of a meaningful crossover operation, was previously solved with the introduction of innovation numbers. Therefore, these same innovation numbers can also be used to determine a compatibility measure, based on the premise that, the more disjoint and excess genes that exist between two networks, the less compatible they are. The similarity of two networks also depends on the how different the weights are for each matching gene. Stanley

therefore defined a *compatibility distance* measure between two networks as:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} \tag{3.26}$$

where E is the number of excess genes between two genotypes, D is the number of disjoint genes, $\bar{W}$ is the average weight differences of matching genes, c_1, c_2 and c_3 are coefficient parameters, and N is a normalisation parameter, normally set to one[2]. If the compatibility distance δ between two networks is below a preset compatibility threshold parameter δ_t, then the networks are considered compatible.

Assigning Genotypes to Species

At the start of every generation, NEAT allocates each genotype to a species as follows. For each species present in the *previous* generation, a random genotype is selected as a representative for that species. For the first generation, a single species is constructed with the first member of the starting population as its representative. Then, each genotype in the population is compared sequentially against each species representative using the compatibility distance measure δ. If a genotype is found to be compatible with a species representative (i.e. $\delta < \delta_t$), then the genotype is assigned to that species and the search stops. That is, the genotype is assigned to the first compatible species encountered during the search. If a genotype is found to be incompatible with all species representatives, then the genotype is added to a new species, for which it also acts as representative for the remainder of the speciation process.

Reproduction

NEAT uses speciation to protect structural innovation by preventing topologically different networks from directly competing with eachother. Instead, competition occurs during the reproduction stage at two levels. At the first level, species compete to achieve a number of allowed offspring. At the second level, individuals within the species compete to become parents.

To allow speciated evolution to support a variety of topologies, it is essential that a single species not be permitted to dominate the population [108]. To prevent this from occurring, the fitness f_i of each genotype i within a species s is adjusted prior to the reproduction phase, according to the number of genotypes in that species $|s|$:

[2]In [108], Stanley states that N can be set to 1 if both genotypes consist of fewer than 20 genes. However, Stanley's actual implementation of NEAT fixes N to 1 for all genotypes, regardless of size.

$$f'_i = \frac{f_i}{|s|} \tag{3.27}$$

This is derived from Goldberg and Richardson's *explicit fitness sharing* method [45]. Each genotype of a species must *share* its fitness with that species. The fitness adjustment encourages the formation of a series of smaller species in the population, and thus promotes diversity of network topologies.

Reproduction in NEAT proceeds so as to maintain a constant population size in each generation. Therefore, the total number of allowed offspring is defined as the population size. Species compete on the following basis: the fitter the genotypes in a particular species s, compared to all species in the population, the more offspring that are assigned to s. Only the adjusted fitness values are considered during reproduction. The number of offspring n_s assigned to a particular species s is calculated as:

$$n_s = \frac{\bar{F}_s}{\sum_k^{|s|} \bar{F}_k} |P| \tag{3.28}$$

where $\bar{F}_s$ is the average adjusted fitness of all genotypes in species s, and $|P|$ is the size of the population P.

Once each species has been assigned a number of offspring, the second level of competition occurs. If the species is allowed more than a certain number of offspring, then the first offspring is simply a clone of the species' best genotype, known as the *species champion*[3]. Genotypes in each species are then ranked by fitness and the lowest performing fraction, defined by a survival threshold parameter, are eliminated. Parents are then chosen randomly (with replacement) from the remaining genotypes to create the required number of offspring.

NEAT also provides a mechanism to tackle the problem of stagnation, no improvement occurring within a certain time frame, within a species or the entire population. If no improvement is seen within a species then a punishment is applied to the fitness of all genotypes in that species, resulting in its eventual removal from the population. If no improvement is seen in the best fitness of the population over a certain time frame, then all species apart from the top two are removed from the population, and each of the top two species are assigned approximately half of the total number of allowed offspring. This has the effect of refocussing evolutionary search to the most promising areas of the search space [108].

[3]In [108], Stanley states that every species is allowed a clone of its champion, whilst his implementation of NEAT allows a clone only if the species has more than five offspring. This corresponds to what is reported in [109].

3.6.6 Initial Population

Before NEAT can evolve a population of neural networks, an initial population, representing a starting point for the evolutionary search, must be defined. Each network in the initial population has an identical minimal topology, in which each input node is connected to every output node. Such a topology allows NEAT to start its search in a low-dimensional weight space, which it can then slowly increase as the search progresses. The connection weights used by each network in the starting population are assigned randomly.

3.6.7 Discussion

In section 3.5, three well-known problems of evolving both topology and weights of neural networks were identified:

- *Competing Conventions*: Assigning different genotypical representations to topologically equivalent neural networks.
- *Meaningful Crossover*: Defining a meaningful crossover operation between networks with different topologies, avoiding damaged offspring.
- *Protecting Innovation*: Preventing new structural innovations which could prove invaluable in the future from being prematurely discarded.

NEAT does not solve the competing conventions problem, but instead defines a crossover operator which is not negatively impacted by the problem. By continually tracking functionally equivalent structure at the connection level, NEAT is able to define a meaningful crossover operator which successfully produces undamaged offspring from two networks with different topologies. Furthermore, this also enables NEAT to effectively speciate the population, thereby protecting new structural innovations until the corresponding networks have had time to optimise their performance with the new structure. Also, unlike the approach used by Han and Cho [55], NEAT does not impose a limit on the maximum size that may be reached by a network during evolution. This frees the user of the burden of deciding precisely what this maximal value should be [108]. Another attractive feature of NEAT is that it can potentially evolve solutions with a low level of complexity for the problem at hand. This is achieved by starting the evolutionary search in a space of minimal dimensionality, which is then increased only slowly (by adding new structure) after sufficient time has been allowed to explore that space.

However, one drawback of NEAT is that it requires a large number of parameters to be set. These parameters are summarised in appendix A. It is unclear as to how closely the optimal parameters depend on the specific problem being solved by NEAT. Finding optimal values manually is an extremely difficult task, because of the high dimensionality of the parameter space. However, it may be possible for NEAT to achieve a good performance in different applications *without* optimal parameter settings. Despite this drawback, NEAT demonstrates a significant advancement in tackling the problem of simultaneously evolving the topology and weights of a neural network.

3.7 Summary

In this chapter, we have described Hosoya *et al.*'s DPC neural network model and examined the operation of an illustrative neural network implementation of this model over a series of artificial visual environments. The ability of this model to separate stimuli from the environment to which it has adapted from stimuli generated by other environments demonstrates the potential of DPC for novelty detection. The model is highly customisable and, due to the minimal assumptions it makes about the properties of the data, we consider it likely to be potentially successful in a wide range of different novelty detection tasks. However, the amount of work required to employ this approach in a particular task is significant. A suitable network topology must be designed, which may be prohibitively difficult for the non-expert user, and values must be chosen for the parameters of the anti-Hebbian learning rule.

One way of overcoming this drawback is to use an evolutionary algorithm to automatically design a suitable neural network structure. However, there are three significant problems to such an approach. First, two networks which implement the same function may be given different encodings, which can result in damaged offspring when the crossover operator is applied to the networks. Second, it is non-trivial to design a meaningful crossover operator that can be applied to networks with wide ranges of differing topologies without resulting in damaged offspring. Finally, the introduction of new structure can initially result in causing the fitness of a network to drop, even though that structure may prove beneficial to the network in the long term. All of these problems are either overcome or avoided by an evolutionary neural network approach, known as Neuroevolution of Augmenting Topologies (NEAT), proposed by Stanley [108]. In this approach, the crossover operator effectively bases the structure of the offspring network on the fitter parent, thus avoiding damaged offspring by preventing both the omission of information

common to both parents and the duplication of information [124]. This also avoids producing damaged offspring when faced with two parents with identical structure but different encodings. To protect new structural innovations, NEAT speciates the population so that only networks which are similar, according to a compatibility distance measure, compete directly with each other. NEAT also tends to evolve neural networks with a low level of complexity, as it starts its search in a space of minimal dimensionality which is then increased slowly as necessary to find an effective solution. However, an important drawback of NEAT is that it requires a large number of parameters to be defined by the user.

Despite this drawback, NEAT may offer an effective solution to the problem of finding an automated method of designing a DPC neural network for a particular novelty detection application. In the next chapter, we present our proposed approach to novelty detection, which we call DPC+NEAT. This approach combines Stanley's NEAT algorithm [108] with Hosoya *et al.*'s DPC model [63] to allow neural network based novelty detectors to be automatically constructed for a particular novelty detection task. We extend the DPC model to include hidden neurons, thus allowing networks of arbitrary topologies to be constructed by NEAT. We demonstrate this approach by first identifying a limitation with the illustrative DPC neural network given by Hosoya *et al.* and then using NEAT to optimise the topology of the network to attempt to overcome this. To address the problem of deciding suitable values for NEAT's many parameters, we choose a series of values given by Stanley [108] and then keep these values fixed throughout all of the experiments performed in this thesis. Finally, in chapter 5 we compare our proposed approach with a number of other novelty detection techniques which were reviewed in chapter 2.

Chapter 4

The Proposed DPC+NEAT Approach

In this chapter, we present our proposed approach to novelty detection. This approach, which we call DPC+NEAT, combines Hosoya *et al.*'s Dynamic Predictive Coding (DPC) neural network model [63] with Stanley's Neuroevolution of Augmenting Topologies (NEAT) algorithm [108]. This aims to overcome the drawback with the DPC model of requiring the user to design the network topology by hand and choose suitable values for the network's parameters. Some of the work presented in this chapter has been published previously by the author [53].

4.1 Introduction

In the previous chapter, we presented Hosoya *et al.*'s Dynamic Predictive Coding (DPC) neural network model. This model was originally proposed as a hypothetical explanation of the visual information processing strategy used by the retina. However, this model is also capable of performing novelty detection, in that it can differentiate between familiar and unfamiliar visual environments. This model is appealing since the sole assumption it makes about the input data is that it has some form of correlational structure which changes in some way between the normal and novel classes. The model also has a minimal level of complexity, since no activation function is used by the neurons and the nonlinearity of the model is provided by a simple anti-Hebbian learning rule. However, it also has the significant drawback of requiring the user to determine the most appropriate neural network structure for their particular task.

One way in which this drawback can be overcome is to use an evolutionary neural network approach to automatically discover suitable network structure and weights. One such approach, Stanley's Neuroevolution of Augmenting Topologies (NEAT) algorithm [108], appears to be particularly effective, addressing some significant challenges faced when evolving neural networks in an unrestrictive manner. In this chapter, we combine Hosoya *et al.*'s DPC model with Stanley's NEAT algorithm to yield a new approach to novelty detection, which we call DPC+NEAT. This approach aims to evolve DPC neural networks which are suitable for use in a particular novelty detection task. The work required from the user is minimised, whilst the potential to customise the network as required for the task is maximised. NEAT's ability to start in a search space of minimal dimensionality, and then increase this dimensionality slowly as necessary, should allow novelty detectors to be evolved with a level of complexity that is justifiable for the novelty detection task. Therefore, we consider this approach to have the potential to fulfil each of the four objectives described in chapter 1.

In the next section, we present the DPC+NEAT approach. Then, in section 4.3 we examine the performance of this approach on a test scenario which is based on the simulation explored in chapter 3. In section 4.4, we compare the response of the best networks found in section 4.3, and the original illustrative neural network described in chapter 3, to the introduction of Gaussian noise. Finally, in section 4.5 we detail the modifications that we must make to DPC+NEAT so that it may be used for other kinds of novelty detection tasks. Some of the work presented in this chapter has been published previously by the author [53].

4.2 The DPC+NEAT Approach

The proposed DPC+NEAT approach allows DPC neural networks to be evolved for novelty detection tasks using the NEAT algorithm. A block diagram illustrating this approach is shown in figure 4.1. Provided with a fitness function and information on the required size of the input layer for the evolved neural network, DPC+NEAT is used to evolve a population of novelty detectors which adhere to a modified version of the DPC neural network model. The best novelty detector evolved can then be used independently of DPC+NEAT in the particular scenario.

The idea of using NEAT to evolve adaptive neural networks is not unprecedented. For example, Stanley proposed a modified version of NEAT, Plastic NEAT, which was capable of evolving neural networks with a series of different Hebbian learning rules [108]. For each connection, one

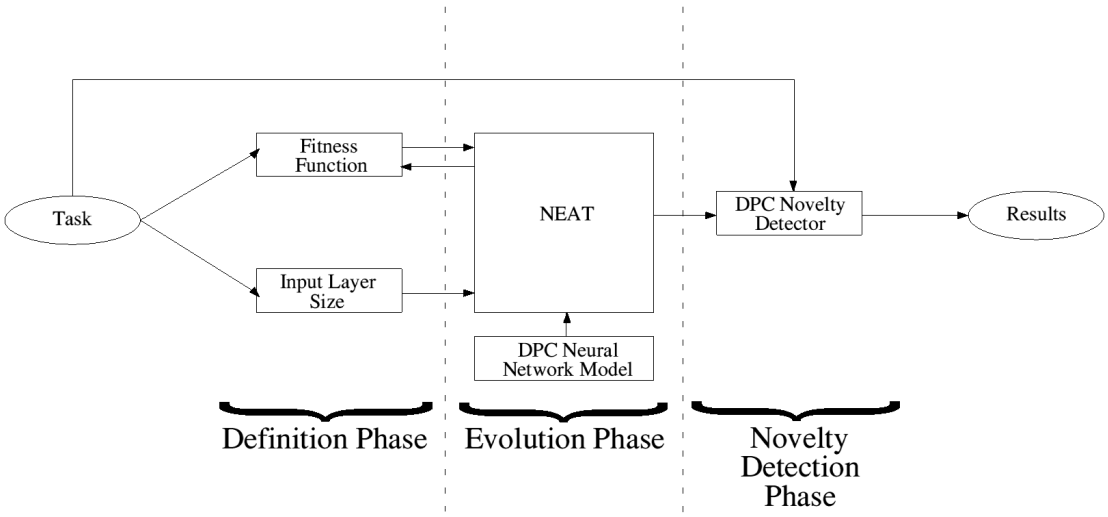


Figure 4.1: A block diagram illustrating the DPC+NEAT approach.

of three rules may be selected: a *plain Hebbian rule*, which increases the connection weight proportionally to correlated activity, a *postsynaptic rule*, which performs in the same manner as the Hebbian rule as well as decreasing the connection weight when the postsynaptic neuron is active only, and a *presynaptic rule*, which performs the same operation as the postsynaptic rule but this time for when the presynaptic neuron is active only. The rules are streamlined into a single general rule for both excitatory and inhibitory connections. Only two learning parameters, η_1 and η_2, are required to express any of the three rules from this general rule. In addition, by setting $\eta_1 = 0$ and $\eta_2 = 0$, a rule representing a fixed-weight connection may be obtained. Rather than evolving a rule for every connection, each genotype holds a single rule set consisting of a finite number of rules. Each gene may then select one of the rules in the rule set, with genes permitted to share the same rule. The rule allocated to each gene is determined through evolution. As in the standard NEAT algorithm, genes in plastic NEAT also have a weight value, again determined through evolution.

In this section, we describe the DPC+NEAT approach. We first examine the representation used to encode DPC neural networks. We also introduce a new mutation operator to NEAT which allows the DPC neural network parameters β, τ and m to be set through evolution. We then extend the DPC neural network model to include hidden neurons, thus providing NEAT with the building blocks necessary to construct any kind of network topology. Finally, we discuss how suitable values for the parameters of the DPC+NEAT approach itself should be determined.

4.2.1 Encoding DPC Networks

Since we wish to evolve DPC neural networks in our approach, unlike Plastic NEAT which permits multiple learning rules, we allow only a single learning rule, the anti-Hebbian rule defined by Hosoya *et al.* [63] (equation 3.3 in section 3.3.1), to be used by the plastic synapses. Therefore, we embed only one additional parameter into each connection gene, *ConnectionType*, which identifies a connection as being either a fixed or plastic synapse. When a new connection is created, it has a probability, defined by a new DPC+NEAT parameter α, of becoming a plastic synapse. The initial weight of the plastic synapses are always set to zero in the corresponding phenotype, allowing these weights to be determined solely by the anti-Hebbian learning rule.

In NEAT, connection genes in different networks which represent the same synaptic connection, i.e. links between the same two nodes, are considered to represent the same structural innovation. Such connection genes may have different synaptic weights, and some may be enabled whilst others are disabled, but they still represent the same physical structure (even if that structure is not implemented in the phenotype). In introducing the *ConnectionType* field, we must consider the impact that this should have on determining whether or not two connection genes represent the same structure. A fixed synaptic connection between two nodes simply scales signals by a given constant, whilst a plastic connection enables learning to occur between the nodes. Therefore, since they provide very different functions to the network, these two types of connection can be considered to be structurally different. Because of this, we consider differences in the *ConnectionType* field between two connection genes to signify that the two genes represent different structural innovations. As such, two connection genes which link the same nodes but represent different connection types would be assigned different innovation numbers by DPC+NEAT and therefore would constitute excess or disjoint genes if their corresponding genotypes were compared.

A DPC neural network has three parameters, β, τ and m, which control the behaviour of the anti-Hebbian learning rule. In Plastic NEAT, Stanley allowed each learning rule to be chosen from a rule set, where each entry had different parameter values [108]. This was because allowing each adaptive synapse to have its learning parameters modified independently would have significantly enlarged the dimensionality of the parameter space, reducing the chance of finding a solution [108]. In our approach, we force the learning rule of each plastic synapse to use the same parameter values. This is compatible with the original specification of the DPC model, as given by Hosoya *et al.* [63], and avoids a possibly unnecessary enlargement of the

parameter space. These network parameters are associated directly with the genotype.

The addition of network parameters to the genotype impacts on the similarity between two networks. Therefore, we amend the compatibility distance measure, given in equation 3.26, to include the absolute difference in these parameters between the two networks, scaled by new parameters c_4, c_5 and c_6. The compatibility distance measure used by DPC+NEAT is given by:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} + c_4|\beta_1 - \beta_2| + c_5|\tau_1 - \tau_2| + c_6|m_1 - m_2| \tag{4.1}$$

where β_i, τ_i and m_i are the parameters associated with network i and $|x|$ represents the absolute value of x.

Finally, so as to allow the DPC network parameters to be evolved by DPC+NEAT, we introduce a new mutation operator, *perturb parameters*, which is controlled by a new rate parameter m_P. When invoked, this mutation operator modifies the network parameter values by using a similar method to Stanley's perturb weights mutation operation: each parameter has a 90% chance of being perturbed and a 10% chance of being replaced with a new value.

Figure 4.2 shows an example of an encoded DPC neural network genotype and its corresponding phenotype (the implemented network). This network has two hidden neurons, which demonstrates an extension of the original DPC neural network model. This extension, to permit hidden neurons in a DPC neural network, increases the range of different network topologies that can be constructed by DPC+NEAT and is discussed next.

4.2.2 Hidden Neurons

In the original model proposed by Hosoya *et al.* [63], a DPC neural network is shown to have an amacrine cell on each plastic link between bipolar and ganglion cells, where the plasticity exists between the amacrine and ganglion cells. These amacrine cells would constitute hidden neurons in an artificial neural network. However, in the mathematical definition of the DPC model (presented in section 3.3.1), this link was shown to be equivalent to a single plastic synaptic connection between the input and output neurons, meaning that the model does not allow hidden neurons. From the mathematical analysis presented in section 3.3.2, it is clear that hidden neurons are not required for the model to fulfil its original function of forming a minimal encoding of stimuli from a particular visual environment.

However, in this work we are interested in using DPC neural networks for a very different purpose: to identify novelty in datasets. In such scenarios, hidden neurons might prove to be

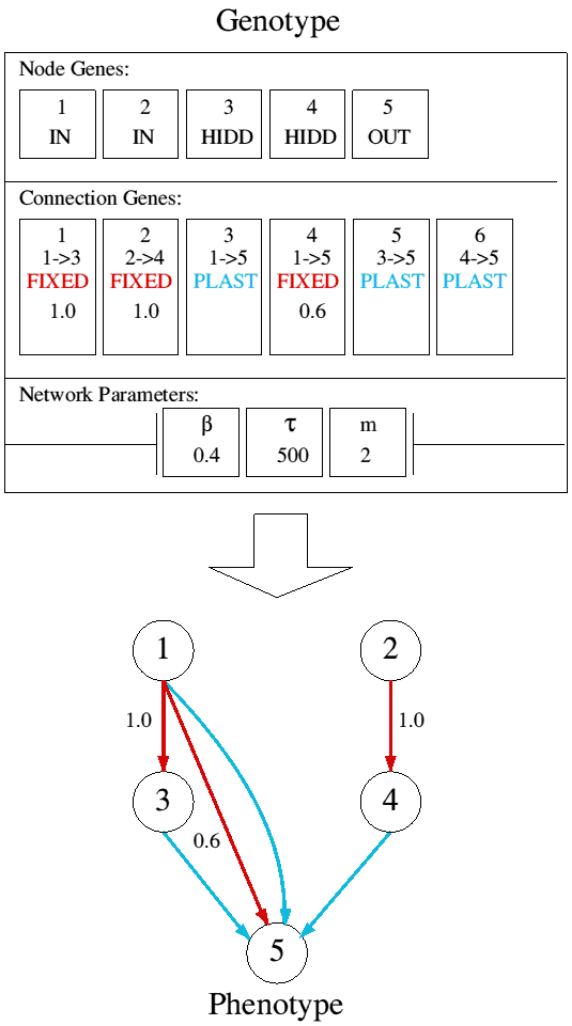


Figure 4.2: A DPC neural network with its corresponding genotype.

invaluable, enabling the network to construct a different representation of an input vector which may subsequently be easier to process. For example, an input which tends to dominate the network may be prescaled so as to reduce its magnitude before determining its correlational relationship with other inputs. This would then allow less dominant inputs to have a greater impact on the network's response, which may be useful in detecting subtle changes in the data. By giving DPC+NEAT the ability to introduce hidden neurons into a DPC neural network, we provide the approach with the building blocks necessary to construct DPC networks of any topology. This is important as it increases the likelihood that DPC+NEAT will be capable of constructing suitable network topologies in a wide range of different novelty detection scenarios.

In this chapter, the hidden neurons that DPC+NEAT may introduce into a network are defined with no activation function, i.e. their output is simply the linear summation of their

inputs. This follows from the definition of output neurons in the original DPC model described in chapter 3. However, as we explain in section 4.5.4, we do allow a threshold to be applied at each hidden neuron in the experiments conducted in chapter 5. This is particularly important in allowing the network to retain a nonlinear behaviour in scenarios where continuous learning is not appropriate.

An important consideration when using hidden neurons in a DPC network is the effect that their introduction will have on plastic synapses. If the inputs to a hidden neuron are provided by plastic synapses alone, then, when the network is in an unadapted state, the neuron will not receive any stimulation. This in turn would prevent the plastic synapses in question from adapting, since the term $\langle y_i x_j \rangle$ in the anti-Hebbian learning rule would always yield zero. Therefore, the substructure of the network which includes the hidden neuron, its input and output synapses, and any other hidden neurons connected in some way, would be redundant in that it would have no influence on the function of the network. To prevent this from occurring, we impose the restriction that each hidden neuron must receive input from at least one fixed synapse and so will be stimulated when the network is in an unadapted state.

To enforce this restriction, we made a modification to the add node mutation operation of NEAT. When a synapse between neurons a and b is split by the introduction of a new hidden neuron h, the new synapse between a and h is forced to have a fixed weight of 1. This agrees with the original implementation of the add node mutation used by NEAT [108] and guarantees that h will receive excitation from the presynaptic neuron a when the network is in an unadapted state. The synapse between h and b then inherits the connection type (and weight if applicable) of the synapse being replaced. We also prevent fixed synapses providing input to hidden or output neurons from being disabled during a crossover operation if this would result in violating the restriction above.

4.2.3 Work Required from the User

The aim of DPC+NEAT is to provide the user with an automated process which develops and configures a novelty detector for their particular novelty detection task. It should therefore make as few demands of the user as possible. However, information is still required on how the constructed novelty detector should interface with the outside world and how the performance of candidate novelty detectors should be judged. We now discuss these requirements in more detail.

The neural network evolved by DPC+NEAT must interface in some way with the data produced in the users novelty detection task. That is, the network must accept input vectors of a particular dimensionality, which defines the size of the input layer (i.e. each input node conveys a unique element of the input vector into the network for processing). Since the user is likely to be an expert in their novelty detection task, or at the very least understand the basic properties of the data involved, it is reasonable to assume that it is trivial for them to provide this information to DPC+NEAT. All networks evolved by DPC+NEAT provide a single output value, the decision on whether or not the corresponding input vector is novel, and so the output layer always contains a single neuron.

DPC+NEAT must also have some way of evaluating the performance of candidate novelty detectors in the user's novelty detection task. As discussed in chapter 3, evolutionary algorithms use a *fitness function* for this purpose. Therefore, DPC+NEAT requires that the user also define a suitable fitness function, which serves to inform DPC+NEAT about the novelty detection task. This may appear to be a non-trivial requirement, but it is actually necessary for the user to define such a quantitative performance measure regardless of the novelty detection technique that they use. For example, if the user chose to employ an existing technique taken from the literature, such as Marsland's Grow When Required (GWR) algorithm [85], then they would need to have some way in which they can (a) determine whether or not that approach is actually effective and (b) tune the relevant parameters to optimise its performance. As we demonstrate in chapter 5, the fitness function provided to DPC+NEAT need not be complex, a simple measure of the classification accuracy of the potential detector would suffice.

4.2.4 DPC+NEAT Parameters

One requirement of DPC+NEAT that a user cannot be reasonably expected to fulfil is the choosing of suitable values for the parameters of the approach. In addition to the original parameters given by Stanley [108], DPC+NEAT includes five new parameters: α, c_4, c_5, c_6 and m_P. These parameters, described in section 4.2.1, control the new functionality introduced to NEAT to allow the inclusion of plastic synapses and the evolution of DPC network parameters. We also introduce two new parameters in section 4.5, the DPC network parameter $dpcNT$ and DPC+NEAT parameter c_7, which are required for the novelty detection tasks in chapter 5. Including these additions, DPC+NEAT requires a total of 21 parameters to be set.

One of the drawbacks of NEAT that we identified in chapter 3 is the high dimensionality of

its parameter space, which we further increase with the addition of the new parameters described above. This drawback appears to cause DPC+NEAT to contradict the objective, as stated in chapter 1, of minimsing the work required from the user in configuring the approach for their particular scenario. Therefore, to cope with this drawback we make the assumption that the parameters governing the behaviour of DPC+NEAT are less tightly linked to the performance of a particular evolved DPC network than the parameters of that network itself. That is, we assume that, over a number of runs, a non-optimal evolutionary search is likely to be capable of constructing at least one novelty detector which achieves a good performance in the relevant task. We test this assumption by keeping the values of the parameters used by DPC+NEAT fixed throughout all experiments reported in this thesis, unless we wish to disable a particular feature of DPC+NEAT such as mutating DPC network parameters, in which case the relevent parameter(s) are set to zero.

In deciding suitable values for the original parameters of NEAT, we restricted ourselves to choosing one of the sets of parameters reported by Stanley in his Ph.D. thesis [108]. Each of these parameter sets have been shown to produce a good performance from NEAT in particular scenarios [108]. We made this choice based on the population size specified in the parameter set. We chose a population size that was small enough to minimise the time required for fitness evaluation in each generation, yet large enough to permit a reasonable level of variation between the individuals in the population. The parameter set we selected was that used by Stanley for his experiments with a real-time variant of NEAT (rtNEAT). This parameter set uses the second smallest population size out of all the sets reported, with the lowest being 16 (which we felt would be too small to ensure sufficient variation within the population). For the additional parameters that we introduced to DPC+NEAT, we set α such that each connection had an equal chance of having a fixed or plastic synaptic weight, and m_P so that networks had the same chance of having either their connection weights or network parameters perturbed. Due to the significant effect that varying the network parameters has on a particular DPC neural network, we set c_4, c_5 and c_6 to be equal to the parameters c_1 and c_2. Appendix A lists the values of all parameters used by DPC+NEAT in the experiments performed in this thesis.

4.3 Evaluating DPC+NEAT in a Test Scenario

In order to gain an indication of whether or not DPC+NEAT was likely to be a successful approach, we devised a test scenario based on the original DPC simulation discussed in chapter

3. In this scenario, we augmented the simulation with two new artificial visual environments, which each approximate a diagonal bar passing through the centre 2×2 pixels of the 4×4 pixel greyscale image. We found that, in the context of novelty detection, the performance of the illustrative neural network in this scenario had the potential to be optimised, making this an ideal test case for DPC+NEAT. In this section, we present this test scenario and the results obtained by DPC+NEAT in this evaluation.

4.3.1 Diagonal Environments

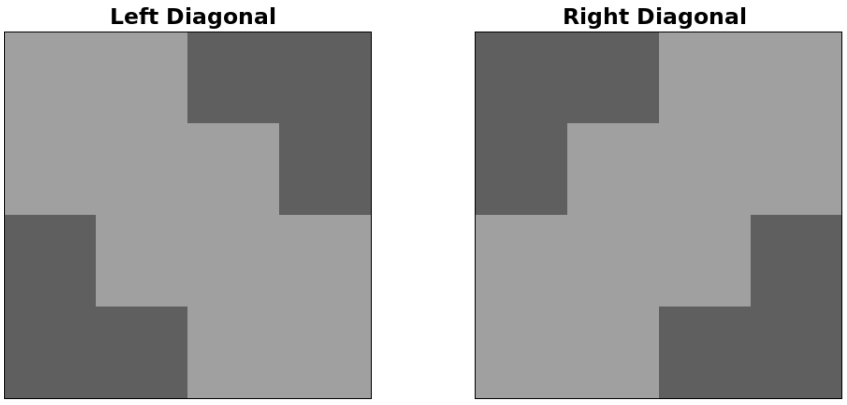


Figure 4.3: The new diagonal environments to be used in the test scenario.

b	b	c	c
b	a	a	c
c	a	a	b
c	c	b	b

Figure 4.4: The structure of the diagonal environments.

The two new diagonal environments that we introduced in this test scenario are shown in figure 4.3. In each environment, the image is expressed using three random variables, $a(t)$, $b(t)$ and $c(t)$, as shown in figure 4.4. For the left diagonal environment, the centre tiles $a(t)$ perfectly correlate with the upper-left and bottom-right tiles $b(t)$ and perfectly anticorrelate with the upper-right and bottom-left tiles $c(t)$:

$$a(t) = b(t) = -c(t) = n_u(t) \tag{4.2}$$

Conversely, for the right diagonal environment, the centre tiles correlate with the upper-right and bottom-left tiles and anticorrelate with upper-left and bottom-right tiles, as shown in equation 4.3:

$$a(t) = -b(t) = c(t) = n_u(t) \tag{4.3}$$

These diagonal environments are interesting as they pose two new challenges to the illustrative DPC neural network presented in chapter 3. First, they bear a stronger similarity to each of the adaptable environments, which is exploited by the network to formulate a partial prediction of one environment when adapted to the other. Second, the peripheral relationships between the two diagonal environments are direct inverses of each other, which causes the output of the network, when adapted to one diagonal environment, to be greater than the sum of the 2×2 centre pixels when the other diagonal environment is applied. We now explore these two challenges in more detail.

Between any of the original adaptable environments discussed in chapter 3 and either of the diagonal environments, 10 pixels maintain their correlational relationship with the centre pixels as opposed to 8 pixels. From equation 3.25 in section 3.4.5, we observe that the activity across the plastic synapses will therefore no longer cancel out but instead continue to inhibit the output neuron, since:

$$\begin{aligned} \mathbf{Ax} &= dap - sap && (4.4)\\ &= 6ap - 10ap && (4.5)\\ &= -4ap && (4.6) \end{aligned}$$

where a is the adapted weight of each plastic synapse and p is the intensity of the centre pixels. This results in the sensitivity of the network to the diagonal environments falling below 1 when adapted to any other adaptable environment.

