Regularised feed forward neural networks for streamed data classification problems

by

Mathys Ellis

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

August 2020

Publication data:

Mathys Ellis. Regularised feed forward neural networks for streamed data classification problems. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, August 2020.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

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

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

Regularised feed forward neural networks for data stream classification problems

by

Mathys Ellis
E-mail: mox.1990@gmail.com

Abstract

Streamed data classification problems (SDCPs) require classifiers with the ability to learn and to adjust to the underlying relationships in data streams, in real-time. This requirement poses a challenge to classifiers, because the learning task is no longer just to find the optimal decision boundaries, but also to track changes in the decision boundaries as new training data is received. The challenge is due to concept drift, i.e. the changing of decision boundaries over time. Changes include disappearing, appearing, or shifting decision boundaries. This thesis proposes an online learning approach for feed forward neural networks (FFNNs) that meets the requirements of SDCPs. The approach uses regularisation to optimise the architecture via the weights, and quantum particle swarm optimisation (QPSO) to dynamically adjust the weights. The learning approach is applied to a FFNN, which uses rectified linear activation functions, to form a novel SDCP classifier. The classifier is empirically investigated on several SDCPs. Both weight decay (WD) and weight elimination (WE) are investigated as regularisers. Empirical results show that using QPSO with no regularisation, causes the classifier to completely saturate. However, using QPSO with regularisation enables the classifier to dynamically adapt both its implicit architecture and weights as decision boundaries change. Furthermore, the results favour WE over WD as a regulariser for QPSO.

Keywords: Data streams, Classification problems, Feed Forward Neural Networks, Quantum Particle Swarm Optimisation, Regularisation, Concept drift.

Supervisors : Dr. A. S. Bosman and Prof. A. P. Engelbrecht (Co-supervisor)
Department : Department of Computer Science
Degree : Master of Science

The true sign of intelligence is not knowledge but imagination.

Albert Einstein (1879 – 1955)

Acknowledgements

I would like to thank the following people and institutions:

- My supervisors, Doctor Anna Bosman and Professor Andries Engelbrecht, for their insight and guidance during the course of this thesis.
- My parents, Thys and Moeka Ellis, for their unwavering support.
- Theonette van Niekerk, for her loving support.
- The National Research Foundation (NRF), for their financial support.

Contents

List of Figures vii

List of Algorithms xi

List of Tables xii

1 Introduction 1

1.1 Motivation . . . 2
1.2 Objectives . . . 3
1.3 Contributions . . . 4
1.4 Thesis Outline . . . 5

2 Computational intelligence problems 7

2.1 Problem classes . . . 8
2.1.1 Optimisation . . . 8
2.1.2 Approximation . . . 10
2.2 Problem environments . . . 13
2.2.1 Static environments . . . 13
2.2.2 Dynamic environments . . . 14
2.3 Summary . . . 16

3 Particle swarm optimisation 18

3.1 How particle swarm optimisation works . . . 19
3.1.1 Momentum component . . . 20
3.1.2 Cognitive and social components . . . 22
3.1.3 Swarm diversity . . . 22

3.1.4 Exploration-exploitation trade-off . . . 23
3.1.5 Controlling velocity . . . 24
3.1.6 Swarm size . . . 25
3.1.7 Neighbourhood topology . . . 26
3.1.8 Particle initialisation . . . 28
3.1.9 Iteration strategies . . . 29
3.1.10 Stopping conditions . . . 29
3.2 Particle swarm optimisation in dynamic environments . . . 30
3.2.1 Issues to consider for dynamic environments . . . 30
3.2.2 Dynamic particle swarm optimisation . . . 31
3.3 Summary . . . 34

4 Artificial neural networks **36**
4.1 How artificial neural networks work . . . 36
4.2 Architecture . . . 38
4.2.1 Feed forward neural networks . . . 39
4.2.2 Neuron unit types . . . 41
4.2.3 Activation functions . . . 42
4.3 Learning . . . 43
4.3.1 Learning strategy . . . 45
4.3.2 Model errors . . . 46
4.3.3 Performance measures . . . 47
4.3.4 Overfitting . . . 51
4.3.5 Underfitting . . . 53
4.3.6 Weight initialisation . . . 53
4.3.7 Stopping conditions . . . 55
4.3.8 Architecture selection . . . 55
4.3.9 Weight adjustment . . . 57
4.4 Training artificial neural networks using particle swarm optimisation . . . 60
4.4.1 Particle swarm optimisation training algorithms . . . 61
4.4.2 Saturation . . . 62
4.4.3 Measuring saturation . . . 63

4.5 Summary 64

5 Stream data classification problems: A real-world concern 65
5.1 Background 65
5.2 Streamed data classifier requirements 67
5.3 Literature review on streamed data classifiers and related works 70
5.3.1 Online learning approaches 70
5.3.2 Decision trees 72
5.3.3 Ensembles 73
5.3.4 Artificial Neural networks 74
5.3.5 Conclusion 76
5.4 Summary 78

6 Quantifying the environment of a streamed data classification problem 79
6.1 Issues with potential severity measures 80
6.2 Identifying environment instances 81
6.3 Spatial severity measure 84
6.4 Temporal severity measure 85
6.5 Normalising severity measures 85
6.5.1 Normalising spatial severity 86
6.5.2 Normalising temporal severity 87
6.6 Dynamism of problem environments 87
6.7 Summary 91

7 Regularised feed forward neural networks as streamed data classifiers 92
7.1 Proposed architecture 92
7.2 Back propagation learning algorithms 94
7.2.1 Back propagation weights adjustment algorithm 94
7.2.2 Weight decay learning algorithm 96
7.2.3 Weight elimination learning algorithm 96
7.3 Quantum particle swarm optimisation learning algorithms 97

7.3.1 Quantum particle swarm optimisation weights adjustment algorithm . . . 97
7.3.2 Weight decay learning algorithm . . . 97
7.3.3 Weight elimination learning algorithm . . . 97
7.4 Proposed streamed data classifiers . . . 98
7.5 Justification of proposed streamed data classifiers . . . 99
7.5.1 Saturation issue . . . 99
7.5.2 Local optimum trapping issue . . . 100
7.5.3 Bounded memory requirement . . . 101
7.5.4 Unbounded dataset requirement . . . 101
7.5.5 Concept drift requirement . . . 102
7.5.6 Random dynamics requirement . . . 102
7.5.7 Online learning requirement . . . 103
7.5.8 High speed data streams requirement . . . 103
7.5.9 One-pass requirement . . . 104
7.5.10 Limited number of tunable control parameters requirement . . . 104
7.5.11 Maintain low model complexity requirement . . . 105
7.5.12 Robustness requirement . . . 106
7.5.13 Fault tolerance requirement . . . 106
7.5.14 Conclusion . . . 106
7.6 Summary . . . 107

8 Empirical process **108**
8.1 Hypotheses about classifiers . . . 108
8.2 Baseline classifiers . . . 110
8.3 Benchmark streamed data classification problems . . . 110
8.3.1 Reasons for using the five problem domains . . . 111
8.3.2 Problem domains . . . 112
8.3.3 Data preparation . . . 118
8.3.4 Construction of benchmark problems . . . 120
8.3.5 Problem environment analysis . . . 123
8.3.6 Problem difficulty analysis . . . 133

8.4 Performance measurement . . . 136
8.4.1 Performance measuring methodology . . . 136
8.4.2 Saturation performance measures . . . 140
8.4.3 Accuracy performance measures . . . 141
8.4.4 Structural complexity performance measures . . . 142
8.4.5 Computational complexity performance measures . . . 145
8.4.6 Overfitting performance measures . . . 146
8.4.7 Control parameter impact on performance measures . . . 146
8.4.8 Weight distribution performance measures . . . 147
8.4.9 Swarm diversity performance measures . . . 148
8.5 Control parameter tuning process . . . 148
8.6 Benchmarking process . . . 152
8.7 Result analysis methodology . . . 153
8.7.1 Descriptive statistics . . . 153
8.7.2 Mann-Whitney-U-based ranking . . . 154
8.7.3 Performance trends . . . 155
8.8 Summary . . . 156

9 Empirical analysis **157**
9.1 Accuracy performance analysis . . . 158
9.2 Saturation analysis . . . 167
9.3 Complexity performance analysis . . . 176
9.4 Saturation, accuracy and complexity performance trends analysis . . . 182
9.5 Overfitting analysis . . . 187
9.6 Overall statistical rank analysis . . . 191
9.7 Control parameters analysis . . . 194
9.8 Weight distribution analysis . . . 198
9.9 Swarm diversity analysis . . . 205
9.10 Conclusion . . . 213
9.10.1 Remarks on the primary objective . . . 213
9.10.2 Remarks on the secondary objectives . . . 215
9.11 Summary . . . 220

10 Conclusions **221**
10.1 Summary of Conclusions 221
10.2 Future Work 224

Bibliography **227**

A Performance trend graphs **241**

B Acronyms **282**

C Symbols **284**
C.1 Chapter 2 284
C.2 Chapter 3 284
C.3 Chapter 4 286
C.4 Chapter 6 288
C.5 Chapter 7 289
C.6 Chapter 8 290

List of Figures

2.1 Illustration of the two types of global optima 9
2.2 Minimisation optimisation problem with local minima 11
2.3 Example of a classification problem 13
2.4 Illustration of a dynamic classification problem 15
2.5 Dynamic environment classifications 16

3.1 PSO topology comparisons 26
3.2 Sampling distribution comparisons 34

4.1 3-layer FFNN architecture 40
4.2 Activation functions comparison 42
4.3 ANN overfitting versus underfitting 54
4.4 Neuron saturation 63

6.1 Contour plot of ζ_{b_w} with respect to Θ'_{b_w} and τ'_{b_w} 90

8.1 Moving hyperplane problem domain illustration 113
8.2 Dynamic sphere problem domain illustration 114
8.3 Slding thresholds problem domain illustration 116
8.4 SEA concepts problem domain illustration 117
8.5 Illustration of sliding window algorithm 121
8.6 Benchmark problems' environment severity levels 130
8.7 Severity trends for streamed data classification problems 131
8.8 Overall benchmark problems' environment severity levels 132
8.9 Streamed data classification problem difficulty 134

9.1 PCC_g versus problem difficulty scatter plot 167

9.2 Raw MSE_g and MSE_t trends versus moving average $MSE_g \pm \sigma_{MSE_g}$ and MSE_t trends, and O_{MSE_g} for the BP classifiers 189
9.3 Raw MSE_g and MSE_t trends versus moving average $MSE_g \pm \sigma_{MSE_g}$ and MSE_t trends, and O_{MSE_g} for the QPSO classifiers 190
9.4 Classifier versus overall inverted MWU-Rank 192
9.5 Classifier versus overall MWU-Comparison winning percentages 193
9.6 Control parameter statistics for classifiers 196
9.7 Weight frequency distribution graphs for the classifiers 203
9.8 Average weight magnitude versus problem difficulty scatter plot 205
9.9 Swarm diverity trend of the QPSO-N classifier for the A1 hyperplane problem 208
9.10 Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for the A1 hyperplane, A1 sphere and A1 thresholds problems 210
9.11 Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for the A1 SEA and A1 electricity problems 211
9.12 Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for selected SEA problems 212

A.1 Legend for the performance trend analysis graphs 241
A.2 Performance trends of the BP classifiers for the A1 – A4 Hyperplane problems 242
A.3 Performance trends of the QPSO classifiers for the A1 – A4 Hyperplane problems 243
A.4 Performance trends of the BP classifiers for the B1 – B4 Hyperplane problems 244
A.5 Performance trends of the QPSO classifiers for the B1 – B4 Hyperplane problems 245
A.6 Performance trends of the BP classifiers for the C1 – C4 Hyperplane problems 246
A.7 Performance trends of the QPSO classifiers for the C1 – C4 Hyperplane problems 247

A.8 Performance trends of the BP classifiers for the D1 – D4 Hyperplane problems . . . 248
A.9 Performance trends of the QPSO classifiers for the D1 – D4 Hyperplane problems . . . 249
A.10 Performance trends of the BP classifiers for the A1 – A4 Sphere problems 250
A.11 Performance trends of the QPSO classifiers for the A1 – A4 Sphere problems . . . 251
A.12 Performance trends of the BP classifiers for the B1 – B4 Sphere problems 252
A.13 Performance trends of the QPSO classifiers for the B1 – B4 Sphere problems . . . 253
A.14 Performance trends of the BP classifiers for the C1 – C4 Sphere problems 254
A.15 Performance trends of the QPSO classifiers for the C1 – C4 Sphere problems . . . 255
A.16 Performance trends of the BP classifiers for the D1 – D4 Sphere problems 256
A.17 Performance trends of the QPSO classifiers for the D1 – D4 Sphere problems . . . 257
A.18 Performance trends of the BP classifiers for the A1 – A4 Thresholds problems . . . 258
A.19 Performance trends of the QPSO classifiers for the A1 – A4 Thresholds problems . . . 259
A.20 Performance trends of the BP classifiers for the B1 – B4 Thresholds problems . . . 260
A.21 Performance trends of the QPSO classifiers for the B1 – B4 Thresholds problems . . . 261
A.22 Performance trends of the BP classifiers for the C1 – C4 Thresholds problems . . . 262
A.23 Performance trends of the QPSO classifiers for the C1 – C4 Thresholds problems . . . 263
A.24 Performance trends of the BP classifiers for the D1 – D4 Thresholds problems . . . 264

A.25 Performance trends of the QPSO classifiers for the D1 – D4 Thresholds problems 265
A.26 Performance trends of the BP classifiers for the A1 – A4 SEA problems 266
A.27 Performance trends of the QPSO classifiers for the A1 – A4 SEA problems 267
A.28 Performance trends of the BP classifiers for the B1 – B4 SEA problems 268
A.29 Performance trends of the QPSO classifiers for the B1 – B4 SEA problems 269
A.30 Performance trends of the BP classifiers for the C1 – C4 SEA problems 270
A.31 Performance trends of the QPSO classifiers for the C1 – C4 SEA problems 271
A.32 Performance trends of the BP classifiers for the D1 – D4 SEA problems 272
A.33 Performance trends of the QPSO classifiers for the D1 – D4 SEA problems 273
A.34 Performance trends of the BP classifiers for the A1 – A4 Electricity problems 274
A.35 Performance trends of the QPSO classifiers for the A1 – A4 Electricity problems 275
A.36 Performance trends of the BP classifiers for the B1 – B4 Electricity problems 276
A.37 Performance trends of the QPSO classifiers for the B1 – B4 Electricity problems 277
A.38 Performance trends of the BP classifiers for the C1 – C4 Electricity problems 278
A.39 Performance trends of the QPSO classifiers for the C1 – C4 Electricity problems 279
A.40 Performance trends of the BP classifiers for the D1 – D4 Electricity problems 280
A.41 Performance trends of the QPSO classifiers for the D1 – D4 Electricity problems 281

List of Algorithms

1 Synchronous Particle Swarm Optimisation algorithm 21
2 Quantum Particle Swarm Optimisation algorithm 35
3 Supervised 3-layer FFNN stochastic BP algorithm 60
4 SDCP environment instance extraction algorithm 82
5 Supervised rectified linear unit (ReLU) summation 3-layer FFNN BP weights adjustment algorithm for an environment instance 95
6 Supervised ReLU summation 3-layer FFNN QPSO weights adjustment algorithm for an environment instance 98
7 Sliding windows algorithm 121

List of Tables

5.1 Comparison of the requirements focused on by the reviewed classifiers . 77

8.1 Sliding window parameter configurations for benchmark problems . . . 123
8.2 Environment analysis of the hyperplane problems 126
8.3 Environment analysis of the sphere problems 127
8.4 Environment analysis of the thresholds problems 128
8.5 Environment analysis of the SEA problems 129
8.6 Environment analysis of the electricity problems 129
8.7 Parameter tuning's parameter sets 150
8.8 Benchmark parameters for BP-N 150
8.9 Benchmark parameters for BP-WD 151
8.10 Benchmark parameters for BP-WE 151
8.11 Benchmark parameters for QPSO-N 151
8.12 Benchmark parameters for QPSO-WD 152
8.13 Benchmark parameters for QPSO-WE 152

9.1 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for accuracy performance measures 160
9.2 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for accuracy performance measures 162
9.3 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for accuracy performance measures 164

9.4 MWU-based pairwise comparison of the classifiers for accuracy performance measures (Wins/Ties/Losses percentages) 165
9.5 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for saturation performance measures 169
9.6 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for saturation performance measures 173
9.7 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for saturation performance measures 174
9.8 MWU-based pairwise comparison of the classifiers for saturation performance measures (Wins/Ties/Losses percentages) 175
9.9 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for complexity performance measures 178
9.10 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for complexity performance measures 179
9.11 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for complexity performance measures 180
9.12 MWU-based pairwise comparison of the classifiers for complexity performance measures (Wins/Ties/Losses percentages) 181
9.13 MWU-based ranking of the classifiers with regards to the performance measures (Wins/Ties/Losses percentages) 191
9.14 Comparison of the control parameters of the classifier and the overall performance of the classifiers 195
9.15 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for weight distribution performance measures 199

9.16 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for weight distribution performance measures 200
9.17 Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for weight distribution performance measures 201
9.18 MWU-based pairwise comparison of the classifiers for weight distribution performance measures (Wins/Ties/Losses percentages) 202
9.19 Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem domains for $\mathcal{D}$ 206
9.20 Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem difficulty for $\mathcal{D}$ 206
9.21 Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem environments for $\mathcal{D}$ 207
9.22 MWU-based pairwise comparison of the classifiers for $\mathcal{D}$ (Wins/Ties/Losses percentages) 208

Chapter 1

Introduction

The power to question is the basis of all human progress.

Indira Gandhi (1917 – 1984)

Data streams are common sources of data in the real world [1]. Because data streams provide sequential access to real-time data, the data they provide is referred to as *streamed data* [1][60]. The task of classifying streamed data according to a set of known labels in real-time is known as the streamed data classification problem (SDCP) [27].

Streamed data can undergo changes due to their real-world sources [98]. These changes cause the decision boundaries of SDCPs to either disappear, appear, or shift [94]. If a classifier is unable to track decision boundary changes, then the classifier will experience concept drift [98]. That is, the classifier will not be able to classify correctly due to the changed decision boundaries [112].

Various studies have been done on SDCPs and their potential classifiers, including feed forward neural networks (FFNNs) [1][13][21][24][60][62][67][76][93][98][102][109]. FFNNs are a powerful class of classifiers provided by the field of computational intelligence (CI) [101][119].

To prevent concept drift in a FFNN, the FFNN must be able to dynamically optimise its architecture and to dynamically adjust its weights [23][96][98][127]. The studies mentioned, however, showed that little work has been done on the subject of dynamic architecture optimisation for FFNNs with regards to SDCPs.

This thesis proposes a novel online learning algorithm that uses regularisation to dynamically optimise the architecture, and quantum particle swarm optimisation (QPSO) to dynamically adjust the weights. QPSO is a dynamic population-based CI optimiser, that combines principles of swarm movement and electron motion within atoms to optimise a problem [6]. The learning algorithm was applied to a 3-layer FFNN, using rectified linear units (ReLUs), to create a classifier for SDCP.

Both weight decay (WD) and weight elimination (WE) were considered as regularisers. The classifiers were empirically investigated on 80 SDCPs from five problem domains. The gradient-based back propagation (BP) optimisation algorithm for FFNNs was used as a benchmark for the classifiers [119].

The remainder of the chapter is organised as follows. Section 1.1 motivates the work reported in this thesis. Section 1.2 discusses the objective and sub-objectives of the research. Section 1.3 presents the contributions that the thesis has made to the field of CI. Lastly, Section 1.4 outlines the organisation of the remainder of the thesis.

1.1 Motivation

This section briefly motivates the work reported in this thesis. SDCP literature on 3-layer FFNN classifiers is limited.

Literature on the superset of SDCPs, i.e. dynamic classification problems, and 3-layer FFNN classifiers have been more abundant. Rakitianskaia and Engelbrecht [98] and Rakitianskaia [94] investigated the use of 3-layer FFNNs, trained by dynamic particle swarm optimisers (PSOs), as classifiers for dynamic classification problems [94]. Rakitianskaia [94] concluded that QPSO is suitable for training FFNNs which experience concept drift. Rakitianskaia [94], however, did not investigate the use of dynamic architecture optimisation in dynamic classification problems.

Aside from Rakitianskaia and Engelbrecht [98] and Rakitianskaia [94], there have not been any other significant investigations into the use of 3-layer FFNN, trained by dynamic PSOs, for dynamic classification problems. Most research on 3-layer FFNNs trained by PSOs focused on static classification problems, i.e. classification problems whose data does not undergo change [68][95][96][97][99][115].

Rakitianskaia and Engelbrecht [98] and Rakitianskaia [94], also did not cater for the effects of saturation that are common for PSO-based training of FFNN [96][115]. Rakitianskaia and Engelbrecht [96][97] did investigate several approaches to overcome the issue of saturation. However, controlling saturation was found to be a non-trivial task. Rakitianskaia and Engelbrecht [95] and Bosman *et al* [10] have suggested that regularisation can aid 3-layer FFNNs trained by PSOs for static classification problems by reducing saturation. Gupta and Lam [50] have shown that using WD regularisation can also improve the performance of FFNNs for static classification problems with noise. Furthermore, studies have shown that WD regularisation and WE regularisation can improve performance with respect to accuracy and complexity for FFNNs on static classification problems [10][68][95][118].

Despite the existence of regularisation techniques and QPSO, no work on using them together to train FFNNs for SDCPs has been done. Lastly, SDCP literature on 3-layer FFNNs use computationally expensive activation functions [76][102][109]. The real-time nature of streamed data, however, requires classifiers to be as computationally inexpense as possible [1][24]. A computationally less expensive alternative to commonly used FFNN activation functions, e.g. sigmoid, is the ReLU activation function [81][108].

Despite ReLU activation function being computationally inexpensive, their main advantage is that ReLU activation function do not saturate in the presences of positive signals [115].

The above gaps in SDCP literature motivate the investigation into regularised FFNNs as classifiers for SDCPs.

1.2 Objectives

The main objective of this thesis is to investigate the use of regularised FFNNs, trained by QPSO, as classifiers for SDCPs. The following sub-objectives are set out in order to ensure that the main objective of the thesis is achieved:

- Provide a sufficient overview of CI, PSOs, artificial neural networks (ANNs) and SDCPs.
- Review the literature on classifiers for SDCPs to motivate this work.

- Identify the requirements and issues that FFNNs need to address in order to be suitable for SDCPs.
- Find a method for simulating SDCPs for FFNNs.
- Find a suitable set of SDCPs which can be used to benchmark the classifiers investigated by this thesis.
- Propose a method to quantitatively analyse the dynamic environments of SDCPs, in order to gain a better understanding of the characteristics of the benchmark SDCPs.
- Propose a number of learning algorithms that use regularisation.
- Propose a viable FFNN design, i.e. architecture, that can be learnt by the proposed learning algorithms.
- Empirically investigate the proposed classifiers using the benchmark SDCPs.
- Determine if any of the proposed classifiers are suitable for SDCPs.
- Identify any shortcomings of the proposed classifiers, and propose possible future enhancements.

1.3 Contributions

The thesis makes the following contributions to the research area of SDCP, within the field of CI:

- A more complete quantification of the requirements that SDCPs impose on FFNN classifiers than currently available in the literature.
- A quantitative method for analysing the dynamic environments of SDCPs.
- A classification scheme for determining how difficult it is for a classifier to accurately learn SDCPs.

- An approach for the application and analysis of FFNNs classifiers on SDCPs.
- A statistically sound empirical investigation of regularised FFNN classifiers for SDCPs.
- A basis for doing future research on regularised and non-regularised FFNN classifiers for SDCPs.
- A basis for doing future research on the use of dynamic PSO and dynamic architecture selection to train FFNN classifiers for SDCPs.

1.4 Thesis Outline

The rest of the thesis is organised as follows:

- **Chapter 2** provides a brief background on the field of CI, and the problems of optimisation and approximation.
- **Chapter 3** discusses PSO, with a focus on PSOs for dynamic environments and the QPSO.
- **Chapter 4** discusses ANNs, with a focus on FFNNs, BP, PSO-based training, and regularisation.
- **Chapter 5** discusses SDCPs and reviews currently available literature on SDCP and their classifiers.
- **Chapter 6** proposes several measures for quantitative analysis of the dynamic environments of SDCPs.
- **Chapter 7** proposes the regularised FFNN classifiers for SDCPs.
- **Chapter 8** presents the hypotheses that were evaluated by the empirical investigation, and the way in which the empirical investigation was conducted.
- **Chapter 9** empirically analyses and discusses the proposed classifiers.

- **Chapter 10** provides the conclusions and future work that were derived from the work done in this thesis.

The following appendices appear at the end of the thesis:

- **Appendix A** provides the performance trend graphs for the performance trend analysis of Chapter 9.
- **Appendix B** provides a list of the important acronyms used, or newly defined, in this thesis, as well as their associated definitions.
- **Appendix C** lists and defines the mathematical symbols used in this thesis, categorised according to the relevant chapters in which they appear.

Chapter 2

Computational intelligence problems

If I had 60 minutes to solve a problem, I'd spend 55 minutes defining it, and 5 minutes solving it.

Albert Einstein (1879 – 1955)

CI is a field within the discipline of artificial intelligence (AI) [33]. CI attempts to create "intelligent" software-based problem solvers for complex problems, whose analytical solutions are too complex or expensive to calculate [33]. Real-world problems, which are problems that occur in real-life situations, tend to be complex [33][39]. Hence, the major driving force behind CI is real-world application [33].

The primary real-world complexities encountered in CI literature are *randomness*, *error*, *change*, *time*, and *space*. Randomness refers to the unpredictable nature of the real-world; error refers to the potential for erroneous problem solver behaviours and erroneous problem information; change describes the potential for problem information to change over time; time describes the computational time constraints of problems; lastly, space refers to the spatial constraints of problem solvers, i.e. available memory [1][2][33][75].

CI problem solvers are generally made up of models and algorithms that are inspired by nature [33][94]. The reason that CI models and algorithms draw on nature for inspiration is because nature has been very successful at solving problems. Such success is illustrated, for example, by the flying patterns formed by a flock of birds that improve the overall aerodynamics of the flock [28]. In the remainder of this thesis, any problem

targeted by CI is referred to as a *CI problem*. CI problems are best understood in light of their categorisation. This chapter introduces and discusses the various categorisations, with the intention of giving sufficient background on the CI problems that are examined in this thesis.

The remainder of the chapter is organised as follows: Section 2.1 discusses two commonly occurring classes of CI problems. Section 2.2 discusses the two types of environments in which CI problems occur, i.e. static and dynamic. Lastly, Section 2.3 summarises the chapter.

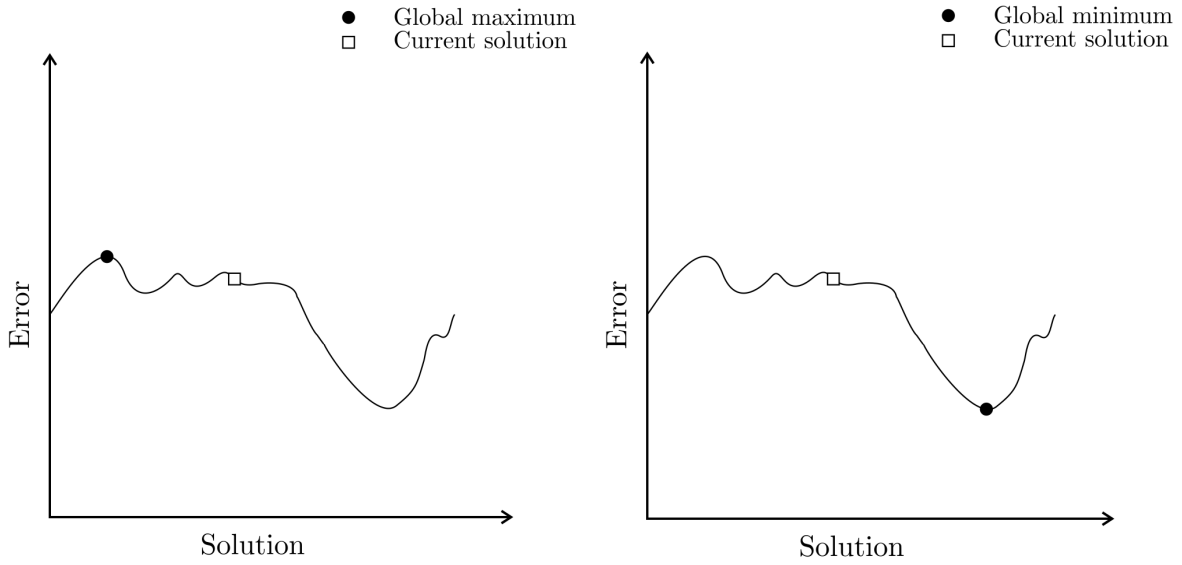
2.1 Problem classes

A CI problem class is the categorisation of what the information available in the *problem domain* represents [33][94]. Knowing what the information represents is important, because it determines the objective of a problem solver [33]. Two fundamental classes of CI problems are *optimisation* and *approximation* [33][39]. These two classes are elaborated on in sections 2.1.1 and 2.1.2, respectively.

2.1.1 Optimisation

An optimisation problem is a class of CI problems that consists of one or more functions, known as objective functions, and zero or more constraints [94]. An objective function represents a metric that is used to evaluate the quality of a solution. The solution is a vector of values assigned to the decision variables of the objective function. Thus, a solution represents a possible answer to the mathematical problem in question [33][63]. A space representing the quality of all the possible solutions, known as the problem's *search space*, is created by the objective function and the decision variables' domains. The objective of an optimisation problem solver is to search the search space to find the best solutions, known as the *global optima*, for the problem [33]. Problem solvers that deal with optimisation problems are called *optimisers*. When an optimiser stops searching because it has found an optimal solution, the optimiser is said to have *converged* on a solution.

Optimisation of an objective function takes one of two forms: *minimisation* or *maximisation* [33]. The former sees global optima as global minima. The latter sees global optima as global maxima. Figure 2.1 presents an illustration of the objective function curves of an arbitrary minimisation and maximisation optimisation problem.



(a) A maximisation problem with global maxima

(b) A minimisation problem with global minima

Figure 2.1: Illustration of the two types of global optima

Optimisation problems can be categorised based on the number of objectives they have, as follows:

- *single-objective*, i.e. one objective function [33].
- *multi-objective*, i.e. two or three objective functions [56].
- *many-objective*, i.e. more than three objective functions [59].

Multi- and many-objective problems are more difficult than single-objective problems, because the sub-objectives tend to work against each other, creating a set of non-dominating optima [56]. Each optimum in the set is a solution that provides a trade-off between the various sub-objectives [59]. Thus, most of the optima need to be found so that an appropriate trade-off solution can be chosen by domain experts [33]. A real-world

example of a multi-objective optimisation function is an attempt to find the best stock portfolio that will provide maximum growth and minimum risk over a specified number of years.

Furthermore, optimisation problems can be *boundary constrained*, *functionally constrained*, or *unconstrained* [33][96]. Boundary constrained optimisation problems are subject to search space boundaries. On the other hand, functionally constrained optimisation problems have constraints that render certain decision variable values infeasible [56]. Unconstrained optimisation problems do not impose any constraints on the problem [97]. The remainder of this thesis considers only unconstrained optimisation problems, because ANNs do not make use of any constraints [39].

A fatal problem that optimisers face is *local optimum trapping*: Any search space can be divided into regions. Each region may have an optimum. However, these optima might not be global optima, i.e. the best possible solutions for the entire search space. Such solutions are referred to as *local optima*, i.e. the optima that represent the best solutions in their corresponding region [33]. Local optimum trapping occurs when an optimiser has converged to one of the local optima rather than to a global optimum, and is unable to escape the region in which the local optimum is located [9][99]. A trapped optimiser cannot escape, because it does not have any information about a better region to which it can move, and does not have the means to gain such information. Therefore, local optimum trapping deteriorates optimiser performance, provided the optimiser does not have means to overcome local optimum trapping [33][39].

Figure 2.2 shows a local region on an objective function curve of a minimisation optimisation problem that can cause local optimum trapping.

2.1.2 Approximation

An approximation problem, also known as a function approximation problem, is a class of CI problems where the objective is to find a functional mapping between an I-dimensional input space and a K-dimensional target space, given a *data set* of *patterns* (D) [39][126]. That is,

$$f : \mathbb{R}^I \rightarrow \mathbb{R}^K$$

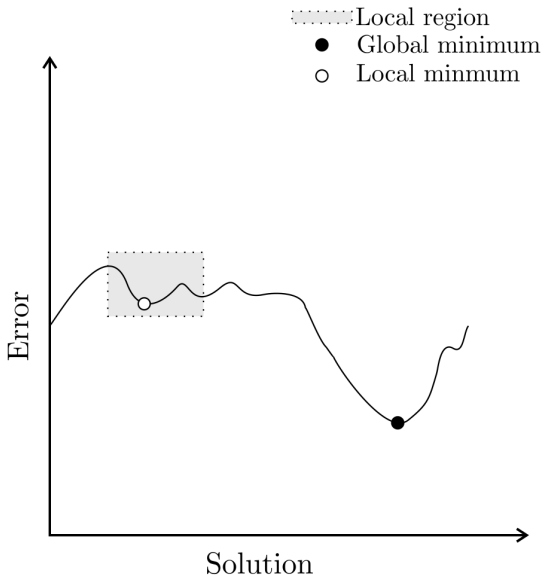


Figure 2.2: Minimisation optimisation problem with local minima

Each pattern, D_p, is a tuple, $(\vec{z}, \vec{t})$, of vectors, where $\vec{z}$ is the *input vector* consisting of I decision variables (independent variables) and $\vec{t}$ is the *target vector* consisting of K output variables (dependent variables) [2][126]. Problem solvers that deal with approximation problems are known as *approximators* [126].

Approximators produce *models*, i.e. parametrised functional mappings that approximate the given function. A classic example of a model is a linear combination of decision variables.

Models have one or more adjustable variables called *model parameters*, e.g. the coefficients in a linear combination of a set of independent variables [31]. Model parameters enable the model to approximate various functions [39]. Finding the model parameters' values that provide the best approximation of a particular function is an optimisation problem [33].

Approximators tend to automate the optimisation of model parameters through the use of optimisers [39][94]. CI-based approximators usually refer to such optimisers as *learning algorithms* [2].

Learning algorithms are guided by one or more *error functions*, i.e. objective functions, that calculate the *model error*. The model error is the deviation between the

model's approximation and target function [39].

The magnitude of the model error is directly related to how well the data set represents the target function, i.e. the distribution and size of the data set [2][126]. The less representative the data set is of the target function, the more difficult it will be to obtain an accurate approximation [33][39]. Thus, the higher the model error tends to be.

To prevent high model errors, the approximation must be made carefully so as to not only represent the provided data set, but also *unseen patterns*. This allows the approximation to *generalise* well, i.e. accurately approximate the target vectors of the patterns not used during model construction [2][33][126]. To produce an approximator that can generalise well requires a fairly-sized data set of well-distributed samples and a mechanism in the learning algorithm that can predict the *generalisation error* of the model [126].

Typical sub-classes of approximation problems include [33][105]:

- *Prediction problems*, which deal with the approximation of complex trends that can be used to forecast future movements, e.g. storm paths [75][105].

- *Classification problems*, which deal with the labelling of patterns according to known *classifications*, e.g. intrusion detection [2][33]. Approximators dealing with classification problems are called *classifiers* [1][116]. A classifier learns to label patterns by approximating the *classification function* that maps an input vector to a discrete class [33][98]:

$$f : \mathbb{R}^I \rightarrow \{t_1, t_2, ..., t_K\}$$

  Classification functions represent the *decision boundaries*, e.g. separating hyperplanes, that separate the patterns of different target classes [94]. Figure 2.3 provides an illustration of a 2-dimensional classification problem, with two classes and non-linear decision boundaries.

The focus of this thesis is on classification problems. Hence, the remainder of the thesis will only consider classification problems in discussions related to approximation.

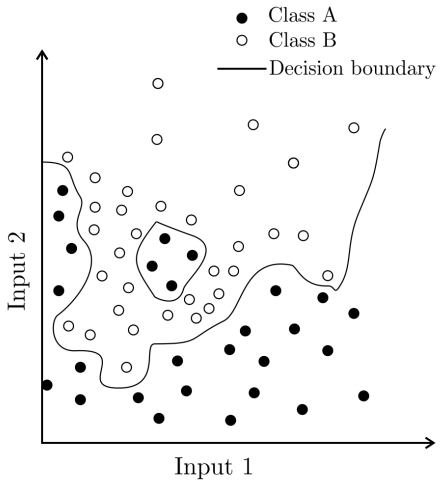


Figure 2.3: Example of a 2-dimensional classification problem with two classes and non-linear boundaries

2.2 Problem environments

A CI problem can further be categorised in terms of its problem environment, i.e. how the available information about the problem changes over time [25][33]. A CI problem can have either a *static environment* or a *dynamic environment* [60].

In the context of a classification problem, the problem environment refers to how its decision boundaries change over time [94]. Static and dynamic environments for classification problems are discussed in Sections 2.2.1 and 2.2.2, respectively.

2.2.1 Static environments

The decision boundaries of a problem with a static environment do not change in any way while the classifier is trained [25]. That is, once an optimal model is found, it does not change. Classification problems with static environments are known as *static classification problems* [30].