Between the two diagonal environments, the peripheral relationships with the centre pixels are direct inverses of each other, for example the upper-left and lower-right corners of the image correlate with the centre pixels in the left diagonal environment but anticorrelate in

the right diagonal environment. This property is not exhibited between any of the original adaptable environments. Again, from equation 3.25 we find that, when stimuli from one diagonal environment is applied to the network when adapted to the other, the contribution of the plastic synapses on the periphery is excitatory rather than inhibitory:

$$\mathbf{Ax} = 12ap - 4ap \tag{4.7}$$

$$= 8ap \tag{4.8}$$

-a	-a	a	a
-a	b-a	b-a	a
a	b-a	b-a	-a
a	a	-a	-a

Figure 4.5: The receptive field of the output neuron of the illustrative neural network model when adapted to the left diagonal environment. Since each pixel is $\pm n$, each plastic synapse converges to the synaptic weight of $\pm a$. Here, b represents the weight of the fixed synapses, which is 1 for each synapse.

-n	-n	n	n
-n	n	n	n
n	n	n	-n
n	n	-n	-n

x

-a	-a	a	a
-a	b-a	b-a	a
a	b-a	b-a	-a
a	a	-a	-a

=

an	an	an	an
an	(b-a)n	(b-a)n	an
an	(b-a)n	(b-a)n	an
an	an	an	an

Figure 4.6: The result of applying stimuli from the right diagonal environment to a network adapted to the left diagonal environment. All inputs on the periphery are enhanced, resulting in the output of the network being larger than the sum of the centre pixels.

This is illustrated in figures 4.5 and 4.6. Figure 4.5 shows the receptive field of the output neuron when the network is adapted to the left diagonal environment. When applying stimuli

from the right diagonal environment, those pixels on the periphery of the image which previously correlated with the centre pixels now anticorrelate, and vice-versa. Thus, the synapses which previously enhanced anticorrelating pixels now enhance the correlating pixels. Because of the cancellation of sign, those synapses which previously suppressed correlating pixels also now enhance the anticorrelating pixels, as shown in figure 4.6. The changes in peripheral correlational relationships between the two diagonal environments results in the input across the plastic synapses contributing to the magnitude of the sum across the fixed synapses, thus resulting in the output of the network being greater than the sum of the centre pixels.

4.3.2 Performance of the Illustrative Neural Network

To further illustrate the findings above, we visually inspected the performance of the illustrative neural network in this test scenario through simulation. Each of the artificial visual environments discussed in chapter 3, and the new diagonal environments introduced in this chapter, were shown to the network for 2,500 time steps. The parameter u, controlling the rate of flicker for each environment, was again set to 1. This configuration was used in all simulations of this test scenario. As in chapter 3, we analysed the performance of DPC networks in this scenario by observing how the networks sensitivity to each adaptable visual environment varied during the time course of the simulation. The sensitivity measure used, defined in section 3.4.3, is a function of the variance of the output neuron of the network when stimuli from a particular environment is applied to the network. At regular intervals during the simulation, the network weights were frozen and the variance of the network to each adaptable environment derived. Again, maximum sensitivity is defined as that reached by the network to the adaptable environments when in an unadapted state. Sensitivity values are normalised to the range $[0, 1]$, assuming this definition of maximum sensitivity.

Figure 4.7 shows how the network's sensitivity to the adaptable environments varied through the simulation. As the network adapted to the Uniform, Checker, Vertical and Horizontal environments from chapter 3, its sensitivity to the diagonal environments fell to approximately 0.75. This is the result of the network exploiting similarities between the environments to formulate a partial prediction. As the network adapted to one diagonal environment, its sensitivity to the other diagonal environment rose above maximum, resulting from the fact that the peripheral relationships are inverted between the two diagonal environments.

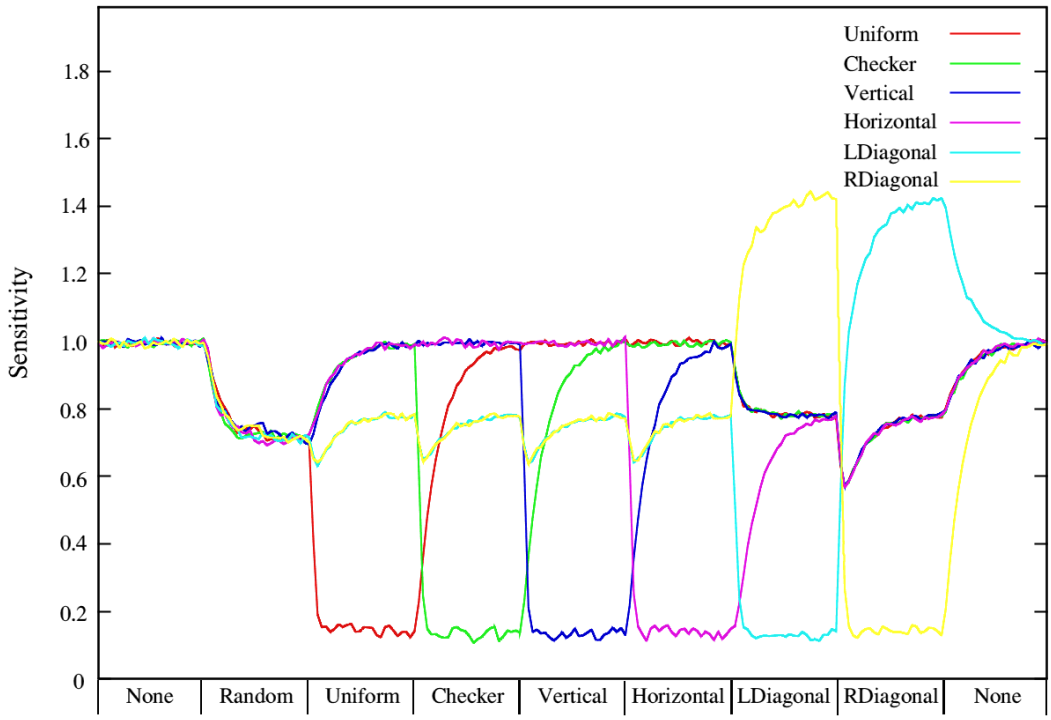


Figure 4.7: The sensitivity graph produced by the illustrative neural network in the test scenario. In addition to the visual environments from chapter 3, this graph shows the network's response to the new left diagonal (LDiagonal) and right diagonal (RDiagonal) environments.

4.3.3 Optimising Performance

In the context of predictive coding, the challenges posed by the diagonal environments in this test scenario reveal both desirable and undesirable behaviour from the illustrative neural network. The ability of the network to exploit similarities between two environments is highly desirable, since it allows a more efficient encoding of information. However, the second challenge posed by the diagonal environments demonstrates a case in which the assumptions implicitly made in the design of the network are invalidated, resulting in a poor response. To improve its performance in this case, the assumptions informing the design of the network need to be modified, meaning that a new network topology is required.

However, novelty detection requires a different behaviour from the network to predictive coding. To ensure correct classification, it is desirable to maximise the separation between known and novel environments. Therefore, the network's reduced sensitivity to the diagonal environments when adapted to a non-diagonal environment is not ideal. However, it could be argued that the network is simply identifying that the degree of novelty is reduced between unknown diagonal and known non-diagonal environments, because of their increased similarity.

Other novelty detectors, such as Marsland's GWR [85], offer such a quantitative measure of novelty instead of a binary decision value. But all environments are similar in some way. For example, the Vertical and Horizontal environments are similar, in that one is simply a rotation of the other, yet the network finds the Horizontal environment completely novel when it is adapted to the Vertical environment and vice-versa. The network also cannot justifiably consider the unknown diagonal environment more novel than an unknown checkerboard or vertical bar environment when adapted to a diagonal environment. These inconsistencies in the network's behaviour therefore discount this argument, and the network thus also has potential to be optimised from the perspective of novelty detection.

Since the performance of the illustrative neural network in this test scenario has the potential to be optimised, it enables us to test the effectiveness of DPC+NEAT. In order to allow DPC+NEAT to perform the optimisation, we must decide on a criterion upon which the performance of a network can be measured. The obvious choice is to base this criterion on the separation that the network achieves between adapted and novel environments, i.e. to assign a higher performance value to those networks which achieve the greatest separation. However, this objective could be trivially accomplished by simply increasing the sensitivity that the network has to novel environments without bound. Therefore, to ensure that this problem is sufficiently challenging, we impose a second requirement on the behaviour of the network: in addition to maximising the separation between adapted and novel environments, the network should not permit its sensitivity to any of the adaptable environments to exceed maximum sensitivity, which we denote here as S_{max}. As previously mentioned, S_{max} is defined *a priori* to be the sensitivity of the original illustrative neural network to the adaptable environments when in an unadapted state. As well as making the problem more challenging, including this second requirement also guarantees that DPC+NEAT will need to make changes to the assumptions implicit in the network's design. Therefore, we require that, when adapted to a particular environment, the ideal network demonstrates insensitivity to that environment and maximum sensitivity to all other environments.

Based on the above discussion, we now formally define a suitable performance criteria. This formal definition is expressed in terms of a fitness function that can be used by DPC+NEAT in this test scenario. We consider the N states of the network which are obtained by allowing it to adapt to the N adaptable environments in the simulation. In each state, we call the environment to which the network is adapted the *known environment* and the remaining adaptable environments the *novel environments*. The principal goal is then to find a network which, for

each state, maximises the separation between the known and novel environments. We therefore define the fitness $F(c)$ of candidate network c to be the sum of the differences between its averaged sensitivity to the novel environments and its sensitivity to the known environment in each state. The mathematical definition of $F(c)$ is given by:

$$F(c) = \sum_{i=1}^{N} \left[\frac{1}{N-1} \left(\sum_{j=1,\ j\neq i}^{N} S_c(i,j) \right) - S_c(i,i) \right] \tag{4.9}$$

where $S_c(i,j)$ denotes the normalised sensitivity of candidate network c to environment j when adapted to environment i. This is measured on the last time step in the simulation that stimulus from environment i is applied to the network. As described in section 3.4.3, all sensitivity values are again normalised to the range $[0,1]$, assuming a maximum sensitivity value of S_{max}. We also require that $S_c(i,j)$ respects the upper bound defined by S_{max}, which should not be exceeded when the network is in an adapted state. Therefore, candidates which allow sensitivity to any environment to rise above 1 (the normalised value of S_{max}) should be punished. However, such candidates should not simply be awarded a fitness of zero as they may yet evolve into good solutions. Also, sensitivity values which only slightly exceed 1 should be awarded a higher fitness than those which are significantly larger than 1. From experimentation, we found that the following nonlinear adjustment applied to $S_c(i,j)$ gave the best results:

$$S_c(i,j) = \begin{cases} 2.0 - S_c(i,j)^4 & S_c(i,j) > 1.0 \\ S_c(i,j) & \text{otherwise} \end{cases} \tag{4.10}$$

After this adjustment, $S_c(i,j)$ may be negative. However, the lowest fitness the network can achieve when adapted to a single environment i is constrained to 0. Thus, a network performing badly when adapted to one environment but not when adapted to another is punished for its poor performance only.

When adapted to environment i, the ideal DPC network should have a sensitivity of 1 to each environment j, where $j \neq i$, and a sensitivity of 0 to environment i. Thus, the highest fitness a network may achieve (i.e. the fitness of the ideal network) is:

$$F(c)_{max} = \sum_{i=1}^{N} \left[\frac{1}{N-1} \left(\sum_{j=1,\ j\neq i}^{N} 1 \right) - 0 \right] \tag{4.11}$$

$$= N \tag{4.12}$$

Since we have 6 adaptable environments, $F(c)_{max} = 6$ in this test scenario. According to this fitness measure, the illustrative neural network achieves a fitness of 3.28 in this test scenario. This defines a baseline measure of performance which we expect the networks evolved by DPC+NEAT to improve upon.

4.3.4 Method

In this test scenario, we wish to determine whether or not DPC+NEAT is able to optimise the performance of the illustrative neural network according to the criteria presented above. However, we also wish to establish the extent to which allowing DPC+NEAT to evolve network parameters, in addition to modifying neural network structure and weights, impacts on the results obtained by DPC+NEAT. Whilst evolving network parameters provides a significant advantage, in that the user is no longer required to find suitable values themselves, are there any further advantages, or any disadvantages, to this strategy?

To try to answer this question, we performed two experiments. In the first experiment, 10 runs of DPC+NEAT were performed with the parameter m_P set to zero. This allowed us to see by how much DPC+NEAT could improve the performance of the illustrative neural network when modifying its structure and weights alone. In the second experiment, 10 runs were again performed but this time with m_P set to 0.8, the value given in appendix A. The results from both experiments were then compared, in terms of the average performance obtained.

We now consider how the fitness of candidate DPC networks are evaluated in these experiments. During a simulation of a particular DPC network, the sensitivity of that network to an environment is calculated based on the sample variance of the network's output when that environment is applied. However, this demands a relatively long runtime, approximately 2 minutes for the illustrative network[1]. Therefore, in order to reduce the time required to perform the evolutionary search, the sensitivity of each candidate was instead derived from the known population variance of the input stimuli using the following method. In this test scenario, each input vector $\mathbf{x}$ presented to a DPC network is based on a single random variable n, which has a Gaussian distribution with zero mean and unit variance. Each element of the input vector scales n by either $+1$ or -1. Thus, an l-dimensional input vector $\mathbf{x}_e$ for environment e may be represented as:

[1]Based on an average over 10 runs, a simulation of the original DPC network on a PC with an AMD64 3700+ 2.2GHz processor, 1GB of RAM, and a 64-bit Linux operating system required approximately 2 minutes to complete

$$\mathbf{x}_e = \mathbf{m}_e n \tag{4.13}$$

$$= \begin{bmatrix} m_{e,1} \\ m_{e,2} \\ ... \\ m_{e,l} \end{bmatrix} n \tag{4.14}$$

where $\forall e, i : m_{e,i} \in \{+1, -1\}$. Then, $\mathbf{m}_e$ may be thought of as a *mask* for environment e, transforming n into a stimulus vector $\mathbf{x}_e$ for that environment. Since the neurons of a DPC network simply output the linear summation of their inputs, the network can be thought of as performing a linearly weighted summation of the elements of the input vector $\mathbf{x}$. Thus, for input stimuli from environment e:

$$y_e = \mathbf{q}^T \mathbf{x}_e \tag{4.15}$$

$$= \mathbf{q}^T (\mathbf{m}_e n) \tag{4.16}$$

$$= n \sum_{i=1}^{l} q_i m_{e,i} \tag{4.17}$$

where $\mathbf{q}$ is an l-dimensional vector representing the weights q_i assigned to each x_i in the summation. Since $var(ax + b) = a^2 var(x)$ and $var(n) = 1$, the variance of y_e may be then derived as:

$$var(y_e) = var\left(n \sum_{i=1}^{l} q_i m_{e,i}\right) \tag{4.18}$$

$$= \left(\sum_{i=1}^{l} q_i m_{e,i}\right)^2 var(n) \tag{4.19}$$

$$= \left(\sum_{i=1}^{l} q_i m_{e,i}\right)^2 \tag{4.20}$$

Therefore, the variance of the output of an adapted network when stimuli from a particular environment e is applied may be derived using the environment mask of e alone. Sensitivity of the network to environment e may then be calculated as:

$$S_e = \sqrt{var(y_e)} \tag{4.21}$$

$$= \sum_{i=1}^{l} q_i m_{e,i} \tag{4.22}$$

The fitness of a candidate network is therefore derived by first adapting the network to each of the adaptable environments in turn, and then, for each adapted state, deriving the sensitivity of the network to each adaptable environment using equation 4.21.

At the end of each run, the best network found by DPC+NEAT was subjected to a full simulation, with sensitivity based once again on sample variance. We used this simulation to obtain a sensitivity graph for the network, which provided us with a visual account of performance as well as allowing us to verify the assigned fitness value and identify any implementation errors.

4.3.5 Results

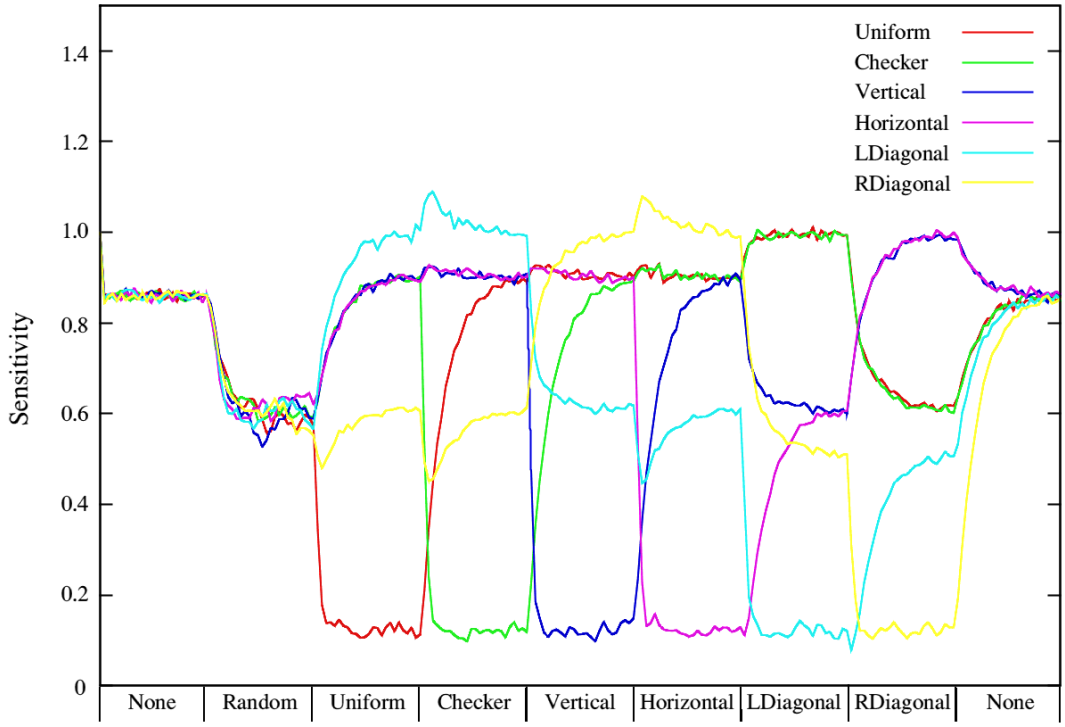


Figure 4.8: The sensitivity graph produced by the best performing network found by DPC+NEAT in the first experiment ($m_P = 0.0$).

We now present the results obtained by DPC+NEAT in this test scenario. In the first experiment, with m_P set to zero, the ten best networks (one for each run) evolved by DPC+NEAT achieved an average fitness of 4.169 (s.d. 0.032), with the best and worst performing networks achieving a fitness of 4.208 and 4.096 respectively. The sensitivity graph produced by the best performing network is shown in figure 4.8. For most novel environments, the network's sensitivity remained at a similar level throughout the simulation. Unlike the original illustrative network, this evolved network did not show a dramatic increase in sensitivity to one diagonal environment when it was adapted to the other diagonal environment. However, a drop in sensitivity to one diagonal environment was still observed when the network was adapted to each of the four non-diagonal environments. The network's sensitivity to multiple environments also fell when it was adapted to either of the diagonal environments. Intriguingly, the sensitivity of the network to all environments when in an unadapted state (i.e. when the None environment was applied) reached the lower-than-expected level of approximately 0.86.

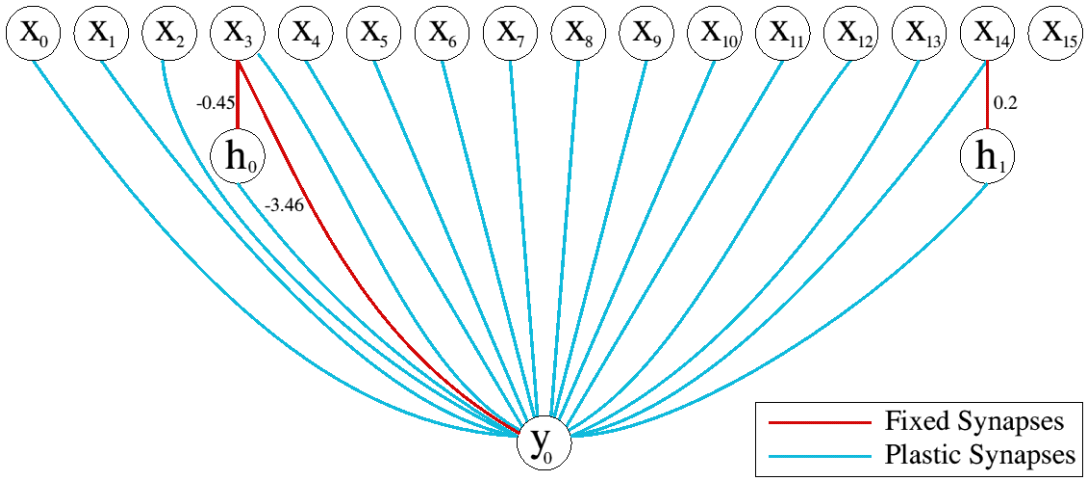


Figure 4.9: The structure of the best performing network found by NEAT in the first experiment. The values next to the fixed synapses give the weight of those synapses.

The topology of the best performing network in the first experiment is shown in figure 4.9. Two hidden neurons have been introduced, which prescale the signals from selected input neurons before allowing the network to use those signals to formulate a prediction. However, these appeared to have only a marginal influence on the output neuron during the simulation. Interestingly, the neurons receiving input from the centre pixels are no longer connected to the output neuron via fixed synapses. Instead, only the upper-right corner peripheral input is connected to the output neuron, with a fixed synapse of weight -3.46. The decreased magnitude of this weight explains the lower-than-expected sensitivity observed for each environment when

the network was in an unadapted state. When adapting to a particular environment, the plastic synapses of the network are modulated so as to suppress the magnitude of the scaled inverse value of the upper-right corner pixel, which we denote here as c. For the left-diagonal environment, $c = -3.46(-n) = 3.46n$ approximates the sum of the centre pixels ($4n$). Therefore, the plastic synapses adapt to this environment in a similar manner to that seen in the illustrative network. However, in the right-diagonal environment, $c = -3.46n$ approximates the inverse of the sum of the centre pixels ($-4n$). When stimuli from the right-diagonal environment is applied to the adapted network, the net excitatory response generated by the plastic synapses serves to reduce the magnitude of c. This causes the network to have a lower sensitivity to the right-diagonal environment when adapted to the left-diagonal environment. When the network is adapted to either Uniform or Checker, c approximates the inverse of the sum of the centre pixels. Here, the network continues to partially suppress stimuli from the right-diagonal environment, since the plastic synaptic weights again fail to cancel themselves out. However, this residual activity from the plastic synapses partially *enhances* stimuli from the left-diagonal environment, where c approximates the actual sum of the centre pixels. The disconnection of the input in the lower-right corner of the stimulus appears to prevent the network's sensitivity to the left-diagonal environment from exceeding maximum in this case. Similar behaviour can also be observed for stimuli from the right-diagonal environment when the network is adapted to either Vertical or Horizontal.

We next performed ten runs of DPC+NEAT where both the structure and parameters of the network were simultaneously evolved ($m_P = 0.8$). The average fitness of the ten best networks (one for each run) evolved by DPC+NEAT was 4.635 (s.d. 0.129), with the best and worst performing networks achieving a fitness of 4.754 and 4.367 respectively. Comparing the average fitness of 4.635 with that obtained in the first set of runs (4.169), we observe that, in this scenario, allowing DPC+NEAT to mutate parameters in addition to structure appears to have a positive impact on its performance. The sensitivity graph produced by the best performing network is shown in figure 4.10. Again, the sensitivity of this network to one diagonal environment did not rise above maximum when the network was adapted to the second diagonal environment. But, the application of a new visual environment sometimes caused the network's sensitivity to certain environments to temporarily rise above maximum. However, as the network continued to adapt to the environment, these sensitivities converged back to maximum. Also, when adapted to the original four adaptable environments this network showed an increased separation between its sensitivity to the adapted environment and its sensitivity to each of the other adaptable

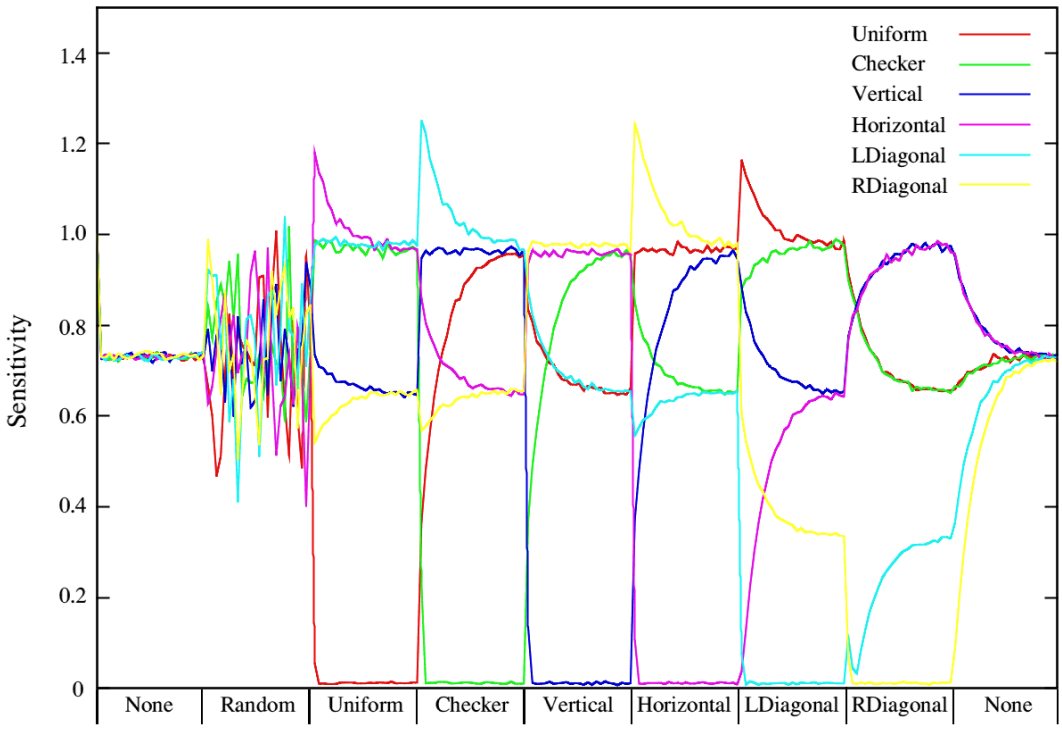


Figure 4.10: The sensitivity graph produced by the best performing network found by NEAT in the second experiment ($m_P = 0.8$).

environments. It accomplished this by increasing the β parameter from 0.4 to 7.658. However, this also resulted in the network becoming much more sensitive to the Random environment. The remaining network parameters τ and m showed little change, with τ being assigned the value 496.282 and m retaining its original value of 2.

The topology of the best performing network found by DPC+NEAT in the second experiment is shown in figure 4.11. In this network, three input neurons have been completely disconnected, indicating little advantage in using the information that they provide. Again, stimuli from the centre pixels pass along only plastic synaptic connections to the output neuron. Four hidden neurons have been introduced, but one has been completely disconnected from the output neuron and therefore has no influence on the behaviour of the network. The remaining hidden neurons were again found to have a minimal influence on the output neuron in the simulation. A single fixed synapse again connects a periphery input to the output neuron, which is used (in the same way as in the best performing network from the first experiment) to prevent sensitivity to diagonal environments from exceeding S_{max}. However, unlike the network from the first experiment, this network demonstrated a lower sensitivity to two environments when adapted

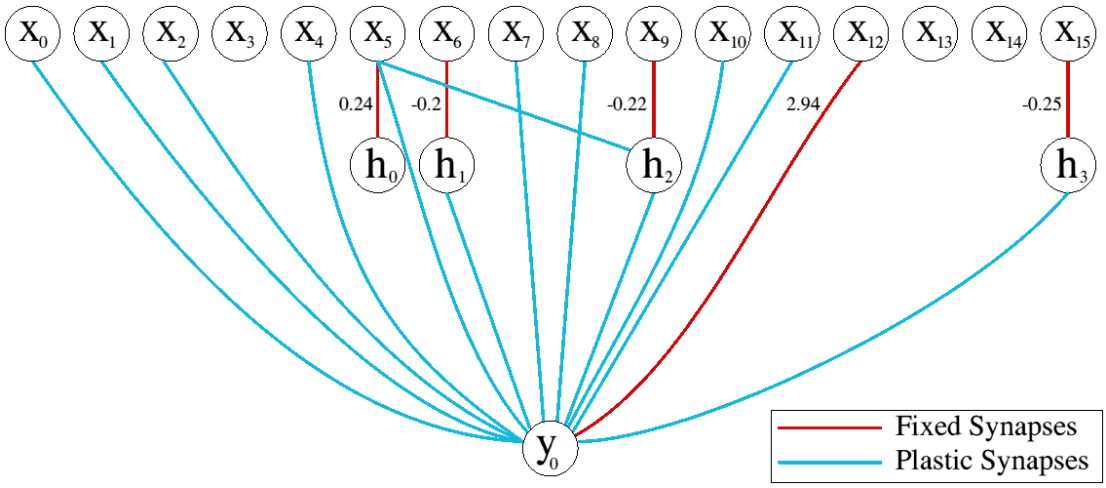


Figure 4.11: The structure of the best performing network found by NEAT in the second experiment. The values next to the fixed synapses give the weight of those synapses.

to each non-diagonal environment. Whilst the network from the first experiment attempted to improve the separability between known and novel environments by increasing its sensitivity to all but one of the novel environments, this network demonstrated a decreased sensitivity to the known environment. This was accomplished not through structural mutation but instead by modifying the network parameter β. This in turn resulted in a greater average separation between known and novel environments than that accomplished by the best performing network in the first experiment, and thus a higher fitness.

4.3.6 Discussion

DPC+NEAT has been shown to successfully optimise the original illustrative neural network according to our predefined performance criteria. The average fitness value of 4.635 given by the 10 best networks evolved in the second experiment is significantly higher than that given by the original network. The best performing networks evolved in the two experiments each attempt to tackle both of the challenges posed to the original illustrative network by this test scenario. They both successfully avoid becoming oversensitive to one diagonal environment when adapted to the other diagonal environment. The networks also demonstrate, when adapted to each non-diagonal environment, an increased separation between the known and novel environments, with each network tackling this challenge in a different way. The best performing network from the first experiment, forced to rely solely on structural modification to achieve this goal, increases the sensitivity reached by all but one of the novel environments when adapted to each non-diagonal

environment. However, in the second experiment the best performing network decreases the minimum sensitivity to each known environment, as well as slightly increasing its sensitivity to all but two of the novel environments. The decrease in sensitivity to each known environment was achieved by increasing the network parameter β. By allowing network parameters to be evolved in addition to structure, we enable NEAT to explore a wider range of possible ways in which the problem at hand may be solved.

The best performing networks in the two experiments were found to have a complexity similar to that of the original DPC network. In each network, multiple hidden neurons were introduced but these were found to have a minimal influence on the output neuron. Instead, the key advantages were gained by moving the focus of each network, defined by the fixed synapses, away from the centre pixels and into the periphery, as well as disconnecting peripheral inputs. The second of these networks also showed a degree of structural redundancy in that one of its hidden neurons was completely detached from the output neuron. This disconnection occurred as a result of the corresponding connection gene between these two neurons becoming *disabled*. Similarly, a number of input neurons were also disconnected, demonstrating that NEAT is capable of discarding inputs which fail to provide an evolutionary advantage. Therefore, despite possibly resulting in some structural redundancy, NEAT's ability to disable connection genes is important as it permits a degree of feature selection to take place.

The minimally effective hidden neurons in both networks, as well as the redundant structure present in the second network, appear to undermine the argument that NEAT tends to evolve solutions of minimal complexity [108]. However, neither network suffers from excessive amounts of redundant or unnecessary structure. In the first network, it may be that the inclusion of the hidden neurons has resulted in a slight, but insignificant, increase in fitness. NEAT interprets any amount of increase in performance as justification for retaining new structure. It may also be true that, whilst they may have been effective at some stage during the evolutionary search, the inclusion of these nodes now provides neither an advantage nor a disadvantage to the network, meaning that there was simply no motivation for NEAT to remove them. In the second network, it is likely that the redundant structure relating to the disconnected hidden neuron was found to be a disadvantage, after modification of other parts of the network. By disconnecting the hidden neuron in this manner NEAT effectively deactivated this structure but retained it in case it proved to be useful once again at some future stage of the evolutionary search. Whilst the two networks considered here clearly do not achieve an optimally minimal structural complexity, they are not significantly overcomplicated either. We examine this issue

of complexity again in chapter 5, when we use DPC+NEAT to evolve networks for different novelty detection scenarios.