2.2.2 Dynamic environments

On the other hand, the decision boundaries of a problem with a dynamic environment can change in any way and at any point during the learning process [94]. Decision boundary changes include *shifting*, *rotation*, *disappearance*, and *appearance* of decision boundaries [62]. Each changed "version" of the decision boundaries is known as an *environment instance* of the classification problem [6]. Classification problems with dynamic environments are known as *dynamic classification problems* [33].

The decision boundaries of dynamic classification problems undergo changes due to changes in the data set of the problem [94]. Changes in the decision boundaries of the problem result in classifiers experiencing the phenomenon known as *concept drift* [98]. In the context of classification problems, concept drift refers to classifiers becoming *stale*, i.e. unable to classify correctly, because the decision boundaries learnt by the classifier no longer approximate the decision boundaries of the problem [13][94]. Concept drift also occurs when the *model architecture*, i.e. the structure of the classifier's model, changes [49].

Figure 2.4 provides an illustration of a dynamic 2-dimensional classification problem, with two classes, that has undergone one environment change. The figure demonstrates that a classifier trained on the previous environment instance will incorrectly classify several patterns as class A instead of B. A classifier experiencing concept drift, therefore, needs to correct its approximation through re-optimisation [62].

Dynamic classification problems are more difficult than static classification problems, because learning algorithms for dynamic classification problems need to track decision boundary changes, in addition to approximating the classification function [8][94]. Real-world classification problems typically are dynamic, because real-world data sets tend to change over time [8].

Decision boundary changes manifest in the error functions of learning algorithms as changing optima [30][99]. The severity, i.e. intensity, of optima changes are described by two parameters [3][11][25]:

- The *spatial severity*, which describes the *magnitude* of change that happens when changes occur. The smaller the spatial severity, the less severe the changes are and vice versa.

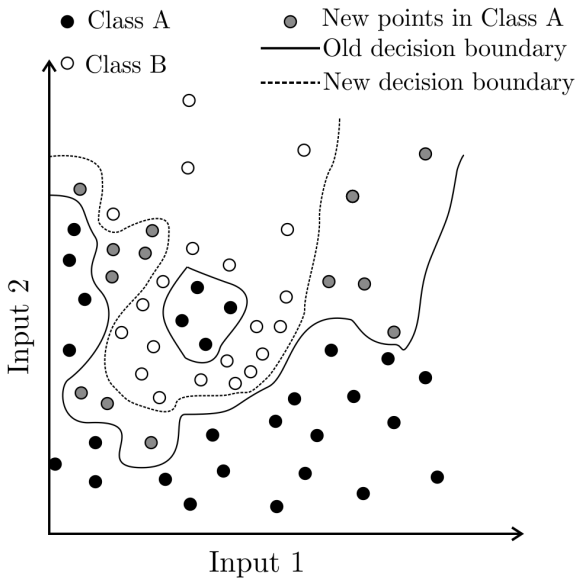


Figure 2.4: Two-dimensional dynamic classification problem with two classes and non-linear boundaries

- The *temporal severity*, which describes the *frequency* at which the changes occur over time. The smaller the temporal severity, the less frequent the changes and vice versa.

Duhain and Engelbrecht [25] presented a holistic classification scheme for dynamic optimisation problems using the two severity parameters. Because of the relationship between decision boundary changes and changing optima, the Duhain and Engelbrecht classification scheme can also be applied to dynamic classification problems. Four classes of dynamic classification problems are defined as illustrated by Figure 2.5 and discussed below:

- *Quasi-static* environments experience low amounts of spatial and temporal severity, i.e. decision boundary changes are not very severe and do not occur often [25]. This type of environment includes static and almost-static environments. Hence, a quasi-static environment should have a low performance impact on a classifier.

- *Progressive* environments experience low spatial severity and high temporal severity [25], i.e. decision boundary changes are not severe, but occur frequently [25].

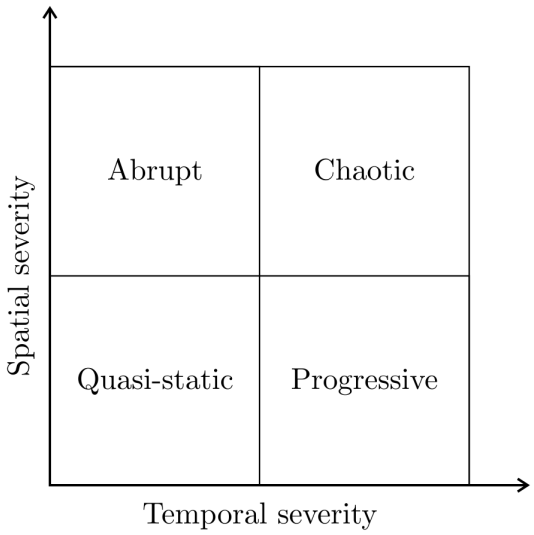


Figure 2.5: Dynamic environment classifications: Spatial severity versus temporal severity

Hence, the decision boundaries change smoothly over time.

- *Abrupt* environments experience high spatial severity and low temporal severity, i.e. decision boundary changes can be quite severe, but occur infrequently [25]. Hence, the decision boundaries do not change often, but, when they do change, the changes are significant.

- *Chaotic* environments experience high amounts of both spatial severity and temporal severity, i.e. decision boundary changes are severe and occur very frequently [25]. This makes the decision boundaries very erratic, and gives learning algorithms little time to train the classifier. Of the four classes, this type of environment will impact a classifier's performance the most [25][33].

2.3 Summary

This chapter discussed the CI problems of optimisation and approximation. Approximation itself is an optimisation problem, i.e. to find the optimal model to approximate a given function. A common type of approximation problem is the classification problem.

Classification problems can either be static or dynamic, based on whether the decision boundaries remained constant or changed over time. These changing decision boundaries result in classifiers experiencing concept drift, i.e. classifiers become outdated. Hence, the classifiers need to re-optimise their models to be able to classify correctly. Additionally, dynamic classification problems can further be classified as either quasi-static, progressive, abrupt or chaotic, using the spatial and temporal severities of their environments.

Chapters 3 and 4 build on the content discussed in this chapter through the introduction of a group of optimisers, known as particle swarm optimisers (PSOs), and a group of approximators, known as artificial neural networks (ANNs), respectively.

Chapter 3

Particle swarm optimisation

The fiercest serpent may be overcome by a swarm...

Isoroku Yamamoto (1884 – 1943)

Particle swarm optimisation (PSO) is a class of stochastic *population-based* optimisers that was introduced by Kennedy and Eberhart [28] in 1995. Population-based optimisers make use of more than one candidate solution at a time to optimise a problem, as opposed to traditional local search optimisation algorithms, such as hill climbers, that use only one solution at a time [33][39].

PSO has been successfully applied to a multitude of optimisation problems, ranging from single-objective to dynamic optimisation problems [9][33][48][52][85][95][98][127]. The remainder of this chapter elaborates on PSO, with the intention of giving sufficient background on PSO as a basis for the discussion in the remainder of this thesis.

The remainder of the chapter is organised as follows: Section 3.1 explains the mechanics of PSOs. Section 3.2 presents the issues that standard PSO algorithms face when dealing with dynamic optimisation problems, and discusses alternative PSOs that are better suited for dynamic optimisation problems. Lastly, Section 3.3 provides a summary of this chapter.

3.1 How particle swarm optimisation works

The first PSOs were derived from the movement patterns of bird flocks [28]. PSO consists of a *swarm*, which consists of n_p *particles*. The position of each particle in the swarm represents a candidate solution. The solution is stored as the particle's N-dimensional *position vector*, $\vec{x}$. Over the course of several optimisation iterations, the particles work together, as a collective, to find optimal solutions in the N-dimensional search space of the objective function [28][33],

$$f : \mathbb{R}^N \rightarrow \mathbb{R}$$

Particle movement is determined by one or more update rules, collectively referred to as the particle's *behaviour*. Behaviour can vary between particles [6]. *Sub-swarms* are thus formed by grouping particles based on their behaviours [34][48].

The standard particle update rules require each particle to have an N-dimensional *velocity vector*, $\vec{v}$. The velocity vector represents the step size and direction of the particle's movement at iteration t. Three components make up the velocity calculation: the *momentum*, *cognitive*, and *social* components [28][33]. Section 3.1.1 elaborates on the momentum component, and Section 3.1.2 elaborates on both the cognitive and social components.

The standard particle update rules consist of a velocity update rule,

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \tag{3.1}$$

and a position update rule,

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3.2}$$

where $v_{ij}(t)$ is the j-th element of the velocity vector of particle i at iteration t; $y_{ij}(t)$ is the j-th element of the personal best position of particle i at the iteration t; $\hat{y}_{ij}(t)$ is the value of the j-th element of the neighbourhood best position of particle i at iteration t; and $x_{ij}(t)$ is the j-th element of particle i's position vector at iteration t; $r_{1j}(t)$ and $r_{2j}(t)$ are the j-th elements of $\vec{r}_1$ and $\vec{r}_2$, respectively [33]. The elements of $\vec{r}_1$ and $\vec{r}_2$ are sampled uniformly from the range $[0, 1]$ [28]. The variables c_1 and c_2 represent the user-specified positive acceleration coefficients of the particle, and ω represents the user-specified inertia weight of the particle.

Together, the three velocity components aid in the control of a swarm's *exploration-exploitation trade-off* through *swarm diversity* [33][89]. Section 3.1.3 discusses the notion of swarm diversity, and Section 3.1.4 discusses the exploration-exploitation trade-off.

Care should be taken when manipulating the three components' parameters because velocity can explode and lead to a diverging swarm. Section 3.1.5 explains how the velocity calculation can be controlled to prevent the swarm from diverging.

Before a PSO can start optimising a problem, the following five design decisions have to be made:

1. How many particles should be used?
2. What kind of *neighbourhood topology*, i.e. social structure, will particles use?
3. How will particles be *initialised*?
4. When will particles be updated during an *iteration*?
5. Which *stopping conditions* should be used to stop the optimisation process?

Sections 3.1.6 to 3.1.10 discuss the effect of these algorithm settings, respectively.

Kennedy and Eberhart [28] initially developed two PSOs, known as the global best (gBest) and local best (lBest) PSOs. Both make use of the standard particle update rule and the synchronous iteration strategy, i.e. particle information is updated at the end of each iteration. Note that ω was initially not included, but implied as being equal to one. Since the introduction of the inertia weight by Shi and Eberhart [104], its use has become the *de facto* in the gBest and lBest PSOs literature.

The difference between the two PSOs is their selection of the social guide, i.e. their neighbourhood topology [33]. The gBest PSO employs the star topology, while the lBest PSO uses the ring topology.

Algorithm 1 presents the pseudocode employed by the gBest and lBest PSOs.

3.1.1 Momentum component

Momentum represents a particle's previous movement direction and magnitude [33]. Thus, momentum prevents the particles from rapidly changing direction, which may

Algorithm 1 Synchronous Particle Swarm Optimisation algorithm

Input: ω, c_1, c_2, n_p, $f(\vec{x})$, stopping conditions
  for $i = 0$ to n_p **do**
    Initialise $\vec{x}_i$ randomly
    Initialise $\vec{v}_i$ to zero vector
    Set $\vec{y}_i$ as $\vec{x}_i$
    Calculate particle i's quality, using $f(\vec{x})$
  end for
  Set t to 0
  while Stopping conditions not true **do**
    for $i = 0$ to n_p **do**
      Determine $\vec{\hat{y}}_i$ by looking for the best personal best in particle i's neighbourhood
      Update $\vec{v}_i$ using Equation (3.1)
      Update $\vec{x}_i$ using Equation (3.2)
      Calculate particle i's quality, using $f(\vec{x}_i)$
    end for
    for $i = 0$ to n_p **do**
      if ($\vec{x}_i$ is better than $\vec{y}_i$) **then**
        $\vec{y}_i = \vec{x}_i$
      end if
    end for
    $t = t + 1$
  end while
  return Best position found so far

cause it to haphazardly search and, as a result, miss good solutions [29]. The momentum component's contribution to the velocity is controlled by the inertia weight, ω [98].

The larger the value of ω is, the larger the momentum component's contribution to the velocity [33]. Thus, the longer it takes for a particle to drastically change course. This can make the particle search more of the search space, i.e. *explore*. However, if the particle keeps moving in the same direction, then a large ω can be bad for the particle because the particle's velocity might explode. This can cause the particle to skip over optimal solutions, and leave the search space all together.

On the other hand, the smaller the value of ω is, the easier it is for the particle to change course; thus, the more chance the particle has to concentrate on a region in the search space, i.e. *exploit*. However, if the ω is too small the particle might not be able to escape the region and become subject to local optimum trapping.

3.1.2 Cognitive and social components

The cognitive component represents the effect of the particle's past success on its current movement, i.e. *nostalgia*, whereas the social component simulates the effect of the particle's neighbours' success on the particle's movement, i.e. *envy* [28][33].

Both components make use of a *guide*. Guides are positions of the best solutions found either by a particle, group of particles, or the entire swarm thus far [64]. The cognitive guide, $\vec{y}_i$, is the personal best of particle i, and the social guide, $\hat{\vec{y}}_i$, is the neighbourhood best of the particle [33][64].

Both the cognitive and the social guides' contributions to the velocity are controlled by their respective user-specified positive acceleration coefficient, c_1 and c_2, and their respective uniformly sampled vectors, $\vec{r}_1$ and $\vec{r}_2$ [28].

The contributions of the cognitive or social component are weighted by the result of multiplying the respective component's positive acceleration coefficient and sampled vector, e.g. $c_1\vec{r}_1$ [28]. Hence, c_1 and c_2 control the maximum possible contributions that their respective component's guide can make to the velocity calculation [33]. If one positive acceleration coefficient is larger than the other, then there exists a greater chance that the respective component might overshadow the other [33].

3.1.3 Swarm diversity

Swarm diversity describes how different the particles' positions in the swarm are, i.e. the spread of the particles in the N-dimensional search space [33][89]. A diverse swarm indicates that particles are not close to one another, i.e. have not yet converged. A non-diverse swarm, on the other hand, indicates that particles have either converged around a point in the search space or collapsed [8]. The more diverse a swarm is, the more information the PSO has about the search space. Diverse swarms are less susceptible to local optimum trapping than non-diverse swarms [33].

Swarm diversity, $\mathcal{D}$, can be measured as the average Euclidean distance between particles and the swarm's centre, as follows [89]:

$$\mathcal{D} = \frac{\sum_{i=1}^{n_p} \sqrt{\sum_{n=1}^{N} (x_{ij} - \bar{x}_j)^2}}{n_p} \tag{3.3}$$

where n_p is the size of the swarm; x_{ij} is the j-th element of the position vector of particle i in the swarm; and $\bar{x}_j$ is the j-th element of the position vector of the swarm centre, which is calculated as the average over all the particle positions.

3.1.4 Exploration-exploitation trade-off

Exploration describes an optimiser's ability to search a wide area of the search space [89]. *Exploitation*, on the other hand, describes the optimiser's ability to concentrate on a particular area of the search space, thereby allowing the optimiser to converge if needed [33].

In the context of PSOs, exploration manifests in diverse swarms and exploitation manifests in non-diverse swarms [89][94]. If the exploration and exploitation levels are not appropriate for the optimisation problem, then the PSO might explore or exploit at the wrong times which may result in sub-optimal performance [33][52][89]. Hence, the user needs to decide on an appropriate exploration and exploitation trade-off [33].

The trade-off, however, is problem dependent, and control of it varies from optimiser to optimiser [33][52][89]. In static environments, the literature argues that optimisers, such as PSOs, should start off by exploring and then progress towards exploiting. This strategy allows the search to first determine the potential of various regions and then to focus on those regions with the most potential in order to find the best solution(s) [94].

In dynamic environments, the same explore-first-exploit-last strategy is recommended, but the strategy should be repeated for each environment instance, because each environment instance can be seen as a standalone problem [7][9][94]. Another suggested approach for dynamic environments is to have some members of the population to always explore while others exploit the findings of the exploring members, i.e. to *maintain diversity* [6].

PSOs that make use of the standard particle update rules allow exploration-exploitation trade-off control through ω, c_1 and c_2 [33][89][98].

The larger ω becomes, the more PSO facilitates exploration, since particles have slower reactions to their cognitive and social guides. Slower reaction means that the particles can move further beyond the boundaries imposed by cognitive and social guides [33]. This allows the PSO to maintain swarm diversity. On the other hand, the smaller ω becomes, the more a PSO facilitates exploitation because particles' movements become

more controlled by the cognitive and social guides. Thus, particles struggle to move beyond the boundaries imposed by the cognitive and social guides [33]. Depending on the values of c_1 and c_2, this leads to a reduction in swarm diversity.

If $c_1 = c_2$, then the exploration-exploitation trade-off is controlled primarily through ω. However, if $c_1 \neq c_2$, then the trade-off can be influenced as follows: if $c_1 > c_2$, then particles will concentrate more on their individual regions in the search space than on the region occupied by their neighbourhood best position, because the cognitive guide is favoured more [33]. This allows the current diversity to be preserved to some degree. If $c_1 < c_2$, then particles will concentrate more on the region occupied by their neighbourhood best position [33]. Hence, the swarm's diversity will be reduced by some degree. Therefore, the former case, i.e. $c_1 > c_2$, allows more exploration than the latter case, i.e. $c_1 < c_2$.

Manipulation of ω, c_1, and c_2 directly effects the acceleration of the particles during optimisation. Thus, the direct effect these control parameters have on the performance of a PSO. Optimal values for ω, c_1, and c_2, however, are problem-dependent [33][94]. Care should however be taken as certain combinations of ω, c_1, and c_2 values can result in exploding velocities [32]. Exploding velocities ultimately lead to divergence, i.e. particles move further and further apart from each other, therefore preventing the swarm from exploiting [32][94]. Cleghorn and Engelbrecht [18] empirically confirmed that there are regions within the value space of ω, c_1, and c_2 that result in a convergent optimisation process.

3.1.5 Controlling velocity

One possible solution to delay exploding velocities is *velocity clamping*, which places an upper bound on the magnitude of the velocity [98]. The upper bound is determined by the vector $\vec{v}_{max}$. The vector $\vec{v}_{max}$, however, is problem-dependent. Velocity clamping also adds more computational complexity to the PSO, specifically with reference to control parameter tuning [33]. Aside from problem dependency, velocity clamping can cause particles to change search direction or overshoot an optimum when at maximum velocity [33]. These two problems can lead to sub-optimal solutions being found, or a lack of convergence.

A more effective way of choosing ω, c_1, and c_2 than manually tuning their values, is to apply the *constriction coefficient* constant,

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \tag{3.4}$$

to the velocity update Equation (3.1) as follows [20][29][99]:

$$v_{ij}(t) = \chi[v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)]] \tag{3.5}$$

where κ is a constant and $\phi = c_1 r_{1j} + c_2 r_{2j}$.