Overall, the results obtained in this test scenario indicate that DPC+NEAT has the potential to be a promising approach. However, some important questions still remain to be answered. First, how sensitive are the evolved networks to various levels of noise introduced to the network inputs? This is important since unpredictable levels of noise are typically present in many real-world scenarios, and we attempt to answer this question in the next section. Second, how should DPC networks be employed in novelty detection tasks? For example, whilst a sensitivity measure is appropriate in the context of differentiating between known and novel artificial visual environments, it would not be suitable for identifying the novelty of a single observation. We address this question in section 4.5.

4.4 Response to Noise

In the vast majority of datasets collected from real-world scenarios, the data is likely to contain a noise component. Therefore, it is important to determine how the presence of noise affects the performance of a DPC neural network. To investigate this, we extended the test scenario from the previous section to consider the introduction of Gaussian noise. For each of the visual environments, apart from Random, we introduced an additive noise component, drawn independently for each pixel from a standard Gaussian distribution ($\mu = 0$, $\sigma = 1$) and scaled by a noise control parameter k. We then observed how increasing noise impacted on the fitness achieved by both the original illustrative neural network and the two best performing networks evolved by DPC+NEAT in the experiments conducted in section 4.3. Finally, we examined the effect of noise on DPC+NEAT itself by introducing noise into the stimuli used by the fitness function to assess performance.

4.4.1 Impact of Noise on Original and Evolved Networks

Figure 4.12 shows how the performances given by the three networks considered here varies with increasing noise. For low levels of noise, the performances of both evolved networks are higher than that of the original network. However, as k increases, the performance of the best performing network from the second experiment conducted in section 4.3, where mutating of the network parameters was permitted, quickly degrades, reaching a fitness of 0 at $k = 1.2$. The performance of the best performing network from the first experiment, without mutating

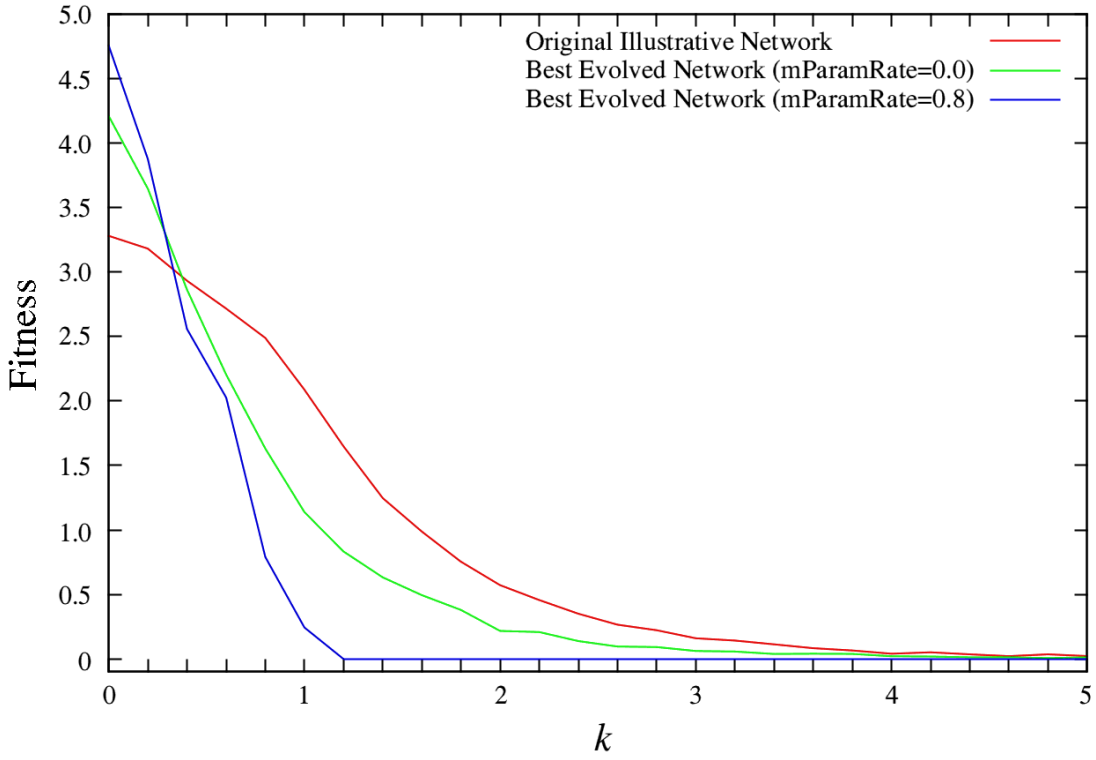
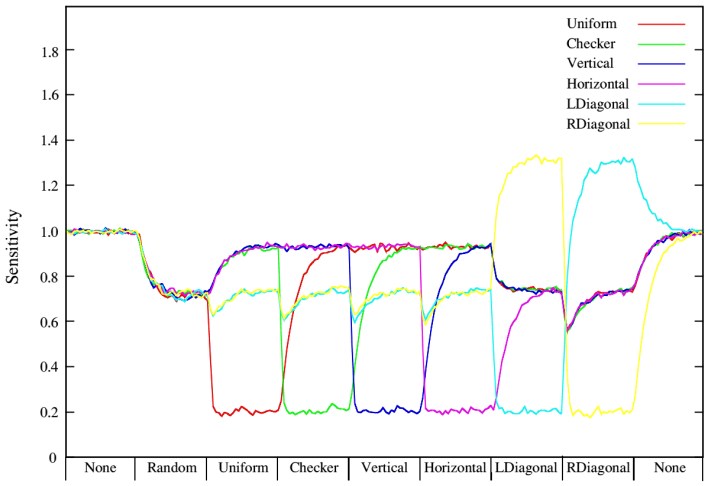


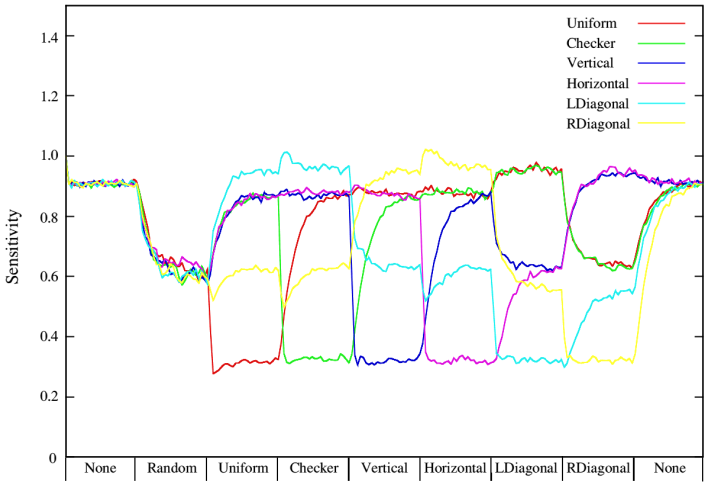
Figure 4.12: The performance achieved by the original illustrative neural network and the best performing networks from the two previous experiments as increasing levels of noise are applied.

network parameters, shows more resistance to noise, reaching a fitness of 0.833 at $k = 1.2$ and a fitness of 0.012 at $k = 5.0$. For $k > 0.4$, the original DPC network outperforms both evolved networks. Tentatively, we found that networks with parameters and topology close to that of the original network tended to give a better performance in the face of increasing noise.

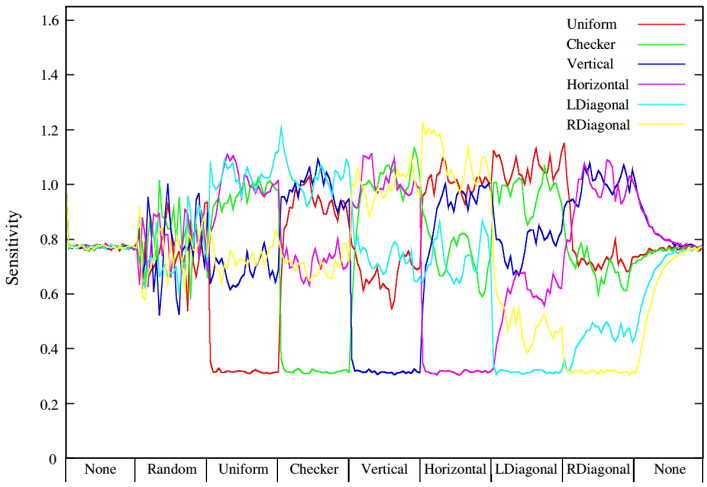
To gain further insight into the effect of Gaussian noise on these three neural networks, we examined the sensitivity graph produced by each network for $k = 0.4$. For this level of noise, the fitness of all three networks lies in the range $[2.5, 3.0]$. The sensitivity graphs produced for the networks are shown in figure 4.13. Contrasting these with the graphs produced by the networks when no noise is present, shown in figures 4.7, 4.8 and 4.10, a number of common differences can be identified. In each case, the minimum sensitivity reached by the network increases in the presence of noise, with the maximum sensitivity seen decreasing slightly for both the original network and the best performing network found by DPC+NEAT without mutating parameters. In the case of the best performing network evolved by DPC+NEAT with mutating parameters, its sensitivity to each novel environment becomes more unstable, frequently exceeding the maximum sensitivity value of 1, which results in a severe degradation



(a)



(b)



(c)

Figure 4.13: The sensitivity graphs produced for $k = 0.4$ by (a) the original illustrative network, (b) the best performing network evolved by DPC+NEAT with $m_P = 0.0$ and (c) the best performing network evolved by DPC+NEAT with $m_P = 0.8$.

in its fitness. This can be explained by the larger value of β used by this network. Whilst such a value allows the network to achieve an excellent performance in the absence of noise, it also makes it more sensitive to the interference caused by the noise. This was also demonstrated in the sensitivity graph shown in figure 4.10, where the Random environment induced erratic changes in the network's sensitivity to the adaptable environments.

To explain why increasing noise results in an increase in the minimum sensitivity and a decrease in the maximum sensitivity reached by the networks, we examined the behaviour of the weights of the plastic synapses in the original network in two simulations, each with a different level of noise. In the first simulation, with $k = 0.0$ (i.e. no noise), we found that after each adaptable environment was shown to the network, all plastic synaptic weights converged to approximately ± 0.22, with the sign of each weight dependent on the correlation between the corresponding pixel and the centre pixels. However, in the second simulation, with $k = 1.0$, the weights instead split into two groups. The weights of the plastic synapses connecting peripheral inputs converged to oscillate in the approximate ranges $[-0.17, -0.12]$ and $[0.12, 0.17]$. However, the weights of the plastic synapses connecting the centre inputs converged to oscillate in the approximate range $[-0.45, -0.40]$. This can be explained by the decreased strength of the correlation between peripheral and centre inputs. With any amount of noise, each centre pixel will have a stronger correlational relationship with the sum of all centre pixels, since its value directly contributes to this sum. Therefore, the corresponding plastic synapses are able to habituate more strongly than those connected to the periphery. The strong inhibition applied at the centre reduces the sensitivity of the adapted network to novel environments, since it cannot be completely cancelled out by the activity at the periphery, thus explaining the decrease in maximum sensitivity. The net effect of the reduction in correlation strength is to reduce the overall inhibition applied to the output neuron, and therefore increase the minimum level of sensitivity reached. Since the evolved networks are based on the original illustrative network, they behave in a similar manner.

4.4.2 Evolving Networks with Noisy Environments

As we have shown, the best performing networks found by DPC+NEAT in section 4.3 are less robust to noise than the original illustrative neural network. This is especially true when the network parameters are also determined by DPC+NEAT. However, if noise is applied to the stimuli used by the fitness function during the evolutionary search, it is reasonable to assume that

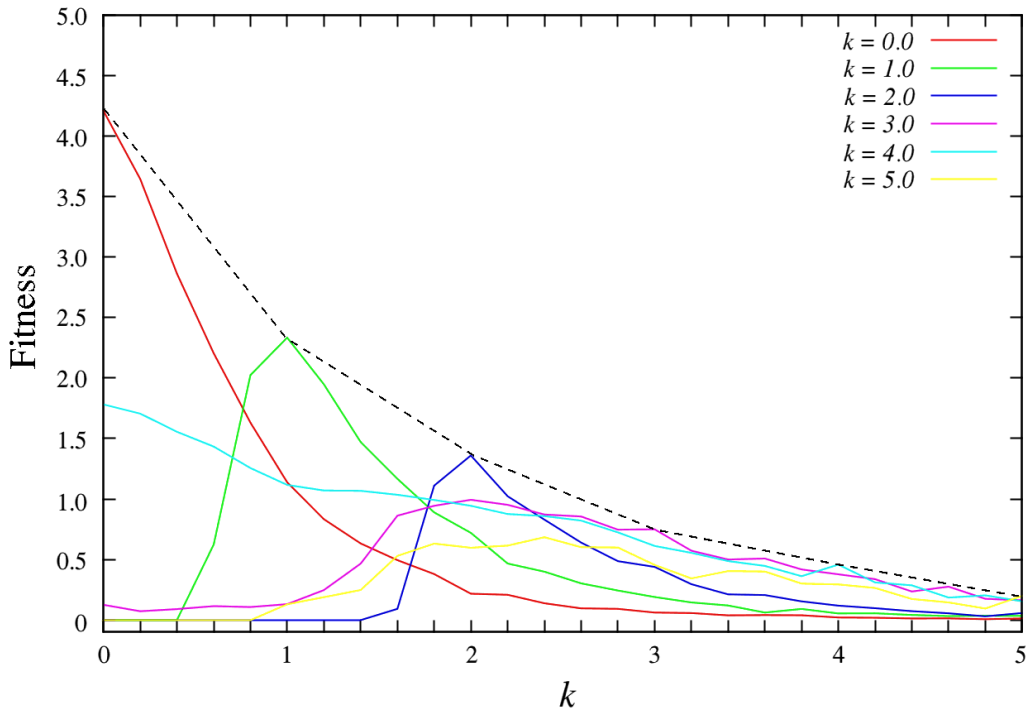
the evolved network should then be optimised for that level of noise. To test this assumption, we investigated whether or not DPC networks could be found which gave a better performance for a particular level of noise than the original network.

From initial experimentation, we found that it was necessary to first amend the fitness function such that the term $S_c(i,i)$ had no adjustment applied if it yielded a value greater than 1. Without this amendment, DPC+NEAT evolved non-adaptive networks whose sensitivity to all environments was greater than 1, with the sensitivity to the adapted environment being sufficiently high so as to force the adjustment term $2 - S_c(i,i)^4$ to be negative. Since $S_c(i,i)$ is subtracted from the average sensitivity of the network to the novel environments in equation 4.9, this resulted in a positive increase in fitness which in turn led to networks being assigned deceptively high fitness values. Therefore, to avoid this from happening, the adjustment applied to $S_c(i,j)$ was amended such that:

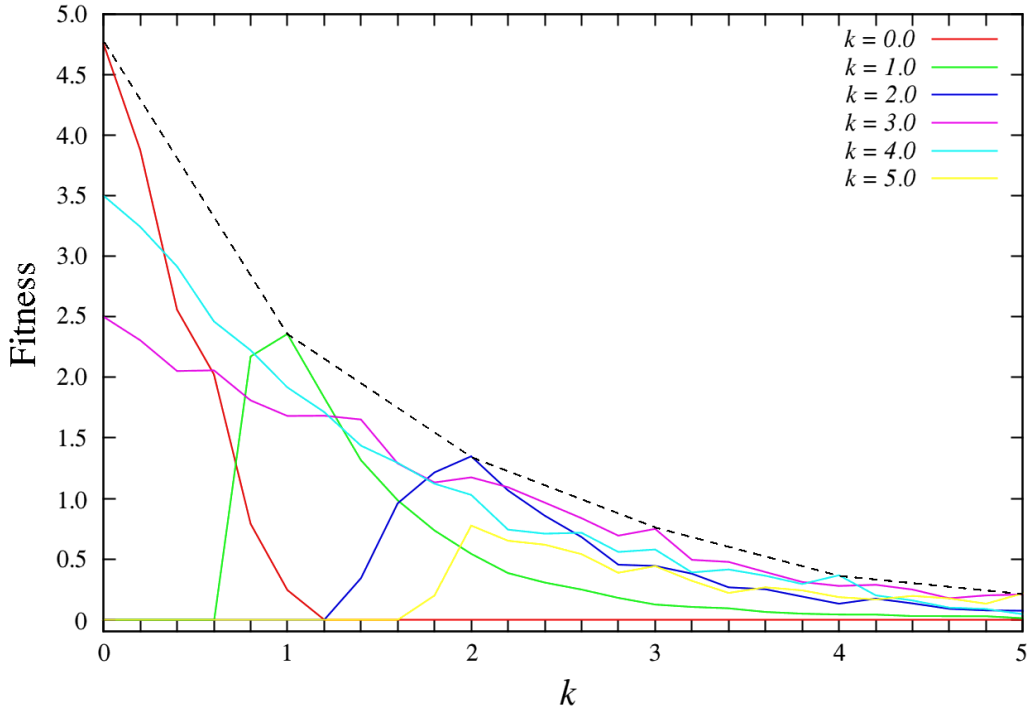
$$S_c(i,j) = \begin{cases} 2.0 - S_c(i,j)^4 & i \neq j,\ S_c(i,j) > 1.0 \\ S_c(i,j) & \text{otherwise} \end{cases} \tag{4.23}$$

Through regression testing, we found that repeating the previous experiments with this amendment gave results which were not significantly different to those described in section 4.3.5.

For each $k \in \{1,2,3,4,5\}$, we evolved ten networks, both with and without mutating network parameters. Figures 4.14(a) and 4.14(b) show how the best of the ten networks evolved for each k performed as the level of noise used in the simulation was varied. For each $k \in \{1,2,3,4,5\}$, the best network evolved achieved a higher fitness than the original illustrative network for that value of k. This can be clearly observed by comparing the dashed curve shown in figures 4.14(a) and 4.14(b) with the curve produced by the illustrative network shown in figure 4.12. These results therefore demonstrate that the assumption that DPC+NEAT can construct networks which offer an optimised performance for particular values of k does in fact appear to hold. However, the degree to which varying k affects different networks is not guaranteed. As shown in figure 4.14(b), the network evolved for $k = 3$, with mutating parameters, gives an increasingly better performance as k decreases (although its performance at $k = 0$ is significantly inferior to that of the best network optimised for $k = 0$), whilst the network evolved for $k = 5$ shows a deteriorating performance as k falls below 2. The same networks evolved for $k = 3$ and $k = 5$ without mutating parameters, shown in figure 4.14(a), both show a decline in performance as k falls below 1.8. The best network evolved both with and without mutating parameters for $k = 1$



(a)



(b)

Figure 4.14: The effect of varying k on the performances of the best networks evolved by DPC+NEAT for particular values of k. Two experiments were performed, each using a different value of m_P. The first graph (a) shows networks that were evolved in the first experiment, with $m_P = 0.0$, whilst the second graph (b) shows networks that were evolved in the second experiment, with $m_P = 0.8$. The dashed curve in each graph is a visual aid, connecting those points in the graph representing the best performance achieved for each k.

shows a declining performance as k moves away from 1 in either direction.

4.4.3 Discussion

The results obtained in this section demonstrate that networks evolved by DPC+NEAT are less robust to noise than the original illustrative neural network. The robustness of an evolved network in this scenario appears to decreases as its topology and weights move away from that of the illustrative network. This seems to indicate that DPC+NEAT optimises neural networks for the particular conditions present in the evolutionary phase. This is further evidenced by the second of the best performing networks from section 4.3.5 that we considered here, which had been evolved by DPC+NEAT with m_P set to 0.8. In this network, a value of β is chosen that gives a good performance in simulations where no noise is present, but a significantly worse performance when noise is introduced.

When evolving networks with noise present during the evolutionary phase, DPC+NEAT successfully found networks which gave a better performance for that level of noise than the original network. In most real-world scenarios, noise is intrinsically embedded in the data and therefore is apparent to DPC+NEAT during the evolutionary phase. The ability of DPC+NEAT to optimise the original network even in the presence of noise therefore demonstrates its potential suitability in such scenarios. However, the results obtained in this section indicate that, whilst small variations in noise will not be likely to heavily impact on the performance of evolved networks, the DPC+NEAT approach may be less suitable in scenarios where large variations in noise occur.

4.5 Novelty Detection with DPC+NEAT

So far, we have examined DPC+NEAT in the original context, given by Hosoya *et al.*, of separating stimuli from different artificial visual environments. In this section, we now describe how we use DPC+NEAT in novelty detection tasks outside of this domain, such as those considered in chapter 5. We first describe how we can interpret the response of a DPC neural network in a novelty detection task without relying on the sensitivity measure defined in chapter 3. We then propose a modification to the anti-Hebbian learning rule which removes an implicit assumption made about the input data. Since many novelty detection tasks involve data without a temporal component, such as record data, we discuss how the learning strategy employed by DPC neural networks should be modified to ensure their suitability in such tasks. This in turn leads us to

propose a modification to the hidden neurons that may be introduced into a DPC network, so as to ensure that the network retains a nonlinear behaviour regardless of the learning strategy employed. Finally, we describe how we construct the neural networks in the initial population provided to DPC+NEAT.

4.5.1 Interpreting Network Response

In the test scenario examined in this chapter, a DPC network's separation of stimuli from different environments is observed through the examination of its sensitivity to those environments. However, this measure of sensitivity is specific to Hosoya *et al.*'s visual experiments [63], and is generally not suitable in other novelty detection tasks. For example, since the sensitivity measure is based on the variance of the output of the network, it prevents the network from providing a decision on the novelty of individual input vectors. Yet, such functionality is required in many novelty detection tasks. An alternative is to instead use the magnitude of the network's output value as an indicator of novelty. In the original DPC neural network model described in chapter 3, the output value describes the error in the prediction constructed by the plastic synapses. Assuming that a novel input vector is likely to incur a greater error than a normal input vector, due to the different properties of the novel vector, then the magnitude of the output value should be a suitable indicator of novelty.

Therefore, in chapter 5 the evolved novelty detectors have a single output neuron, whose magnitude indicates the novelty of individual input vectors. In each novelty detector, the output layer is constrained to have a single output neuron and a new network parameter, $dpcNT$, is introduced to translate this magnitude into a binary decision value (i.e. "normal" or "novel"). If the magnitude of the output of an evolved novelty detector is greater than or equal to $dpcNT$ then the corresponding input vector is labelled as novel, otherwise it is considered to be normal. The value of $dpcNT$ is decided through evolution, along with the other network parameters β, τ and m. The compatibility distance measure, given in equation 4.1, is also modified to consider $dpcNT$. With this modification, the compatibility distance measure becomes:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} + c_4|\beta_1 - \beta_2| + c_5|\tau_1 - \tau_2| + c_6|m_1 - m_2| + c_7|dpcNT_1 - dpcNT_2| \quad (4.24)$$

where $dpcNT_i$ is the value of $dpcNT$ used by network i, and c_7 is a DPC+NEAT parameter controlling the influence of $dpcNT$ on equation 4.24. We regard differences in $dpcNT$ to be just

as significant as differences in any of the other network parameters, and so therefore set c_7 to 1.

4.5.2 Correlation Measure

The learning rule given in equation 3.3 in section 3.3.1 for plastic synapse a_{ij} is driven by the correlation between the input x_j and network output y_i over recent stimulus [63]. If the correlation between y_i and x_j is defined as $R_{y_i x_j}$, then the anti-Hebbian learning rule given in equation 3.3 may be expressed as:

$$\frac{da_{ij}}{dt} = \frac{-a_{ij} - \beta R_{y_i x_j}}{\tau} \quad \tau, \beta > 0 \tag{4.25}$$

As shown in chapter 3, Hosoya *et al.* determine the correlation between y_i and x_j using:

$$R_{y_i x_j} = \langle y_i x_j \rangle \tag{4.26}$$

$$= \frac{1}{m} \sum_{k=t-m+1}^{t} y_i(k)\, x_j(k)\,, m > 1 \tag{4.27}$$

However, the above definition of the correlation term $R_{y_i x_j}$ makes the implicit assumption that the distribution of input values x_j will have zero mean and unit variance [94]. This assumption is undesirable since the mean and variance of the data used in a particular novelty detection task are usually not known *a priori*. Whilst it is possible that a data preprocessing technique could be used to transform the data to have the required mean and variance, this is undesirable as it would increase the work required from the user in applying a DPC-based novelty detector to their particular task. This correlation measure is also unbounded, with a magnitude that is sensitive to that of individual values of x_j and y_i, especially when m is small. This makes it potentially possible for high-magnitude input values to destabilise the neural network.

An alternative measure of correlation that could be used by the learning rule is Pearson's sample correlation coefficient, which is defined as [24]:

$$R_{y_i x_j} = \rho_{y_i x_j} \tag{4.28}$$

$$= \frac{cov(y_i, x_j)}{\sqrt{var(y_i) var(x_j)}} \tag{4.29}$$

$$= \frac{\frac{1}{m}\sum_{k=1}^{m} y_i(k) x_j(k) - \bar{y_i}\bar{x_j}}{\sqrt{\frac{1}{m}\sum_{k=1}^{m}\left(y_i(k) - \bar{y_i}\right)^2 \frac{1}{m}\sum_{k=1}^{m}\left(x_j(k) - \bar{x_j}\right)^2}} \tag{4.30}$$

This measure of correlation provides two important advantages over equation 4.26. First, the mean and variance of the input distribution are not assumed, but instead approximated using sample means and variances. Second, it can be shown that the coefficient obtained from equation 4.28 will always lie in the range $[-1, +1]$ [24], regardless of the magnitudes of x_j or y_i. We anticipate that this would allow a more stable performance from the network in scenarios involving high-magnitude input values. As in equation 4.26, the DPC parameter m may again be used to define the period over which the correlation between the activity of two neurons in the network is measured.

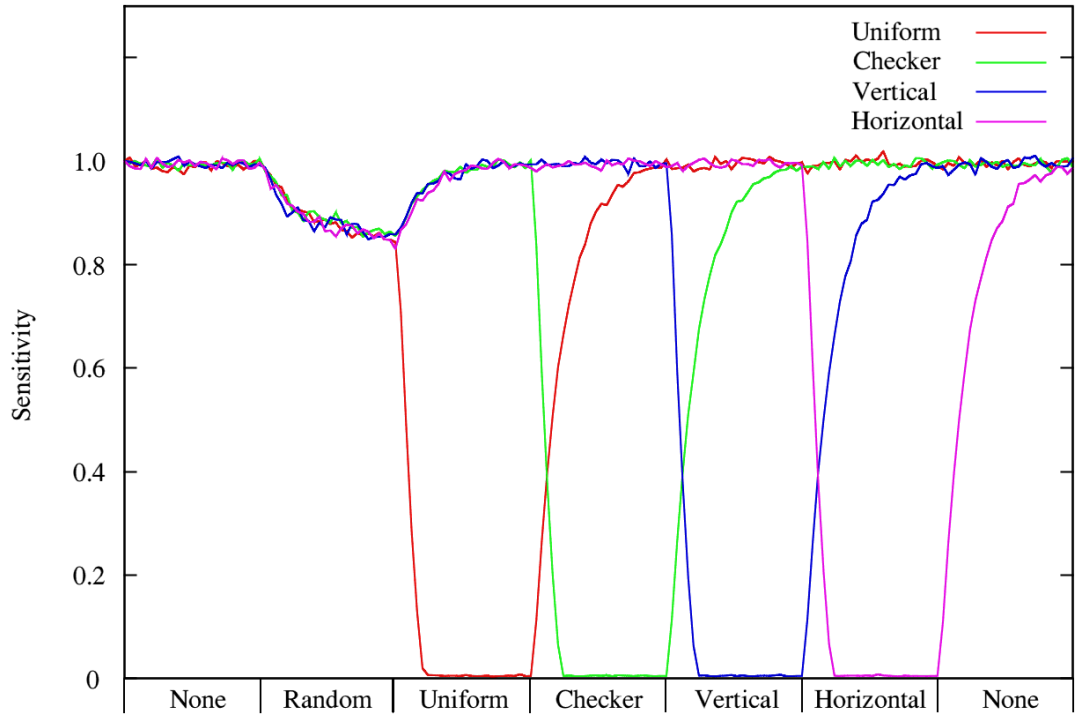


Figure 4.15: The sensitivity graph produced by the original illustrative neural network when Pearson's correlation coefficient is used in the anti-Hebbian learning rule.

In order to gauge the effect of using the sample correlation coefficient on the DPC model, we performed a simulation of the original illustrative neural network with the new correlation measure. The resulting sensitivity graph is shown in figure 4.15. Comparing this against the original sensitivity graph (shown in figure 3.10 in section 3.4.4), we observe that, with precisely the same parameters, the network achieves a much lower adapted sensitivity to the adaptable environments. Furthermore, a much less significant drop in the network's sensitivity to the adaptable environments is observed when the Random environment is applied. In the experiments conducted in chapter 5, we therefore continue to use the sample correlation coefficient in

the anti-Hebbian learning rule.

4.5.3 Non-Adaptive Networks

Whilst the novelty detectors evolved by DPC+NEAT are designed to be adaptive, this is not appropriate in all scenarios. An adaptive novelty detector is only required when the data in question has a temporal dimension and so has the potential to change over time. Thus, in record datasets containing no temporal information, an adaptive approach is not required. Moreover, an adaptive approach can actually prove to be a disadvantage when used on such datasets, since the approach is sensitive to the *ordering* of the data which is not guaranteed. The novelty of a data element in a record dataset is not dependent on any other element of the dataset outside the training phase. Yet, an adaptive novelty detector may undergo changes in its state as it processes each element of the dataset, adjusting its acquired model of normality according to what it has observed "recently". By doing this, the novelty detector fails to classify elements independently of those seen in the "recent" past.

To address this problem, we simply allow the user to disable the adaptive learning capability of the DPC neural network model when necessary. In scenarios not requiring an adaptive solution, the plastic synapses are modulated during a training phase, after which their weights become fixed. The anti-Hebbian learning rule is still used in the training phase, meaning that the network continues to use an unsupervised learning technique. However, whilst allowing the user to prevent DPC networks from continually learning about the data makes those networks suitable for use in scenarios where adaptive behaviour is not appropriate, it also prevents the network from operating in a nonlinear fashion. We discuss how to compensate for this next.

4.5.4 Hidden Neurons

In this chapter, we extended the DPC neural network model to include hidden neurons. In doing so, we gave DPC+NEAT the building blocks required to construct networks with any topology, in the hope that this would allow the approach to discover suitable topologies in a wide range of different novelty detection scenarios. In a typical neural network, it is important that hidden neurons exhibit a nonlinear functionality. If they do not, then the capability of the network is reduced to that of a single-layer perceptron [57].

Despite this, the hidden neurons that we have considered in this chapter are linear, since they simply output the summation of their inputs. However, the networks evolved here do not reduce

to linear single-layer networks because of the nonlinearity provided by the plastic synapses. But this is true only when these synapses are allowed to adapt, meaning that the nonlinearity of the network is lost when, as discussed in section 4.5.3 above, this adaptive property is disabled. To enable DPC neural networks to retain their nonlinearity in such scenarios, it is necessary to replace the hidden neurons used in this chapter with ones that employ nonlinear activation functions. This gives the networks evolved in chapter 5 the capability to construct multiple decision boundaries as necessary for the particular novelty detection task at hand. The activation function we use is the familiar threshold function, defined as [57]:

$$\varphi(v) = \begin{cases} 1 & v \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.31}$$

This function is chosen over other possible activation functions, such as the logistical function given in chapter 2, because of its simplicity, thus reducing the impact of increasing numbers of hidden neurons on computational complexity. With this threshold activation function, the output h_i of hidden neuron i becomes:

$$h_i = \varphi\left(\sum_j (b_{ij} + a_{ij})\, x_j\right) \tag{4.32}$$

The threshold activation function given in equation 4.31 makes the assumption that zero is a suitable threshold value. However, it may be that a different value would be more appropriate, or that different hidden neurons require different threshold values. To permit further customisability, one option is to make the threshold at each hidden neuron an evolvable network parameter, similar to the output neuron's threshold $dpcNT$. However, this would increase the dimensionality of the parameter space each time a new hidden neuron is introduced. An alternative to this is to introduce a *bias node* into the network. Such a node outputs a constant value (typically +1), which may be tuned by modifying the weight of any synaptic connections extending from the node. This method of adjusting the threshold of activation functions is commonly employed in neural networks [57]. This alternative also has the advantage that the bias node need only be connected to those hidden neurons whose threshold value actually requires an adjustment. If the threshold required by a particular hidden neuron is in fact zero, then no connection to the bias node is required. We therefore provide all networks with a bias node, but do not automatically connect this node to new hidden neurons. Through its mutation operators, DPC+NEAT may adjust the threshold of a particular hidden neuron by introducing a new connection between

this neuron and the bias node and then modifying the weight of this connection.