The key advantage of using the constriction coefficient is that if Equation (3.5) is used under the constraints $\phi \geq 4$ and $\kappa \in [0, 1]$, then the swarm will converge in terms of statistical expectation to a point that may or may not be an optimum [20][29]. This ensures that velocities will not explode, therefore eliminating the need for velocity clamping [33].

Another alternative is to assign values to ω, c_1, and c_2 that satisfy the theoretically derived convergence conditions of the constriction coefficient. A set of well-known, empirically tested values are $\omega = 0.729844$ and $c_1 = c_2 = 1.496180$ [29][98][115].

Theoretical analysis of the PSO has further restricted the convergence conditions, to guarantee a higher order of stability as follows [15][16][17][19]:

$$|w| < 1$$

$$0 < c_1 + c_2 < \frac{24(1 - \omega^2)}{7 - 5\omega}$$

For the purpose of this thesis, it is sufficient to know that the values suggested above for ω, c_1, and c_2 conform to these convergence conditions. For more detailed works on the subject, the reader is referred to the works of Cleghorn and Engelbrecht [16][17][19].

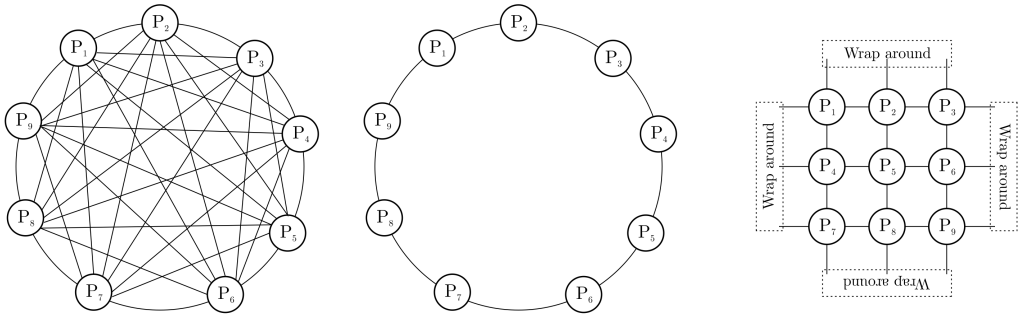
3.1.6 Swarm size

One of the most important PSO parameters to consider is the swarm size, n_p. A small swarm size means that there are only a few particles searching the search space. On the other hand, a large swarm size means that there are many particles searching the search space. Larger swarms allow for more exploration than smaller swarms [94]. However, the larger the swarm, the more computationally expensive the PSO [82].

Malan and Engelbrecht [82] showed that there is a critical swarm size, i.e. a limit, for each problem that, if exceeded, will result in the PSO's performance deteriorating. The choice for n_p is thus problem dependent, though studies indicate a good range to be between 10 and 30 particles [33][98][115].

3.1.7 Neighbourhood topology

The neighbourhood topology of a PSO describes the social structure of particles, defining how social guides are selected [64][94]. Three widely used neighbourhood topologies are the *star*, *ring*, and *Von Neumann* topologies [33][94]. Figure 3.1 provides illustrations of the three topologies for a swarm with nine particles. Each circle represents a particle, and a line between two particles indicate that they can communicate with each other.



(a) Star topology (b) Ring topology (c) Von Neumann topology

Figure 3.1: PSO topology comparisons for a swarm with 9 particles

Star topology

The star topology allows all particles in the swarm to exchange information about their best positions with one another [64]. The social guide is the best particle found by the entire swarm, because the swarm forms one neighbourhood [35].

The star topology allows information about the best positions in the search space to flow very quickly within the swarm; this in turn can lead to rapid convergence of the swarm [64][94]. Rapid convergence also entails a quick loss in swarm diversity, because particles are moving towards the global best position. Literature argues that the star

topology tends to favour exploitation [64]. However, Engelbrecht [35] empirically showed the best topology to be problem dependent.

Ring topology

The ring topology forms neighbourhoods of three particles based on the particle's index in the swarm [35]. Each particle i forms a neighbourhood with the particles to its left, i.e. $(i-1) \bmod n_p$, and right, i.e. $(i+1) \bmod n_p$ [64]. The social structure forms a ring-like shape, with overlapping neighbourhoods [64].

The small overlapping neighbourhoods slow down the flow of information about the best positions found among particles in the swarm. This slows down swarm convergence and the rate at which swarm diversity is lost [64]. Literature argues that the ring topology tends to favour exploration. However, Engelbrecht [35] empirically showed that the ring topology performance is problem dependent.

Von Neumann topology

The Von Neumann topology, named after John Von Neumann for his pioneering work in cellular automata, structures the swarm into a grid, the edges of which are connected to the edge on the opposite side of the grid [64][94]. In this way, each particle has a neighbour in each of the four basic directions, north, east, south and west [64][94]. Note that like the ring topology, the neighbourhood of the Von Neumann topology is based on particle indices.

The lattice social structure results in a slower flow of information about the best positions found among particles than the star topology [33]. Information about best positions flows faster than in the ring topology, because information flows in four directions instead of two [94].

The Von Neumann topology, therefore, tends to balance exploration and exploitation better than the other two topologies. On average, the Von Neumann topology tends to perform better than the star and ring topologies, across various problems, including dynamic optimisation problems [33][64][94][115].

3.1.8 Particle initialisation

Before starting the optimisation process, the positions and velocities of the n_p particles need to be initialised. Initialisation of these two vectors is important because the starting positions and velocities of particles affect the PSO's likelihood of finding a global optimum [33].

Position initialisation

Due to starting positions having an impact on the efficiency of a PSO, the positions should be initialised to cover the entire search space in a uniform manner [33][37]. This prevents biased searches, and reduces the likelihood of the swarm not exploring certain regions of the search space [98].

Literature therefore recommends that particle positions be uniformly initialised within the boundary constraints of the search space [33]. The boundary constraint of an optimisation problem is the valid value range of each vector element in the position vector of a solution [33].

Some optimisation problems do not have boundary constraints, in which case the positions should be initialised to a reasonable range within the search space [33][37]. The *roaming behaviour* of standard PSO particles allows the particles to move outside their initialisation range in search for optimal solutions [37]. Smaller position initialisation ranges should, therefore, be preferred over larger ranges for unbounded optimisation problems [37].

Velocity initialisation

PSO performance can be degraded significantly if the initial velocity values are too high, because the resulting large step sizes may result in particles skipping over global optima and leaving the search space. A common way to initialise velocities is to initialise them uniformly in a range that is a fraction of the boundary constraints, for instance, a quarter [33]. However, this approach tends to lead to the problem of to large step sizes.

As an alternative, Engelbrecht [32] suggested initialising velocities to zero, because it enhances PSO performance by reducing the time wasted searching in infeasible regions.

3.1.9 Iteration strategies

An iteration for a population-based optimiser is a single step of the optimisation process that iterated through all the entities in the population once. The iteration strategy of a PSO can be classified as either *asynchronous* or *synchronous*. This choice is based on when guides are updated, i.e. when the change in position of a particle becomes visible to the other particles [36].

Asynchronous strategy

Asynchronous PSOs update the personal best position and neighbourhood best positions of the particles immediately after the new position of a particle has been calculated [36]. Any information about the search space that is found by the particle is, thus, immediately conveyed to the particles that still need to move during that iteration [33].

The literature suggests that this type of iteration generally works better with the ring topology than with the star topology, because information can flow quicker through the swarm [33]. Engelbrecht [36] showed that the choice of iteration strategy should be based on the characteristics of the optimisation problem rather than on the type of topology. This observation was also supported by Engelbrecht's findings in [38].

Synchronous strategy

Synchronous PSOs update the particles' personal best positions and neighbourhood best positions only after all the particles' new positions have been calculated [36]. Information about the best positions found by a particle in the current iteration is, thus, only available to the other particles during the next iteration [33].

3.1.10 Stopping conditions

A PSO's optimisation process makes use of a set of *stopping conditions*, to determine when to terminate the search process [33].

Common PSO stopping conditions include the maximum number of objective function evaluations, maximum number of iterations and an acceptable level of error has

been reached [33]. Note that parameters used by these three stopping conditions are problem dependent [33].

3.2 Particle swarm optimisation in dynamic environments

PSOs, such as the gBest and lBest PSOs, are considered to be static PSOs. That is, they fail to work for dynamic optimisation problems. Section 3.2.1 discusses the reasons why static PSOs are not suitable for dynamic optimisation problem. Lastly, Section 3.2.2 presents examples of PSOs that have been developed to work in dynamic environments.

3.2.1 Issues to consider for dynamic environments

Static PSOs are not suitable for dynamic optimisation problems for three reasons [8]:

1. They have no means of detecting environment changes.
2. They have no means of coping with personal and neighbourhood best positions that have become stale, i.e. no longer be the best.
3. They have no means to overcome loss of swarm diversity.

These three issues are discussed in the remainder of this section.

Change detection

PSOs need to know when environment changes occur, so that a PSO can ensure that its particles do not become trapped in unfruitful regions of the search space. If the particles do become trapped, the PSO's performance will be degraded, as discussed in Section 2.2.2.

A way for PSOs to detect changes is to use *sentry particles* [33]. Sentries are initialised to random, fixed positions in the search space at the start of the search, and do not move during the search [9]. Since the positions of sentries do not change during the search, sentries monitor for changes in the environment over time [9][33]. If there is a change in

particle fitness between the beginning and the end of an iteration for a sentry particle, then the environment has changed.

Outdated memory

Memory refers to the personal and neighbourhood best positions of a particle. When an environment change happens, the memory of a particle may no longer be the best position found by the particle and its neighbours [98]. Memory that is no longer valid is called *outdated memory* [6][8]. Any outdated memory needs to be corrected, otherwise, particles will be misinformed and be attracted to sub-optimal regions [8].

A way to handle outdated memory is to ensure that the quality of personal and neighbourhood best positions are recalculated and checked against the current positions, every time a change is detected, or at every iteration [6][8].

Swarm diversity loss

Losing swarm diversity is a natural consequence of the PSO converging, as discussed in Section 3.1.3. However, if the current environment instance changes, the PSO needs to start searching again, because optimality of the current solutions cannot be assumed. To stimulate exploration of the new environment, swarm diversity needs to be restored or increased [8].

A way to overcome this issue to is to re-initialise the PSO whenever a change is detected [94]. Another way is to prevent complete convergence of the swarm by maintaining swarm diversity [6].

3.2.2 Dynamic particle swarm optimisation

Various PSOs have been designed to address the three shortcomings of non-dynamic PSOs. Three dynamic PSOs, namely the reinitialising PSO, the charged particle swarm optimisation (CPSO) algorithm, and the QPSO algorithm, are discussed in the remainder of this section.

Note that the remainder of this thesis focuses only on the QPSO algorithm. The other two dynamic PSOs are provided to demonstrate to the reader that there are various ways

of making a PSO dynamic.

Reinitialising particle swarm optimisation

The reinitialising PSO is a dynamic PSO that uses sentry particles to detect change, and complete swarm reinitialisation to restore swarm diversity and to update outdated memory [33][98].

This approach might fail to detect changes if there are not enough particles in the search space, because the positions of sentries need to experience change for change detection to occur [33][98]. Furthermore, reinitialisation of the swarm prevents previously acquired knowledge about the search space to be reused, thus, potentially wasting computational effort [33][98].

Charged particle swarm optimisation

Blackwell and Bentley [9] proposed the CPSO for dynamic optimisation problems. The CPSO is inspired by the physics of electrostatic charges. Some particles are assigned positive charges and the velocity Equation (3.1) is modified to take the charges into account [9]. Particles with a charge higher than zero repel each other when they are too close to one another.

This prevents the loss of swarm diversity through the prevention of complete convergence [98]. The issue of outdated memory is addressed by allowing the fitness of personal best and neighbourhood best positions to be recalculated during each iteration.

Quantum particle swarm optimisation

Blackwell and Branke [6] proposed a simpler and computationally less expensive alternative to the CPSO, known as the QPSO. QPSO is inspired by the quantum mechanics model of an atom where positions of electrons around the atom are sampled from some probability distribution. The standard PSO model is adapted by introducing *quantum particles* [6].

Quantum particles act as if they are electrons around the nucleus of an atom, where the nucleus is the social guide of the respective particle [6]. The quantum mechanics

model of an atom dictates that an electron's position is somewhere within the sphere around the nucleus of an atom [52]. The exact position of an electron, however, can only be guessed with some degree of certainty as it is constantly changing [6]. Hence, quantum particles' positions are estimated to be somewhere in a hypersphere around the social guide at a specific point in time.

These concepts are implemented by dividing the swarm of the QPSO into two sub-swarms, each with a different set of behaviours [6]. Note that splitting the swarm equally between the two sub-swarms has been found to be very effective [6][98].

The first sub-swarm uses the standard particle update rules, and the second sub-swarm uses the *quantum update rules* [6][98]. Note that the original implementation uses the star topology, though any topology can be used [6][94].

The quantum update rule set, unlike the standard update rule set, only consists of a position update rule defined as follows [52]:

$$x_{ij}(t+1) \sim d(\hat{y}_{ij}(t), r) \tag{3.6}$$

where d refers to a user-defined sampling distribution and r refers to the radius around $\vec{\hat{y}}$. Quantum particles' positions are re-sampled every iteration, using the statistical distribution d, from a hypersphere around the $\vec{\hat{y}}$ particle, with a radius r [6].

Each particle is sampled from the hypersphere as follows:

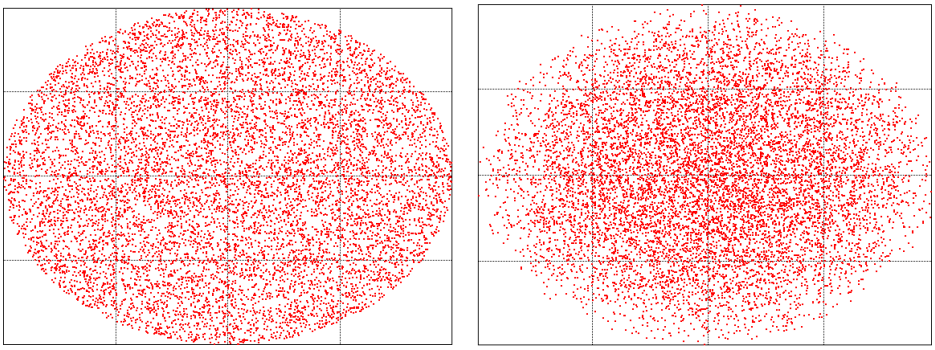
1. Sample a vector whose elements are $\in [-1, 1]$, using the uniform distribution.
2. Convert the resulting vector in step 1 into a unit vector.
3. Sample a value in the range of $[0, r]$, using the user-defined distribution.
4. Multiply the vector found in step 2 by the value found in step 3.
5. The sampled position vector is equal to $\vec{\hat{y}}$ plus the resulting vector of step 4.

The quantum update rule prevents complete convergence of the swarm, thus preventing swarm diversity loss. The combination of standard and quantum update rules allow the quantum particles to continuously explore the search space, while the standard particles exploit the quantum particles' findings [6]. A larger value for r results in a

larger sample area around $\vec{\hat{y}}$, thus enabling quantum particles to explore more of the search space [6]. The inverse of this occurs for smaller values of r.

A commonly used sampling distribution is the uniform distribution [6][52]. However, Harrison *et al.* [52] investigated the effect of various sampling distributions on QPSO performance, and found that the *de facto* uniform distribution performs neither the best nor worst in dynamic environments. Harrison *et al.* [52] suggested the use of the linear decreasing distribution with a small radius.

Figure 3.2 provides a comparison of the uniform and linear decreasing distributions on a 2-dimensional sampling area using 10000 samples.



(a) Uniform sampling distribution (b) Linear decreasing sampling distribution

Figure 3.2: Sampling distribution comparisons for 2-dimensional vector: 10000 samples

Algorithm 2 presents the pseudo code for the QPSO, which can make use of any topology.

Note that QPSO is focussed on by this thesis instead of the other dynamic PSOs variants, because QPSO is computationally less complex than other dynamic PSO. Furthermore, there is a significant body of research on QPSO, including improvements and its successful use to train ANNs.

3.3 Summary

This chapter discussed a class of powerful stochastic population-based optimisation algorithms called PSO. PSOs have various configurations ranging from particle behaviour to neighbourhood topologies, each with their own set of best practices.

The two original PSOs, gBest and lBest, are inefficient for dynamic optimisation problems due to their inability to detect changes, to overcome swarm diversity loss, and to cope with outdated memory. A simple, but highly efficient PSO alternative for dynamic optimisation problems is the QPSO.

The next chapter, discusses ANNs, and also discuses how the QPSO can be applied as a learning algorithm for ANNs.

Algorithm 2 Quantum Particle Swarm Optimisation algorithm

Input: ω, c_1, c_2, n_p, r, distribution d, $f(\vec{x})$, stopping conditions
  for $i = 0$ to n_p **do**
    Initialise $\vec{x}_i$ randomly
    Initialise $\vec{v}_i$ to zero vector
    Set $\vec{y}_i$ as $\vec{x}_i$
    Calculate the quality of particle i using $f(\vec{x})$
  end for
  Assign standard particle update rules randomly to half of the swarm
  Assign quantum update rule to the remaining half of the swarm
  Set t to 0
  while Stopping conditions not true **do**
    for $i = 0$ to n_p **do**
      Determine $\vec{\hat{y}}_i$ by looking for the best personal best position the in neighbourhood of particle i
      if (particle i is a quantum particle) **then**
        Update $\vec{x}_i$ using Equation (3.6)
      else
        Update $\vec{v}_i$ using Equation (3.1)
        Update $\vec{x}_i$ using Equation (3.2)
      end if
      Calculate the quality of particle i using $f(\vec{x})$
    end for
    for $i = 0$ to n_p **do**
      if ($\vec{x}_i$ is better than $\vec{y}_i$) **then**
        $\vec{y}_i = \vec{x}_i$
      end if
    end for
    $t = t + 1$
  end while
  return Best position found so far

Chapter 4

Artificial neural networks

All models are wrong; some models are useful.

George Edward Pelham Box (1919 – 2013)

Over the course of the last couple of decades, artificial neural networks (ANNs) have evolved to become very successful approximators for a broad spectrum of approximation problems [33][39][106]. This chapter discusses ANNs, with focus on ANNs as classifiers.

The remainder of the chapter is organised as follows. Section 4.1 discusses the general mechanics of ANNs. Section 4.2 discusses the concept of ANN architecture and introduces the FFNN architecture, commonly used for classification problems. Section 4.3 explains how ANNs learn a functional mapping, with focus on the supervised learning approach for FFNNs. Section 4.4 discusses the usage and issues of PSO-based learning for FFNNs. Lastly, Section 4.5 summaries the chapter.