4.5.5 Initial Population

In the test scenario considered in this chapter, the objective was to optimise the performance of the original DPC neural network. Therefore, in the first generation, each individual in the population was a copy of the original network with a weight and network parameter mutation (if $m_P > 0$) applied. However, when evolving a DPC network for their particular novelty detection application, the user will not normally know of a suitable network to start the search with. Instead, as advocated by Stanley [108], the search should begin with networks of minimal topology.

Stanley suggested that each network in the first generation have the same topology, with each input being connected to every output, and randomly chosen weights [108]. We proceed in a similar manner: each input of a network is directly connected to the output neuron by a plastic synapse, with the first input also connected by a fixed synapse (so that the output neuron is stimulated when the network is in an unadapted state). As previously mentioned, a bias node is also provided in the input layer to allow adjustment of hidden neuron thresholds. This bias node is also connected to the output neuron by a fixed synapse. The function provided by this network topology is to predict and suppress the values seen at the first input, based on the correlational relationship between this first input and all other input nodes.

4.6 Summary

By combining Hosoya *et al.*'s DPC neural network model [63] with Stanley's Neuroevolution of Augmenting Topologies (NEAT) algorithm [108], we have devised DPC+NEAT, a system which evolves DPC neural networks for novelty detection applications. We have examined the ability of DPC+NEAT to optimise the original illustrative neural network described in chapter 3 according to a predefined performance criteria. DPC+NEAT successfully found networks with a better performance than the original network, both with and without mutating the parameters of the DPC model. This suggests that DPC+NEAT has the potential to be a promising approach. Whilst, on average, better performing networks were found with mutating parameters, the performance of the best of these networks was also the fastest to degrade when Gaussian noise was introduced to its inputs. However, when a particular level of noise was introduced to the stimuli used to evaluate networks during the evolutionary phase, DPC+NEAT was able to evolve

networks which outperformed the original illustrative neural network for that level of noise. This indicates that DPC+NEAT optimises neural networks for the particular conditions present in the evolutionary phase. The ability of DPC+NEAT to evolve novelty detectors that give a good performance over noisy data is important, since most real-world scenarios involve data with a noise component.

We have also discussed how DPC+NEAT will be applied to novelty detection tasks in general, such as those considered in the next chapter. Instead of using a sensitivity measure, the network will indicate the novelty of a particular input vector with the magnitude of its output value. This is thresholded, according to a network threshold parameter $dpcNT$, to obtain a binary decision value (with the value of $dpcNT$ being decided through evolution). A modified version of the anti-Hebbian learning rule is used, which removes an implicit assumption that the input data has zero mean and unit variance. This modification substitutes Hosoya *et al.*'s original correlation term with Pearson's sample correlation coefficient. To allow DPC+NEAT to evolve networks for nonlinear novelty detection tasks which do not require adaptive learning, threshold activation functions are introduced to the hidden neurons. A bias node, present in the input layer, can be used to adjust the threshold value for each neuron. Finally, a generic network topology has been selected that gives a suitable starting point for the evolutionary search.

In the next chapter, we evaluate DPC+NEAT against the objectives set out in chapter 1. We test the approach in a series of very different novelty detection tasks, involving either record or time series datasets. We also compare the performance achieved by DPC+NEAT in each task against that given by a number of existing approaches to novelty detection from the literature.

Chapter 5

Evaluating DPC+NEAT

In the previous chapter, we presented the DPC+NEAT approach, the combination of Hosoya *et al.*'s Dynamic Predictive Coding (DPC) neural network model [63] with Stanley's Neuroevolution of Augmenting Topologies (NEAT) algorithm [108], which we consider to have the potential to fulfil each of the four objectives stated in chapter 1. In this chapter, we evaluate DPC+NEAT over a series of novelty detection tasks. We also compare DPC+NEAT with three existing approaches from the literature in each task.

5.1 Introduction

The system proposed in the previous chapter, DPC+NEAT, appears to represent a promising new approach to novelty detection. In a test scenario, this approach was shown to successfully improve the performance of an illustrative neural network according to a predefined criteria. The performance of the network was also optimised by DPC+NEAT when different levels of noise were present in the data used during the evolutionary search. Its apparent ability to evolve highly customised neural networks for a particular novelty detection task, slowly increasing network complexity only as necessary to achieve gains in performance, without requiring a significant amount of work from the user leads us to hypothesise that DPC+NEAT has the potential to meet the four objectives stated in chapter 1. Before evaluating DPC+NEAT, we first reconsider the four objectives stated in chapter 1.

These objectives define the requirements that an approach would need to fulfil in order to allow novelty detection to be performed easily, efficiently and effectively in a wide range of different scenarios. They provide us with the criteria against which we evaluated existing

approaches to novelty detection in chapter 2. These objectives are restated below:

- *General Suitability:* The approach should be suitable for and effective in a wide range of different novelty detection tasks.
- *Customisability:* The approach should permit a high degree of customisability.
- *Work:* The effort required from the user to customise the approach for a particular task should be minimal.
- *Complexity:* The complexity of the approach should be justifiable for the particular task.

In this chapter, we evaluate DPC+NEAT against these objectives. To evaluate DPC+NEAT's ability to fulfil the first objective, general suitability, we examined its performance in six very different novelty detection tasks involving either record or time series datasets. From this, we hope to identify any limitations that may affect the applicability of DPC+NEAT in other problems not considered here. A good performance from DPC+NEAT in a particular novelty detection task suggests that it is capable of evolving novelty detectors which are suitable for that task.

But what constitutes a "good" performance? Some novelty detection tasks are inevitably harder than others, which suggests that the performance of an approach in a particular problem should be considered in the context of that achieved by other approaches. To provide context to the results achieved by DPC+NEAT, we therefore compared its performance in each task against that given by three existing approaches, which we evaluated in chapter 2. Two of these approaches have been demonstrated over a range of different novelty detection tasks, and so are used in every experiment considered in this chapter. However, the third approach used in each task is specifically designed for the type of data (i.e. record or time series) considered in that task.

Since DPC+NEAT designs a neural network specifically for the novelty detection task at hand, it can take advantage of a high degree of customisability, the second of the four objectives listed above. However, not only should DPC+NEAT design novelty detectors which give a high level of performance, it should also minimise their complexity, so as to achieve the fourth objective of our evaluation criteria. By tailoring to a particular task, it should be possible for DPC+NEAT to discover unique properties of the data in that task and exploit this information in the design of the novelty detector. To determine whether or not DPC+NEAT does this in

each task, we examine the structural complexity of the best networks evolved in light of the perceived difficulty of the task, as well as identifying any redundant structure.

Finally, according to the third objective above, DPC+NEAT should require a minimal amount of work from the user in its configuration for a particular task. This is achieved in two ways. First, we allow the parameters of the DPC neural network model to be determined by DPC+NEAT during the evolutionary search. By allowing DPC+NEAT to determine the network structure, synaptic weights and model parameters, we are able to achieve a high degree of customisability without increasing the work required from the user. Second, as discussed in chapter 4, we use the same parameters for DPC+NEAT in all experiments conducted throughout this thesis. We aim to demonstrate that solutions can successfully be found even when DPC+NEAT's own parameters are not optimised for the particular problem. The parameters used throughout this chapter are given in appendix A.

We now present the evaluation of the DPC+NEAT approach. In the next section, we present the six novelty detection tasks that we used to test the DPC+NEAT approach. Then, in section 5.3 we detail the approaches from the literature, originally evaluated in chapter 2, with which we compare DPC+NEAT's performance over these six novelty detection tasks. In section 5.4, we describe the performance measure used in this work to assess each novelty detector. We then detail two cross validation techniques that were used to ensure that favourable results were not achieved through overfitting in section 5.5, and in section 5.6 describe the statistical test that we used when comparing DPC+NEAT with the existing approaches. In section 5.7, we describe the experimental setup used to evaluate DPC+NEAT and in section 5.8 present the results achieved by DPC+NEAT and the comparison approaches in each of the novelty detection tasks. Finally, we evaluate DPC+NEAT in the context of these results in section 5.9.

5.2 Novelty Detection Tasks

We first present the novelty detection tasks which we used to evaluate DPC+NEAT. The tasks used in this evaluation should be different enough to allow us to judge whether or not DPC+NEAT is likely to be suitable for a wide range of novelty detection problems. However, as we mentioned in chapter 1, in this thesis we restricted ourselves to those tasks which involve two very common forms of data: record data and time series data. We considered six novelty detection tasks in total: three based on real-world record datasets, two based on artificially generated time series and one based on a challenging real-world time series.

5.2.1 Novelty Detection over Record Data

Detecting anomalies or irregularities in a set of data records is an important task in many real-world scenarios. For example, given a series of blood measurement observations, where each observation consists of a series of measurements taken from a particular patient, novelty detection can be used to identify unusual measurements, or combinations of measurements, which may indicate the presence of disease. However, one problem with testing novelty detectors on record data is the lack of suitable benchmark datasets, i.e. record datasets that have been specifically designed to represent novelty detection problems.

In the classification literature, a large number of benchmark datasets are widely available, in which the data is provided with class labels. These class labels are critical for determining the performance of both classifiers and novelty detectors, since they provide us with an independently defined "ground-truth" which can be compared against the responses given by the detector. The lack of a suitable novelty detection benchmark dataset has led many researchers to resort to using classification datasets when testing their novelty detection approach on record data. Typically, one class in the chosen dataset is labelled as "normal" with the remaining class or classes being used to represent the "novel" class. This has the drawback that the "novel" class is no longer truly novel, as it is generally well represented in the dataset.

The lack of a suitable alternative has meant that, despite this drawback, we are also forced to use classification datasets to test our approach on record data. In this work, we use three such datasets: Cox *et al.*'s Biomedical dataset [27], the Wisconsin Breast Cancer dataset and Fisher's classical Iris dataset [41]. As we will show, each of these datasets has been used before to test various approaches to novelty detection. We now discuss these datasets in more detail.

Biomedical

The first dataset that we consider is Cox *et al.*'s 1982 Biomedical dataset[1] [27], which is also sometimes referred to as Biomed [85]. The Biomedical dataset has been used by many researchers to evaluate different approaches to novelty detection [13, 66, 85, 111]. The dataset has 209 observations, with each observation consisting of four blood measurements taken from a single patient. 15 of these observations have one or more blood measurements missing and so are discarded. Of the remaining 194 observations, 127 are taken from healthy patients and so are considered to be normal. The remaining 67 observations are taken from patients carrying a rare

[1] Available from: http://lib.stat.cmu.edu/datasets/

genetic disease, and are considered to be novel. The task is, after training on normal data alone, to label normal and novel observations correctly.

Breast Cancer

We next consider the Wisconsin Breast Cancer dataset[2] [126], originally obtained from the University of Wisconsin Hospitals. Again, this has also been used as a benchmark dataset for many proposed approaches to novelty detection [47, 75, 115]. This is a larger dataset than Biomedical, consisting of 683 observations without missing data. Each observation consists of 9 attributes, which describe characteristics of cell nuclei present in an image of a fine needle aspirate of a breast mass. Of the 683 observations without missing data, 444 are taken from benign masses, and are thus classed as normal, and 239 are taken from malignant masses, which are classed as novel. Again, the task is, after training on normal observations alone, to identify those novel observations corresponding to malignant masses, whilst correctly labelling normal observations. This dataset is distinguished from the others we consider by both its higher dimensionality (9 attributes as opposed to 4) and the larger number of data records.

Iris

Finally, we consider Fisher's classical Iris dataset[3] [41]. Like the previous two datasets, this is yet another popular benchmark among researchers evaluating novelty detection approaches [66, 111, 115]. This dataset consists of observations belonging to *three* classes, where one class is linearly separable from the other two. The dataset consists of 150 observations, equally divided between the three classes. Each observation consists of 4 measurements (sepal length, sepal width, petal length and petal width) taken from a type of Iris plant. The three classes identify the precise family to which the plant belongs: setosa, versicolor or virginica.

To make this dataset suitable for use in a novelty detection scenario, three versions of the dataset are constructed [66, 111]. In each version, one class is labelled as the normal class, and the other two classes are labelled as novel. A different class is labelled as the normal class in each version. The task is then, after training with instances from the normal class, to identify whether or not a given observation belongs to that class. For ease of reference, we name the variant datasets as *Iris-Setosa*, *Iris-Versicolor*, and *Iris-Virginica*, where the named class is the normal class for that version of the dataset.

[2] Available from the UCI Machine Learning Repository at: http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)

[3] Available from the UCI Machine Learning Repository at: http://archive.ics.uci.edu/ml/datasets/Iris

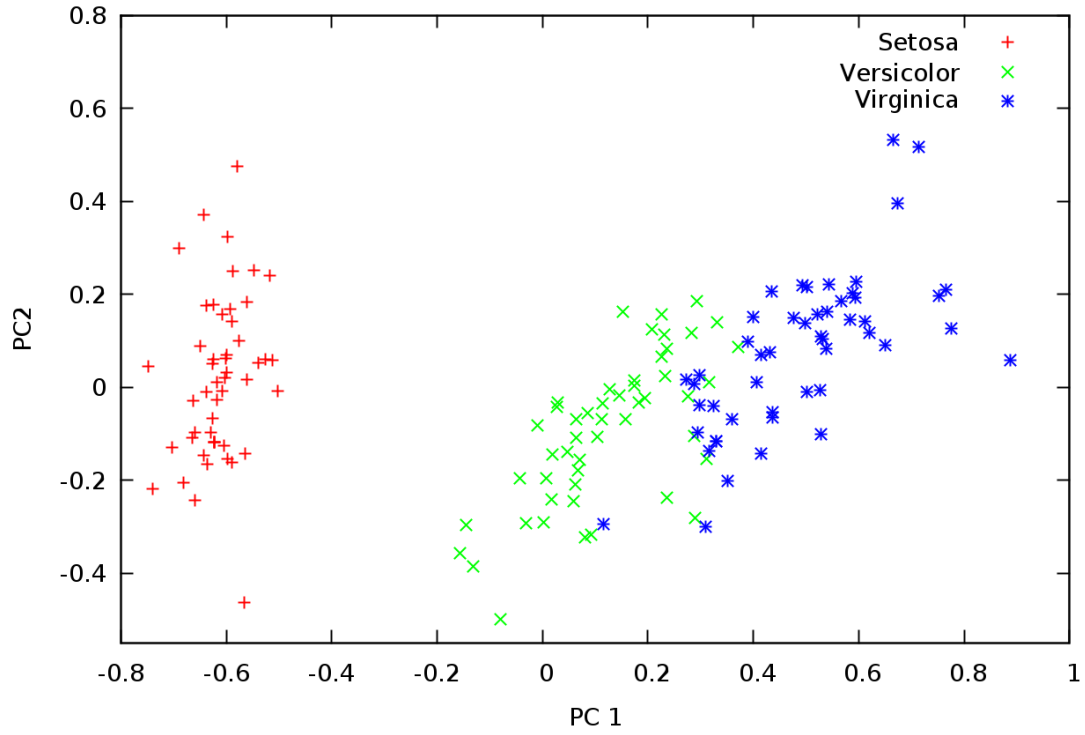


Figure 5.1: A visualisation of first two principal components of Fisher's classical Iris dataset. The setosa class is clearly linearly separable from the versicolor and virginica classes.

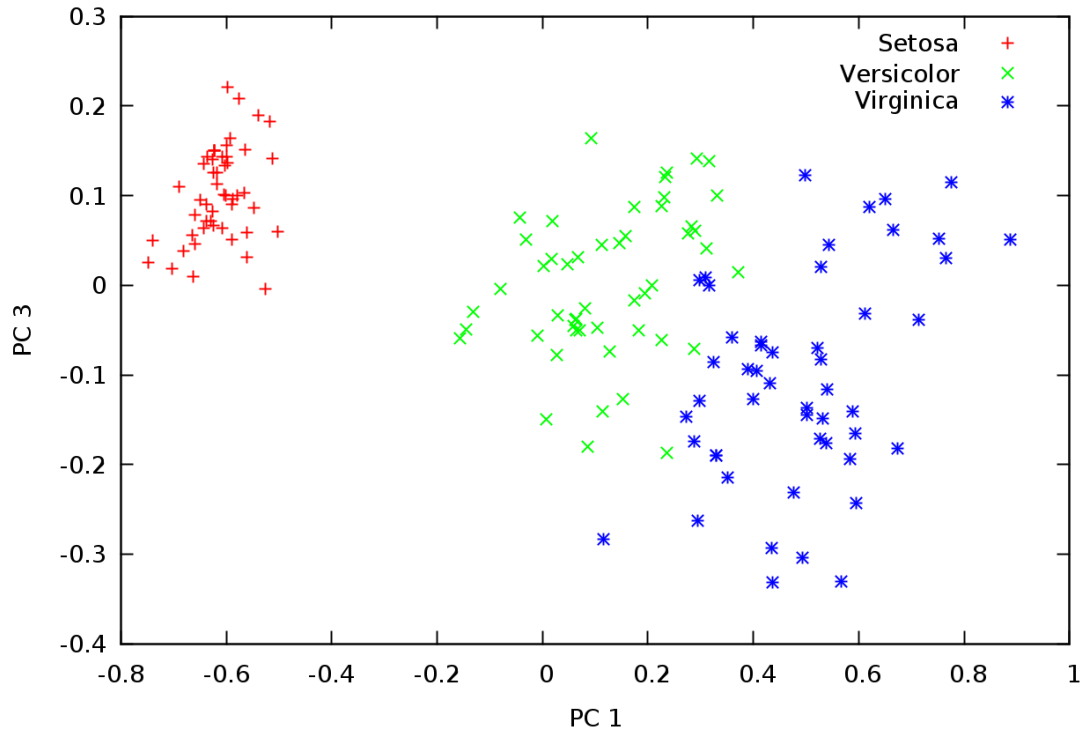


Figure 5.2: A visualisation of Fisher's classical Iris dataset along the first and third principal components. In this view, it can clearly be seen that the nonlinear separation between the versicolor and virginica classes can be effectively approximated by a linear decision boundary.

A visualisation of the first two principal components of this dataset is shown in figure 5.1. In this dataset, the setosa class is clearly linearly separable from the other two, meaning that, when this class is considered normal, DPC+NEAT should be easily able to find a solution. The trivial nature of this problem should result in a solution with a minimal number of hidden neurons and synapses. In fact, solutions should be found which employ no hidden neurons since only a single decision boundary is required. The versicolor and virginica classes, on the other hand, are nonlinearly separable. However, viewing the data along the directions of the first and third principal components (shown in figure 5.2), it is clear that a linear decision boundary can provide a high degree of classification accuracy. Another interesting aspect of this dataset, shown in figures 5.1 and 5.2, is that the versicolor class lies *between* the other two classes. This allows us to test whether or not DPC+NEAT can construct a novelty detector capable of forming two decision boundaries in the input space, one on either side of the normal class.

5.2.2 Novelty Detection over Time Series Data

The next three novelty detection tasks we consider concern both univariate and multivariate time series datasets. A univariate time series is constructed by measuring some quantity, or variable, at regular intervals. This is extended to the multivariate case by considering multiple variables simultaneously. For example, a univariate time series may describe the indoor temperature of a single room of an office building over the course of a year. By combining this with those time series describing temperatures in the other rooms of the building over the same period, one obtains a multivariate time series describing the temperature variations throughout the building over the year.

As with the record datasets, to measure the performance of a novelty detector over a particular time series we require that series to be labelled so that we know beforehand which novel events occur and when. In the case of record data, finding datasets in which each element was suitably labelled was not difficult. However, the majority of the publicly available real-world time series datasets are provided without labels. This is most likely due to the difficulty in reliably assigning such labels, especially in large time series. Some authors [78] have solved this problem by validating the response of the novelty detector through a visual inspection of the time series. However, this is unreliable since the detector's response to non-trivial novel events which are not visually identifiable will not be evaluated. Also, it is more rigorous to require an independent, impartial validation of the novelty detector's performance. This is achieved if the

time series is independently labelled by an expert.

Due to the lack of available labelled time series data, the first two novelty detection tasks we used in this work are based on synthetic data. Each time series is generated by an underlying mathematical model, which is then perturbed in some way to generate novel events. The final dataset that we consider is a real-world medical time series, with labels which have been assigned independently by multiple experts.

Mackey-Glass

The first synthetic series we consider is the famous Mackey-Glass dynamic time series [81]. This dataset was used by González and Dasgupta to test their hybrid approach to novelty detection based on real-valued negative selection [47]. The time series is defined by the differential equation:

$$\frac{dx}{dt} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t) \tag{5.1}$$

where a, b, c and τ are parameters. Importantly, τ controls the chaotic nature of the series [47]. In this work, we used an experimental setup similar to that employed by González and Dasgupta [47]. Equation 5.1 was solved numerically using the fourth-order Runge-Kutta method, with an integration step of 0.02 and a sampling rate of 12. The initial condition was chosen randomly in the range $[0.2, 1.4]$. The values used for the parameters were $a = 0.2$, $b = 0.1$, $c = 10$ and, for normal data, $\tau = 30$. We then generated an 11,000 point time series, with the first 1,000 points being discarded to eliminate the initial-value effect [47]. Of the remaining series, we used 1,000 points, with no novel events, as the training set. The remaining 9,000 points were used as the test set. Within this test set, 42 novel events with varying duration were generated by randomly assigning a new value to τ.

Since this time series is univariate, and the novelty detection methods we consider in this chapter require multivariate vector inputs, we also had to perform a preprocessing step on the series. One commonly used preprocessing step to translate a univariate time series into a series of multivariate vectors is the *sliding window* technique. This involves placing a window with a size of n data points over the time series, and assigning all data points within that window to a single vector. This constitutes the input for a single time step t. The vector for the next time step $t+1$ is then obtained by moving the window m data points along the series and extracting those data points which subsequently fall inside the window. In this work, we used a sliding

window of size 4 and moved the window along 1 data point after each time step. Therefore, the sliding windows for subsequent time steps *overlap*. This experimental setup is the same as that used by González and Dasgupta [47].

One problem with employing the sliding window technique is that it gives a novelty detector a series of different perspectives of a single novel event. These perspectives may be categorised as follows [88]:

- *Boundary Condition:* a window consisting of normal and novel data points (i.e. encompassing the boundary of the anomalous event).
- *Encompassing:* a window which encompasses the entire anomalous event, whilst also including one or more normal data points.
- *Whole:* a window which exactly covers the entire anomalous event.
- *Internal:* a window which falls inside the anomalous event.

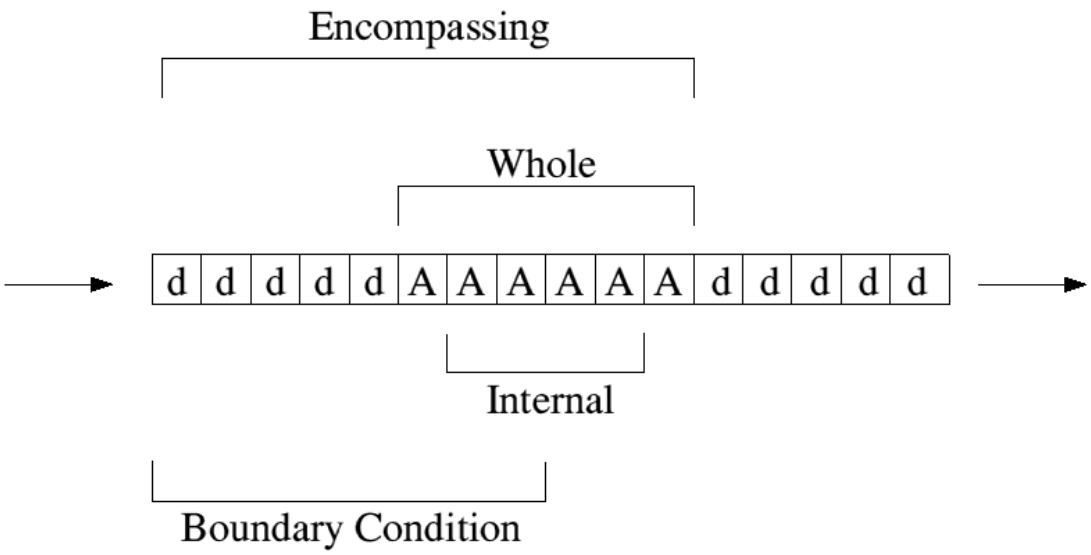


Figure 5.3: Different perspectives of a time series generated by a sliding window technique [88]. The normal data points in the time series are represented by "d" and the novel event is indicated by "AAAAA".

These different perspectives are illustrated in figure 5.3 [88]. In deciding how to assess a novelty detector, we must decide exactly what we want it to identify. Those windows which fall entirely on normal data should always be classified as normal. However, in the case of a novel event, the detector could be required to classify each and every window containing one or more anomalous data points pertaining to that event as novel. Alternatively, it may be sufficient to classify

at least one of the windows covering the event as novel, thus still successfully highlighting the event to the user. To classify each window as novel leads to the problem of differing degrees of difficulty. For example, a boundary condition window with a single anomalous data point is considerably harder to classify than an internal window. Identifying whether or not the precise data point at time step t is anomalous is not possible for the comparative approaches, as these attempt to determine the class of the entire vector. To avoid these difficulties in this work, we used the alternative approach given above, i.e. classifying at least one of the windows covering a particular novel event as novel.

Hard Disk

We next consider a simple numerical model of a hard disk read/write head controller. This model is given by Mathworks as an example case study for the MATLAB Control Systems Toolbox [87] and is used by Sohn *et al.* to test their auto-associative novelty detection approach [106].

The read/write head is modelled by the differential equation:

$$J\frac{d^2\theta}{dt^2} + C\frac{d\theta}{dt} + K\theta = K_i i. \tag{5.2}$$

where J is the inertia of the head assembly, C is the viscous damping coefficient of the bearings, K is the return spring constant, K_i is the motor torque constant, θ is the angular position of the head, and i is the input current [87]. In Sohn *et al.*'s work [106], each of these variables is a function of an ambient temperature T, as defined below.

$$K = \frac{6}{87}(0.1T - 1.5)^3 + \frac{4}{87}(0.1T - 1.5) + 10 \tag{5.3}$$

$$K_i = \frac{0.01}{30}[(0.1T - 1.5)^3 + (0.1T - 1.5)^2 + (0.1T - 1.5) + 1] + \frac{0.14}{3} \tag{5.4}$$

$$J = 0.01\left(\frac{T}{15} + 9\right) \tag{5.5}$$

$$C = 0.004tanh\left(\frac{T}{30}\pi - \frac{\pi}{2}\right) \tag{5.6}$$

If the Laplace transform of equation 5.2 is taken and discretised using the zero-order-hold

method, the following discrete transfer function, from i to θ, is obtained [106].

$$H(z) = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2} \tag{5.7}$$

Case	K	C
(a)	$[0.85K_{20}, 0.95K_{20}]$	C_{20}
(b)	K_{20}	$[0.90C_{20}, 1.10C_{20}]$
(c)	$[0.95K_{20}, 1.05K_{20}]$	C_{20}
(d)	$[0.95K_{20}, 1.05K_{20}]$	$[0.90C_{20}, 1.10C_{20}]$

Table 5.1: The damage cases used by Sohn *et al.* in the Hard Disk model [106]. K_{20} and C_{20} denote the values of K and C respectively, derived using equations 5.3 and 5.6, with T fixed at 20 °C.

The coefficients a_1, a_2, b_1, and b_2 were used as input to the novelty detector in Sohn *et al.*'s work. Each coefficient was corrupted with Gaussian noise, which is 1% of the root-mean-square (RMS) of that coefficient. Different kinds of novel data were artificially generated by introducing four damage cases, described in table 5.1. In each case, the ambient temperature T was fixed at 20 °C in the model and K and C were perturbed as indicated in table 5.1.

In our work, each instance of this time series consists of 2,500 data points, where the first 250 points constitute training data, free of anomalies, and the remaining 2,250 points constitute test data. The results presented by Sohn *et al.* [106] indicate that damage case (b) from table 5.1 is the most challenging. Therefore, the test data contains anomalous events corresponding to this damage case. Unlike the experimental setup used by Sohn *et al.*, anomalous events are randomly interdispersed with normal data points. The task here is to identify those data points which are anomalous, without misclassifying normal data points.

ECG

The final time series we consider is an electrocardiogram (ECG) series from the MIT-BIH Arrhythmia Database, available from PhysioNet[4] [46]. An ECG is a time series of the electrical potential between two points on the surface of the body caused by a beating heart [68], consisting of a series of spikes corresponding to individual heart beats.

The MIT-BIH Arrhythmia Database contains 48 half-hour 2-channel ECG recordings. Each recording has been independently annotated by two or more cardiologists. The annotations consist of a label associated with each beat indicating whether that beat is normal or abnormal.

[4] http://www.physionet.org/physiobank/database/mitdb/

Different labels are used for different types of anomalous beats. Here, the task is to identify those beats labelled as abnormal. We consider recording 201 from the MIT-BIH database, as this contains the widest variety of different abnormal beat types [7]. The recording consists of 650,000 data points with 1,963 beats. Of these, 338 beats are labelled as abnormal, with 6 different abnormal beat types[5]. The first 3,000 points contain 11 normal beats, and we use this subset as the training set.

The task of identifying abnormal beats is similar to that of identifying anomalous events in the Mackey-Glass time series. Each novelty detector is required to highlight an anomalous beat by identifying an anomaly in any of the time steps covered by that beat. As this is a multivariate time series, the inputs to the detector at a given time step are the single data points from each of the 2 channels of the recording for that time step. One problem is identifying the precise duration of each beat [7]. Since we have no way of determining this, we considered all responses from each detector in the 150 time steps centred around the step for which the beat label was provided to relate to that beat. Furthermore, in this task we discarded all responses which do not directly relate to a beat. In doing this, we make the assumption that the central point of a beat, to which each annotation corresponds, is easily identifiable.

5.3 Comparison Approaches

In order to provide context to the results achieved by DPC+NEAT in each task, we compared its performance to that of a number of existing approaches to novelty detection from the literature. The comparison approaches we selected are Marsland's Grow When Required (GWR) algorithm [85], Ji's V-detector approach [66], Schölkopf *et al.*'s one-class support vector machine (1-SVM) [99] (for tasks considering record data) and Ahmed *et al.*'s Kernel-based Online Anomaly Detection algorithm (KOAD) [3] (for tasks considering time series data). We now briefly discuss these approaches.

As detailed in section 2.3.4, GWR is an approach based on Kohonen's self-organising map (SOM). Despite being specifically proposed to tackle an online novelty detection task based on a mobile robot, Marsland also applied GWR to a selection of novelty detection tasks based on record datasets [85]. This apparent applicability to both record and time series data makes this method an ideal competitor to our approach. However, since GWR is a clustering approach, it may not be effective in univariate time series tasks in which input vectors consist of subsequences

[5]Note that whilst Bay *et al.* report 7 abnormal beat types [7], our examination revealed there to be only 6.

of the series (formed by the sliding windows preprocessing technique) [67]. This may result in a poor performance being exhibited for the Mackey-Glass time series in this chapter. We used the publicly available MATLAB implementation of GWR provided by Marsland[6].

V-detector, described in section 2.2.2, is a real-valued negative selection (RVNS) algorithm proposed by Ji [66] in which each detector has an independent radius. Like GWR, V-detector has also been demonstrated on novelty detection tasks over both record and time series datasets [66]. This again makes V-detector a good comparison approach. In this work, we used the publicly available Java implementation of V-detector provided by Ji[7].

1-SVM, described in section 2.2.3, was proposed by Schölkoph *et al.* [99] as a method of employing SVM for novelty detection. However, this approach was primarily designed for data without a temporal dimension. According to Ma and Perkins, 1-SVM may potentially achieve poor results in time series tasks where the data has not been specifically preprocessed for use with 1-SVM [79]. Indeed, the preprocessing technique used over the data has a critical influence on the success of SVM-based approaches [119]. Therefore, we compared the performance of DPC+NEAT against 1-SVM only in those novelty detection tasks involving record datasets. 1-SVM has previously been applied to two of the datasets considered here, Biomedical and Iris, by Stibor *et al.* [111]. In their work, the sole preprocessing step performed was to normalise the datasets to the range $[0, 1]$, which we also did here. We used the publicly available implementation of 1-SVM provided by Chang and Lin's LIBSVM library[8] [23].

Finally, KOAD, described in section 2.2.4, is a recent approach proposed by Ahmed *et al.* [3] for novelty detection over time series datasets. The approach has been shown to give good results over very different time series anomaly detection tasks, such as identifying anomalies in data describing traffic behaviour in a computer network [3, 4] and identifying unusual events from images taken from a series of cameras monitoring a road network. This demonstrates an impressive versatility, which makes KOAD a good comparison approach for the time series tasks we consider in this chapter. We use the publicly available MATLAB implementation of KOAD provided by the authors[9].

[6]Available at: http://www-ist.massey.ac.nz/smarsland/GWR.html
[7]Available at: http://www.zhouji.net/prof/vdetector.html
[8]Available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm/
[9]Available at: http://www.tsp.ece.mcgill.ca/Networks/projects/monit-tarem-results.html

5.4 Measuring Performance

In this section, we define the measure of performance that we used to assess the novelty detectors evolved by DPC+NEAT, as well as the comparison approaches discussed above. In a particular novelty detection task, the job of a novelty detector is to determine whether or not a given data element falls outside of the normal class. Each data element is therefore assigned one of two labels: "normal" or "novel".

The similarities between novelty detection and classification allow us to assess the performance of novelty detectors using measures originally intended to evaluate two-class classifiers. Consider such a classifier which maps each data element to one of the two classes *positive* and *negative* [40]. For a particular data element, one of four outcomes may occur. The data element may belong to the positive class and be mapped correctly by the classifier. This is known as a *true positive*, "true" reflecting the accuracy of the mapping and "positive" identifying the class chosen by the classifier. Alternatively, this same data element may be misclassified as negative, which is referred to as a *false negative*. The remaining two outcomes, a negative data element correctly classified as negative and a negative data element misclassified as positive, are referred to as *true negative* and *false positive* respectively.

By considering the positive class to be the novel class, and the negative class to be the normal class, the four outcomes identified above also apply to the decision made by a novelty detector for a particular data element. Explicitly, in the context of novelty detection, these outcomes are defined as:

- *True Negative (TN):* A normal data element is correctly labelled as normal.
- *False Positive (FP):* A normal data element is incorrectly labelled as novel.
- *True Positive (TP):* A novel data element is correctly labelled as novel.
- *False Negative (FN):* A novel data element is incorrectly labelled as normal.

These outcomes are commonly presented as a *confusion matrix* [40], shown in table 5.2. As with a two-class classifier, a novelty detector has the goal of minimising false positives and false negatives, whilst maximising true positives and true negatives.

Some performance metrics that have been commonly associated with confusion matrices in the classification literature are listed below [40]:

Hypothesised Class	True Class	
	Normal	Novel
Normal	TN	FN
Novel	FP	TP
$\sum$	N	P

Table 5.2: A confusion matrix for two-class classification and novelty detection problems. $P = FN + TP$ denotes the total number of actual novel data elements, or positive examples, and $N = TN + FP$ denotes the total number of actual normal data elements, or negative examples.

$$\text{true positive rate} = \frac{TP}{TP + FN} \tag{5.8}$$

$$\text{false positive rate} = \frac{FP}{TN + FP} \tag{5.9}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{5.10}$$

$$\text{accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \tag{5.11}$$

The true positive rate is also sometimes referred to as the *hit rate*, *recall* or *sensitivity* of the classifier, whilst the false positive rate may be sometimes referred to as the *false alarm rate* [40]. At first glance, these metrics all appear to be potentially suitable for expressing the performance of the various novelty detectors in the experiments considered in this chapter. However, as we will now discuss, some of these metrics suffer from an important drawback which would make them a poor choice for this work.

5.4.1 Class Skew

One important consideration when choosing an appropriate performance metric is the sensitivity of that metric to *skews* in the class distribution of the data. Whilst this is not the case in the record datasets considered in this chapter, in a typical novelty detection task the number of normal data elements tends to grossly outweigh the number of novel data elements. This means that the data is usually heavily skewed towards the normal class. A performance metric which is sensitive to this skew will be biased towards the accurate labelling of those data elements from the normal class, and give values which vary with the ratio of normal to novel data elements.

For example, consider a novelty detection task with 100 data elements, of which only 5 are novel, and some "black box" function which consistently labels each and every data element presented to it as normal. This black box function would achieve a TN count of 95, a FN count of 5, and TP and FP counts of 0. According to the accuracy measure defined in equation 5.11 above, such a performance would achieve a score of 0.95. But this score is obviously misleading since, by always labelling every data element as normal, the black box is naturally incapable of performing novelty detection. Over a second dataset, this time with 50 normal data elements and 50 novel data elements, the same black box function would achieve a score of 0.5. Despite the fact that the black box function has not changed between the two datasets, it receives radically different performance scores. This is caused by the sensitivity of the performance metric to the different distributions of normal and novel data elements in the two datasets.

A performance metric can eliminate its sensitivity to class skew by restricting itself to values from a single column of the confusion matrix. Therefore, of the metrics defined in equations 5.8-5.11, the true positive and false positive rates are both insensitive to class skew, and so are suitable for our use. Each of these metrics consists of an entry of the confusion matrix, normalised by the sum of entries in the same column. As we will see next, this normalisation allows the metrics to be combined whilst still avoiding class skew.

5.4.2 ROC Analysis

The true and false positive rates defined in equations 5.8 and 5.9 can be used to define a two-dimensional space known as the Receiver Operating Characteristics (ROC) space [40]. For a given novelty detector, a curve can be generated in ROC space by varying one of its parameters, typically a novelty or decision threshold parameter, to obtain different true and false positive rates. Since neither of these rates are sensitive to class skew, this curve is robust to changes in the number of normal and novel data elements [40]. ROC curves have been used to analyse a number of novelty detectors in the literature, for example [3, 5, 21, 47, 111].

The ROC graph, illustrated in figure 5.4, has a number of interesting features [40]. The points (0,0) and (1,1) represent the performance of detectors which classify data elements as all normal or all novel respectively. Such detectors are not capable of differentiating between the two classes. The black box function considered previously would lie at point (0,0), since it is unable to label any data elements as novel. The point (0,1) represents the perfect detector, i.e. one which correctly classifies each and every data element. Its inverse, the point (1,0), represents

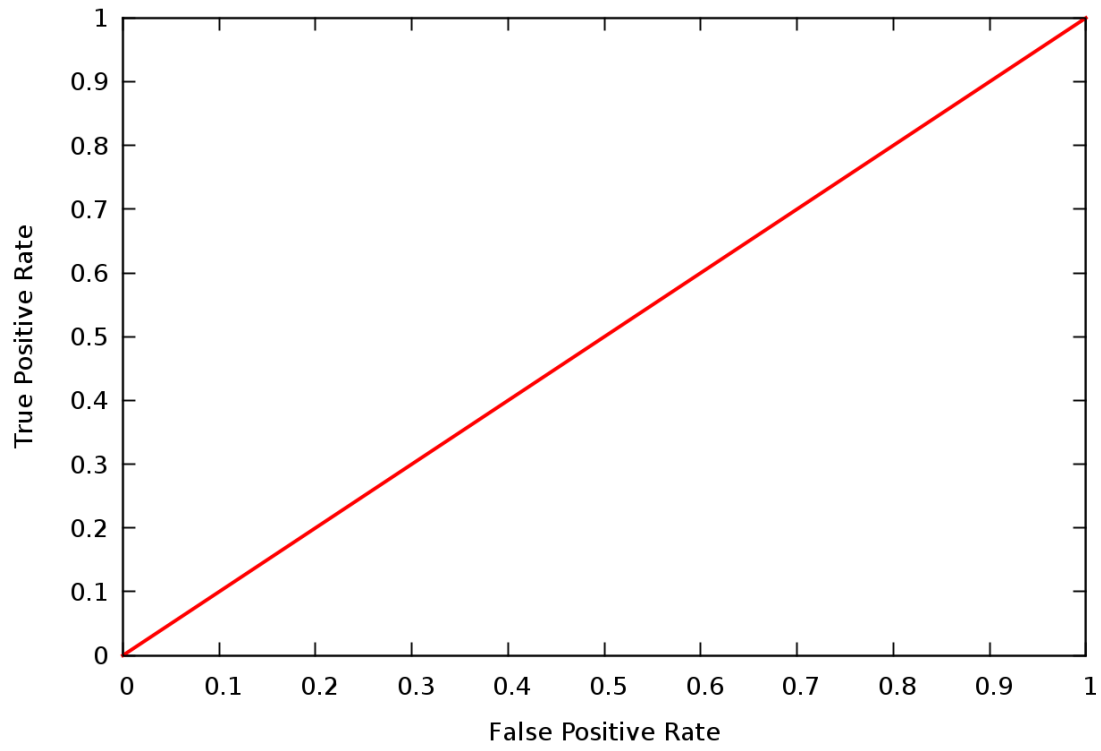


Figure 5.4: A basic Receiver Operating Characteristics (ROC) graph.

an "inverse" novelty detector, which is capable of perfectly separating the normal and novel classes but assigns opposite labels (i.e. labelling all normal data as novel and vice-versa) to every data element. The diagonal line in figure 5.4 represents the performance obtained through random guessing (i.e. labelling each data element as novel with probability $p = 0.5$). When considering the single performance of two detectors, informally one detector outperforms another if its corresponding point in ROC space is further towards the upper-left corner [40]. To obtain a single scalar performance measure from a ROC curve, one may calculate the Area Under the Curve (AUC) [40]. A detector that achieves a better average performance than another receives a higher AUC value. A performance obtained through random guessing (the diagonal line in figure 5.4) would receive an AUC value of 0.5. An additional performance measure associated with ROC curves is specificity, defined as [40]:

$$\text{specificity} = \frac{TN}{FP + TN} \tag{5.12}$$

$$= 1 - \frac{FP}{FP + TN} \tag{5.13}$$

$$= 1 - \text{false positive rate} \tag{5.14}$$

As mentioned before, to obtain a ROC curve for a novelty detector, one would vary one of the parameters of that detector. In our approach all parameters are chosen through evolutionary search, yet it would be possible to allow any one of these to instead be chosen by way of a ROC analysis. However, this would require the fitness function to use a performance metric based on such an analysis, such as AUC. This presents a significant disadvantage in that multiple evaluations, using different values for the varied parameter, would be required to determine the fitness of a particular candidate, which would significantly increase the amount of time required by DPC+NEAT to perform the evolutionary search. Because of this, we avoid such an approach in this chapter.

5.4.3 Performance Measure

As previously mentioned, the true and false positive rates are not affected by class skew and so are ideal performance metrics to use for novelty detection. This is confirmed by the fact that ROC analysis, which is based on these measures, has been used to examine many novelty detection approaches. However, DPC+NEAT requires that a *single* performance value be used to define the fitness of a particular novelty detector. Because a novelty detector must be constructed which *maximises* the true positive rate whilst *minimising* the false positive rate, the false positive rate must first be replaced with a value that can be maximised before the two rates can be combined. This inversion is equal to specificity, as defined in equation 5.12. Therefore, the two performance measures we consider are the true positive rate, or sensitivity, and the true negative rate, or specificity.

Furthermore, since the sensitivity and specificity measures are normalised (by the number of novel and normal data elements respectively), we can easily combine these into a single performance measure whilst maintaining indifference to class skew. This single performance measure is defined formally as:

$$PA = \frac{1}{2}\left(\text{sensitivity} + \text{specificity}\right) \tag{5.15}$$

$$= \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{FP+TN}\right) \tag{5.16}$$

In this chapter, the performance measure above defines the *predictive accuracy* of a particular novelty detector. During the evolutionary search, this describes the fitness of candidate networks.

Unlike measures based on ROC curves, this performance measure can be obtained after a single evaluation of a detector. Because the measure is an average of the true positive and true negative rates, it describes a line in ROC space rather than a particular point. For example, given the relationship between true negative and false positive rates described in equation 5.14, it can be shown that each point on the line shown in the ROC graph in figure 5.4 yields a value of 0.5 according to our measure. As the information describing the exact position of the relevant candidate network on this line is lost, it becomes harder to intuitively interpret this value. To solve this, we also examined the average mean true positive and true negative rates obtained by the detectors evolved by DPC+NEAT, and the mean rates achieved by each comparison approach.

5.5 Overfitting

To evaluate the fitness of a candidate novelty detector, DPC+NEAT must examine the performance of that detector in labelling both normal and novel data. If normal data were used alone, then detectors that always label all presented data elements as normal, such as the black box function considered earlier, would be given high fitness scores, which is clearly undesirable. However, whilst it is necessary to use both kinds of data, doing so could potentially lead to the problem of *overfitting*. This is when a particular model, such as a neural network or statistical classifier, learns, or fits itself to, a dataset too precisely, resulting in a lack of generalisability to new data.

A number of techniques can be used to determine how well a particular novelty detector generalises to unseen data. The simplest approach, the *holdout method*, partitions the available data into a training set and a test set [113]. The novelty detector is then trained using data from the training set alone, after which its performance is evaluated solely over the test set. However, since a subset of the available data is reserved for testing, it restricts the amount of data that can be used to train the detector, which may result in a worse performance estimator [113]. The performance estimate given by the method may vary depending on how the data has been partitioned into the two sets.

An alternative to the holdout method is the *k-fold cross validation technique*. In this approach, the data is partitioned into k approximately equally sized subsets [113]. The novelty detector is then evaluated k times, where, in each evaluation, one of the k partitions is chosen as the test set and the remaining $k-1$ partitions are used to train the detector. Each partition is

used as the test set in exactly one of the k evaluations. The results over the k evaluations may then be combined to give a performance estimator for the detector. This technique increases the amount of data available for training, and offers less variance since all data elements feature in one of the k test sets. However, it also is more computationally intensive as it requires k evaluations to be performed. A typical value used for k is 10, since this has been shown to produce a good performance estimator over numerous datasets [125].

A special case of k-fold cross validation is N-fold cross validation, where N is the total number of data elements available. This is more commonly known as the leave-one-out cross validation method [113]. This uses the maximum possible number of data elements to train the novelty detector, since only one element is used in each test set. However, it is again computationally expensive and, due to the extremely small size of the test set, tends to provide a performance estimator with high variance [113].

In a neural network, overfitting can result from the structure of the network being overly complex for a particular problem [62]. However, since NEAT aims to find solutions with minimal complexity [108], we hypothesised that novelty detectors evolved by DPC+NEAT will not be as susceptible to the overfitting problem. We tested this hypothesis when examining the performance of DPC+NEAT over the six novelty detection tasks, using two of the techniques described above as follows. First, when evolving novelty detectors, DPC+NEAT was permitted to use only a subset of the normal and novel elements in the test dataset for each task. This enabled us to test evolved detectors on data that was unseen during the evolutionary phase. Second, for the tasks based on record datasets, we evaluated each detector (including the comparison approaches) using the 10-fold cross validation technique. This tests for overfitting of the normal class by the detectors during the training phase. However, 10-fold cross validation cannot be used on time series datasets, since moving a subset of a time series to a different part of that series would introduce *artifacts*. That is, an unusual change would occur between the first data point of the subset and the previous data point in the series. This artifact may then be highlighted by a novelty detector as an anomaly. Because of this, we used the holdout method again when evaluating detectors over the time series datasets.

5.6 Statistical Comparison

In evaluating DPC+NEAT, we wanted to determine whether or not this approach was able to evolve detectors in the six novelty detection tasks which outperform each of the comparison

approaches. We consider an evolved novelty detector to outperform a comparison approach on a given task if it achieves a better average performance. Statistically, this may be phrased as: is the *expected value*, or *population mean*, of the probability distribution of results achievable by a particular evolved novelty detector on a particular dataset greater than that of each comparison approach?

To answer this question, we need to establish whether or not there is a difference between the population means of the probability distribution of results for the two detectors being compared. If a difference exists, then the approach which gives the best performance is that with the greater population mean. Based on this, we may construct two hypotheses:

- H_0: There is no difference between the population means, i.e. the two detectors are likely to give the same performance.
- H_1: A difference does exist between the population means, meaning that the better performing detector is the one with the highest population mean.

The first hypothesis, H_0, is known as the *null hypothesis* [24], whilst the second hypothesis, H_1, is referred to as the *alternate hypothesis*. In comparing the two detectors, we must decide whether or not to reject H_0 in favour of H_1. In making this decision, there is always a probability that we will commit one of two kinds of error [24]:

- *Type I*: We reject the null hypothesis H_0 when it is in fact correct.
- *Type II*: We do not reject the null hypothesis H_0 when it is in fact false.

Some authors, for example Stibor *et al.* [111], have compared the performance of different approaches by calculating and comparing the mean and standard deviation of the results observed from each approach over a certain number of runs. This assumes the *law of large numbers*, which states that the mean of a sample tends to the population mean as the size of the sample tends to infinity. However, the accuracy of the approximation of the population mean achieved by the sample mean is unknown [24]. It could be that the observed differences in the sample means occurred by chance, meaning that we may commit a type I error if we reject the null hypothesis based on this information. To minimise the chance of committing a type I error, we need to test if this difference is *statistically significant*. Such a test is called a significance test [24].

A simple way of testing that the two sets of results are significantly different is to obtain an estimate of the interval into which the population mean for each result set has a high probability

p of falling. This estimate is known as the *confidence interval* for the population mean [24], and may be determined when the sample mean follows a Gaussian distribution [24]. If the confidence intervals obtained for both results sets do not overlap, then one may reject the null hypothesis with a probability $1 - p$ of incurring a type I error. However, if the intervals do overlap then a more sensitive significance test is required, such as Student's t-test. This again makes the assumption that the sample mean follows a Gaussian distribution.

These significance testing methods are parametric methods, since they make assumptions about the properties of the probability distribution of the data. But in most cases this distribution is unknown and so it appears unclear whether or not these assumptions hold. However, for large sample sizes, the assumption that the sample mean of the data follows a Gaussian distribution can be justified by the *central limit theorem*, which states that the distribution of the sample mean tends to a Gaussian distribution as the size of the sample tends to infinity [24]. But the number of samples required before the distribution of the sample mean can sufficiently approximate a Gaussian distribution is still unknown.

Alternatively, one can avoid the Gaussian assumption by using a non-parametric significance test, also referred to as a *distribution-free* test [24], which typically tests the data in ranked form. Two such tests are the Mann-Whitney U-test and the Wilcoxon signed ranks test. The first of these tests is suitable when the comparison between two detectors is *independent*, that is each detector is run over a completely different dataset. The second test is suitable for use when the same dataset is used to test each detector; the comparison is said to be a *paired* comparison since, on each run, the two detectors are connected by the chosen dataset. In this work, we therefore used the Wilcoxon signed ranks significance test. We now describe this test in more detail.

The Wilcoxon signed ranks test operates on both the sign and the magnitude of the difference in the results obtained by the detectors [24]. The test proceeds as follows. For each run, the difference in the result obtained for each approach is calculated and stored. Then, the results are ranked according to the absolute values of these differences (lowest first, with tied ranks being averaged), with each rank value subsequently being given the sign of the corresponding difference value. Those ranks with positive sign are then summed to give a quantity T_+. If the null hypothesis H_0 is correct, then each rank is equally likely to receive either a positive or negative sign since the differences in results are equally likely to be positive or negative. Therefore, we would expect T_+ to constitute the sum of approximately half the ranks and so have a high value. If the number of runs performed exceeds 16, then T_+ will have a probability

distribution that is approximately Gaussian [24]. This can be exploited in order to determine the probability, or p-value, of obtaining a value greater than or equal to T_+ assuming that H_0 is true. This is done by first calculating the z-score of T_+, thus transforming T_+ such that its distribution becomes a standard Gaussian distribution, and then determining the p-value from this distribution. If the p-value is below a certain threshold, then the null hypothesis may be considered sufficiently unlikely and be rejected. In this case, we may say that the difference in the performance of the two detectors is statistically significant and infer from the sign of the z-score which of the two detectors gives the better performance. With a threshold of 0.01, we may assert that one detector outperforms the other with a confidence level of 99%.

As well as determining whether or not a given novelty detector constructed by DPC+NEAT outperforms a comparison approach, we also wanted to determine whether or not DPC+NEAT *tended* to evolve superior novelty detectors. This means that we had to compare a number of novelty detectors evolved by DPC+NEAT against the comparison approach. Each detector in the sample can be compared against the comparison approach using the Wilcoxon signed ranks test described above. In a particular novelty detection task, we might find that, after evolving 100 novelty detectors, the statistical test confirms that each detector outperforms the comparison approach with a confidence level of 99%. That is, for each detector, the probability that the null hypothesis is false is 0.99. However, the probability of the null hypothesis being false for *all* of the evolved detectors is $0.99^{100} = 0.366$. Therefore, we can only state that all of the evolved detectors outperform the comparison approach with a confidence level of 36.6%.

To allow us to make the above statement with a confidence level of 99%, we must apply a correction to the threshold used in each of the 100 Wilcoxon signed ranks tests performed. For a particular threshold p, this correction, known as the *Bonferroni correction*, defines the corrected threshold p' as [1]:

$$p' = \frac{p}{N} \tag{5.17}$$

where N is the number of tests being performed. In our example above, to obtain a confidence level of 99% over the 100 tests, the corrected threshold p' that should be used is $\frac{0.01}{100} = 0.0001$. However, whilst the Bonferroni correction decreases the probability of a type I error, some authors regard it is being a very conservative correction [1]. For this reason, all statistical tests in this chapter were performed both with and without the Bonferroni correction. It is also worth noting that, aside from the Bonferroni correction, a range of different techniques for

testing of multiple hypotheses have been proposed in the literature [102]. However, these are not considered further in this work.

5.7 Experimental Setup

We use the following experimental setup to evaluate DPC+NEAT. This aims to provide statistically robust results as well as to guard against overfitting.

5.7.1 Normalisation

Many approaches to novelty detection make the assumption that the input data has been normalised to a particular range, usually $[-1, 1]^n$ or $[0, 1]^n$, where n denotes the dimensionality of the data. For example, GWR makes the assumption that all input data lies in the range $[-1, 1]^n$ [85]. Ahmed *et al.* also normalised input data to this range in their experiments on computer network traffic flows [3]. Ji also noted that data is usually normalised before being passed to V-detector [66] and Stibor *et al.* reported normalising the input data to the range $[0, 1]^n$ in their experiments with both 1-SVM and V-detector [111].

In the case of DPC+NEAT, it is feasible for networks to be evolved which automatically transform the data as required to achieve a good performance. Intuitively, this should allow novelty detectors to be evolved for tasks in which no normalisation is applied to the data. Furthermore, Hosoya *et al.* did not report normalising the inputs, drawn from a standard Gaussian distribution, to their prototypical DPC model [63]. However, to avoid creating a disadvantage for the comparison approaches, we normalised all datasets to the range $[0, 1]^n$ before performing the evaluation.

5.7.2 Tasks based on Record Datasets

For those novelty detection tasks based on record datasets, we used the 10-fold cross validation method as described in section 5.5. For each of the datasets, the normal and novel data elements were segregated into different sets. The normal set was then randomly divided into 10 subsets and 10 training sets were generated, where each set consisted of all but one of the 10 subsets (with a different subset being left out each time). For each of the 10 training sets, a corresponding test set was created, consisting of the left out subset as well as all novel data elements. Unfortunately, this resulted in test sets being produced with more novel than normal data elements, further

emphasising the drawback identified in section 5.2.1 of using classification benchmark datasets to evaluate novelty detectors. Each of the 10 training and test set pairs were used 10 times, with different orderings of the data elements, giving 100 pairs in total. All approaches were tested on the same 100 pairs.

For each of the datasets, we assessed each novelty detector as follows. The detector was run over each of the 100 training and test set pairs. The performance of that novelty detector was then judged over all 100 runs. For DPC+NEAT, we evolved one novelty detector for each of the 100 training sets. We then evaluated each of the 100 evolved novelty detectors individually using the same method employed for the comparison approaches. Finally, we compared the performance of each evolved detector with each of the comparison candidates, using the Wilcoxon signed ranks significance test. Each test was performed both with and without the Bonferroni correction.

As we have previously discussed in section 4.5.3, it is undesirable to use an online learning approach when processing record datasets which contain no temporal information. This is because an online learning mechanism would be sensitive to the ordering of the records, which is not guaranteed. Therefore, in the experiments in this chapter which involve record datasets, the novelty detectors evolved by DPC+NEAT were permitted to learn only during the training phase. In the test phase, the anti-Hebbian learning rule was disabled, meaning that detector's plastic synaptic weights became fixed. As discussed in section 4.5.4, the thresholding activation function used in the hidden neurons of evolved detectors allowed those detectors to retain a nonlinear functionality in the test phase, despite the loss of their adaptive behaviour.

5.7.3 Time Series Tasks

As explained in section 5.5, we had to use the holdout method in place of the 10-fold cross validation technique for those novelty detection tasks based on time series datasets, so as to avoid the introduction of artifacts. For a given time series, the first n data points were taken as the training set, with the remaining data points used to form the test set. For the Mackey-Glass and Hard Disk tasks, we generated 100 different time series. For the Mackey-Glass task, each generated series had a different randomly chosen starting condition. For the Hard Disk task, each data point was based on a randomly chosen temperature value. Each detector was evaluated over the 100 series in each task, and DPC+NEAT detectors were compared with comparison approaches using the Wilcoxon signed ranks test. For the ECG time series, we ran each detector

over the series once only. This is because the performance of a given novelty detector evolved by DPC+NEAT is always consistent in multiple runs over the same time series, since the neural network has no stochastic element. In each time series task considered in this chapter, the detectors evolved by DPC+NEAT were permitted to perform online learning during the test phase.

5.7.4 Fitness Evaluation

To evaluate the fitness of candidate novelty detectors during DPC+NEAT's evolutionary search, we used the following method. In the tasks based on record datasets, each run of DPC+NEAT was associated with one of the 100 generated training and test sets, with each training and test set pair being used in exactly one run. Each candidate network evolved by DPC+NEAT was trained using the first 90% of the data elements from associated training set. The remaining 10% of elements from the training set were then combined with the first 10% of the data elements from the corresponding test set to yield an *evolutionary test set*. This was done to ensure that, in addition to novel elements, the evolutionary test set had a sufficient number of examples from the normal class. The fitness of the candidate detector was then determined based on the predictive accuracy score, as defined in equation 5.15, achieved over this set. The best novelty detector evolved by DPC+NEAT for the run was finally evaluated using all 100 training and test sets in their entirety, with the resulting averaged score being reported as the mean predictive accuracy score for that detector.

For the Mackey-Glass and Hard Disk time series tasks, each run of DPC+NEAT was again uniquely associated with one of the 100 generated training and test set pairs. The entire training set was used to train candidate novelty detectors during the evolutionary phase, with an evolutionary test set consisting of the first 10% of the data points from the associated test set being used to determine the fitness of the detectors. Because of the much larger size of the datasets considered in these tasks, the first 10% of the generated test sets always contained an ample number of normal data points, meaning that it was sufficient to use only data points from the generated test set in the evolutionary test set. Since the ECG task is based on a single real-world dataset, each run of DPC+NEAT was forced to use the same training and test sets for this task. Here, so as to ensure that enough examples of anomalous beats were included, the first 25% of the data points from the original test set were used as the evolutionary test set. In the first two time series tasks, the best novelty detector evolved by DPC+NEAT in each run

Approach	Parameter	Description
GWR	a_T	Activity Threshold
	h_T	Habituation Threshold
	ϵ_b	Learning Rate (best node)
	ϵ_n	Learning Rate (neighbour nodes)
	τ_b	Habituation Parameter (best node)
	τ_n	Habituation Parameter (neighbour nodes)
	α	Habituation Parameter
	y_0	Initial synapse strength
	t	Novelty Threshold
V-detector	r_s	Self Threshold
	c_0	Target Coverage
	l	Maximum Number of Detectors
1-SVM	ν	Control parameter for support vector count and number of training errors
	$K(.,.)$	Kernel Function
	γ	Kernel Parameter
	C	Constraint in Lagrange Formulation
	t	Tolerance (Stopping Condition Parameter)
KOAD	ν_1	Orange alarm threshold
	ν_2	Red alarm threshold
	$K(.,.)$	Kernel function
	σ	Kernel parameter
	d	Match threshold
	L	Number of previous measurements to consider for dictionary element evaluation
	ϵ	Constant for resolving orange alarms
	ι	Time steps to keep track of the contribution of an input vector for which an orange alarm was raised
	γ	Forgetting factor

Table 5.3: The application-specific parameters for the four comparison approaches used in this work.

was evaluated over all 100 generated training and test set pairs to obtain the reported mean predictive accuracy score for that detector. In the ECG task, the best evolved detector was evaluated once over the entire time series to yield its predictive accuracy score.

5.7.5 Parameters

The parameters used for DPC+NEAT in this work retained the same values as they had in the experiments performed in chapter 4, which are given in appendix A. However, the comparison approaches also required a number of parameters to be set, whose optimal values depend on the particular novelty detection task.

The application-specific parameters for each comparison approach are shown in table 5.3.

Approach	Parameter	Record Tasks		Time Series Tasks	
		Value	Source	Value	Source
GWR	h_T	0.1	[85]	0.1	[85]
	ϵ_b	0.2	[85]	0.1	[85]
	ϵ_n	0.05	[85]	0.001	[85]
	τ_b	0.333	[85]	0.333	[85]
	τ_n	0.143	[85]	0.143	[85]
	α	1.05	[85]	1.05	[85]
	y_0	1	[85]	1	[85]
V-detector	c_0	0.99	[66]	0.99	[66]
	l	1000	[66]	1000	[66]
1-SVM	ν	0.05	[111]	N/A	
	$K(.,.)$	RBF	[111]		
	γ	$1/dim$	[23]		
	C	1	[23]		
	t	0.001	[23]		
KOAD	$K(.,.)$	N/A		Gaussian	[4]
	σ			0.6	[4]
	d			0.9	[4]
	L			100	[4]
	ϵ			0.2	[4]
	ι			20	[4]
	γ			1	[4]

Table 5.4: The parameters for each comparison approach not found experimentally. Each value was taken from the literature as indicated by the source column. *dim* is defined as the dimensionality of the input space.

For Marsland's GWR [85], we introduced an additional threshold parameter t to allow the quantitative response given by GWR to be translated into a binary decision value (i.e. normal or novel). We have also omitted those parameters that are unique to LIBSVM from the list given for 1-SVM. These omitted parameters retain default values [23]. When faced with the application-specific parameters of a given novelty detector, the user would be forced to discover for themselves the best values to use in their specific scenario. Therefore, in this work we determined suitable parameter values for each comparison approach "experimentally" or, in other words, by trial-and-error.

A manual search over the parameter space is naturally limited in its ability to effectively explore that space. In particular, it is difficult to explore the various relationships *between* parameters. Here, we limited ourselves to varying only the threshold parameters for each method, which are critical in novelty detection tasks. The remaining parameters were taken from the literature, as shown in table 5.4. This includes all parameters for the 1-SVM approach, which were set to the same values used by Stibor *et al.* in their experiments over the Biomedical and

Dataset	GWR		V-detector	KOAD	
	a_T	t	r_s	ν_1	ν_2
Biomed	0.97	0.1	0.05	N/A	
Breast	0.92	0.1	0.2	N/A	
Iris-S	0.92	0.1	0.21	N/A	
Iris-Ve	0.88	0.3	0.4	N/A	
Iris-Vi	0.92	0.1	0.8	N/A	
Mackey	0.99	0.3	0.1	0.02	0.03
Hard Disk	0.90	0.1	0.11	0.001	0.05
ECG	0.92	0.3	0.45	0.01	0.25

Table 5.5: The parameters for the comparison approaches that were found experimentally. For each dataset, the parameters of each approach were optimised manually over 10 trial runs.

Iris datasets [111]. For GWR, the threshold parameters are the activity threshold a_T, which decides the level of activity required to indicate a good association between an input pattern and a map field node, and our threshold parameter t, which converts the continuous output of GWR into a binary decision value. For V-detector, the only threshold parameter is the self threshold r_s, which decides the maximum distance that an input pattern may be from a detector for a match to occur. For KOAD, the threshold parameters are ν_1 and ν_2, which define the maximum tolerable dissimilarity between a data point and the model of the normal class before that data point is considered either unusual or novel.