4.1 How artificial neural networks work

ANNs are mathematical graph-like models that can approximate mathematical functions [33][126]. ANNs draw their inspiration from the human brain. The human brain is a complex network of cells, called *neurons*, that are connected by pathways, called *synapses* [84]. McCulloch and Pitts [84] first captured these structures and the activity within them in 1943, by designing a mathematical model for constructing and analysing single neurons in the human brain.

Biological neural networks use electro-chemical charges to carry signals within them [84][101]. These charges flow from neuron to neuron via synapses. Charges from incoming synapses accumulate in a neuron, until some threshold is reached [84][101]. Upon reaching the thresholds, the neuron *activates* by discharging the built-up charge via its outgoing synapses [39]. The electro-chemical impulse is split across the outgoing synapses, in proportion to the strengths of each synapse [33].

In the context of ANNs, neurons are seen as information processing units [106]. Synapses, on the other hand, are seen as unidirectional weighted edges that dictate the flow of information between neurons [39]. Note that this thesis refers to the neuron from which a synapse flows as the *source neuron*, and the neuron to which the synapse flows as the *destination neuron*.

Furthermore, neuron activation is seen as the mathematical transformation of the information, represented by incoming synapses, into a single value, known as the *activation value*. An *activation function* and neuron unit type specification dictate activation [33][101][106]. The activation function describes how the activation value is produced [119]. The neuron unit type specifies how the information from incoming synapses is fed into the activation function [106]. These two aspects are known as neuron components.

Hence, neuron components and synapses represent the adjustable variables, i.e. model parameters, of an ANN. Besides neurons and synapses, all ANNs make use of *input neurons* and *output neurons*. Input neurons allow the inputs, i.e. the values of the independent variables, of a pattern to be fed into an ANN. Output neurons provide the outcome of processing the pattern, i.e. the values of the dependent variables [39][101]. The ANN needs to learn the function, i.e. $f(\vec{z}) = \vec{t} + \epsilon$, which maps the independent variables to the dependent variables of an approximation problem with an ϵ amount of error [33][121]. Note that the terms "*training*" and "learning" are used interchangeably in ANN literature [31][33][39][106].

4.2 Architecture

The *architecture* of an ANN describes the overall structure of an ANN [39][69][121]. The structural aspects of an ANN can be grouped into two groups [39][69][121]:

- Firstly, operation and interconnectivity of neurons and synapses; in other words, how the neurons and synapses are organised, and which activation functions and neuron unit types are used.
- Secondly, the graph structure of an ANN; in other words, the total number of neurons (n_n), synapses (n_s) and neuron layers (n_l) that are in the ANN. Neuron layers are sub-groups of neurons in an ANN, e.g. input neurons form the input layer.

Neuron and synapse operation and interconnectivity are strongly influenced by the type of approximation problem, e.g. classification; whereas the actual structure is more influenced by the specific instance of the problem, e.g. the number of independent and dependent variables of a classification problem. Architecture, therefore, has a direct influence on the approximation ability of an ANN [33].

Examples of architectures include [33][71]:

- Recurrent neural networks (RNNs), which are usually used for problems where the function is dependent on previous outputs or inputs. Recursive mathematical functions or text prediction are typical examples of problems addressed by RNNs.
- Convolutional neural networks (CNNs), which are used for problems where the function is buried in deep layers of features. Image and audio recognition are typical examples of problems addressed by CNNs.
- FFNNs, which are commonly used for classification problems.

The remainder of this thesis focuses on FFNNs, because of their ability as classifiers. Section 4.2.1 discusses FFNNs. Section 4.2.2 elaborates on neuron unit types commonly used in FFNNs. Lastly, Section 4.2.3 discusses the activation functions commonly used in FFNNs.

4.2.1 Feed forward neural networks

FFNNs, also known as multi-layer perceptrons (MLPs), were derived from Rosenblatt's perceptron [101][106]. FFNNs group input and output neurons into two separate layers [69][101]. Between these two layers, multiple *hidden layers* are allowed [33]. Each hidden layer consists of variable numbers of *hidden neurons* [39]. Note that a FFNN with more than one hidden layer is classified as a deep neural network (DNN) [71]. DNNs are beyond the scope of this thesis.

All activation functions for hidden and output neurons require a *bias*, thus a single *bias neuron* is placed in the preceding layer [39]. A FFNN bias neuron has a constant value of -1 [115]. The weights of synapses connected to bias neurons are called *threshold values* [33].

Each neuron, excluding bias neurons, is connected to all the neurons in the preceding layer [33]. Information flows from the input neurons in a forward direction towards the output neurons [33][101].

The commonly used 3-layer FFNN consists of an input layer, one hidden layer, and an output layer [33]. Three-layer FFNNs are commonly used for two reasons:

- Firstly, 3-layer FFNNs are intuitively easier to understand and less complex to design than FFNNs with more than one hidden layer [33][39].

- Secondly, Cybenko [22] proved that 3-layer FFNNs, using sigmoid activation functions, can learn any continuous function. Later, Hornik [57] generalised even further by showing that 3-layer FFNNs can learn any continuous function provided that the hidden layer has enough hidden neurons for the function in question, and that the activation functions are non-linear and arbitrarily bounded. Therefore, 3-layer FFNNs are universal approximators [57][126]. Sonoda and Murata [108] further proved that, if activation functions are non-polynomial, they do not need to be bounded for the universal approximation theorem to hold. Note that the number of hidden neurons needed in the case of a 3-layer FFNNs can be exponentially more than that needed by FFNNs to accurate classifications [86]. This is not a concern for this thesis, because the thesis investigates classification problems whose number of hidden neurons have been empirically determined.

From a notational point of view, this thesis uses the notation

$$n_i\text{-}n_h\text{-}n_k$$

to represent the neuron layout of a 3-layer FFNN, where n_i is the number of input neurons, n_h is the number of hidden neurons, and n_k is the number of output neurons. Note that n_i and n_h excludes the bias neurons.

Figure 4.1 presents a structural diagram of a 3-layer FFNN where the circles represent non-bias neurons, the rounded squares represent bias neurons, the lines between neurons represent synapses, and the dashed rectangles represent the neuron layers.

In Figure 4.1, z_i is the activation value of input neuron i that corresponds to the i-th element in input vector $\vec{z}$, h_h is the activation value of the hidden neuron h, and o_k is the activation value of the output neuron k that corresponds to the k-th element in target vector $\vec{t}$. Furthermore, w_{hi} is the weight of the synapse from input neuron i to hidden neuron h; and w_{kh} is the weight of the synapse from hidden neuron h to output neuron k.

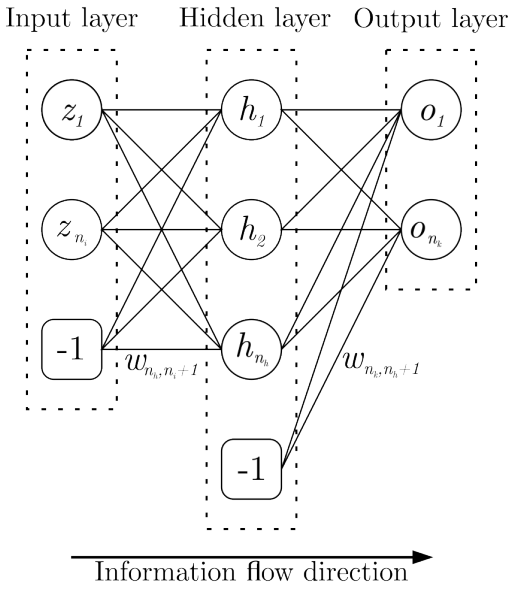


Figure 4.1: 3-layer FFNN architecture

4.2.2 Neuron unit types

Neuron unit types determine the way in which the weights of incoming synapses and the values of source neurons are used to produce the weighted net input signal, net, used by the activation function [33][39].

Two commonly used neuron unit types are product units and summation units [33].

Product unit

Product units compute the net input signal using

$$net_{in} = \prod_{j=1}^{n_{in}} v_{in,j}^{w_{in,j}} \tag{4.1}$$

where n_{in} is the number of incoming synapses for the activating neuron which includes any synapses from bias neurons, $w_{in,j}$ is the weight of the incoming synapse j, and $v_{in,j}$ is the value of the source neuron of the j-th incoming synapse [26].

Summation unit

Summation units compute the net input signal as the weighted sum [33], i.e.

$$net_{in} = \sum_{j=1}^{n_{in}} v_{in,j} w_{in,j} \tag{4.2}$$

Compared to product units, summation units store less information [26][74]. That is, more summation units than product units are required to approximate a function to the same accuracy [33]. Product units, however, tend to increase the number of local optima and the magnitude of gradients [74]. This usually hinders the learning process of an FFNN, especially for learning algorithms that use gradients [74]. Population-based optimisation algorithms, such as PSO, have been more successful than gradient-based learning algorithms at training product unit FFNNs for static classification problems [61][114].

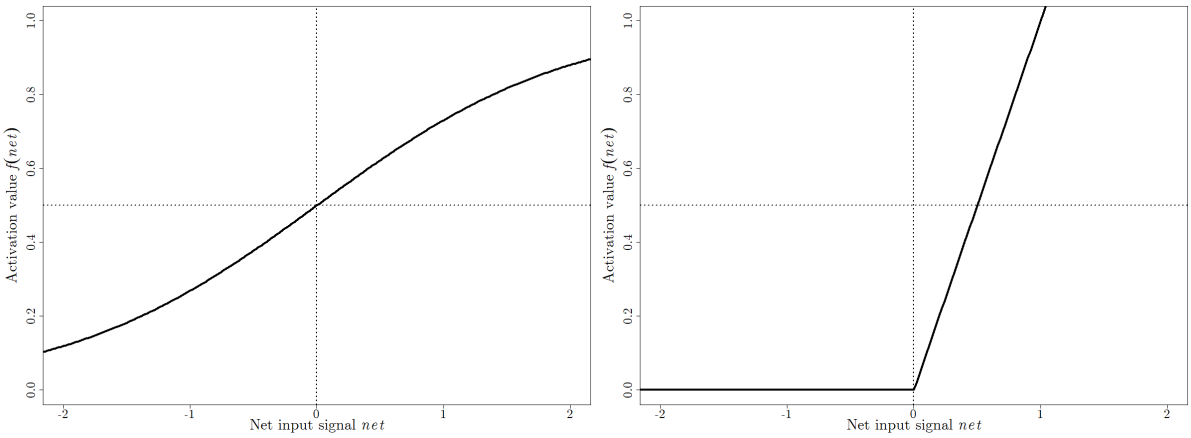
At the time of writing it is unknown how effective product units are in dynamic classification problems. Only summation units will be used in the remainder of this thesis.

4.2.3 Activation functions

Many different activation functions have been proposed in FFNN literature. Activation functions dictate how the activation value of a neuron is calculated from the net input signal [39][115]. Because the curve of a particular activation function might be more suitable for the function curve of a particular approximation problem than another activation function's curve, choosing the correct activation function is essential for good performance [2][39][115].

Two commonly used non-linear activation functions are the *sigmoid* and *rectified linear* functions [39][71][85][98][115].

Figure 4.2 presents a visual comparison of the two activation functions with a zero bias.



(a) Sigmoid (b) Rectified linear

Figure 4.2: Activation functions comparison: sigmoid versus rectified linear

Sigmoid

The sigmoid activation function is defined as

$$f_{Sig}(net) = \frac{1}{1 + e^{-\lambda(net-\theta)}} \tag{4.3}$$

where θ is the bias, and λ controls the steepness of the curve [72][115]. The net input signal is in the range $(-\infty, \infty)$, but the *active range* of the sigmoid function, i.e. the

range where the most visible changes in $f_{Sig}(net)$ exist, is $[-\sqrt{3}, \sqrt{3}]$ when $\lambda = 1$ and $\theta = 0$ [33][115].

The sigmoid function smooths the continuous activation curve in the range $(0, 1)$ [72]. This property has allowed the sigmoid function to be applied successfully in a number of 3-layer FFNN applications [33][39][72][81][115]. Sigmoid is therefore commonly used as the default activation function for 3-layer FFNN [81][98][115].

Rectified linear

The ReLU activation function is a more recent activation function than the sigmoid function, and has become the *de facto* activation function for DNNs, e.g. CNNs [71][81][115]. The reason for the popularity of the ReLU function over bounded activation functions, such as sigmoid, is its low level of computational complexity and that it does not suffer from the vanishing gradient problem [71][115]. The ReLU activation function is defined as

$$f_{ReL}(net) = max\{\lambda(net - \theta), 0\} \tag{4.4}$$

It is easy to see that the ReLU activation function is a modified version of the *linear* activation function, which is defined as $f_L(net) = \lambda(net - \theta)$ [33][115]. The simple modification of applying the max function ensures that the range of the activation value is bounded below by 0, i.e left bounded.

Unlike the sigmoid function, the active range of the ReLU function is $[0, \infty)$, when $\theta = 0$. Furthermore, the gradient of the ReLU is non-continuous, i.e. $\frac{d}{dnet} f_{ReL}(net) = 0$ when $(net - \theta) \leq 0$ and $\frac{d}{dnet} f_{ReL}(net) = \lambda$ when $(net - \theta) > 0$.

4.3 Learning

Finding the best ANN for the problem, i.e. training an ANN, is an optimisation problem, called the ANN *learning problem* [31][49]. The ANN learning problem comprises of two objectives [2][39][79]:

1. find the architecture that minimises the complexity of the ANN, i.e. the number of neurons and synapses, and

2. find the synapse weight values that minimises the error function.

Typically, the two objectives are seen as two separate optimisation problems [23][31]. Because the optimal weights are not just dependent on the problem, but also on the architecture, the two optimisation problems are typically done in sequence. That is, the optimal architecture must first be found before the optimal synapse weights [79]. It is also possible to optimise the architecture and synapse weights simultaneously [95][118].

These two optimisation problems are handled by the *architecture selection* and *weight adjustment*, respectively [33]. Together, these optimisation processes form the learning algorithm for an ANN. A learning algorithm iteration, i.e. one traversal through the entire data set used to train the ANN, is known as an *epoch* [2][33][39]. In their most basic form, learning algorithms consist of the following two steps [31][121]:

1. Select an initial architecture through trial-and-error, from domain knowledge, or by using an architecture selection algorithm.

2. Adjust the weights using an optimisation algorithm.

It is possible for architecture selection to be done dynamically, i.e. during weight adjustment or every time before the weights are adjusted. If a learning algorithm performs architecture selection during weight adjustment, then the ANN learning problem is a multi-objective optimisation problem, because both objectives are optimised at the same time. This thesis focuses on such learning algorithms.

Before applying learning algorithms to ANNs, there are several issues that need to be considered [2][33][94][115][121]:

1. How will the learning algorithm use the information in the dataset to learn an ANN?

2. What error function should be used to determine how well the problem has been approximated?

3. What measurements can be used to evaluate the performance of an ANN?

4. How can the inability to generalise, because of exact curve-fitting to a dataset, i.e. *overfitting*, be prevented?

5. How can the inability to generalise, because of too weak curve-fitting to a dataset, i.e. *underfitting*, be prevented?

6. How will the weights be initialised?

7. Under what conditions should the learning process stop?

If the above issues are not properly addressed, learning will more than likely result in sub-optimal ANNs [2]. Sections 4.3.1 to 4.3.7 discuss these issues, respectively. Section 4.3.8 elaborates on architecture selection. Lastly, Section 4.3.9 elaborates on weight adjustment.

4.3.1 Learning strategy

The approach a learning algorithm takes to train an ANN is known as a *learning strategy* [33]. The learning strategy used depends on the amount of information available in the patterns of a data set, e.g. do the patterns have an input and target vector or just an input vector [39][62]. There are three well-known learning strategies [33][39][69]:

1. *Unsupervised learning*, which is a *learn-by-exploration* strategy designed for approximation problems where patterns have only inputs and decision boundaries that can be potentially used to label the input vectors need to be found. Unsupervised learning strategies are typically applied to clustering problems.

2. *Reinforcement learning*, which is a *learn-by-interaction* strategy designed for approximation problems where patterns have only inputs, but environmental feedback on the output of a model is available. Reinforcement learning strategies are typically applied to game problems.

3. *Supervised learning*, which is a *learn-by-example* strategy designed for approximation problems where patterns have both inputs and targets available. Supervised learning strategies are typically applied to classification problems. Supervised learning calculates the model error of the ANN, by applying some error function to the targets of the patterns and outputs generated by the processing of the patterns. The model error, determined by the error function, is then used by the learning algorithm to optimise the current ANN [106].

The focus of this thesis is on classification problems. Unsupervised and reinforcement learning strategies will, therefore, not be considered further in this thesis.

4.3.2 Model errors

Model errors guide learning algorithms (refer to Section 2.1.2 for a definition of model error). How informative a model error is for weight adjustment and architecture selection depends on [2][113]:

- The error function used to calculate the model error. If the error function is not relevant for the problem, e.g. a maximisation function for a classification problem, then weight adjustment and architecture selection will fail.
- The number of patterns in the data set used to calculate the model error. The less patterns there are, the less informative the model error becomes, because less is known about the function space of the approximation problem. Hence, weight adjustment and architecture selection stand a greater chance of failing.
- The quality of the patterns in the data set used to calculate the model error. The more errors there are in the data set, or less representative the data set is of the approximation problem, the more inaccurate the model error will be. Hence, weight adjustment and architecture selection stand a greater chance of failing.

There are three types of model errors typically used when training ANNs [33][98][113]:

1. The *training error*, E_t, is calculated using the training set, D_t. The training error quantifies how well the ANN has learned the given training patterns, i.e. how well the function represented by the training patterns has been approximated. Hence, E_t is used to determine the weight adjustments.
2. The *generalisation error*, E_g, is calculated using the generalisation set, D_g. The generalisation error quantifies the ability of the ANN to predict the targets associated with the patterns not used during training, and is calculated after the ANN has been completely trained.

3. The *validation error*, E_v, is calculated using the validation set, D_v. The validation error is an estimate of the generalisation error of the ANN during training. It is used by learning algorithms that need an estimate of the generalisation error during training.

D_t, D_g and D_v are mutually exclusive, randomly selected subsets of the data set [33]. For dynamic environments, the three sets are selected at the start of every environment instance. The number of patterns per set is usually a percentage of the entire data set D. The splitting percentages are presented using the notation

$$p_g\text{-}p_v\text{-}p_t$$

where each is the percentage of patterns in D that is used for D_g, D_v and D_t, respectively. This approach for defining model errors is known as the *hold-out error* approach [45].