For each dataset, we attempted to find suitable values for the threshold parameters of each comparison candidate over 10 trials. Each trial used the same data as that used by DPC+NEAT during its evolution phase. In order to reduce variability, each approach was run 10 times over the subset and then evaluated based on its averaged performance. At the end of the trial, we manually perturbed the parameters in an attempt to improve performance. The parameter values yielding the best performance were then selected for use in the comparison experiments. The best parameter values found for each approach are given in table 5.5.

5.8 Results

We now present the results obtained for DPC+NEAT and the comparison approaches over the six novelty detection tasks. We first consider the three tasks based on record datasets and then move on to the three time series tasks.

Dataset	Avg. Mean Predictive Accuracy	St. Dev.	Best Mean Predictive Accuracy	Worst Mean Predictive Accuracy	Average Detector Complexity		
					No. Hidden Nodes	No. Plastic Synapses	No. Fixed Synapses
Biomed	0.771	0.046	0.852	0.588	3.1	3.33	7.98
Breast	0.942	0.024	0.972	0.865	3.31	6.34	9.12
Iris-S	0.947	0.057	1.000	0.763	0.18	3.88	2.88
Iris-Ve	0.904	0.070	0.970	0.605	1.67	3.25	5.52
Iris-Vi	0.901	0.073	0.985	0.546	2.61	2.92	6.79

Table 5.6: The performance achieved by DPC+NEAT for the record datasets.

Dataset	GWR		V-detector		1-SVM	
	Mean Predictive Accuracy	St. Dev.	Mean Predictive Accuracy	St. Dev.	Mean Predictive Accuracy	St. Dev.
Biomed	0.756	0.060	0.814	0.044	0.665	0.026
Breast	0.914	0.028	0.962	0.020	0.955	0.012
Iris-S	0.921	0.111	0.835	0.092	0.970	0.064
Iris-Ve	0.879	0.085	0.647	0.060	0.895	0.051
Iris-Vi	0.783	0.126	0.843	0.087	0.804	0.056

Table 5.7: The performances achieved by the comparison approaches for the record datasets.

Dataset	DPC+NEAT		GWR		V-detector		1-SVM	
	Avg. Mean TP Rate	Avg. Mean TN Rate	Mean TP Rate	Mean TN Rate	Mean TP Rate	Mean TN Rate	Mean TP Rate	Mean TN Rate
Biomed	0.722	0.821	0.772	0.740	0.884	0.744	0.393	0.938
Breast	0.932	0.952	0.998	0.831	0.986	0.938	0.962	0.947
Iris-S	0.957	0.936	0.980	0.862	1.000	0.670	1.000	0.940
Iris-Ve	0.938	0.870	0.921	0.836	0.992	0.302	0.889	0.895
Iris-Vi	0.938	0.865	0.870	0.696	0.992	0.694	0.688	0.920

Table 5.8: The average mean true positive and true negative rates achieved by the 100 novelty detectors evolved by DPC+NEAT and the mean rates achieved by each comparison candidate for the record datasets.

Dataset	GWR			V-detector			1-SVM		
	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than
With No Correction ($p = 0.01$):									
Biomed	53	29	18	3	31	66	95	3	2
Breast	83	6	11	7	20	73	22	22	56
Iris-S	53	22	25	92	6	2	36	19	45
Iris-Ve	60	27	13	98	1	1	59	19	22
Iris-Vi	87	10	3	72	18	10	86	9	5
With Bonferroni Correction ($p = 0.0001$):									
Biomed	47	38	15	1	46	53	95	3	2
Breast	80	9	11	5	29	66	14	37	49
Iris-S	47	39	14	87	12	1	28	33	39
Iris-Ve	57	32	11	96	4	0	43	44	13
Iris-Vi	80	17	3	70	22	8	81	15	4

Table 5.9: Statistical comparison of the novelty detectors evolved by DPC+NEAT with the comparison approaches for the record datasets, both with and without Bonferroni correction (99% confidence level). "Detectors Equiv. To" refers to the number of novelty detectors evolved by DPC+NEAT whose performances are not significantly different to the respective comparison candidate.

5.8.1 Record Datasets

Tables 5.6 and 5.7 show the results obtained by DPC+NEAT and the comparison approaches respectively for the three novelty detection tasks based on record datasets. For each dataset, apart from Biomedical, the novelty detectors constructed by DPC+NEAT achieved a consistently high average mean performance, with the best detectors achieving a predictive accuracy of at least 0.97. In each dataset, the best novelty detector evolved by DPC+NEAT outperformed every comparison approach. Table 5.8 compares the average mean true positive (TP) and true negative (TN) rates achieved by the 100 detectors evolved by DPC+NEAT with the mean rates achieved by the comparison approaches. In general, DPC+NEAT tended to achieve similar values for both the TP and TN rates, unlike the comparison approaches which sometimes achieved a higher performance in one rate than the other. This may be the result of the fitness function used by DPC+NEAT, defined in equation 5.15, which gives equal weighting to each of the two rates.

We note from table 5.7 that the performances of 1-SVM over the Iris datasets are slightly lower than those reported by Stibor *et al.* [111], with the results in table 5.8 generally indicating a lower TN rate. The results obtained for V-detector are also significantly worse here than those reported by both Stibor *et al.* and Ji over the Iris datasets [66, 111]. We also observe different results for V-detector over the Biomedical dataset, even though we used the same self-radius r_s of 0.05. Further investigation revealed that this difference in results for both approaches was due to fact that we used a different experimental setup. In both Stibor *et al.*'s and Ji's work, the approaches were trained using 100%, 50% and (for the Biomedical dataset only) 25% of the data elements from one class, drawn randomly from the dataset. The approaches were then tested over *all* data elements from the dataset, including those already shown during the training phase. When we tested both approaches (using the same parameters as those used by Stibor *et al.* and Ji) under this experimental setup, we obtained similar results to those reported by both Stibor *et al.* and Ji for both 1-SVM and V-detector over the Iris and Biomedical datasets. However, we consider our experimental setup to be more robust against overfitting, since each test set contains only those normal and novel data elements not present in the corresponding training set.

Table 5.9 presents the results of the statistical comparison, using the Wilcoxon signed ranks test both with and without the Bonferroni correction, between the novelty detectors evolved by DPC+NEAT and the comparison approaches. With the Bonferroni correction, at least 47

detectors evolved by DPC+NEAT were shown to outperform GWR in each task, with this number reaching as high as 80 for the Breast Cancer and Iris-Virginica datasets. V-detector was shown to outperform the majority of the detectors evolved by DPC+NEAT for the Biomedical and Breast Cancer datasets, but not for the three Iris datasets. Here, the majority of detectors evolved by DPC+NEAT gave a statistically significantly higher performance. The comparison against 1-SVM also shows mixed results. For the Biomedical and Iris-Virginica datasets, the vast majority of networks evolved by DPC+NEAT gave a statistically significantly better performance than 1-SVM. However, for the Breast Cancer and Iris-Setosa datasets, the majority of the evolved networks were either as good as or worse than 1-SVM. For the Iris-Versicolor dataset, with the Bonferroni correction applied, just under half of the networks evolved by DPC+NEAT outperformed 1-SVM.

As we anticipated, the complexity of the evolved networks varied for each dataset. The average number of hidden neurons used by the evolved networks is highest for the Biomedical and Breast Cancer datasets. The results obtained by each of the three comparison approaches suggest that the Biomedical dataset represents a difficult novelty detection task. This would therefore justify a higher level of complexity from the novelty detectors evolved by DPC+NEAT. However, it is less clear if the detectors evolved for the Breast Cancer dataset have a justifiable level of complexity, since the performances achieved by the comparison approaches indicate that the dataset represents an easier novelty detection task. The networks with the lowest average number of hidden neurons and synapses were those evolved for the Iris-Setosa dataset. As shown in section 5.2.1, this dataset represents a trivial novelty detection problem, since the normal class is linearly separable from the two novel classes.

To gain further insight into the complexity of the evolved novelty detectors, we examined the properties of the best performing detector evolved by DPC+NEAT for each dataset. Figures 5.5 and 5.6 show the neural network structure of the best detectors evolved by DPC+NEAT for the Biomedical and Breast Cancer datasets respectively. Both detectors have a complex network structure, with the Biomedical detector having 8 hidden neurons, 19 fixed synapses and 5 plastic synapses, and the Breast Cancer detector having 9 hidden neurons, 19 fixed synapses and 8 plastic synapses. Intriguingly, the novelty detector evolved for the Breast Cancer dataset has disconnected 3 of its 9 input nodes, demonstrating a degree of feature selection. Figures 5.7, 5.8 and 5.9 show the structure of the best detectors evolved for the Iris-Setosa, Iris-Versicolor and Iris-Virginica datasets respectively. For the Iris-Setosa dataset, the best evolved detector is linear in nature, which is a justifiable level of complexity for this trivial linearly separable

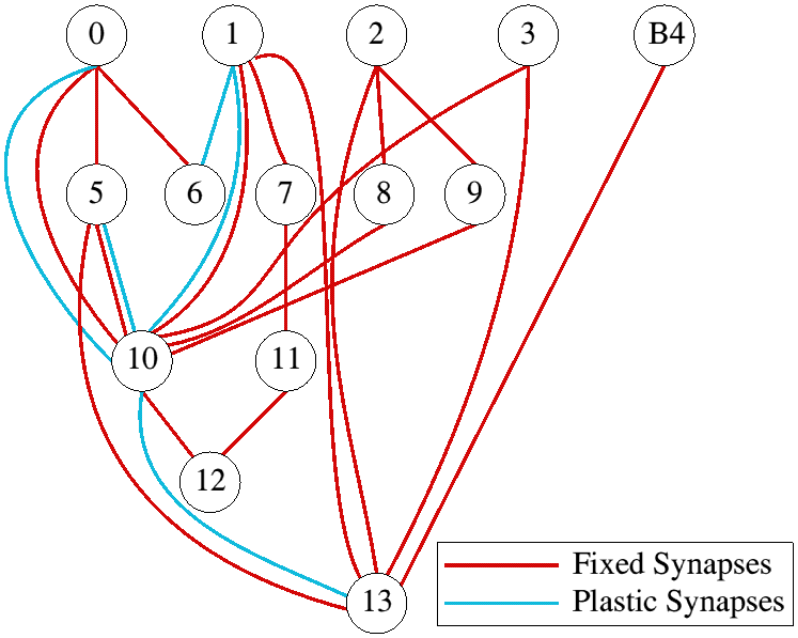


Figure 5.5: The structure of the best evolved novelty detector for the Biomedical dataset. Nodes 0,1 and 2 represent input neurons, whilst node 13 is the output neuron. Node B4 represents the bias node, which outputs a constant value of +1. Hidden neurons 6 and 12 do not have any outgoing synapses and so represent redundant structure.

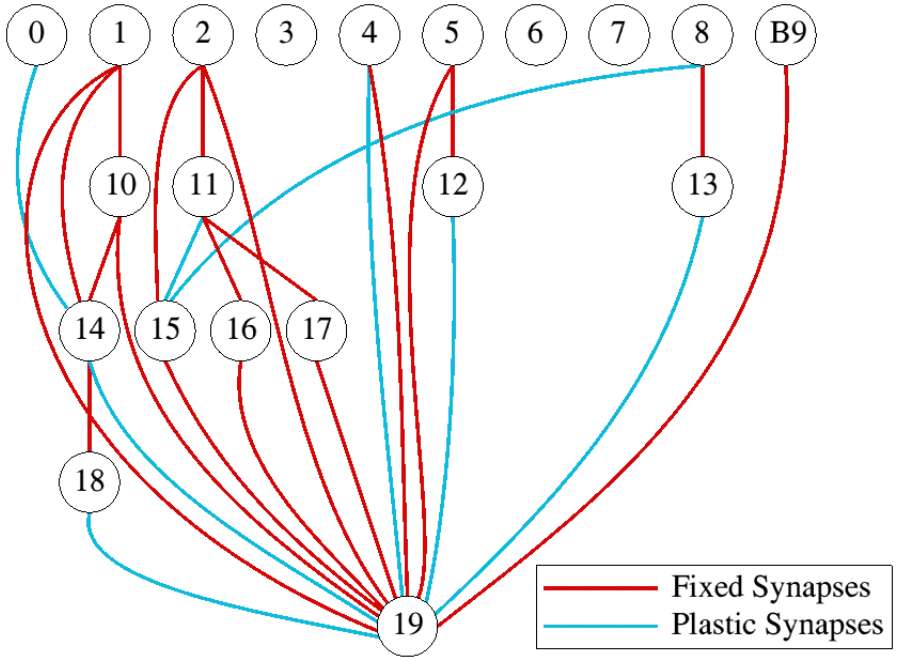


Figure 5.6: The structure of the best evolved novelty detector for the Breast Cancer dataset. Input neurons 3, 6 and 7 are completely disconnected from the network, demonstrating a degree of feature selection.

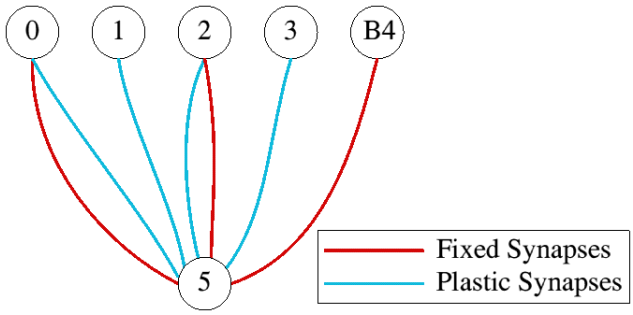


Figure 5.7: The structure of the best evolved novelty detector for the Iris-Setosa dataset. This detector has no hidden neurons and so demonstrates a linear behaviour.

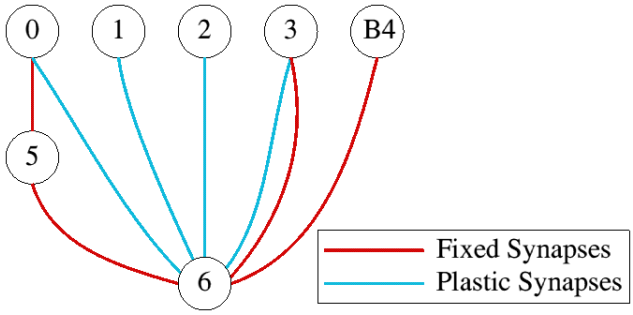


Figure 5.8: The structure of the best evolved novelty detector for the Iris-Versicolor dataset. Despite having a single hidden neuron, this detector is still linear in nature since the hidden neuron always fires.

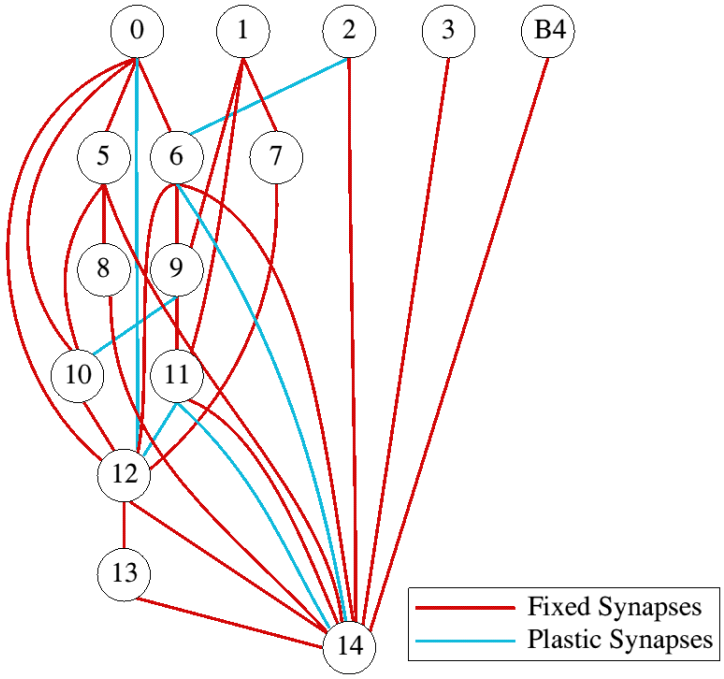


Figure 5.9: The structure of the best evolved novelty detector for the Iris-Virginica dataset. Despite the increased complexity of this detector, it only achieves a slightly better separation between the versicolor and virginica classes than the best detector for the Iris-Versicolor dataset (shown in figure 5.8).

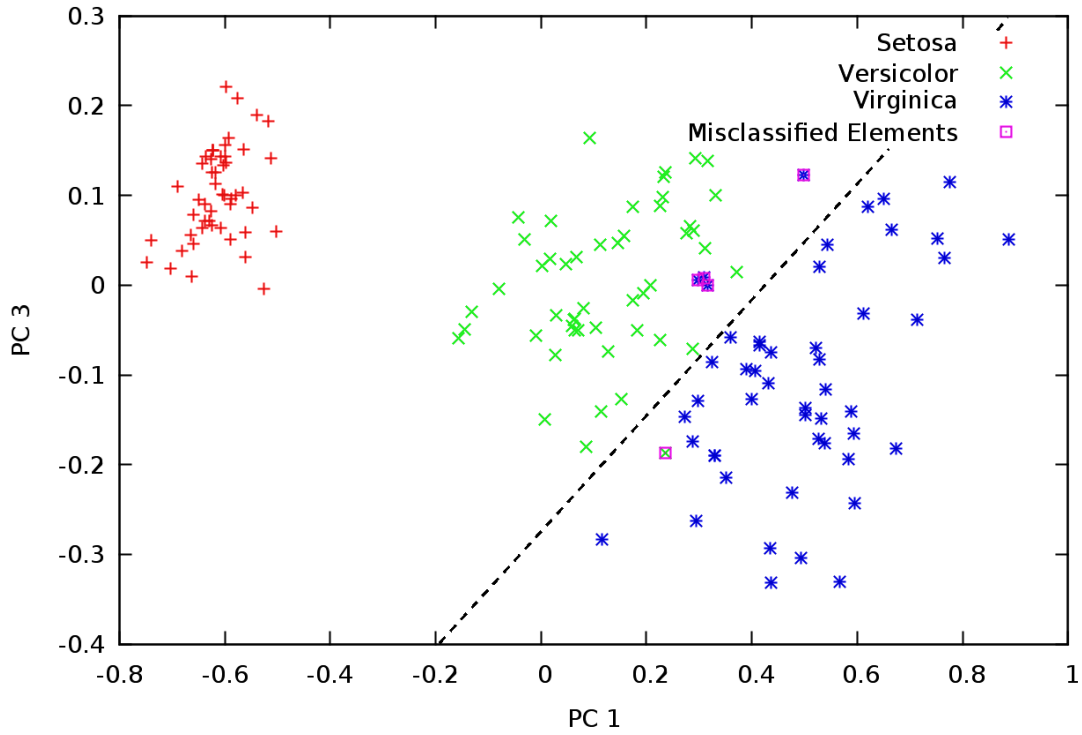


Figure 5.10: The positions of the data elements in the Iris-Versicolor dataset misclassified by the best evolved novelty detector for that dataset. This visualisation is taken along the first and third principal components of the dataset, as in figure 5.2. The dashed line sketches the decision boundary suggested by the positions of the misclassified data elements.

problem. As expected, this detector is able to accurately label every data element in the dataset, thus achieving a predictive accuracy score of 1 as shown in table 5.6. The best detector evolved for the Iris-Versicolor dataset has a similar level of complexity but with a single hidden neuron. However, upon closer examination we found that this hidden neuron is guaranteed to fire for any input, since each attribute of the data is normalised to the range $[0, 1]$. Therefore, this detector is also linear. But since the output neuron is thresholded on magnitude alone, the detector is able to establish two decision boundaries in the input space. The first decision boundary completely separates the versicolor class from the setosa class, whilst the second boundary attempts to separate the versicolor and virginica classes. However, the nonlinearity of the true separation between these classes cannot be captured, thus resulting in some misclassification of data elements. When examined along the directions of the first and third principal components, as shown in figure 5.10, the positions of the misclassified data elements indicate that the decision boundary between the versicolor and virginica classes is positioned to exploit the near-linearity of the separation between these classes. The best detector found for the Iris-Virginica dataset

achieves a slightly better separation between the versicolor and virginica classes, misclassifying only three data elements from the versicolor class. This network is again very complex, which is surprising given the simplicity of the network found for the Iris-Versicolor dataset. However, as indicated in table 5.6, we found that on average the majority of the other novelty detectors evolved had a complexity only slightly greater than those detectors evolved for the Iris-Versicolor dataset, with the second best network (achieving a mean predictive accuracy score of 0.973) having only 3 hidden neurons, 6 fixed synapses and 4 plastic synapses.

5.8.2 Time Series Tasks

Tables 5.10 and 5.11 show the results obtained by DPC+NEAT and the comparison approaches respectively for the Mackey-Glass and Hard Disk time series datasets. DPC+NEAT demonstrated a generally consistent performance over both datasets, with the best evolved novelty detectors again outperforming the comparison approaches. This is most notable for the Hard Disk dataset, where the comparison approaches achieved very poor mean predictive accuracy scores of between 0.485 and 0.504, compared with an average of 0.818 achieved by the detectors evolved by DPC+NEAT. Table 5.12 shows the average mean TP and TN rates achieved by the evolved detectors and the mean rates achieved by each comparison approach for each time series. For the Mackey-Glass dataset, both DPC+NEAT and KOAD achieved similar mean TP and TN rates, whilst GWR and V-detector demonstrated a performance skewed towards the TN and TP rates respectively. For the Hard Disk dataset, both GWR and KOAD struggled to identify novel events, with V-detector achieving a low, albeit consistent, performance for both the mean TP and TN rates. The average performance of the evolved novelty detectors for this series also appears to be skewed to some extent towards the TP rate.

Table 5.13 again compares the performance of the detectors evolved by DPC+NEAT with that of the comparison approaches using the Wilcoxon significance test. Even with the Bonferroni correction applied, the vast majority of evolved detectors are shown to have achieved a better performance over both time series tasks. Without the Bonferroni correction, we found that only 2 of the 100 evolved detectors were outperformed by the comparison approaches, with a further two giving a performance equivalent to KOAD.

The average complexity of the novelty detectors evolved by DPC+NEAT are similar for both time series datasets. The best detector evolved for the Mackey-Glass dataset, shown in figure 5.11, has 7 hidden neurons, 4 plastic synapses and 14 fixed synapses, whilst the best detector

Dataset	Avg. Mean Predictive Accuracy	St. Dev.	Best Mean Predictive Accuracy	Worst Mean Predictive Accuracy	Average Detector Complexity		
					No. Hidden Nodes	No. Plastic Synapses	No. Fixed Synapses
Mackey	0.859	0.034	0.891	0.641	4.85	4.07	10.6
Hard Disk	0.818	0.085	0.933	0.542	5.86	4.2	11.96

Table 5.10: The performance achieved by DPC+NEAT for the first two time series datasets.

Dataset	GWR		V-detector		KOAD	
	Mean Predictive Accuracy	St. Dev.	Mean Predictive Accuracy	St. Dev.	Mean Predictive Accuracy	St. Dev.
Mackey	0.674	0.012	0.802	0.012	0.809	0.024
Hard Disk	0.492	0.004	0.485	0.035	0.504	0.007

Table 5.11: The performances achieved by the comparison approaches for the first two time series datasets.

Dataset	DPC+NEAT		GWR		V-detector		KOAD	
	Avg. Mean TP Rate	Avg. Mean TN Rate	Mean TP Rate	Mean TN Rate	Mean TP Rate	Mean TN Rate	Mean TP Rate	Mean TN Rate
Mackey	0.865	0.853	0.419	0.929	0.880	0.724	0.818	0.801
Hard Disk	0.891	0.744	0.013	0.972	0.495	0.476	0.053	0.955

Table 5.12: The average mean true positive and true negative rates achieved by the 100 novelty detectors evolved by DPC+NEAT and the mean rates achieved by each comparison candidate for the first two time series datasets.

Dataset	GWR			V-detector			KOAD		
	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than	DPC+NEAT Detectors Better Than	DPC+NEAT Detectors Equiv. To	DPC+NEAT Detectors Worse Than
With No Correction ($p = 0.01$):									
Mackey	98	0	2	97	1	2	96	2	2
Hard Disk	100	0	0	100	0	0	100	0	0
With Bonferroni Correction ($p = 0.0001$):									
Mackey	98	2	0	97	1	2	96	2	2
Hard Disk	100	0	0	100	0	0	100	0	0

Table 5.13: Statistical comparison of the novelty detectors evolved by DPC+NEAT with each comparison approach for the first two time series datasets, both with and without Bonferroni correction (99% confidence level). "Detectors Equiv. To" refers to the number of novelty detectors evolved by DPC+NEAT whose performances are not significantly different to the respective comparison candidate.

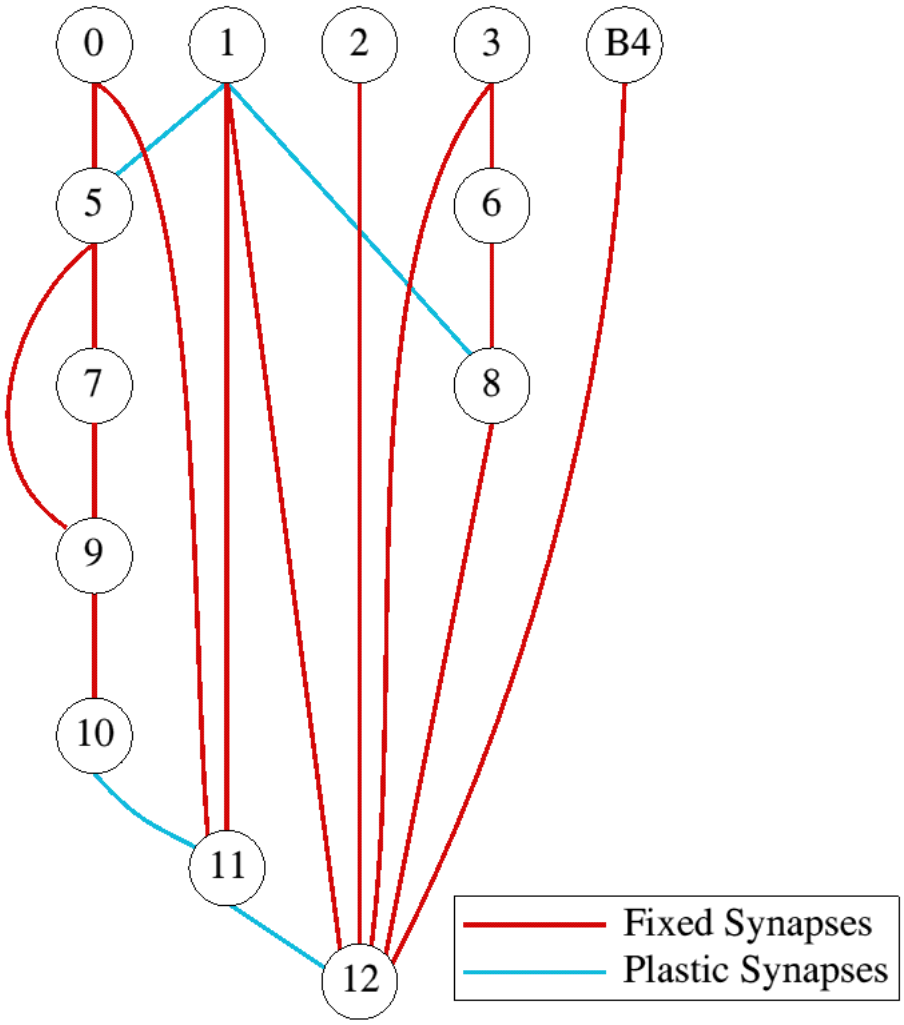


Figure 5.11: The structure of the best evolved novelty detector for the Mackey-Glass dataset.

evolved for the Hard Disk dataset, shown in figure 5.12, has 6 hidden neurons, 4 plastic synapses and 10 fixed synapses. In particular, this second detector has a number of interesting properties. First, three of the four plastic synapses consistently maintain a synaptic weight of 0. This is because the hidden neurons which drive these synapses always output a value of 1, as they receive input values which are always greater than or equal to 0. This means that a significant portion of the network structure, comprising hidden neurons 7, 8 and 10, is redundant. Hidden neuron 6 is also redundant in that it does not provide stimulus to any other neuron in the network. When this redundant substructure is discarded, we observe that the neural network performs two functions. The circuit comprising input node 3, the bias node and the output neuron performs linear novelty detection, based on the fourth attribute of the data alone. The remainder of the network attempts to correct misclassification of data points from the normal

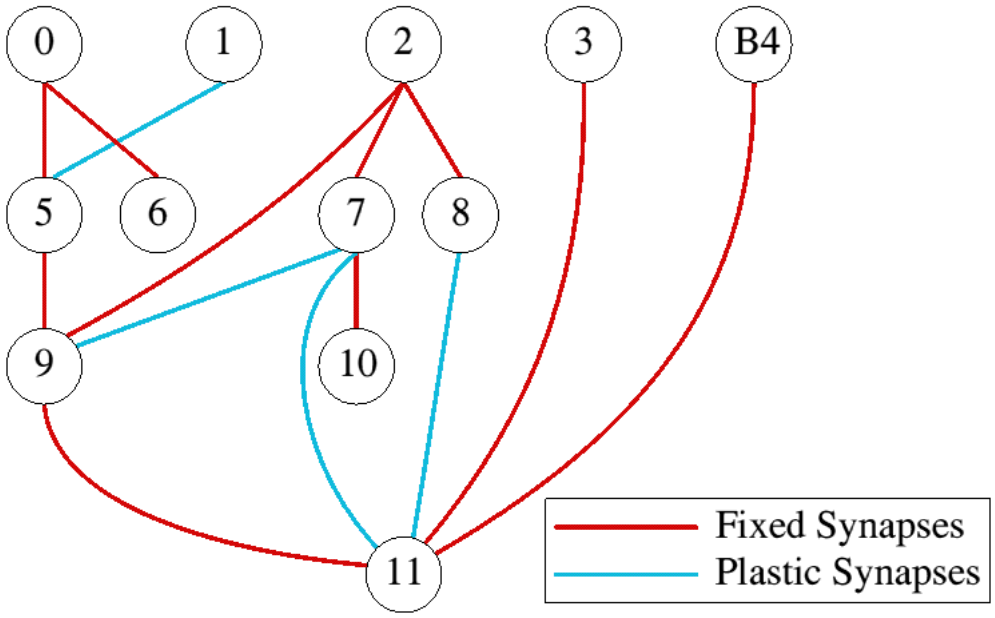


Figure 5.12: The structure of the best evolved novelty detector for the Hard Disk dataset. The substructure comprising hidden neurons 7, 8 and 10 is redundant, since the three plastic synapses in this substructure never adapt and neuron 10 has no outgoing synapses. Hidden neuron 6 is also redundant as it has no outgoing synapses.

Method		DPC+NEAT Performance				Avg. Detector Complexity		
Name	Predictive Accuracy	Avg. Pred. Acc.	St. Dev.	Best Pred. Acc.	Worst Pred. Acc.	No. Hidden Nodes	No. Plastic Synapses	No. Fixed Synapses
GWR	0.534	0.654	0.148	0.846	0.422	0.160	1.960	2.460
V-detector	0.540							
KOAD	0.760							

Table 5.14: The performances achieved by DPC+NEAT and the comparison approaches for the ECG time series dataset.

class. This function is activated for a given data point when the first attribute of the data point is greater than zero and the third attribute is less than or equal to 0.115. Its activation results in an instantaneous shift in the range of values of the fourth attribute that are taken to indicate normality. We found that, over a 2,500 point time series with 1,955 normal data points and 545 novel data points, this correction function prevented 374 normal data points from being mislabelled as novel.

Table 5.14 shows the results obtained by DPC+NEAT and the comparison approaches for the ECG task. Since only one time series is used, with one run for each detector, a statistical analysis was not possible. In this time series, the performance of DPC+NEAT was disappointing. Of the 100 novelty detectors evolved, we found that 65 were outperformed by KOAD, the best

performing comparison candidate for this time series. The best evolved detector achieved a TP rate of 0.698 and a TN rate of 0.994, whilst the average TP and TN rates achieved by the 100 evolved novelty detectors were 0.549 and 0.759 respectively. This compares to a TP rate of 0.559 and a TN rate of 0.962 achieved by KOAD. Both approaches demonstrated a performance clearly skewed towards the correct identification of normal beats.

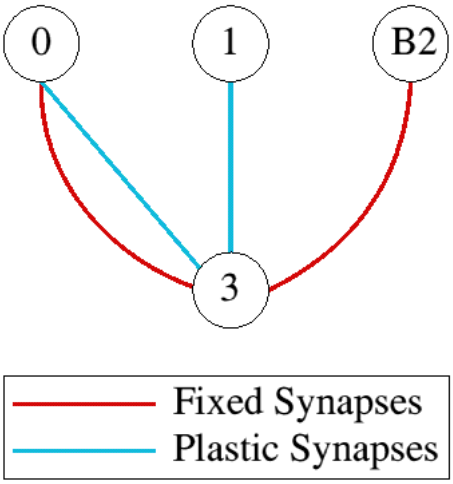


Figure 5.13: The structure of the best evolved novelty detector for the ECG dataset. This detector is structurally identical to those in the initial population used by DPC+NEAT.

The structure of the best detector evolved by DPC+NEAT is shown in figure 5.13. This network shows no structural differences from those in the initial population (used as the starting point of the evolutionary search), instead differing only in the weights chosen for the fixed synapses. Intriguingly, the average neural network complexity of the evolved detectors indicates that hidden neurons were rarely used. DPC+NEAT therefore appeared to find no evolutionary advantage in using the functionality provided by such neurons in this task. However, whilst thresholding hidden neurons appear to provide no benefit, this does not mean that no kind of hidden neuron could prove useful in this scenario. It is entirely possible, for example, that the very different function provided by the hidden neurons used in chapter 4 would be more suitable here.

To explore this possibility, we reran the ECG experiment whilst allowing DPC+NEAT to use the same kind of hidden neurons as those used in chapter 4 (and with no bias node in the input layer). The results obtained are shown in table 5.15. DPC+NEAT achieved a substantially better performance, with only 17 evolved novelty detectors giving a performance worse than or equivalent to KOAD. The average TP and TN rates achieved by the evolved detectors were 0.878 and 0.713 respectively, demonstrating less of a skewed performance. The best detector

Method		DPC+NEAT Performance				Average Detector Complexity		
Name	Predictive Accuracy	Avg. Pred. Acc.	St. Dev.	Best Pred. Acc.	Worst Pred. Acc.	No. Hidden Nodes	No. Plastic Synapses	No. Fixed Synapses
GWR	0.534	0.796	0.036	0.869	0.695	1.860	2.200	4.170
V-detector	0.540							
KOAD	0.760							

Table 5.15: The performances achieved by DPC+NEAT, with non-thresholding hidden neurons, and the comparison approaches for the ECG time series dataset.

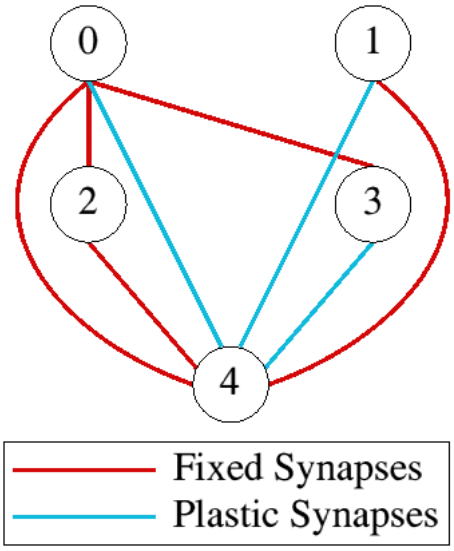


Figure 5.14: The structure of the best evolved novelty detector for the ECG dataset with non-thresholding hidden neurons.

evolved by DPC+NEAT, shown in figure 5.14, achieved a TP rate of 0.902 and a TN rate of 0.835. This detector features two hidden neurons, three plastic synapses and 5 fixed synapses. Looking at the structure of this detector, we observe that some structural redundancy exists. The connection from input neuron 0 to the output neuron, via hidden neuron 2, may just as well be represented as a direct connection between the input and output neurons. However, the second hidden neuron in the network does provide an important function. It permits the value at input neuron 0 to be amplified, before the network judges the correlation between this value and that at the output neuron.

5.9 Evaluation

Over six very different novelty detection tasks, DPC+NEAT has successfully constructed suitable novelty detectors which outperformed a series of comparative approaches to novelty detection from the literature. We now evaluate this approach in the context of these results and against the objectives set out in chapter 1.

The first objective requires that DPC+NEAT be generally suitable for and effective in a wide range of different problems. In each of the six novelty detection tasks considered in this chapter, DPC+NEAT successfully evolved novelty detectors which were subsequently shown to outperform a series of existing approaches to novelty detection in the literature. In achieving a high performance, these novelty detectors demonstrated their suitability to the task at hand. This in turn suggests that DPC+NEAT is indeed a suitable approach in a wide range of different novelty detection tasks. However, in each of the three novelty detection tasks based on record datasets, a significant proportion of the evolved detectors achieved a performance that was worse than one or more of the comparison approaches. This could indicate that the efficiency of the evolutionary search needs to be improved, since DPC+NEAT has demonstrated that it has the potential to evolve superior novelty detectors in all three tasks. This improvement may be achievable through tuning of the DPC+NEAT parameters, although requiring the user to tune these parameters manually would undermine the third objective of this research. In chapter 6, we consider ways in which these parameters could potentially be automatically tuned during the evolutionary search.

To meet the second objective, DPC+NEAT must permit a high degree of customisability. The best novelty detectors evolved by DPC+NEAT in this study use a variety of different neural network structures which allow an enhanced performance in each particular novelty detection task. This demonstrates DPC+NEAT's ability to customise the novelty detector for the specific application. However, despite this, we found that DPC+NEAT achieved a disappointing performance in the ECG task when using thresholding hidden neurons. But when the threshold activation function was removed, the performance of the detectors subsequently evolved showed a significant improvement. The two kinds of hidden neuron provide functionality that may or may not prove useful in a particular time series scenario. In the Hard Disk dataset, we found that thresholding hidden neurons allowed the best evolved network to instantaneously modify its decision boundary in the face of particular data points, which in turn resulted in the correct classification of a number of normal data points that would have otherwise been mislabelled.

This instantaneous response would be unachievable using plastic synapses alone, at least without limiting their use throughout the rest of the network. However, in the best performing novelty detector evolved for the ECG time series, hidden neurons without a thresholding activation function allowed the network to amplify particular input values before extracting correlational information. Such a function would not be achievable with thresholding hidden neurons. By defining the type of activation function that must be used by the hidden neurons *a priori*, we introduced an assumption into the DPC neural network model which, in some novelty detection tasks, was shown to be invalid. To avoid introducing such an assumption, the decision on which activation function to use should instead be made by DPC+NEAT.

Unlike the comparison approaches, which have all required suitable parameter values to be defined before being used on each dataset, DPC+NEAT required no additional work from the user, thus satisfying the third objective in chapter 1. The results reported here were achieved with the same set of parameter values for DPC+NEAT, given in appendix A, that were used in chapter 4. The values for the original NEAT parameters were found experimentally by Stanley [108] to give a good performance. However, as we will discuss further in chapter 6, if the user wished to use different values in a particular scenario, then a significant amount of work would likely be required. It may also be argued that the performance of each comparison approach was hampered in some way because of the chosen parameter values. But we have no way of knowing the optimum values, and can only explore the parameter space manually. It is reasonable to assume that, since a non-expert user would be in a similar position, these performances are representative of those which such a user may themselves observe.

For most of the novelty detection tasks, the average complexity of the detectors evolved by DPC+NEAT appears to vary with the perceivable difficulty of the task, indicated by the performances given by the comparison approaches. Of the novelty detection tasks based on record data, the performances of the comparison approaches suggest that the Biomedical, Iris-Versicolor and Iris-Virginica datasets represent harder novelty detection tasks. This corresponds to an increased level of average complexity exhibited by the novelty detectors evolved by DPC+NEAT for these datasets. The Breast Cancer and Iris-Setosa datasets appear to constitute easier novelty detection tasks. The evolved novelty detectors for the Iris-Setosa dataset, in which the normal and novel classes are linearly separable, demonstrate a notable decrease in complexity in that the majority of these detectors are linear in nature. However, the average complexity of the novelty detectors evolved by DPC+NEAT for the Breast Cancer dataset was higher than expected, possibly indicating that DPC+NEAT's ability to find solutions of a justifiable

level of complexity is not always guaranteed. For the Mackey-Glass and Hard Disk time series datasets, DPC+NEAT again tended to evolve complex novelty detectors and the performances of the comparison approaches suggest that these tasks were indeed difficult. However for the ECG task, DPC+NEAT, using the non-thresholding hidden neurons from chapter 4, evolved significantly less complex networks on average, despite the apparent difficulty of the task as suggested by the comparison approaches performances. Overall, aside from the Breast Cancer dataset, DPC+NEAT did appear to evolve novelty detectors of a justifiable level of complexity for the novelty detection task at hand. However, further investigation is required to understand why DPC+NEAT may potentially evolve overly complex novelty detectors in some tasks.

Overall, the results presented in this chapter suggest that DPC+NEAT does indeed fulfil the objectives stated in chapter 1. Its success in six very different novelty detection tasks suggests that it is likely to be suitable for use in a wide range of different novelty detection scenarios. However, it is likely that the efficiency of the evolutionary search could be improved in some cases. DPC+NEAT does permit a high degree of customisability, yet this could also be improved by allowing it to choose the activation function used in the hidden neurons. The work required from the user is minimal in this approach, since DPC+NEAT's parameters do not need to be tuned for each particular scenario before successful novelty detectors can be evolved. However, it is also true that optimising these parameters could potentially lead to a more efficient search. A significant amount of work would likely be incurred by the user, should they wish to use different parameter values. Finally, in most of the novelty detection tasks considered here, the complexity of the evolved detectors did appear to be proportional to the perceivable difficulty of the task. However, in some cases redundant structure was found in the best performing novelty detectors and the complexity of the detectors evolved in one task did appear to be unjustifiably high.

Unfortunately, whilst these results are promising, they are also overly optimistic for two reasons. First, during evolution, DPC+NEAT must evaluate the performance of candidate novelty detectors over both normal and novel data. This allows knowledge of the novel class to be built into detectors (through their structure and fixed weights) and so gives DPC+NEAT an unfair advantage over the comparison approaches, which were trained using normal data alone. The requirement of novel data during the evolutionary phase is a significant drawback of our approach, which we discuss further in chapter 6. Second, in the experimental setup used in this chapter, the performances of evolved novelty detectors were evaluated using test datasets which included data elements that were used to evaluate fitness during the evolutionary phase.

It is reasonable to assume that correctly classifying these elements was relatively easy for the evolved detectors, since they were optimised to give the correct response for the majority of the data elements in the evolutionary test set. Ideally, the detectors should have been evaluated exclusively over data not used during the evolutionary phase. However, despite the limitations of the results obtained in this chapter, we believe that they still indicate that our proposed approach shows promise in meeting the objectives of this research.

5.10 Summary

In this chapter, we evaluated the DPC+NEAT approach that we proposed in chapter 4. We chose six novelty detection tasks that were different enough to allow us to judge whether or not DPC+NEAT was likely to be suitable for use in a wide range of different novelty detection scenarios. The first three tasks considered record datasets with different characteristics, whilst the second three tasks considered both artificial and real-world time series datasets. In each novelty detection task, we compared the performance achieved by the novelty detectors evolved by DPC+NEAT against three existing approaches that we reviewed in chapter 2. Two of these approaches, Marsland's Grow When Required (GWR) [85] algorithm and Ji's V-detector [66] method, have been demonstrated, by the authors, over novelty detection tasks involving both record and time series datasets and so were used in all six novelty detection tasks. The final approach in each task was chosen specifically for the type of data involved: for record datasets, Schölkopf's one-class support vector machine (1-SVM) [99] approach was used, whilst for time series datasets we used Ahmed *et al.*'s Kernel-based Online Anomaly Detection (KOAD) [3] algorithm.

Over the six novelty detection tasks, we found that DPC+NEAT successfully evolved novelty detectors which outperformed each of the comparison approaches. However, in some tasks a significant number of the detectors evolved were outperformed by one or more comparison approaches, potentially suggesting that the efficiency of the evolutionary search could be improved. The best evolved novelty detectors used a variety of different neural network structures, demonstrating DPC+NEAT's ability to highly customise detectors for the particular task at hand. However, in one task the performance of the evolved detectors was hampered by DPC+NEAT being forced to use an unsuitable kind of hidden neuron (which we selected *a priori*), suggesting that the degree of customisability permitted by DPC+NEAT could be further improved. The work required from the user is minimal, since DPC+NEAT's parameters do not need to be

tuned for each particular scenario before successful novelty detectors can be evolved. However, this work would be significantly increased if the user decided to use different parameter values. Finally, in all but one of the six tasks, the level of complexity of the evolved detectors appeared to be justified, given the perceivable difficulty of the task as indicated by the performances achieved by the comparison approaches.

Overall, based on the results obtained in this chapter, DPC+NEAT does appear to fulfil the objectives stated in chapter 1. However, these results are overly optimistic, since DPC+NEAT was given an unfair advantage over comparison approaches by being permitted to use novel data when evolving detectors. The requirement of novel data during the evolutionary phase represents a significant drawback of our approach, which we discuss further in the next chapter. Also, the performances of evolved novelty detectors were evaluated over test datasets which included some data elements that had previously been used to evaluate fitness during the evolutionary phase. This is likely to have contributed to a further positive increase in these results. However, despite their limitations, we believe that the results obtained in this chapter still indicate that our proposed approach shows promise in meeting the objectives of this research.

Chapter 6

Summary and Conclusions

In this final chapter, we provide a summary of the work presented in this thesis. We then examine a significant drawback of our proposed approach, DPC+NEAT, that was identified in chapter 5 and consider ways in which this might be overcome. We also examine other drawbacks which, whilst less serious, define directions in which this work may be extended. We relate our proposed approach to the aims and objectives of this research, as described in chapter 1, and consider the extent to which they have been satisfied. Finally, we conclude this chapter by summarising the key points of this work and considering the contribution that the proposed approach makes to the novelty detection field.

6.1 Summary of the Thesis

In this thesis, we have proposed a new approach, DPC+NEAT, which aims to construct novelty detection neural networks for a given novelty detection task. In chapter 1, we presented the problem of novelty detection and identified the challenges that an end user, not expert in either novelty detection or the data mining field in general, is faced with when trying to use novelty detection in a real-world scenario. They must first identify from the literature an approach which is likely to be successful in their scenario and then determine suitable values for any parameters of that approach. These challenges are at best time consuming and at worst prohibitively difficult for the user to tackle. This motivated us to develop a system which allows the user to utilise the power of novelty detection but without having to face the challenges above. We identified a number of requirements that such a system would need to satisfy, which defined the objectives of our work.

In chapter 2, we evaluated a number of existing approaches to novelty detection from the literature against the objectives identified in chapter 1. These approaches, employing techniques from a wide range of areas within machine learning, could generally be divided into two groups: statistical approaches, such as k-nearest neighbour, negative selection and support vector machines, and neural network approaches, such as multilayer perceptrons, self-organising maps and evolutionary neural networks. We found that each approach that we evaluated failed to fulfil one or more of the objectives given in chapter 1, therefore justifying the need for a new approach. From the various techniques covered in chapter 2, we considered evolutionary neural networks to be the most promising, since they have the potential to construct neural networks which are specifically customised to the particular novelty detection task. We therefore chose to base our proposed approach on evolutionary neural networks.

Chapter 3 described the techniques used in this research. In the first part of the chapter, we presented a biologically inspired neural network model that appeared to have the potential to be effective in a wide range of different novelty detection scenarios. This model, Dynamic Predictive Coding (DPC), was originally proposed by Hosoya *et al.* as a hypothetical model of the adaptive encoding capabilities of the retina [63]. Its intended function is to separate familiar visual patterns from the unfamiliar or, in other words, perform a kind of novelty detection. Both fixed and plastic synapses are used in the model, where the plastic synapses have their weights modified using an anti-Hebbian learning rule. The model is based on the minimal assumption that novel data tends to have a different correlational structure to normal data, thus giving it the potential to be used in many different novelty detection tasks. However, the structure of a DPC neural network must be designed specifically for the particular task to which it is to be applied. Similarly, the values for the three parameters of the model, which control the behaviour of the learning rule, are also application-specific. In the second part of chapter 3, we identified an evolutionary neural network approach, Stanley's Neuroevolution of Augmenting Topologies (NEAT) [108], which simultaneously discovers the topology and weights required by a neural network in order to solve a particular problem. NEAT addresses three significant problems encountered when evolving both neural network structure and weights in an unrestricted fashion: crossing over identical networks which have different encodings, defining a meaningful crossover operation over parents with very different topologies and protecting new structural innovations until the corresponding networks have had sufficient time in which to optimise their performance with the new structure. None of these problems were addressed by the most promising evolutionary neural network approach found in chapter 2. NEAT is also

appealing as it tends to evolve neural networks with a low level of complexity for a particular problem.

In chapter 4, we presented our proposed approach, DPC+NEAT, which combines the DPC neural network model with NEAT to allow the evolution of DPC neural networks for novelty detection tasks. The goal of this approach is, when provided with a suitable fitness function which describes the user's novelty detection task, to automatically construct a novelty detector which achieves a good performance in that task. To ensure that DPC+NEAT was provided with the building blocks necessary to construct networks of any topology, we extended the DPC neural network model described in chapter 3 to permit the inclusion of hidden neurons. We also introduced a new mutation operator which allowed DPC+NEAT to vary the values of the parameters defined by the DPC neural network model. To gain an insight into whether or not DPC+NEAT was likely to be a successful approach, we examined its ability to optimise the structure of an illustrative DPC neural network in an extension of the visual encoding task considered in chapter 3. We also investigated how the optimised networks, and DPC+NEAT itself, were affected by different levels of noise, a factor which is commonly found in most real-world datasets. The results obtained in chapter 4 did indeed suggest that DPC+NEAT would be an effective approach, and as such was worthy of further investigation. However, a number of changes were required to be made to DPC+NEAT before it could be used in tasks other than those considered in chapter 4. We concluded the chapter by describing these changes, thus completing our presentation of the approach.

Finally, in chapter 5 we evaluated DPC+NEAT against the objectives given in chapter 1. The performance of DPC+NEAT was assessed over a number of very different novelty detection tasks, such as identifying unusual blood measurements taken from patients, detecting data not relating to a known kind of flower, and identifying unusual heartbeats in an electrocardiogram series. We tried to choose very different novelty detection tasks so that we could gain an indication of the likelihood of DPC+NEAT being suitable for use in a wide range of different novelty detection scenarios. To enable us to judge what constituted a "good" performance in each task, we also compared the performance given by detectors evolved by DPC+NEAT against that given by a number of existing approaches to novelty detection, which we reviewed in chapter 2. On each novelty detection task, we found that DPC+NEAT successfully evolved novelty detectors which outperformed the chosen comparison approaches. This suggested that DPC+NEAT was in fact suitable for use in a wide range of different novelty detection scenarios. The best detectors evolved for each task used a variety of different neural network structures,

demonstrating DPC+NEAT's ability to highly customise detectors for the particular task at hand. The work required in configuring DPC+NEAT was minimal, since the same parameters could be used in all novelty detection tasks. However, as we will discuss in the next section, this work would likely be significantly increased if the user wished to use a different set of parameter values. In most tasks, the level of the complexity of the evolved detectors also appeared to be justified, given the apparent difficulty of the task. Based on these observations, we concluded that DPC+NEAT did appear to fulfil each of the objectives stated in chapter 1. However, whilst these results were promising, we considered two reasons as to why they are overly optimistic. First, DPC+NEAT was given an unfair advantage over comparison approaches as it requires novel data during the evolutionary phase. We discuss this important drawback further in the next section. Second, the performances of evolved novelty detectors were evaluated over test datasets which included some data elements that had previously been used to evaluate fitness during the evolutionary phase. This is likely to have further contributed positively to these results. Despite their limitations, however, we believe that the results obtained in chapter 5 still indicate that our proposed approach shows promise in meeting the objectives of this research.

6.2 Drawbacks of DPC+NEAT

We now examine in detail four important drawbacks of the DPC+NEAT approach. Three of these drawbacks have been previously highlighted in chapter 5 but are considered here in detail, whilst the remaining drawback (time required for evolution) will be discussed here for the first time. We also consider ways in which each of these drawbacks could potentially be overcome.

6.2.1 Fitness Function Requires Novel Data

As we have described in chapters 1 and 2, a novelty detector is required to classify data elements as normal or novel based on knowledge acquired from the normal class alone. This is because, in most scenarios, large amounts of normal data can be easily provided, whilst only a very limited number of examples of the novel class may be available. This requirement of a novelty detector is satisfied by the DPC neural network model. As was clearly shown in chapter 3, a DPC neural network acquires its knowledge from the data it sees, demonstrating insensitivity to the kinds of data seen frequently whilst remaining sensitive to data which is different in some way.

However, as highlighted in chapter 5, examples of the novel class *are* required by DPC+NEAT. This requirement stems from the fitness function, which is critical in guiding the evolutionary

search. In order to determine the performance of a particular novelty detector, the fitness function must examine the detector's ability to classify both normal *and* novel data. This is because if data from the normal class alone was used, then a novelty detector could achieve a maximum fitness score by consistently outputting a value below the established novelty threshold, regardless of the input it sees. This requirement of novel data by DPC+NEAT constitutes the most significant drawback of the approach.

The problem of requiring both normal and novel data during the evolutionary search also impacts on the evolutionary neural network approaches to novelty detection identified in chapter 2. The approaches proposed by Samanta [98] and by Hofmann and Sick [59] fail to address this issue, instead additionally relying on supervised learning methods to train the evolved detectors. However, the approach proposed by Han and Cho [55] does attempt to provide a solution. Instead of using actual novel data, randomly generated data is used to approximate examples of the novel class. Using this method, Han and Cho successfully evolved novelty detectors which compared favourably against two other techniques in an intrusion detection scenario. However, one drawback of this method is that there is no guarantee that randomly generated data points will fall outside of the normal class. Forcing a novelty detector to label a randomly generated point as novel when it actually lies inside the normal class would inevitably result in poor decision boundaries being constructed by the detector.

A more promising approach, which we identified in chapter 2, is that proposed by González *et al.* [47, 48], in which a negative selection algorithm is used to artificially generate novel data. This approach, real-valued negative selection (RVNS), works by constructing a number of "detectors" which cover those areas of the input space not already covered by the normal class. Each "detector" is a data point in the input space which lies outside of the normal class and so constitutes an example of the novel class. In RVNS, the detectors are also kept separated from one another so that they collectively maximise the coverage of the novel class. González and Dasgupta used RVNS to train a multilayer perceptron to perform novelty detection over a number of different datasets [47], including the Wisconsin Breast Cancer record dataset and the Mackey-Glass time series dataset, which we considered in chapter 5. It was shown to achieve a good performance over both of these datasets, which suggests that it may be a potentially promising approach.

However, using RVNS to provide artificial novel data would become problematic in scenarios where the concept of normality changes over time. For example, consider a time series constructed from the data collected by a temperature sensor. The range of temperature values

expected at any particular time varies with seasonality, i.e. this range is generally expected to be higher in the summer months than in the winter months. Therefore, to ensure that novelty detectors are evolved which are capable of adapting to seasonal variations, the time series used by the fitness function should capture these variations. However, this also means that artificially generated novel records would need to be sensitive to this seasonality. One way in which this could be accomplished is to dynamically adjust the detectors generated by RVNS through the course of the time series. That is, one could generate a set of detectors, based on an initial segment of the time series, and then continuously attempt to match these detectors to subsequent segments of the series. If a match occurred, then the relevant detector could be adjusted or replaced so as to ensure that the set of detectors continued to represent the novel class. Another problem is that artificially generated novel data may lie at a significant distance from the normal class, possibly misrepresenting the difficulty of the novelty detection task to DPC+NEAT. This would then result in detectors being evolved which perform poorly when faced with real novel data. Therefore, it would be important to ensure that at a proportion of the artificial novel data produced lies in close proximity to the normal class.

6.2.2 Customisation of Hidden Neurons

In the novelty detection tasks considered in chapter 5, we forced each hidden neuron of the evolved novelty detectors to use a threshold activation function. However, in one of the tasks, we found that this activation function resulted in a disappointing performance from the approach. When the hidden neurons were permitted to have no activation function, like those used in chapter 4, DPC+NEAT achieved a much better performance. By deciding the type of activation function that should be used *a priori*, we had introduced an assumption into the design of the networks (i.e. that thresholding hidden neurons are always suitable), which was shown to be invalid in this task. The fact that the choice of activation function is not made by DPC+NEAT, but instead forced onto the user, constitutes another drawback of this approach.

This drawback can be overcome in one of two ways. A new network parameter could be introduced into the novelty detectors evolved by DPC+NEAT which then determines the type of activation function to be used in those detectors. This parameter could then be mutated in the same way as the existing network parameters. However, changing the activation function used in the hidden neurons of a network would result in an instantaneous and significant change in the behaviour of that network. This sudden change could severely impact on the fitness of

the network, thus possibly resulting in the loss of once promising candidates.

Alternatively, the parameter controlling the chosen activation function could be placed inside the hidden neurons themselves. The value of this parameter could then be decided independently for each hidden neuron as it is introduced into the network. In this way, DPC+NEAT could potentially exploit the benefits of multiple kinds of activation function in a single network, should it prove advantageous to do so. This also increases the degree of customisability permitted by DPC+NEAT. A new mutation operator could also be introduced which allows DPC+NEAT to change the activation function used by a particular hidden neuron. However, determining whether or not such an operator would actually be beneficial to DPC+NEAT requires further investigation.

6.2.3 Time Required For Evolution

One drawback of DPC+NEAT which we have not previously discussed is that, depending on the complexity of the novelty detection task and the amount of data involved, the process of evolving suitable novelty detectors can become significantly time consuming. This was especially apparent in the time series experiments conducted in chapter 5, which were of a substantially larger size than the record datasets considered. For example, on a Linux cluster with 30 nodes it took approximately 5 days to evolve 100 novelty detectors with DPC+NEAT for the Mackey-Glass time series, which consisted of 10,000 data points. This suggests that each detector was evolved in an average of 36 hours. Without such computing power available, and when larger datasets are considered, the time required to evolve suitable novelty detectors may become prohibitively large.

The most time consuming part of the evolutionary search is the execution of the fitness function. Therefore, a faster search can be achieved by DPC+NEAT by simply reducing the number of fitness evaluations that are performed. This can trivially be accomplished by reducing the size of the population. However, this also reduces the potential variability that can exist within the population in any given generation. This reduction in population size may also require other parameters of DPC+NEAT to be modified, such as those parameters controlling mutation and crossover rates.

6.2.4 DPC+NEAT Parameters

DPC+NEAT has a total of 21 parameters which control its behaviour. Requiring the user to find suitable values for each of these parameters undermines our objective of requiring a minimal amount of work from the user in configuration for a particular scenario. Therefore, in this thesis we have investigated whether or not DPC+NEAT can achieve a good performance, that is evolve detectors which outperform a number of comparison approaches, in very different novelty detection tasks whilst using the same set of parameters. Whilst this was shown to be successful, it also exposes another drawback of DPC+NEAT: if the user does not wish to use these "default" parameters, for example to optimise the evolutionary search in some way or to adjust the runtime of the search, then they are left with the original problem of determining suitable values through a limited manual search. Furthermore, in some of the novelty detection tasks considered in chapter 5, the performances achieved by the detectors evolved by DPC+NEAT appeared to indicate that the efficiency of the evolutionary search could be improved. Such an improvement might be achieved by optimising the parameters of DPC+NEAT for the particular scenario.

The problem of determining suitable values for the parameters of an evolutionary algorithm is not new. One way in which this has been tackled in the past in genetic algorithms (GA's) is to use a second GA to optimise the parameters of the original GA [89]. However, not only is this process time consuming, since each fitness evaluation performed in the second GA consists of a run of the original GA, it also simply moves the problem from setting the parameters of the original GA to setting those of the second GA. In our view, a more promising way of tackling this task would be to allow the parameters to be *self-adaptive*, that is to allow DPC+NEAT to change its own parameters as necessary during the evolutionary search. Certain parameters, such as population size for example, could be chosen solely by the user if necessary, with all other parameters being dynamically adapted. Alternatively, all parameters could be chosen by DPC+NEAT automatically.

6.3 Relating DPC+NEAT to the Aims and Objectives in Chapter 1

We now consider whether or not the DPC+NEAT approach proposed in this thesis has successfully accomplished the aims and objectives set out in chapter 1. As a reminder, the aims of this research are succinctly described by the following sentence:

To develop an approach which enables novelty detection to be performed easily, efficiently and effectively in a wide range of different scenarios

In chapter 1, this was distilled into four objectives which we argued such a system should fulfil:

- *General Suitability:* The approach should be suitable for and effective in a wide range of different novelty detection tasks.
- *Customisability:* The approach should permit a high degree of customisability.
- *Work:* The effort required from the user to customise the approach for a particular task should be minimal.
- *Complexity:* The complexity of the approach should be justifiable for the particular task.

These objectives formed a set of evaluation criteria, which we used in chapters 2 and 5 to evaluate both existing approaches to novelty detection in the literature and the new approach proposed in this thesis. However, the evaluation in chapter 5 did not take into account all of the drawbacks identified in the previous section. Therefore, in this section, we again relate DPC+NEAT to each of the four objectives in turn, and, based on both its success in the evaluation conducted in chapter 5 and the drawbacks discussed in the previous section, consider whether or not the approach can be considered to have fulfilled each objective.

Objective 1: General Suitability

In order to determine whether or not DPC+NEAT was likely to be suitable for use in a wide range of novelty detection tasks, in chapter 5 we evaluated its performance over six very different tasks. So as to allow us to interpret the performance achieved in each task, we also made a comparison between the detectors evolved by DPC+NEAT and three existing approaches to novelty detection from the literature. In each task, DPC+NEAT was able to construct at least one detector which outperformed each of the existing approaches considered. Two of these existing approaches used in all six novelty detection tasks, Marsland's Grow When Required (GWR) approach and Ji's V-detector algorithm, which have each been shown to be versatile novelty detectors in other studies [66, 85], failed to demonstrate the same level of versatility as that shown by DPC+NEAT. By refocusing the problem from constructing a versatile novelty detector to instead constructing a system which itself can build specialist novelty detectors, DPC+NEAT is able to generalise to different novelty detection tasks without sacrificing performance.

However, as we have discussed in section 6.2.1, in order to evaluate the performance of candidate novelty detectors during the evolutionary search, DPC+NEAT requires examples of both the normal *and* novel classes. This serious drawback of DPC+NEAT impacts on its suitability for use in novelty detection tasks, where little or no novel data is typically available. If no novel data is available, then DPC+NEAT cannot be used. If very little novel data is present, then evolved novelty detectors could potentially deliver a poor performance once deployed.

Objective 2: Customisability

DPC+NEAT clearly permits a high degree of customisability, thus satisfying the second objective above. The possible network topologies that can be evolved are not restricted in any way, unlike other evolutionary neural network approaches identified in chapter 2. In addition to modifying structure and weights, DPC+NEAT is also free to choose suitable values for the parameters of evolved detectors. The high degree of customisability afforded to DPC+NEAT was clearly demonstrated in the experiments conducted in chapter 5, where networks with very different topologies were evolved in each novelty detection task, exploiting the unique properties of the data in the task.

However, one decision not left to DPC+NEAT was the determination of the type of activation function to be used by the hidden neurons, which we decided ourselves *a priori*. Unfortunately, this resulted in a poorer performance from the evolved novelty detectors in one task considered in chapter 5, where the activation function we had chosen was actually unsuitable.

Objective 3: Work

As we demonstrated in chapter 2, a common problem in many approaches is that increasing the customisability of the approach also increases the work required from the user in configuring the approach for their particular novelty detection task. For example, whilst a multilayer perceptron supports a range of different topologies, and so is highly customisable, the user is required to select the appropriate topology, a task which represents a significant amount of work and demands the skills of a neural network expert. Marsland's GWR approach has a number of parameters which can be used to customise its behaviour. But again this requires that the user find suitable values, a process which is time consuming and so constitutes a large amount of work. DPC+NEAT, on the other hand, avoids this problem by determining suitable network structure, weights and parameters itself through evolutionary search. Whilst DPC+NEAT itself has a large number of parameters which can be configured, we have shown in chapter 5 that

good results can be obtained in a range of different tasks when using the same set of parameters for DPC+NEAT. However, as we discussed in section 6.2.4, if the user wishes to *optimise* the evolutionary search for their problem, they must still resort to determining suitable parameter values for DPC+NEAT manually.