The sum square error (SSE) and the mean square error (MSE) are commonly used to calculate E_t, E_v, and E_g for a supervised ANN [2][39][119]. SSE is calculated using

$$SSE = \sum_{p=1}^{|D|} \sum_{k=1}^{K} (t_{p,k} - o_{p,k})^2 \tag{4.5}$$

and MSE is calculated as

$$MSE = \frac{SSE}{|D| \times K} \tag{4.6}$$

where D is a data set consisting of $|D|$ patterns. Each pattern $p = 1, ..., |D|$ consists of an input vector and a target vector, i.e. $(\vec{z}_p, \vec{t}_p)$ [2][10][94]. Therefore, $D = \{(\vec{z}_p, \vec{t}_p) | p = 1, ..., |D|\}$, $t_{p,k}$ is the k-th target of pattern p, and $o_{p,k}$ is the output value of the k-th output neuron when $\vec{z}_p$ is processed by the ANN. Note that MSE and SSE are minimised.

4.3.3 Performance measures

Performance measures are used to evaluate and compare trained ANNs. There are two important categories of performance measures to consider, namely *accuracy* and *complexity* [33].

Accuracy

Accuracy performance measures, in the case of supervised learning algorithms, evaluate how close the outputs of the ANN are to their expected values, i.e. the targets [113]. Accuracy is the primary category of ANN performance measures used in literature [2][39][113].

SSE and MSE are regarded as accuracy measures, however, they are not the only accuracy measures. Another accuracy measure for classification problems is the percentage good classifications (PGC), which is defined as

$$PGC = \frac{\sum_{p=1}^{|D|} y_p}{|D|} \tag{4.7}$$

where y_p is defined as

$$y_p = \begin{cases} 0, & \text{if } \epsilon < |\vec{t}_p - \vec{o}_p| \\ 1, & \text{otherwise} \end{cases} \tag{4.8}$$

and ϵ is the error threshold; $\vec{o}_p$ is the output vector of the ANN when processing pattern p [113].

Another accuracy measure can be derived from the PGC by setting $\epsilon = 0$. This measure is called the percentage correct classifications (PCC) [113].

MSE is one of the commonly used accuracy measure [33][85][94][95][113][115]. However, the literature also recommends the use of PCC, since MSE on its own does not represent a fair view of the accuracy performance of an ANN for classification problems due to MSE not evaluating correct pattern classification, but rather error magnitude [33][113]. The difference in the unit of measure between error magnitude and correct classification also makes interpreting MSE values difficult. Note that, unlike MSE and SSE, PCC is maximised.

Confusion matrices are also commonly used to analyse the accuracy performance of a classifier [41]. A confusion matrix is constructed by creating a square matrix whose rows represent the predicated classes and whose columns represent the expected classes [41]. Each element of the matrix is filled in with a count of the number of times the expected class is classified as the predicated class of the element [41][123]. A confusion matrix allows the levels of confusion, i.e. incorrect classifications, per class to be quantified [41].

Another accuracy performance visualisation, using confusion matrices, are reciever operating characteristic (ROC) curves [65]. ROC curves visualise the trade–off between classifying a pattern as a particular class correctly or incorrectly [41]. To create a ROC curve for a particular class, a ROC space for the class must first be created. A ROC space is a two dimensional space that sees a classifier classifying a pattern as the particular class in the following ways [41]:

1. *True positive*, when the pattern is correctly predicted to be a particular class.
2. *False positive*, when the pattern is incorrectly predicted to be a particular class.
3. *True negative*, when the pattern is correctly predicted to not be a particular class.
4. *False negative*, when the pattern is incorrectly predicted to not be a particular class.

Note that the four classifications above can be seen as elements of a confusion matrix of a binary classifier.

Points on a ROC curve are formed by calculating two classification rates from the ROC space [65]. First, the *true positive rate* is calculated as the ratio between the number of correct positive classifications and the actual number of positive patterns [41]. Second, the *false positive rate* is calculated as the ratio between the number of incorrect positive classifications and the actual number of negative patterns [123]. Using a variable error threshold, like PGC does, various pairs of true positive and false positive rates can be determined for a particular class [123].

Interpretation of the two rates is relatively straight forward. When both rates are zero, the classifier is classifying all patterns as negative [41]. On the other hand when both rates are one the classifier is classifying all patterns as positive. The classifier is perfect when the true positive rate is one and the false positive rate is zero and vice versa [41].

Plotting the true positive rate against the false positive rate results in a ROC curve for the classifier [41]. A linear ROC curve means that the classifier is as good as a random classifier [123]. A ROC that curves above the linear line indicates a classifier that is better than a random classifier and vice versa [65].

ROC curves have the added advantage of being immune to data sets that have skewed class distributions, because true positive and false positive rates are ratios based on the performance of a single class [41][65][123]. The construction of ROC curves, however, can be time consuming. Furthermore ROC curves work best for classification problems which have only two classes. Classification problems with more than two classes require a ROC curve for each class.

Complexity

Complexity performance measures evaluate the *structural* and *computational* complexity of an ANN [33].

Structural complexity refers to the size of the ANN [10][31][49][79][95]. Two simple structural complexity measures are the total number of synapses (n_s) and the total number of neurons (n_n).

These two measures give limited insight into the structural complexity of an ANN, because they do not consider the weight values of a synapse. For example, consider a FFNN with summation units. If a weight is zero then according to Equation (4.2) the synapse will not have any effect on the net input signal [31][118]. The synapse can therefore be considered *irrelevant* in determining the output of neurons in the FFNN [31][33][68].

It is possible for a neuron in the FFNN to also be considered irrelevant when its activation value is always zero [31][49]. This is because any synapse from a neuron with an activation value of zero will not have any effect on the net input signal.

The above discussion leads to two other alternative measures, namely the effective number of synapses (n_{s_e}), and the effective number of neurons, n_{n_e}. Together, n_{s_e} and n_{n_e} represent the *effective structural complexity* of an ANN.

The weight values of irrelevant synapses or the activation values of irrelevant neurons are seldom exactly zero. These values can, however, be close enough to zero so that the contributions to the outputs of ANN are negligible [118]. Unfortunately, there are no unanimously agreed upon "irrelevant" threshold for weights and activation values, because such thresholds depend mostly on the activation functions [10][31][33][95][118]. The use of statistical hypothesis tests have been found to be very effective in determining irrelevant synapses and neurons [31]. These hypothesis tests generally test the signifi-

cance of a weight and the sensitivity of the network to that weight [31]. Any attempt made at determining the effective structural complexity should therefore consider such statistical tests.

Computational complexity, on the other hand, refers to two aspects:

- The computational effort required to train the ANN. This aspect is dependent on the training set and the computational complexity of the optimisation algorithm employed by the learning algorithm.
- The computational effort used by the ANN to process a pattern. This aspect is dependent on n_s, n_n, the activation functions, and the neuron unit types used by the ANN.

This thesis focuses on architecture selection and the use of classifiers. Hence, only the measures relevant to measuring the structural complexity and computational effort of using a FFNN are considered in the remainder of this thesis.

4.3.4 Overfitting

Overfitting is generally described in literature through the concepts of *bias*, i.e. how far the outputs of the ANN are from their targets, and *variance*, i.e. how scattered the outputs are for similar unseen inputs [2][23][47]. An ANN overfits when it has a low bias and high variance [2][47].

If the error function is minimised, then the occurrence of a low bias and high variance is indicated by a decrease in the training error while the validation error increases as learning progresses [33]. Hence, an overfitted ANN can accurately approximate training patterns, but fails to generalise test patterns [23][33][47]; the ANN effectively learns the noise in the training data. Overfitting generally occurs when

- the ANN is too complex for the problem at hand,
- the ANN is trained for too long, and
- the training set contains noise.

If E_v is calculated using the error function, then overfitting can be detected by the trigger,

$$E_v > \bar{E}_v + \sigma_{E_v} \tag{4.9}$$

where $\bar{E}_v$ is the moving average of E_v and σ_{E_v} is the standard deviation of the validation errors used to calculate $\bar{E}_v$ [33].

The moving average period for $\bar{E}_v$ is specified in epochs. The larger the moving average period is, the less sensitive the detection method becomes to fluctuations in E_v, because the moving average is including more validation errors. Large moving average periods can result in false negatives when detecting overfitting. On the other hand, the smaller the moving average period is, the more sensitive the method becomes. Too small moving average periods can lead to false positives when detecting overfitting. The moving average period thus controls the overfitting detection sensitivity [33].

Two disadvantages of Equation (4.9) are that the equation does not consider E_t, and that the equation introduces an additional control parameter that is problem dependent [33].

If the training error is not considered, false positives for overfitting can occur. For example, consider the case in which both training and validation errors increase over time. In such an event Equation (4.9) would detect overfitting. It cannot be said, however, that the training set is being overfitted, because the ANN is also failing to learn the training set.

Robël [100] suggested an alternative overfitting detection trigger when using MSE to calculate E_t and E_v. The trigger is defined as,

$$\rho(t) > \varphi_\rho(t) \tag{4.10}$$

where $\rho(t)$ is called the *generalisation factor* at epoch t and $\varphi_\rho(t)$ is the threshold at epoch t. The two terms are defined as follows [100]:

$$\rho(t) = \frac{E_v(t)}{E_t(t)} \tag{4.11}$$

with

$$\varphi_\rho(t) = min\{\varphi_\rho(t-1), \bar{\rho} + \sigma_\rho, 1.0\} \tag{4.12}$$

where $\bar{\rho}$ is the moving average of ρ over a given number of epochs, and σ_ρ is the standard deviation of the moving average [33][100]. The overfitting detection approach therefore results in overfitting being detected when the validation error is more than the training error, or when the two errors are moving significantly away from each other [33][100].

4.3.5 Underfitting

Underfitting is a less commonly referred to phenomenon in ANN literature, and is essentially the inverse of overfitting, i.e. high bias and low variance [2][47]. Underfitting translates to both the training and validation errors eventually stagnating at an unacceptable level.

Underfitting typically occurs when the architecture of an ANN is not complex enough to model the function of an approximation problem [33]. Underfitting, however, also occurs when the architecture is sufficient, but the ANN has not been trained enough. An underfitted ANN, therefore, struggles to specialise, i.e. the ANN cannot approximate both training and test patterns [2][47].

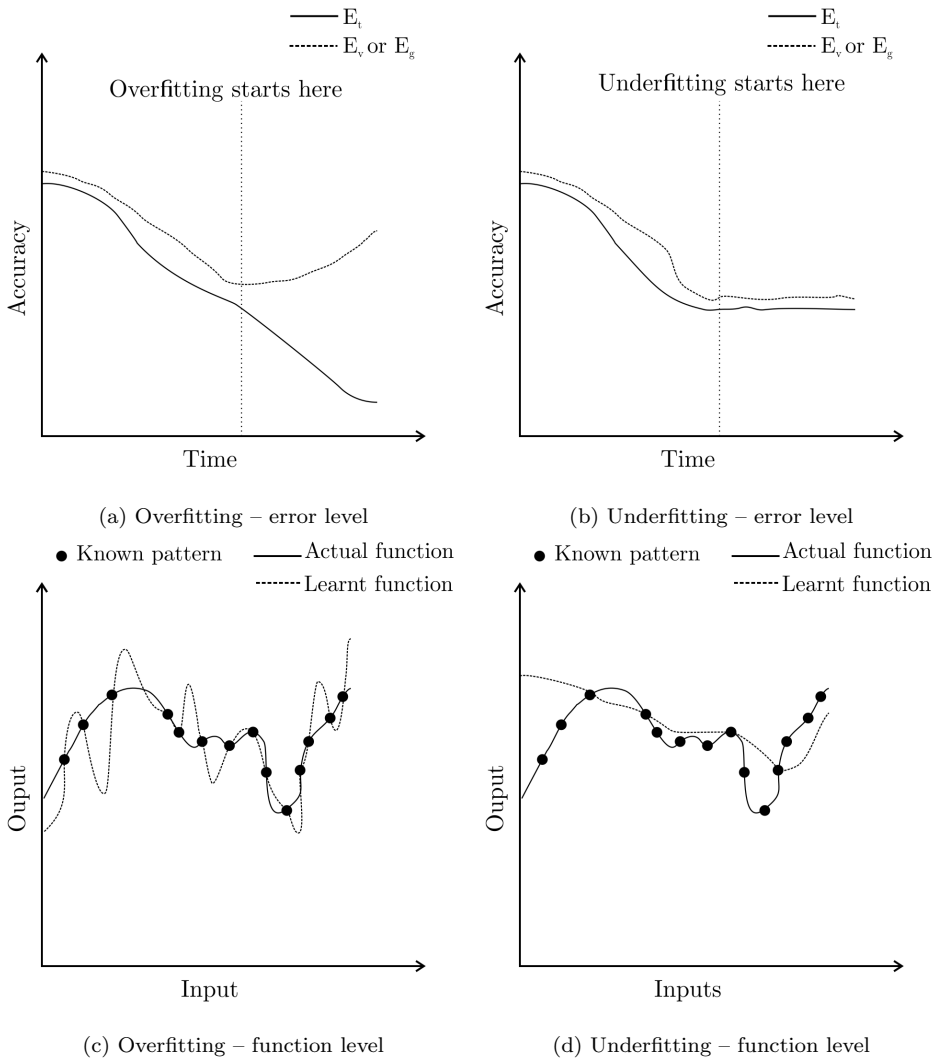
Figure 4.3 illustrates how overfitting contrasts to underfitting by demonstrating their effect on the training and validation errors, and by illustrating the difference between the expected and approximated functions. The approximation problem illustrated in the figure has one input variable and one output variable.

4.3.6 Weight initialisation

Weight initialisation is the process of setting the synapse weights to an initial value [42]. Weight initialisation should be done with care, because the speed and performance of most weight adjustment algorithms are sensitive to the initial weight values [33][69].

It is considered good practice to initialise weights uniformly within a range around zero, because there will be no bias towards certain regions in the search space [72]. Initialising the weights in this manner also prevents initially large weights that could lead to premature convergence [33][42][72]. A sensible initialisation range for weights, is

$$\left[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}\right] \tag{4.13}$$



(a) Overfitting – error level

(b) Underfitting – error level

(c) Overfitting – function level

(d) Underfitting – function level

Figure 4.3: ANN overfitting versus underfitting

where $fanin$ is the number of incoming synapses to the neuron, whose synapses are being initialised [72][120].

Note that He *et al.* [55] proposed *He normal initialisation* for ReLU units, however, the approach is used primarily for DNN with more than one hidden layer. This thesis focuses on 3-layer FFNNs with one hidden layer thus Equation 4.13 is used instead of He normal initialisation.

4.3.7 Stopping conditions

Learning algorithms make use of stopping conditions to determine when ANN training should be terminated [39]. Stopping conditions determine how long an ANN trains on a given data set, and thereby influences what the ANN can learn [98]. As already indicated, if training is terminated too early, the ANN may underfit, while terminating training too late may lead to overfitting.

Stopping conditions, such as maximum number of epochs, overfitting detection and acceptable E_v level, are commonly used by ANN learning algorithms [33]. For the purpose of this thesis, it is sufficient for the reader to know that the maximum number of epochs stopping condition limits the number of training epochs n_e to the value n_e^* [2][33].

Further discussion on stopping conditions are considered to be outside the scope of this thesis, because this thesis requires ANNs to train on streams of data. Therefore, in practice there is no stopping condition, and the ANN will continue training as data becomes available in the stream.

4.3.8 Architecture selection

The objective of architecture selection is to prevent overfitting and underfitting by ensuring that the complexity of the ANN architecture is appropriate for the approximation problem [79]. Architecture selection algorithms are broadly characterised into three groups [33]:

1. *Construction* algorithms, which select small architectures initially and then grow these architectures until overfitting is observed [79].

2. *Pruning* algorithms, which initially select *oversized* architectures, and then shrink these architectures until underfitting is observed [31].

3. *Regularisation* algorithms, which do not explicitly change the architecture, but rather neutralise the contribution of unnecessary weights by driving their weight values to zero [10][118].

This thesis focuses on regularisation for 3-layer FFNNs. Therefore, construction and pruning approaches fall outside the scope of the thesis.

Regularisation augments the error function by adding a regularisation (or penalty) term, E_r, to penalise structural complexity as follows [118]:

$$E'_t = E_t + \lambda_r E_r \tag{4.14}$$

where E'_t is the regularised error function, and λ_r is the regularisation coefficient.

The regularisation coefficient weighs the influence of the regularisation term on the overall error. The regularisation terms considered by this thesis need to be minimised thus the larger λ_r is, the more complexity is penalised resulting in more weight values to be driven towards zero at the cost of increasing the model error [50][68]. Too large λ_r values may result in underfitting, because important weights might also be driven to zero [95]. On the other hand, too small λ_r values may result in overfitting, because E'_t will be more concerned with fitting the training set than reducing model complexity, and therefore fit noise [118]. The exact value for λ_r is problem dependent and has to be carefully tuned in order to achieve a balance between model error and model complexity [33].

Regularisation is attractive because it is a simple on-line approach to architecture optimisation, and there is no need to decide on a threshold value to determine when a weight/neuron should be removed [50][118]. Regularisation, however, does introduce an additional problem dependent parameter, i.e. λ_r.

Two well-known regularisation terms are weight decay (WD) and weight elimination (WE), discussed below [33][95].

Weight decay

WD is defined as

$$E_r = \frac{1}{2}\sum_{j=1}^{n_s} w_j^2 \tag{4.15}$$

In essence, WD counts the number of synapses based on their weight value [95]. ANNs with larger weight values will be penalised more than ANNs with smaller weight values [95].

Because WD penalises the total amount of weight that can be assigned to the synapses, the training algorithm is forced to allocate weights to only the necessary

synapses. Therefore, the training algorithm will drive the weights of irrelevant synapses to zero [50].

A drawback of WD is that it does not associate any notion of relevance to the values of weights. WD, therefore, penalises a weight value with the same aggressiveness regardless if the value is relevant or not [10][68].

Weight elimination

WE was proposed by Weigend *et al.* [118] to overcome the issue of equal aggressive penalisation of relevant and irrelevant weight values seen in WD, and is defined as

$$E_r = \sum_{j=1}^{n_s} \frac{\frac{w_j^2}{w_0^2}}{1 + \frac{w_j^2}{w_0^2}} \tag{4.16}$$

where w_0 is a non-zero threshold, which controls the level at which weight values become irrelevant. The larger w_0 is, the less large weights are penalised, because they are considered more relevant, i.e. needed to model the problem better, and vice versa [118]. The main drawback of WE is that it introduces another control parameter that needs to be tuned [33].

Bosman *et al.* [10] found that WE both smooths the gradients of the minima and introduces additional minima into the search space. The weight value threshold parameter, w_0, controls the sharpness of the introduced minima, i.e. the magnitude of their gradients. The value of w_0, however, is problem dependent [10][94].

4.3.9 Weight adjustment

Optimisation algorithms adjust the weights of an ANN such that the given error function is optimised [47][121]. Weight adjustment is a static optimisation problem if the training set and architecture does not undergo changes. On the other hand, when either the training set or architecture changes during training, weight adjustment becomes a dynamic optimisation problem [49][79].

The original perceptron model adjusted weights by using simple supervised methods, e.g. trial-and-error [101]. These methods were later replaced by the BP algorithm

which adjusted the weights using gradient descent (GD) [119]. Various algorithms for adjusting the weights of FFNNs have since come into existence, e.g. LeapFrog, quickprop, scaled conjugate gradient, and PSO [5][39][40][98][107]. Regardless of these new algorithms, BP is still used as a baseline for learning algorithm comparisons in FFNNs literature [85][98]. This thesis investigates learning algorithms that train FFNNs on dynamic classification problems, therefore, BP will be used to benchmark the investigated optimisation algorithms against. The remainder of this section, therefore, provides the reader with sufficient background on BP.

BP is a static, local-search, gradient-based optimisation algorithm proposed by Werbos [119] in 1974. BP was originally developed as a supervised algorithm for adjusting the weights of a 3-layer FFNN, but since has been improved and adapted for other architectures [33][106].

BP has three modes of learning, namely *batch learning*, *mini-batch learning* and *stochastic learning* [39]. Batch learning accumulates all the weight changes required by the training patterns into a single weight update. Mini-batch learning is similar to batch learning, however, it differs in that it breaks the training set up into small batches such that the number of patterns in a batch is $1 < |D| < |D_t|$ [33][122]. On the other hand, stochastic learning changes the weights after each pattern presentation [33]. With regards to weight adjustment, this thesis focuses on data streams, therefore, only the stochastic BP is considered for the remainder of this thesis.

Stochastic BP consists of two phases, i.e. *feed forward* and *back propagation* [119]:

1. The feed forward phase takes each pattern in the training set, and calculates the output signals for all hidden and output neurons in the FFNN [33].

2. The back propagation phase starts by calculating the error between the targets and actual outputs for the pattern [119]. Next, the error is used to calculate an *error signal* for each output neuron. This error signal is propagated backwards to the hidden layer, and is used to adjust the weights between the hidden and output layers. The error signal for each hidden neuron is then calculated, and propagated backwards to the input layers so that the weights between the input and hidden layers can be adjusted [69][119].

Because of the use of gradient calculations, the exact implementation of BP is dependent on the error function, activation functions, and neuron unit types [72]. Furthermore, BP requires that the weights are not initialised to the same value, otherwise the optimisation will fail due to all weights making equal contribution to the error, resulting in all weights being changed by the same amount [33][72].

BP is subject to local optimum trapping, because it conducts a local-search and is sensitive to initial conditions [72][94]. Stochastic BP can further suffer from haphazard changes in the search trajectory, due to constant changes in the sign of the weight or the overshooting of a local minimum [39][72]. Various extensions have been devised to overcome these shortcomings, such as the momentum term (α) and the learning rate (η) control parameters [10][33][39]. These two control parameters are implemented by augmenting the general weight update equation as follows

$$w_j(t+1) = w_j(t) - \eta \Delta w_j(t) + \alpha \Delta w_j(t-1) \tag{4.17}$$

where w_j is the j-th weight and $\Delta w_j(t)$ is the change that weight w_j experiences at epoch t. The two control parameters work as follows:

- The momentum parameter controls the contribution of the previous weight change to the current weight change. Thus, it makes changes in the sign of the change more difficult [10]. A larger α smooths the search trajectory. If too large, then the search can miss fruitful regions of the search space due the search trajectory taking a long time to change [39]. If sufficient momentum is used, it can help the search to skip over regions that can cause local optima trapping [39][72].

- The learning rate parameter controls the change actually experienced by the particular weight [10]. That is, the parameter controls the rate at which the FFNN learns a pattern. The larger η is, the larger the change [10]. If η is too small, the search will follow the gradient path almost exactly and take a long time to converge to the minimum, or even worse, become stuck in sub-optimal regions "en route" to the minimum [39]. On the other hand, too large η values will cause the search to converge quickly. The quick convergence can cause the search to overshoot a good minimum. The learning rate, therefore, ultimately controls the exploration-

exploitation trade-off for BP, where a smaller η allows more exploration and vice versa [33][69].

Note that the values of both control parameters are problem dependent.

Algorithm 3 presents generalised pseudo code for the stochastic BP algorithm for a supervised 3-layer FFNN [33][115].

Algorithm 3 Supervised 3-layer FFNN stochastic BP algorithm

Input: η, α, initialised FFNN, D, p_g, p_v, p_t, stopping conditions
  Set t to 0
  Set D_g as $p_g \times |D|$ randomly chosen patterns from D
  Set D_v to $p_v \times |D|$ randomly chosen patterns from the remaining patterns in D
  Set D_t to the remaining patterns in D
  while Stopping condition not true **do**
    Shuffle D_t
    for $p = 0$ to $|D_t|$ **do**
      Feed forward pattern p into the FFNN
      Calculate neuron activations using the activation function
      Calculate the error of the pattern using the error function
      Calculate error signals
      Adjust FFNN's output to hidden layer weights using Equation (4.17)
      Adjust FFNN's hidden to input layer weights using Equation (4.17)
    end for
    $t = t + 1$
  end while
  return Trained FFNN

4.4 Training artificial neural networks using particle swarm optimisation

PSOs have been successfully employed to select architectures for ANNs and to adjust their weights [58][61][85][98][114][115][127]. While PSOs have been successful in training ANNs, PSO does suffer from the problem of *saturation* [95][96][97].

Section 4.4.1 discusses several training algorithms that use PSO to select architectures and to adjust weights. Section 4.4.2 elaborates on the issue of saturation. Lastly, Section 4.4.3 discusses an approach to quantify saturation.

4.4.1 Particle swarm optimisation training algorithms

Algorithms that use PSO to adjust ANN weights start off by initialising n_p instances of an ANN [85]. Each particle's position vector represents the weights and biases of an ANN [33][85]. These position vectors are commonly referred to as *weight vectors*. Weight vectors are usually initialised using the approaches discussed in Section 4.3.6. The objective function is the ANN's error function [98]. MSE is typically used by PSO-based weight adjustment algorithms [98][115].

Five advantages that PSO-based weight adjustment algorithms have over BP are:

1. their resilience to local optimum trapping [98],
2. their ability to handle concept drift, if a dynamic PSO is used, e.g. QPSO [52],
3. they do not use computationally expensive derivative calculations [85],
4. they are computationally simple and easy to implement [61], and
5. there is empirically and theoretically derived guidance on control parameter value assignment to guarantee that an equilibrium state will be reached [16].

Rakitianskaia and Engelbrecht [98] investigated the use of QPSO and CPSO to train 3-layer FFNNs for dynamic classification problems. It was found that QPSO and CPSO performed similarly to each other, and tended to outperform BP in terms of accuracy [98]. Rakitianskaia and Engelbrecht [98], however, did not consider optimising the architecture after an environment change.

Zang *et al.* [127] proposed a PSO-based learning algorithm for 3-layer FFNNs that incorporates architecture selection. The gBest PSO tracks and optimises the number of hidden neurons and connection density of potential architectures. The quality of each architecture is the quality of the best particle found by applying the gBest PSO to the weights of the particular architecture. All the weights for the architecture are adjusted if hidden neurons are removed. If hidden neurons are added, only the new weights are adjusted [127]. However, Zang *et al.* [127] did not consider that dynamic architecture selection results in a dynamic optimisation problem itself, where the search

space dimensionality changes over time. Neither did they consider dynamic classification problems.

A key issue that is neglected in the above works pertaining to the use of PSO to train ANN is *saturation* [97].

4.4.2 Saturation

A neuron is *saturated*, when its activation value is always near the asymptotic ends of a bounded activation function [72][97]. For example, if the sigmoid activation function used, the asymptotic ends are either

$$\lim_{net \to \infty} f_{Sig}(net) = 1 \tag{4.18}$$

or

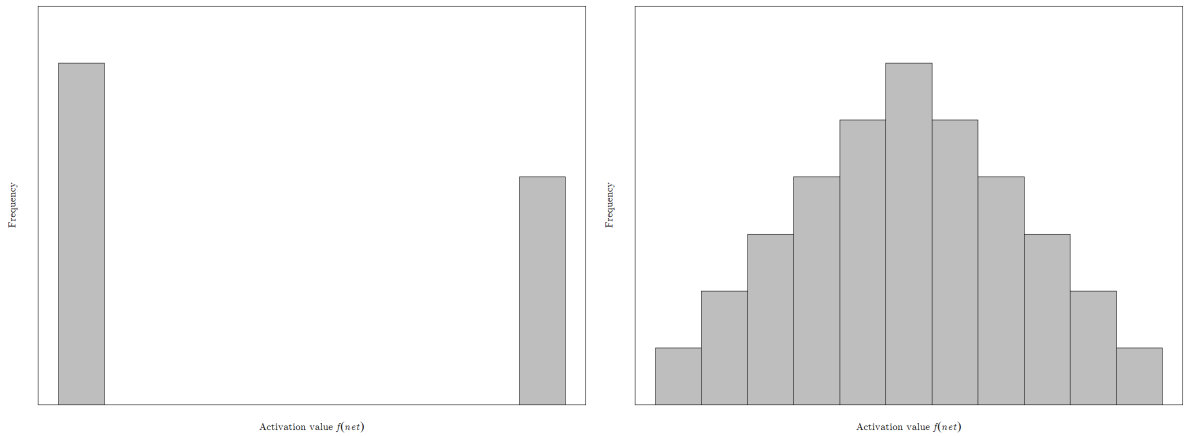
$$\lim_{net \to -\infty} f_{Sig}(net) = 0 \tag{4.19}$$

Saturation hinders the ability of an ANN to approximate problems, because it reduces the set of possible activation values of a neuron to a binary set of activation values [72][96]. The set of possible activation values of a neuron is called its *information capacity* [96]. Figure 4.4 illustrates the reduction in information capacity of a neuron by comparing the activation value distributions of a saturated and unsaturated neuron. Note that saturation limits the range of activation values produced by the activation function of a neuron.

When PSO is used to adjust the weights, saturation can have a serious negative effect on the performance and also lead to overfitting [96][115]. If the ANN begins to saturate, then the error function constantly produces similar information about the search space. Velocities thus either stagnate or explode because the memory of the PSOs never changes. Both cases usually prevent the search from converging to an optimal solution [95][96][97].

Rakitianskaia and Engelbrecht [96] showed that controlling saturation in PSOs is a difficult task because of the mechanics of PSOs. One method to reduce premature saturation is to initialise the weights around zero, because large weights tend to lead to premature saturation [96].

Saturation can potentially be reduced through the use of ReLU activation functions, because they limit saturation to one side [115]. Furthermore, it was shown that ReLU



(a) Saturated neuron $f(net)$ distribution

(b) Unsaturated neuron $f(net)$ distribution

Figure 4.4: Neuron saturation: Saturated versus unsaturated neuron activation value distributions example

activation functions are statistically equivalent in performance to the commonly used sigmoid activation function, when using PSO to adjust the weights [115].

Furthermore, Rakitianskaia and Engelbrecht [99] theorised that regularisation may potentially reduce unwanted saturation, due to regularisation reducing the weight magnitudes. Later, Rakitianskaia and Engelbrecht [95] showed that regularisation techniques, such as WD, do indeed help to reduce saturation in 3-layer FFNNs. The study in [95] was, however, limited to static classification problems.

4.4.3 Measuring saturation

Rakitianskaia and Engelbrecht [96] originally proposed to measure saturation as the average absolute net input signal of hidden neurons in a 3-layer FFNN. Later, Rakitianskaia and Engelbrecht [97] proposed

$$\varphi_{b_w} = \frac{\sum_{b=1}^{B} |\bar{g}'_b| f_b}{\sum_{b=1}^{B} f_b} \tag{4.20}$$

as an alternative saturation measure, where B is the number of activation value bins created by using the binning width b_w, f_b is the frequency, i.e. number of hits, for bin b, and $\bar{g}'_b$ is the average activation value, scaled to the range $[-1, 1]$.

The saturation level measured by φ_{b_w} is in the range $[0, 1]$. If φ_{b_w} is zero, then there is no saturation. On the other hand, if the value is one, then the neurons are completely saturated [97]. A value less than 0.5 indicates a normal distribution of activation values, while a value of 0.5 indicates an uniform distribution of activation values [97].

Rakitianskaia and Engelbrecht [97] argued that the alternative measure in Equation (4.20) should be used, because it takes into account that saturation is bounded in nature and affects the activation values, whereas Rakitianskaia and Engelbrecht's [96] measure was only bounded below and did not consider the activation values but net input signals.

4.5 Summary

This chapter reviewed ANNs as a powerful class of approximators. The concept of architecture was discussed, and the 3-layer FFNN was suggested for classification problems. Furthermore, learning a classification problem using an ANN was shown to be a multi-objective optimisation problem that consists of architecture selection, i.e. optimising the architecture to find the simplest representation for the problem, and weight adjustment, i.e. optimising the weights to accurately fit the problem.

Regularisation approaches, such as WD and WE, were discussed as simple yet effective approaches to FFNN architecture selection. The BP algorithm was presented as the benchmark approach to weight adjustment for FFNNs. Two key problems with BP, however, is its lack of mechanisms to handle environment changes and susceptibility to local optimum trapping.

Dynamic PSO-based training was presented as an alternative to BP, because of the ability of a dynamic PSO to address the issues of concept drift and local optimum trapping. A prevalent issue with PSO-based training, however, is saturation. The use of ReLU activation functions and regularisation was discussed as a means to alleviate saturation in PSO-based training.

The next chapter introduces streamed data classification problems (SDCPs), and discusses how ANNs have been used as classifiers for SDCPs.

Chapter 5

Stream data classification problems: A real-world concern

I know that I know nothing.

Socratic paradox

This chapter expands on Section 2.1.2 by introducing a class of classification problems called streamed data classification problems (SDCPs). SDCPs are the core problems dealt with by this thesis. Note that this thesis refers to classifiers for SDCPs as *streamed data classifiers*.

The remainder of the chapter is organised as follows: Section 5.1 presents a brief overview of SDCPs to demonstrate why classifiers for SDCPs must be found. Section 5.2 elaborates on the issues that streamed data classifiers must address when working with SDCPs. Section 5.3 provides a review of the literature on streamed data classifiers. Lastly, Section 5.4 summaries the chapter.

5.1 Background

Any set of quantitative or qualitative values originating from a subject is called *data* [103][117]. Data can be described using five different dimensions, i.e. *volume*, *variety*, *velocity*, *veracity* and *value* [103]. Volume describes the amount of data. Variety describes the structural differences between data. Velocity describes the speed at which

data becomes available. Veracity describes the accuracy and authenticity of the data. Value describes the usefulness of the information contained in the data [103].

Most computational devices generate data over the course of time [103][117]. Such devices are therefore said to generate *data streams* [1][102][103]. The rise of the internet and other forms of device connectivity has resulted in data streams being shared between devices, across organisations and individuals [1]. Data streams can be synthesised into information about the subjects, e.g. users or processes, generating the data streams [1][75][102][103].

Information that can be used to provide insight or to optimise processes, such as the information in data streams, is considered a currency in today's information society [70][117]. Data streams are thus becoming more and more valuable for various organisations [103].

The increase in both the computational power and uses of computational devices have resulted in data streams that contain large amounts of data that are generated at high speeds, and that vary in structure, veracity, and value [21][60][102][103]. Data that exhibits a large magnitude in at least volume, variety and velocity are termed *big data* [103]. Aside from the big data nature of data streams, data streams are based on real-world subjects, e.g. users and processes. Data streams are therefore subject to the real-world complexities of randomness, error, change, time, and space [75]. The task of extracting useful information from data streams is therefore a non-trivial real-world task, referred to as a streamed data problem (SDP) [60][75][102].

CI categorises SDPs as dynamic approximation problems [1][75]. However, not all dynamic approximation problems are SDPs, because potential approximators need to cater for the big data nature and real-world complexities of sequentially-accessible streams of data in addition to just learning in the presence of concept drift [1]. A SDP requiring classification is known as a SDCP. The remainder of this thesis focuses on SDCPs. The reader is referred to the work by Aggarwal [1] for more information on SDPs.

To illustrate the difference between SDCPs and a regular dynamic classification problem, consider the problem of modelling the price movement of a financial market. This problem requires that the price movement is classified in terms of trends, e.g. upward, downward, or flat. Like any regular dynamic classification problem, the decision bound-

aries of the given problem are subject to change. Theses decision boundaries changes are due to the underlying market conditions, market structure and participants changing over time. Unlike regular dynamic classification problems, the data set of the given problem, i.e. financial market pricing data, is considered big data in nature [103]. The given problem, therefore, also requires the classifier to process and learn from a continuous sequential stream of real-world data, in real-time [76].

5.2 Streamed data classifier requirements

SDCPs impose eleven additional requirements on classifiers when compared with those requirements imposed by static classification problems [1][24][44][60][67][70][75]. These requirements are derived from the big data nature and real-world complexities of data streams [1]. The eleven requirements are as follows:

1. **Bounded memory**: A computer does not have unlimited memory. A streamed data classifier must therefore not exceed the amount of available memory, otherwise the classifier will fail [1]. Alternatively, the classifier can employ a swapping or sampling mechanism that can reduce the amount of data held in memory.

2. **Unbounded dataset**: Real-world data streams are infinite, thus, a streamed data classifier can not store all the patterns. The classifier must only use the patterns necessary to approximate the classification function [1][60].

3. **Concept drift**: Real-world data streams can cause classifiers to experience concept drift (refer to Section 2.2.2). The unbounded data set requirement makes it impossible to know whether or not concept drift occurs at some point in the data stream. A streamed data classifier must therefore be able to handle concept drift at all times [1][67].

4. **Random dynamics**: The temporal and spatial severity levels experienced by real-world data streams are subject to randomness in the problem domain. Not only can data change in SDCPs, but the temporal and spatial severities can also change over time [1][88]. A streamed data classifier must thus be effective in quasi-static, abrupt, progressive, and chaotic environments.

5. **Online learning**: Training the classifier only once is not sufficient for solving SDCPs, because there is a chance that concept drift can occur later on in the data stream. A streamed data classifier must therefore be *online*, i.e. continuously learning [21][27][67].

6. **High speed data streams**: Real-world data streams tend to flow at high speeds, due to the number of users and uses of computational devices [60]. A streamed data classifier must thus be able to process patterns fast enough so that the classifier can be used before the classifier becomes *stale*, i.e. outdated [1][92][98].

7. **One-pass**: Literature suggests that a streamed data classifier has one shot at learning a pattern if the classifier is to stand a chance at adhering to the high speed requirement [1][24][60][67]. Because the classifier has only one attempt to learn each pattern, the classifier must be an efficient learner, i.e be able to extract as much useful information from a pattern without overfitting to the pattern, in order to make accurate classifications [21][60].