Objective 4: Complexity

The final objective listed above states that the "complexity of the approach should be justifiable for the particular task". In our work, this objective is met not by the DPC+NEAT approach itself but instead by the novelty detectors that it evolves. This is accomplished through NEAT's ability to start the evolutionary search with networks of minimal complexity and increase this complexity only gradually during the search, after allowing sufficient time to explore the weight and parameter spaces. DPC+NEAT's adherence to this objective is also vividly demonstrated in chapter 5. In a novelty detection task where the normal and novel classes were linearly separable, DPC+NEAT tended to evolve simplistic linear novelty detectors. Whilst nonlinear behaviour is important in a neural network, it is only justifiable if the problem itself is nonlinear. In trivial problems which can be solved linearly, the complexity introduced to the network to achieve nonlinearity would not be justifiable. In the tasks involving time series datasets, the detectors evolved by DPC+NEAT had a higher level of complexity on average, which corresponded to the increased difficulty of these tasks as evidenced by the lower performances achieved by the existing approaches. However, in one novelty detection task in chapter 5, the complexity of the detectors evolved by DPC+NEAT appeared to be unjustifiably high, given that a very good performance was achieved by the comparison approaches. This may suggest that, whilst this objective tends to be fulfilled by DPC+NEAT, this is not guaranteed in every scenario.

Does DPC+NEAT Achieve Our Aims?

We now consider whether or not DPC+NEAT successfully achieves the aims of this research. From the evaluation in chapter 5, we see that DPC+NEAT generally appears to fulfil all four objectives. In a series of very different novelty detection tasks, detectors were evolved by DPC+NEAT which outperformed a number of comparison approaches. Since its parameters were kept fixed, DPC+NEAT required a minimal amount of work from the user whilst permitting a high degree of customisability in the novelty detectors it evolved. The complexity of these detectors also generally appeared to correspond to the difficulty of each task.

However, after taking into account the drawbacks discussed in section 6.2, we find that

DPC+NEAT does not entirely fulfil one of the four objectives. The first drawback, requiring novel data during the evolutionary search, makes DPC+NEAT potentially unsuitable for use in novelty detection tasks where no, or very little, novel data is available. Unfortunately, as we saw in chapter 2, the lack of available novel data is characteristic of most novelty detection problems. Therefore, we cannot argue that DPC+NEAT successfully fulfils the first objective above, since its suitability is limited to novelty detection tasks where enough novel data is available to allow the robust evolution of detectors.

6.4 Further Directions for Improvement

From the drawbacks discussed in section 6.2, we have already identified two important ways in which our proposed approach may be improved. First, to overcome the drawback of requiring novel data during evolution, we suggested that DPC+NEAT could be combined with a RVNS algorithm, which can be used to effectively generate artificial novel data. Indeed, the original application of RVNS, as proposed by González *et al.* [47, 48], was to generate artificial novel data to permit supervised classifiers, such as the multilayer perceptron, to be used in novelty detection tasks. The second improvement that could be made to DPC+NEAT is to allow its parameters to be chosen in an adaptive fashion during the evolutionary search. These parameters could be optimised to improve the performance of the evolutionary search in a particular scenario. Furthermore, the user could opt to fix the values of certain parameters, and allow DPC+NEAT to optimise the remaining parameters. For example, the user may wish to achieve an improved performance from the evolutionary search with the population size constrained to 20, so as to reduce the number of fitness evaluations that need to be performed.

In addition to improving DPC+NEAT itself, there are also other ways in which the work presented in this thesis may be extended. One such direction would be to investigate evolving novelty detectors which use learning rules different to that given by the DPC neural network model. It may be that allowing plastic synapses to be updated from one of a possible selection of rules would be beneficial, with different rules making different assumptions about the data. Such a strategy was used in Stanley's Plastic NEAT approach [108], where each synapse was able to choose a learning rule from a predefined rule set, through the mutation of two control parameters. This had the advantage of ensuring that the dimensionality of the search space did not explode as new synapses were introduced. However, the number of rules that should be defined in the ruleset, and what these rules should be, remain open questions.

Another direction in which this research could be continued would be to investigate using different algorithms to perform the evolutionary search. For example, Siebel and Sommer [103] recently proposed an approach, Evolutionary Acquisition of Neural Topologies version 2 (EANT2), that was shown to outperform NEAT in a simulated task. Like NEAT, EANT2 starts its evolutionary search from networks with minimal structure. However, during the search, rather than evolving the structure and weights of networks simultaneously, EANT2 first optimises the weights of each network in the population, and then, if this fails to result in a sufficiently high fitness being reached, allows structural mutations to take place. EANT2 also requires fewer parameters to be set by the user than NEAT. A comparison of the performances achieved by EANT2 and NEAT when evolving DPC-based novelty detectors would be intriguing.

Finally, as we have discussed in chapter 5, there are a distinct lack of available benchmark datasets in the novelty detection literature. This is especially true in the case of record datasets, which has caused many researchers to resort to adapting datasets that were originally intended to allow benchmarking of classifiers. The drawback of this is that novelty detectors are then being tested on datasets where the "novel" class is well represented, and so cannot be truly considered as novel. Therefore, we would consider the creation of benchmark datasets which specifically represent novelty detection problems to be a significant contribution to the field. To facilitate this, a repository could be created to store these datasets and make them publicly available. Similar repositories have been established in the data classification field, such as the UCI Machine Learning Repository[1] (from which two of the three record datasets considered in chapter 5 were taken). By using novelty detection benchmark datasets instead of classification benchmark sets, we believe a more realistic evaluation of proposed novelty detectors would be obtained.

6.5 Conclusions

The research presented in this thesis was motivated by the goal of developing a system to allow novelty detection to be performed on a wide variety of different problems. We have tried to tackle this problem by employing an evolutionary algorithm to evolve neural networks, with adaptive learning capabilities inspired by biological processes observed in the retina, for individual novelty detection tasks. The evolutionary algorithm, Stanley's Neuroevolution of Augmenting Topologies (NEAT) [108], is capable of constructing neural networks of any topology. The learning

[1] Available at: http://archive.ics.uci.edu/ml/

rule used by the networks, which adheres to Hosoya *et al.*'s DPC model [63], makes the basic assumption that unusual data elements have a different correlational structure to that seen in normal data. From evaluation over a number of very different novelty detection tasks, we found that the proposed approach, which we called DPC+NEAT, showed promise in meeting a series of objectives that we derived from the primary goal of this research.

However, DPC+NEAT does suffer from an important drawback which prevents us from regarding it as a complete solution. In order to evolve novelty detectors for a particular task, DPC+NEAT must periodically evaluate the performance of potential candidates, which requires access to both normal *and* novel data. But the availability of novel data tends to be extremely limited in most novelty detection scenarios, which has a considerable impact on the suitability of this approach. In this chapter we have identified a potentially promising way in which this drawback can be overcome. Instead of requiring real novel data in a particular scenario, a technique can be used to generate artificial data belonging to the novel class. This technique, real-valued negative selection (RVNS), has previously been used to do just this [47, 48]. Therefore, whilst we have failed to address this drawback in the work presented in this thesis, we feel that it can potentially be overcome.

Another drawback of DPC+NEAT is that its behaviour is governed by a large number of parameters, which are described in appendix A. In this work, we demonstrated that DPC+NEAT can evolve novelty detectors which give a good performance in very different novelty detection tasks whilst using the same set of parameters. However, if for some reason the user does not want to accept these "default" parameters, for example because they wish to use a different population size, then they are again faced with the problem of determining new values themselves through a limited manual search. One intriguing way in which this drawback could be overcome would be to enable DPC+NEAT to automatically tune some or all of its own parameters as required during the evolutionary search.

Despite these drawbacks, we consider DPC+NEAT to be a significant contribution to the novelty detection field. It shifts the focus away from the original problem of designing novelty detectors for specific applications, or groups of applications, to constructing systems which themselves then tackle this problem. Since they do not require the user to have expert knowledge of either data mining or machine learning, these systems have the potential to open up the power of novelty detection to a vast number of people working with data in all kinds of disciplines. Whilst, as we have shown in this work, there are still important challenges that remain to be tackled, we hope that this thesis has demonstrated the potential of this exciting new branch of

research.

Appendix A

DPC+NEAT Parameters

This appendix lists the 21 parameters of the DPC+NEAT approach and gives the values used for these parameters in the experiments conducted in this thesis.

Parameter	Description	Value
$\delta_t(0)$	Initial compatibility threshold value	4.0
c_1	Coefficient for excess genes	1.0
c_2	Coefficient for disjoint genes	1.0
c_3	Coefficient for average weight difference	0.4
s_t	Target number of species	4
S_t	Survival threshold	1.0
m_{CG}	Probability of genes being chosen from either parent during crossover	0.6
m_{AG}	Probability of genes being averaged from both parents during crossover	0.4
m_{NC}	Probability of an offspring being created by mutation alone	0.25
m_O	Probability of an offspring being created by crossover alone	0.2
m_{AC}	Probability of mutation adding a connection to the network	0.05
m_{AN}	Probability of mutation adding a node to the network	0.03
i_R	Probability of two genomes from different species mating	0.05
w	Maximum magnitude of a mutation that changes a connection weight	0.5
p	Population size	50
c_4	Coefficient for DPC β parameter	1.0
c_5	Coefficient for DPC τ parameter	1.0
c_6	Coefficient for DPC m parameter	1.0
c_7	Coefficient for DPC $dpcNT$ parameter	1.0
m_P	Probability of mutation modifying DPC parameter values	0.8
α	Probability of a new connection being a plastic synapse	0.5

Table A.1: The parameters used by DPC+NEAT throughout this thesis. The coefficient parameters c_i are used in the compatability distance measure, defined by equation 4.24 in section 4.5.1. The first 15 parameters were inherited from NEAT, whilst the remaining 6 parameters were introduced specifically for DPC+NEAT.

Table A.1 details DPC+NEAT's parameters. The first 15 of these parameters were defined originally by Stanley's Neuroevolution of Augmenting Topologies (NEAT) algorithm [108] and

so were inherited by DPC+NEAT. Two of NEAT's parameters, controlling the probability of introducting recurrent connections into a network and stagnation of a non-improving population, are omitted from table A.1 as these functions are not used in this work.

The parameter values given in table A.1 are used in all experiments, apart from those conducted in chapter 4 where different values for the parameter m_P are used. In this chapter, m_P is set to 0.0 in some experiments and 0.8 in others, so that we could investigate the advantages gained by allowing DPC+NEAT to evolve network parameters in addition to structure and weights in a test scenario.

Glossary of Key Terms

Activation Function — A function which generates an output value for a neuron in a neural network, based on the sum of that neuron's inputs.

Adaptable Environment — In the dynamic predictive coding simulations considered in chapters 3 and 4, a visual environment with correlational relationships between pixels, which can therefore be learned by the dynamic predictive coding networks considered in these chapters.

Adaptive Learning — A online learning technique which allows an approach to adapt to changes in the properties of the input data as and when these changes occur.

Anomaly Detection — See *Novelty Detection*

Anti-Hebbian Learning — The inverse of Hebbian learning, in that correlated activity between two neurons in a neural network leads to a decrease in the weight of the synapse connecting them, whilst uncorrelated activity leads to an increase.

Class — In classification, a group of data elements that share some common property or properties.

Class Boundary — A boundary which indicates the area(s) of an input or feature space covered by a particular class.

Class Label An attribute associated with a particular data element which identifies the class to which that element belongs.

Classification A data mining task in which the group, or class, to which a given data element belongs is predicted, based on the properties of the data element and knowledge of these classes acquired during a training phase.

Cluster See *Cluster Analysis*

Cluster Analysis A data mining task in which data elements are automatically placed into groups, or clusters, based on information directly available from the data itself.

Clustering See *Cluster Analysis*

Data Element An element of a set of data. For record data, this is a record (i.e. a row in the table structure holding the data). For time series data, this is the value of a particular series at a particular point in time. A data element may also be referred to as a data point, since it constitutes a single point in the input space.

Data Mining The process of extracting, or mining, information (knowledge) embedded in large amounts of data.

Data Point See *Data Element*

Decision Boundary See *Class Boundary*

Decision Surface See *Class Boundary*

Evolutionary Algorithm A search algorithm that employs Darwinian evolution principles to find a solution which optimises some measure of performance.

False Negative The misclassification of a data element from the positive (novel) class.

False Positive The misclassification of a data element from the negative (normal) class.

Feature Space An abstract n-dimensional geometrical space in which each element of a dataset may be described in terms of n features. The dimensionality of the feature space may be higher or lower than that of the original input space.

Fitness Function A function which assesses the performance of each individual within a population being evolved by an evolutionary algorithm. Each individual is assigned a fitness value based on this assessment.

Fitness Value A scalar value describing the performance of an individual within a population being evolved by an evolutionary algorithm.

Fixed Synapse A neural network synapse whose weight remains constant.

Forgetting The ability of an approach to selectively disregard knowledge previously acquired.

Genotype The representation used by an evolutionary algorithm of an individual candidate solution within a population evolved by that algorithm.

Hebbian Learning A learning method based on Donald Hebb's postulate of learning, which states that the repeated activation of two biological neurons strengthens the connection between those neurons. A Hebbian learning rule increases the weight of a synapse when the activity between the two neurons it connects is correlated, and decreases the weight when this activity is anti-correlated.

Hidden Layer A layer of hidden neurons in a neural network.

Hidden Neuron A neuron in a neural network which lies between the input and output layers.

Hyperplane The generalisation of the geometrical concept of a plane in three dimensional space to n-dimensional space.

Hypersphere The generalisation of the geometrical concept of a sphere in three dimensional space to n-dimensional space.

Hypersurface The generalisation of the geometrical concept of a surface in three dimensional space to n-dimensional space.

Input Space A geometrical space in which elements of a dataset may be located, where the dimensionality of the space corresponds to the dimensionality of the dataset.

Multivariate Time Series Time series data consisting of two or more variables (i.e. multiple series).

Neural Network	A collection of processing units, called neurons, connected in some manner with directed links known as synapses with some strength or weight.
Neuron	See *Neural Network*.
Normal Data	Data that lies inside a known class.
Novel Data	Data that falls outside of any known classes.
Novelty Detection	The detection, or identification, of data that is unusual or unknown, deviating from some acquired model of normality.
Offline Learning	A learning strategy in which a specially constructed training dataset is required. A supervised or unsupervised learning technique may be used in this strategy.
One-Class Classification	See *Novelty Detection*
Online Learning	A learning strategy in which an approach is trained in-situ using real data that has not been specifically processed or selected for the learning phase. This generally requires an unsupervised learning technique to be used. An approach capable of online learning may also support adaptive learning.
Outlier	A data point lying away from the majority of points in the normal class or classes. This may represent an anomaly or simply an unusual instance of normality.
Overfitting	When a particular model learns, or fits itself to, a dataset too precisely, resulting in a lack of generalisability to new data.

Phenotype The realisation of an individual candidate solution described by a genotype.

Plastic Synapse A neural network synapse whose weight changes continuously.

Predictive Coding A technique, originally proposed by Peter Elias for the transmission of magnitudes, which encodes elements for transmission in terms of prediction error. The prediction of the current element is based on the properties of previously transmitted elements.

Principal Component Analysis A technique which re-expresses n-dimensional data in an m-dimensional space, where $m \leq n$, with minimal error.

Prototype A data element which is representative of some group or class. Prototypes commonly define the centres of clusters in a clustering technique.

Record Data Data that is organised in tabular form, comprising a series of records, which define the rows of the table, with each record consisting of a number of attributes, defining the columns of the table.

Sliding Window A data preprocessing technique which converts a time series into a collection of vectors. A window of n data points is moved over the time series, with the segment of the time series covered by the window at a particular time step constituting the input vector for that time step. Between any two time steps, the window moves m points along the series. If $m < n$, the sliding window is said to be overlapping.

Supervised Learning The process of learning over training data elements for which the desired response is known. The performance of the learner is repeatedly evaluated over the training dataset, and this information is used to further inform its learning.

Synapse See *Neural Network*

Test Dataset See *Test Phase*

Test Phase A phase in which the knowledge acquired by an approach in a training phase can be assessed. The test dataset used in this phase consists of data not previously seen by the approach, allowing one to determine whether the approach has successfully learnt the general characteristics of the training data or if the approach overfits the training data.

Time Series Data Data that is organised as one or more series of values, where each series describes how a single variable changes over time.

Training Dataset See *Training Phase*

Training Phase A phase in which an approach tries to learn the general characteristics of the elements of a training dataset. This learning is subsequently assessed in a test phase.

True Negative The correct classification of a data element from the negative (normal) class.

True Positive The correct classification of a data element from the positive (novel) class.

Univariate Time Series Time series data consisting of a single variable (i.e. a single series)

Unsupervised Learning The process of learning over training data for which the desired response is unknown. In this case, the performance of the learner cannot be evaluated during the learning phase.

Bibliography

[1] H. Abdi. The Bonferroni and Sidak corrections for multiple comparisons. In N. Salkind, editor, *Encyclopedia of Measurement and Statistics*, chapter 2, pages 103–107. Thousand Oaks, 2007.

[2] D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[3] T. Ahmed, M. Coates, and A. Lakhina. Multivariate online anomaly detection using kernel recursive least squares. In *Proceedings of the 26th IEEE International Conference on Computer Communications.*, pages 625–633. IEEE, 2007.

[4] T. Ahmed, B. Oreshkin, and M. Coates. Machine learning approaches to network anomaly detection. In *Proceedings of the Second Workshop on Tackling Computer Systems Problems with Machine Learning*. USENIX Association, 2007.

[5] F. Angiulli. Condensed Nearest Neighbor Data Domain Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1746–1758, 2007.

[6] California Polytechnic State University Architecture Department. Architecture department weather data. http://www.caed.calpoly.edu/sites/ehhf/weatherstation.html. (URL correct as of April 2009).

[7] S. Bay, K. Saito, N. Ueda, and P. Langley. A framework for discovering anomalous regimes in multivariate time-series data with local models. http://www.isle.org/ sbay/papers/darts.pdf, 2004. (URL correct as of April 2009).

[8] M. Berthold and J. Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In *Advances in Neural Information Processing Systems*, volume 7, pages 521–528. MIT Press, 1995.

[9] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[10] R. Bogacz, M. Brown, and C. Giraud-Carrier. High capacity neural networks for familiarity discrimination. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 2, pages 773–778. IEEE, 1999.

[11] R. Bolton and D. Hand. Statistical fraud detection: A review. *Statistical Science*, 17:235–255, 2002.

[12] O. Bretscher. *Linear Algebra with Applications.* Prentice Hall, 2001.

[13] C. Campbell and K. Bennett. A linear programming approach to novelty detection. In *Advances in Neural Information Processing Systems*, volume 13, pages 395–401. MIT Press, 2000.

[14] G. Carpenter and S. Grossberg. ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

[15] G. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[16] G. Carpenter and S. Grossberg. ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3:129–152, 1990.

[17] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, and D. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3:698–713, 1992.

[18] G. Carpenter, S. Grossberg, and D. Rosen. ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*, 4:493–504, 1991.

[19] G. Carpenter, S. Grossberg, and D. Rosen. Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns. In *Proceedings of the 1991 International Joint Conference on Neural Networks*, volume 2, pages 411–416. IEEE, 1991.

[20] G. Carpenter, S. Grossberg, and D. Rosen. Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.

[21] G. Carpenter, M. Rubin, and W. Streilein. ARTMAP-FD: Familiarity discrimination applied to radar target recognition. In *Proceedings of the 1997 International Conference on Neural Networks*, volume 3, pages 1459–1464. IEEE, 1997.

[22] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems*, volume 13, pages 409–415. MIT Press, 2001.

[23] C. Chang and C. Lin. *LIBSVM: A library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm (URL correct as of April 2009).

[24] G. Clarke and D. Cooke. *A Basic Course in Statistics*. Arnold, 2004.

[25] L. Cordella, C. De Stefano, F. Tortorella, and M. Vento. A method for improving classification reliability of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 6:1140–1147, 1995.

[26] T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[27] L. Cox, M. Johnson, and K. Kafadar. Exposition of statistical graphics technology. In *ASA Proceedings of the Statistical Computation Section*, pages 55–56, 1982.

[28] P. Crook and G. Hayes. A robot implementation of a biologically inspired method for novelty detection. In *Proceedings of Towards Intelligent Mobile Robots Conference*. Department of Computer Science, University of Manchester, 2001.

[29] P. Crook, S. Marsland, G. Hayes, and U. Nehmzow. A tale of two filters - on-line novelty detection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3894–3899. IEEE, 2002.

[30] F. Desobry, M. Davy, and C. Doncarli. An online kernel change detection algorithm. *IEEE Transaction on Signal Processing*, 53:2961–2974, 2005.

[31] D. Dong and J. Atick. Statistics of natural time-varying images. *Network: Computation in Neural Systems*, 6:345–358, 1995.

[32] Y. Dong, Z. Sun, and H. Jia. A cosine similarity-based negative selection algorithm for time series novelty detection. *Mechanical Systems and Signal Processing*, 20:1461–1472, 2006.

[33] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley, 2nd edition, 2001.

[34] E. Eiben and J. Smith. *Introduction to Evolutionary Computing.* Springer, 2003.

[35] P. Elias. *Predictive Coding.* PhD thesis, Faculty of Arts and Sciences, Harvard University, 1950.

[36] P. Elias. Predictive coding - parts I and II. *IRE Transactions on Information Theory*, 1:16–33, 1955.

[37] Y. Engel, S. Mannor, and R. Meir. The kernel recursive least-squares algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285, 2004.

[38] C. Fang, S. Chen, and C. Fuh. Automatic change detection of driving environments in a vision-based driver assistance system. *IEEE Transactions on Neural Networks*, 14:646–657, 2003.

[39] L. Fausett. *Fundamentals of Neural Networks.* Prentice Hall International, 1994.

[40] T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Labs, 2003.

[41] R. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188, 1936.

[42] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley, 1966.

[43] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 202–212. IEEE, 1994.

[44] A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer, 2002.

[45] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 41–49. Erlbaum, 1987.

[46] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101:e215–e220, 2000.

[47] F. González and D. Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4:383–403, 2003.

[48] F. González, D. Dasgupta, and R. Kozma. Combining negative selection and classification techniques for anomaly detection. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 1, pages 705–710. IEEE, 2002.

[49] E. Granger, S. Grossberg, M. Rubin, and W. Streilein. Familiarity discrimination of radar pulses. In *Advances in Neural Information Processing Systems*, volume 11, pages 875–881. MIT Press, 1999.

[50] S. Grossberg. Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions. *Biological Cybernetics*, 23:187–202, 1976.

[51] S. Grossman. *Elementary Linear Algebra*. Wadsworth, 1987.

[52] S. Guttormsson, R. Marks II, M. El-Sharkawi, and I. Kerszenbaum. Elliptical novelty grouping for on-line short-turn detection of excited running rotors. *IEEE Transactions on Energy Conversion*, 14:16–22, 1999.

[53] S. Haggett, D. Chu, and I. Marshall. Evolving a dynamic predictive coding mechanism for novelty detection. *Knowledge-Based Systems*, 21:217–224, 2008.

[54] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[55] S. Han and S. Cho. Evolutionary neural networks for anomaly detection based on the behavior of a program. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 36:559–570, 2006.

[56] C. Harrison. Experiments with linear prediction in television. *Bell System Technical Journal*, 31:764–783, 1952.

[57] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, international edition, 1999.

[58] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.

[59] A. Hofmann and B. Sick. Evolutionary optimization of radial basis function networks for intrusion detection. *Proceedings of the International Joint Conference on Neural Networks*, 1:415–420, 2003.

[60] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[61] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United Stated of America*, 79:2554–2558, 1982.

[62] A. Hopgood. *Intelligent Systems for Engineers and Scientists.* CRC Press, 2001.

[63] T. Hosoya, S. Baccus, and M. Meister. Dynamic predictive coding by the retina. *Nature*, 436:71–77, 2005.

[64] J. Jackson. *A User's Guide to Principal Components.* Wiley, 1991.

[65] N. Japkowicz, C. Myers, and M. Gluck. A novelty detection approach to classification. In *Proceedings of the 14th International Conference on Artificial Intelligence*, pages 518–523, 1995.

[66] Z. Ji. *Negative Selection Algorithms: from the Thymus to V-detector.* PhD thesis, University of Memphis, 2006.

[67] E. Keogh and J. Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and Information Systems*, 8:154–177, 2005.

[68] E. Keogh, J. Lin, and A. Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 226–233. IEEE, 2005.

[69] E. Keogh, J. Lin, and W. Truppel. Clustering of time series subsequences is meaningless: implications for past and future research. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 115–122. IEEE, 2003.

[70] T. Kohonen. *Self-Organizing Maps.* Springer, 3rd edition, 2001.

[71] T. Kohonen. Self-organized maps of sensory events. *Philosophical Transactions of the Royal Society A*, 361:1177–1186, 2003.

[72] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[73] M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233–243, 1991.

[74] S. Kuffler. Discharge patterns and functional organisation of mammalian retina. *Journal of Neurophysiology*, 16:37–68, 1953.

[75] H. Lee and S. Cho. The novelty detection approach for different degrees of class imbalance. In *Lecture Notes in Computer Science*, volume 4233, pages 21–30. Springer, 2006.

[76] Y. Li, M. Pont, and N. Jones. Improving the performance of radial basis function classifiers in condition monitoring and fault diagnosis applications where 'unknown' faults may occur. *Pattern Recognition Letters*, 23:569–577, 2002.

[77] Y. Liao, V. Vemuri, and A. Pasos. Adaptive anomaly detection with evolving connectionist systems. *Journal of Network and Computer Applications*, 30:60–80, 2007.

[78] J. Ma and S. Perkins. Online novelty detection on temporal sequences. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 613–618. ACM, 2003.

[79] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1741–1745. IEEE, 2003.

[80] J. Ma, J. Theiler, and S. Perkins. Accurate on-line support vector regression. *Neural Computation*, 15:2683–2703, 2003.

[81] M. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[82] J. Madera and B. Dorronsoro. Estimation of distribution algorithms. In E. Alba and R. Martí, editors, *Metaheuristic Procedures for Training Neural Networks*, chapter 6, pages 87–108. Springer, 2006.

[83] M. Markou and S. Singh. Novelty detection: a review - part 1: statistical approaches. *Signal Processing*, 83:2481–2497, 2003.

[84] M. Markou and S. Singh. Novelty detection: a review - part 2: neural network based approaches. *Signal Processing*, 83:2499–2521, 2003.

[85] S. Marsland. *On-line novelty detection through self-organisation, with application to inspection robotics*. PhD thesis, Department of Computer Science, University of Manchester, 2001.

[86] M. Martinelli, E. Tronci, G. Dipoppa, and C. Balducelli. Electric power system anomaly detection using neural networks. In *Lecture Notes in Computer Science*, volume 3123, pages 1242–1248. Springer, 2004.

[87] Mathworks. Control System Toolbox: Hard-disk read/write head controller. http://www.mathworks.com/access/helpdesk/help/toolbox/control/casestudies/f0-1000868.html. (URL correct as of April 2009).

[88] R. Maxion and K. Tan. Anomaly detection in embedded systems. *IEEE Transactions on Computers*, 51:108–120, 2002.

[89] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1999.

[90] T. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, international edition, 1997.

[91] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In *Lecture Notes in Computer Science*, volume 141, pages 178–187. Springer, 1996.

[92] A. Oliveira, F. Neto, and S. Meira. Novelty detection for short time series with neural networks. In *Design and Application of Hybrid Intelligent Systems*, pages 66–75. IOS Press, 2003.

[93] A. Oliveira, F. Neto, and S. Meira. Improving novelty detection in short time series through RBF-DDA parameter adjustment. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, volume 3, pages 2123–2128. IEEE, 2004.

[94] C. O'Reilly and Y. Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press, 2000.

[95] N. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1:67–90, 1993.

[96] F. Rosenblatt. The perceptron: A probabilistic model for information storage in the brain. *Psychological Review*, 65:386–408, 1958.

[97] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1:318–362, 1986.

[98] B. Samanta. Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. *Mechanical Systems and Signal Processing*, 18:625–644, 2004.

[99] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

[100] B. Schölkopf, J. Platt, and A. Smola. Kernel method for percentile feature extraction. Technical Report MSR-TR-2000-22, Microsoft Research, 2000.

[101] A. Secker and A. Freitas. WAIRS: Improving classification accuracy by weighting attributes in the AIRS classifier. In *IEEE Congress on Evolutionary Computation*, pages 3759–3765. IEEE, 2007.

[102] J. Shaffer. Multiple hypothesis testing. *Annual Review of Psychology*, 46:561–584, 1995.

[103] N. Siebel and G. Sommer. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems*, 4:171–183, 2007.

[104] S. Singh and M. Markou. An approach to novelty detection applied to the classification of image regions. *IEEE Transactions on Knowledge and Data Engineering*, 16:396–407, 2004.

[105] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[106] H. Sohn, K. Worden, and C. Farrar. Novelty detection under changing environmental conditions. In *Proceedings of the 8th Annual SPIE International Symposium on Smart Structures and Materials*, pages 108–118. SPIE, 2001.

[107] M. Srinivasan, S. Laughlin, and A. Dubs. Predictive Coding: A fresh view of inhibition in the retina. *Proceedings of the Royal Society of London, Series B*, 216:427–459, 1982.

[108] K. Stanley. *Efficient Evolution of Neural Networks through Complexification.* PhD thesis, Artificial Intelligence Laboratory, University of Texas at Austin, 2004.

[109] K. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1757–1762. IEEE, 2002.

[110] P. Sterling. Retina. In G. Shepherd, editor, *The Synaptic Organization of the Brain*, chapter 6, pages 170–213. Oxford University Press, 3rd edition, 1990.

[111] T. Stibor, P. Mohr, and J. Timmis. Is negative selection appropriate for anomaly detection? In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 321–328. ACM, 2005.

[112] J. Takeuchi and K. Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering*, 18:482–492, 2006.

[113] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Pearson, international edition, 2006.

[114] D. Tax and R. Duin. Data domain description using support vectors. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 251–256, 1999.

[115] D. Tax and R. Duin. Support vector domain description. *Pattern Recognition Letters*, 20:1191–1199, 1999.

[116] D. Tax and R. Duin. Data description in subspaces. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 672–675. IEEE, 2000.

[117] D. Theofilou, V. Steuber, and E. De Schutter. Novelty detection in a Kohonen-like network with a long-term depression learning rule. *Neurocomputing*, 52-54:411–417, 2003.

[118] B. Thompson, R. Marks II, J. Choi, M. El-Sharkawi, M. Huang, and C. Bunje. Implicit learning in autoencoder novelty assessment. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 3, pages 2878–2883. IEEE, 2002.

[119] R. Unnthorsson, T. Runarsson, and M. Jonsson. Model selection in one-class ν-SVMs using RBF kernels. In *Proceedings of the 16th International Congress and Exhibition on Condition Monitoring And Diagnostic Engineering Management*, 2003.

[120] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 2000.

[121] G. Vasconcelos, M. Fairhurst, and D. Bisset. A bootstrap-like rejection mechanism for multilayer perceptron networks. In *II Simpósio Brasileiro de Redes Neurais*, pages 167–172, 1995.

[122] H. Vieira Neto and U. Nehmzow. Visual novelty detection with automatic scale selection. *Robotics and Autonomous Systems*, 55:693–701, 2007.

[123] A. Weigend and N. Gershenfeld, editors. *Time Series Prediction: Forecasting the future and Understanding the Past*. Addison-Wesley, 1994.

[124] D. Whitley. Genetic algorithms and neural networks. In J. Periaux, M. Galan, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, pages 203–216. Wiley, 1995.

[125] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.

[126] W. Wollberg and O. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences of the United States of America*, 87:9193–9196, 1990.

[127] S. Wu and T. Chow. Induction machine fault detection using SOM-based RBF neural networks. *IEEE Transactions on Industrial Electronics*, 51:183–194, 2004.

[128] X. Xuan and K. Murphy. Modeling changing dependency structure in multivariate time series. In *Proceedings of the 24th International Conference on Machine learning*, pages 1055–1062. ACM, 2007.

[129] Y Yang, J. Zhang, J. Carbonell, and C. Jin. Topic-conditioned novelty detection. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 688–693. ACM, 2002.

[130] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447, 1999.