8. **Limited number of tunable control parameters**: Optimisation of the control parameters of a learning algorithm is known as control parameter tuning. Control parameter tuning is generally done when a classifier is *offline*, i.e. not learning [53][115]. The online learning requirement, however, allows a streamed data classifier to be offline only at the start of the learning process. Control parameter tuning can thus only be performed using a portion of the data stream. Self-adaptive approaches that tune control parameters during training do exist, but add additional complexity [1][53]. Regardless of the approach used to tune the control parameters, the process can be simplified by reducing the number of control parameters, and reducing the size of the value ranges searched. Learning algorithms should therefore have as few as possible control parameters to optimise, and should allow for searching for control parameter values which provide good performance over a majority of problems [67][102].

9. **Maintain low model complexity**: *Model complexity* refers to the number of model parameters in a classifier (refer to Section 2.1.2). Every model parameter

adds time and space complexity to both the learning and execution of the classifier [33][67]. If the time complexity is too high, then high speed requirements may be violated. If the space complexity is too high, then bounded memory requirements may be violated. A streamed data classifier should thus have as few model parameters as possible [1]. Unfortunately, the minimum model complexity required is dependent on the decision boundaries [94]. Thus, the minimum model complexity can change as decision boundaries appear, disappear, shift, and rotate [44]. A streamed data classifier must therefore also be able to adapt its model complexity to environment changes.

10. **Robustness**: A real-world data stream most likely contains erroneous patterns which may degrade the accuracy of a classifier [13][102]. The literature refers to these erroneous patterns as *noise* [1][13][94][102]. Note that the literature also uses the term noise to describe an excess of valid but similar patterns [33], for the purpose of this thesis noise will mean any erroneous patterns. Noise generally occurs due to machine faults or user error, e.g. precision errors or inaccurate data capture [13][33]. A streamed data classifier needs to be robust enough not to be affected by noise [1].

11. **Fault tolerance**: The implementation of a streamed data classifier needs to be able to handle faults resulting from hardware failures upon which the classifier is implemented [21]. This requirement, however, only becomes more prevalent when classifiers are implemented using specialised hardware designed just for the classifier, e.g. field-programmable gate array circuits, or when the implementation is scaled across various hardware resources using virtualisation [1][21].

Naturally, the more of the above issues a classifier can address, the greater the chance of the classifier to succeed when applied to SDCPs. These requirements should be considered during the design of any streamed data classifier, and the evaluation of potential classifiers for SDCP.

5.3 Literature review on streamed data classifiers and related works

Research on streamed data classifiers has gained popularity over the last two decades [1][102][105]. The majority of early research mostly focused on online learning and concept drift [1][13][24][60][76][98][111][112][116]. Overtime, research has begun to include the other streamed data classifier requirements discussed in Section 5.2, for example [1][21][27][67][70][80][88][92][102][109].

The purpose of the literature review presented in this section is two fold:

- Firstly, the review sets out to identify what SDCP literature has investigated so far.
- Secondly, the review sets out to motivate the work of this thesis. This is done by showing that there is no significant research into the use of regularised 3-layer FFNNs, trained by dynamic PSO-based algorithms, as streamed data classifiers.

Section 5.3.1 reviews online learning approaches. Sections 5.3.2, 5.3.3 and 5.3.4 respectively review decision trees, ensembles and ANN as streamed data classifiers. Section 5.3.5 presents the conclusions drawn from the review, and whether the two goals were achieved.

5.3.1 Online learning approaches

Various approaches that allow a classifier to continuously learn, i.e. online learning, have been proposed [54][80][102]. Online learning literature broadly classifies online approaches into two categories [76][80][111]:

- *Chuck-by-chunk*, which accumulates the next n patterns in the data stream into a chunk, and then trains the classifier using the chunk. Once a chuck has been processed sufficiently by the classifier the next chunk is processed.
- *Pattern-by-pattern*, which processes each pattern in the data stream individually.

This section focuses on the latter category, because learning pattern-by-pattern adheres to the one-pass requirement of the SDCP. Some well-known pattern-by-pattern approaches are listed below:

- *Stochastic gradient descent* is a commonly used pattern-by-pattern approach [33][80]. Each pattern is learnt by adjusting the model parameters of the classifier according to their instantaneous error gradients [80].

- *Online convex optimisation* is an approach that converts the optimisation process of the classifier into an iterative game in which the classifier is an online player [54]. At each iteration of the game the classifier makes a decision about an input, i.e. classifies the input [54]. The goal is for the classifier to try and minimize the regret that the classifier experienced by making a decision [54]. Regret is calculated as the difference between the loss of the decision and the loss of best possible decision known at the time [54]. The loss function is based on a convex set of the all the possible classifications, i.e. a set in which all elements are within a continuous, bounded Euclidean space [54]. Note that stochastic gradient descent is a variant of online convex optimisation [54].

- The *Winnow algorithm* uses a binary classifier to classify each pattern that is presented. The Winnow algorithm uses reinforcement learning, and takes a user-defined constant as control parameter to guide demotion and promotion step sizes [77]. If the classification is incorrect, then the model parameters of the classifier is adjusted by the Winnow algorithm, as follows [77]:

  - If the classification was suppose to be zero, then weights linked to each input that was equal to one are set to zero.
  - Otherwise, the weights linked to each input that was equal to one are multiplied by the user-defined constant.

There exist various other approaches to online learning, however, they are beyond the scope of this review. The reader is directed to the comprehensive survey by Losing *et al.* [80] for more information on online learning approaches for SDCPs.

5.3.2 Decision trees

Decision trees are approximators that use a tree data structure to approximate functions [39][60]. The non-terminal nodes represent evaluation criteria, i.e. a set of logical condition statements [24]. On the other hand, terminal nodes represent final outcomes. Decision trees, therefore, provide a series of logical steps to determine approximations [1][24][39][60]. There are three types of decision trees [4][39][43][60][66]:

- *Regression trees*, which are used to approximate mathematical functions. The terminal nodes of regression trees represent real number values.
- *Model trees*, which are used to approximate piecewise mathematical functions. The terminal nodes of model trees represent linear functions.
- *Classification trees*, which are used as classifiers. The terminal nodes of classification trees represent classifications.

Regression and model trees are considered to be outside the scope of this thesis, because this thesis deals with classification problems. Note that the literature generally refers to classification trees as decision trees, in accordance with this convention any further references to decision trees will mean classification trees [1][24][39][60].

Domingos and Hulten [24] proposed a constant memory and time per pattern, i.e. one-pass, decision tree, called the very fast decision tree (VFDT), for learning high speed data streams. The VFDT achieves the one-pass requirement by making use of Hoeffding decision trees. Unfortunately, the VFDT does not cater for concept drift.

Later, Hulten *et al.* [60] modified VFDT to handle concept drift, and called the new decision tree algorithm concept-adapting very fast decision tree (CVFDT). CVFDT is able to handle concept drift by growing an alternative decision in the background, when the CVFDT detects incorrect classifications in the current decision tree [60][66]. The alternative tree replaces the current decision tree once it has become more accurate than the current decision tree. Hulten *et al.* [60] showed that the CVFDT is more effective at handling concept drift compared to traditional decision tree algorithms.

Decision trees are outside the scope of this thesis. The reader is, however, referred to the comprehensive survey of decision trees by Kotsiantis [66].

5.3.3 Ensembles

An ensemble is a group of high variance, low bias, i.e. overfitted, classifiers working together to reduce the high variance in the outcomes of the ensemble members [33][109][116]. There are various ways in which the outcomes of an ensemble is determined. Such as aggregating the outcomes, using outcomes of the best generalising classifiers, voting on the outcomes, or a linear combination of outcomes [33][67][78][94][109][116].

Ensemble learning techniques, such as *bagging* and *boosting*, have been developed [33][67]. A bagging approach trains each ensemble member with a randomly sampled subset of the training set [12]. Boosting, on the other hand, orders the ensemble members sequentially and then trains them in that order instead of in parallel. The first ensemble member is trained on all the training patterns. The next ensemble member in the sequence is trained on the residual, i.e. the patterns that the previous ensemble members in the sequence could not learn [125].

Chu *et al.* [13] also showed that ensembles of classifiers were more robust than single classifiers. Care should, however, be taken with the size of the ensemble, because too many ensemble members will make the ensemble too computationally complex, while too few ensemble members will lead to poor generalisation performance [73].

In addition to their robustness, ensembles are good at handling concept drift [67][94]. Ensembles generally handle concept drift by applying the notion of age or minimum generalisation error to its members [94]. This allows too old or too inaccurate ensemble members to be detected and replaced with a classifier trained on new data [13][112][116]. Another approach to address concept drift in ensembles is to simply add newly trained ensemble members over time [67][116].

The main benefits of ensembles are their robustness to noise and ability to adjust to concept drift. Their computational complexity, however, might not make them suitable for high speed data streams. Furthermore, the classifiers that make up the ensembles must have some effectiveness in solving SDCPs, if the classifiers are to provide any meaningful contribution to the ensembles.

Ensembles have been developed to successfully deal with SDCPs [67][73][109][116]. Wang *et al.* [116] proposed an ensemble of classifiers that can be used for SDCPs. The ensemble weights the relevance of each classifier in the current epoch using an expected

accuracy for the current chunk of patterns in the data stream [116]. The approach has the advantages of handling concept drift, coping with high speed, robustness and online learning [116].

Telec *et al.* [109] investigated the use of an ensemble of different ANNs for modelling a data stream of real-estate market prices. Telec *et al.* [109] showed ANN ensembles to be more effective than ensembles which make use of non-ANN classifiers. Furthermore, Telec *et al.* [109] showed that larger ensembles provide better generalisation performance than smaller ensembles.

Ensembles are outside the scope of this thesis. However, the reader is referred to the comprehensive survey of ensembles by Krawczyk *et al.* [67].

5.3.4 Artificial Neural networks

Various ANN approaches have been developed to solve SDCPs [21][76][93][94][98][102].

Cui *et al.* [21] introduced a new one-pass *sequence learning* ANN architecture called hierarchical temporal memory (HTM), based on the recent neuroscience discovery of synaptic integration in the cerebral cortex. Sequence learning is a type of classification problem that requires classification of the outcome from a sequence of past patterns [21]. To do so, the classifier must maintain *temporal context* [21][33]. Temporal context refers to the time-based contextual dependencies that can exist between patterns, i.e. a sequence of events that provides context for the next event [21]. HTM allows for *temporal context* to be built up and maintained by the ANN. Cui *et al.* [21] showed that HTM was more effective, robust, and fault tolerant than RNNs and other traditional ANNs in streamed data sequence learning problems. The HTM, however, does not provide mechanisms to control model complexity, handle random dynamics, and reduce the number of tunable control parameters. Furthermore, the algorithm was not evaluated on SDCPs but streamed data sequence learning problems. Thus, the applicability of HTM to SDCPs is unknown.

Pratama *et al.* [93] proposed an online random neural network (RdNN), called the recurrent type-2 random vector functional link network (RT2McRVFLN), to address SDPs. RdNNs, first proposed by Gelenbe [46], are ANNs which are based on the biological behaviour of neuron circuits. The mechanics of RdNNs are outside the scope of

this thesis. The reader is however referred to Gelenbe [46] for more information. Note that the RT2McRVFLN is a type of RNN and not a FFNN. The RT2McRVFLN was found to be competitive against other streamed data classifiers in-terms the accuracy-to-simplicity trade-off, and was able to control model complexity to some extent [93]. The algorithm, however, did not provide any explicit means to deal with bounded memory, noisy patterns, system faults, reduce the number of tunable control parameters, and ensure patterns were only processed once during training. Furthermore, the RT2McRVFLN was not compared to traditional benchmarks algorithms such as stochastic BP, and the empirical analysis was done on a limited set of three benchmark problems.

Liang *et al.* [76] proposed a fast online sequence learning FFNN based on extreme learning machine (ELM) concepts, called online sequential extreme learning machine (OS-ELM) [76]. ELMs are FFNNs that allow fast training by inverting the training set matrix and output matrix, in order to find the optimal weights. Because there is no guarantee that the exact inverse might exist, a pseudo matrix inversion formula, i.e. the Moore-Penrose generalised inverse, is used instead of normal matrix inversion [76]. The pseudo matrix inversion attempts to find an approximate inverse that minimises the least square error [76]. The advantages of OS-ELM is its ability to handle situations requiring high speed data processing and little parameter tuning. The algorithm, however, did not provide any explicit means to deal with concept drift, noisy patters, system faults, and model complexity.

Sancho-Asensio *et al.* [102] proposed a robust, online, concept-drift handling classifier system for SDCPs called supervised neural constructivist system (SNCS). SNCS makes use of FFNNs, stochastic BP, and genetic algorithms (GAs) [102]. GAs are population-based optimisers like PSOs. GAs, however, are based on evolutionary concepts and not swarm movements [33][39]. The GA is used to select an FFNN architecture, while BP is used to adjust the weights. The algorithm, however, did not provide any explicit means to deal with bounded memory, high-speed data streams, system faults, reduce the number of tunable control parameters, and ensure patterns were only processed once during training.

Rakitianskaia and Engelbrecht [98] and Rakitianskaia [94] investigated the use of 3-layer FFNNs, trained by dynamic PSOs, as classifiers for dynamic classification prob-

lems [94]. Rakitianskaia [94] concluded that the dynamic PSOs, including the QPSO, are suitable for learning FFNNs which experience concept drift in online situations, and can in some cases outperform stochastic BP. Rakitianskaia [94], however, did not investigate classification problems with respect to the other SDCP requirements, discussed in Section 5.2. Furthermore, Rakitianskaia [94] did not cater for the effects of saturation that plague PSO-based FFNN learning algorithms [96][115].

Aside from Rakitianskaia and Engelbrecht [98] and Rakitianskaia [94], there has not been any other significant investigations into the use of 3-layer FFNNs, trained by dynamic PSOs. Most research on 3-layer FFNNs trained by PSOs focused on static classification problems and classification problems that require only the handling of concept drift, and not SDCPs [10][95][98][99][102][115].

Gupta and Lam [50] have shown that WD combined with the BP algorithm can handle static classification problems with noise. Furthermore, studies have shown that WD and WE can improve the accuracy and complexity performance of both BP and PSO weights adjustment algorithms [10][68][95][118]. However, current studies, i.e. [10][50][68][95][118], considered only static classification problems and not dynamic classification problems.

5.3.5 Conclusion

Table 5.1 summaries the streamed data classifier requirements focused on by the classifiers reviewed in this Section 5.3. From the above review of current SDCP literature and table 5.1, the following conclusions are made:

- Most streamed data classifiers that were investigated considered limited subsets of the SDCP requirements presented in Section 5.2.
- There is no significant research into the use of regularised 3-layer FFNNs to deal with SDCPs.
- The issue of saturation has been addressed for 3-layer FFNNs trained by static PSOs, but the suggested approaches have not been tested on streamed data FFNN classifiers trained by dynamic PSOs.

Table 5.1: Comparison of the requirements focused on by the reviewed classifiers

Requirements focused on	Reviewed classifiers									
	VFDT [24]	*CVDFT* [60]	*Ensemble-based* [109][116]	*HTM* [21]	*OS-ELM* [76]	*RT2MCRuFLN* [93]	*SNCS* [102]	*Stocastic BP FFNN* [94][98]	*QPSO FFNN* [94][98]
Bounded memory	X	X		X	X			X	X
Unbounded data set	X	X		X	X	X	X	X	X
Concept drift		X	X	X		X	X		X
Random dynamics						X	X		X
Online learning	X	X	X	X	X	X	X	X	X
High speed	X	X	X	X	X	X			
One-pass	X	X		X	X				
Limited number of tunable control parameters				X	X			X	X
Maintain low model complexity						X	X		
Robustness			X	X			X		
Fault tolerance				X					

- There is no significant research into the use of combining regularisation and dynamic PSOs to train 3-layer FFNNs for SDCPs.

The work of this thesis is thus necessary, because there is a gap in SDCP literature with regards to regularised 3-layer FFNNs.

5.4 Summary

The chapter defined data streams as a sequence of data points. SDCPs were then introduced as a class of non-trivial classification problems based on data streams. The big data and real-world nature of data streams, however, was shown to impose eleven additional requirements on streamed data classifiers. These requirements make SDCP non-trivial classification problems. Furthermore, background on streamed data classifiers and related works was provided. This review of related work showed that the work done by this thesis is necessary.

This chapter concludes the background part of this thesis. The next two chapters presents the original proposals made by this thesis starting with Chapter 6. Chapter 6 proposes a quantitative method for analysing and classifying the dynamic environments of SDCPs.

Chapter 6

Quantifying the environment of a streamed data classification problem

If you can't measure it, you can't improve it.

Peter Ferdinand Drucker (1909 – 2005)

Classification of the environment of a CI problem according to the four dynamic environments discussed in Section 2.2 requires the spatial and temporal severity of the problem to be known. Current literature thus far has made no suggestions on how to measure the spatial and temporal severities of streamed data classification problems (SDCPs). This chapter proposes several measures to quantify the spatial and temporal severities of SDCPs so that their dynamic environments can be classified as either quasi-static, abrupt, progressive, or chaotic. This thesis uses the term *severity measure* to describe any method that can quantify either the temporal or spatial severity of an optimisation problem.

The remainder of this chapter is organised as follows: Section 6.1 discusses the issues that need to be considered when designing severity measures for SDCPs. Section 6.2 discusses the identification of environment instances in SDCPs. Section 6.3 proposes a method for measuring the spatial severity of SDCPs. Section 6.4 proposes a method for measuring the temporal severity of SDCPs. Section 6.5 proposes a way of normalising the proposed severity measures so that they can be used to classify the environment of an

SDCP. Section 6.6 discusses how to classify SDCP environments, using the normalised severity measures. Lastly, Section 6.7 summaries the chapter.

6.1 Issues with potential severity measures

Any severity measure for SDCPs will need to consider the following three issues:

- *Environment instance identification*: To be able to measure the severity of a change or the frequency between changes, a severity measure requires knowledge of how the distribution of the targets changed and when the change occurred. The most effective way of knowing what changes occurred and when, is to determine the environment instances of a problem. SDCPs, however, present the classification problem as a continuous sequence of patterns, instead of sets of patterns that represent environment instances. A way of defining and identifying environment instances in a data stream is therefore required for the severity measures of SDCPs.

- *Severity measures provide estimations of actual severities*: Real-world data streams are unbounded. Any reading from a severity measure will therefore be an estimate of the actual severity, because the reading will only be based on a sub-sequence of patterns in the data stream. Furthermore, the reading will represent an average because the severity levels can change during the course of the sub-sequence.

- *Environment classifications based on severity measures are subjective*: The transition between quasi-static, abrupt, progressive and chaotic environments are relative to the upper bounds of both the spatial and temporal severities (refer to Figure 2.5). However, the environment classification scheme of Duhain and Engelbrecht [25] does not specify any definitive upper bounds for both types of severities. Some user-defined upper bounds, therefore, need to be chosen to make environment classifications comparable across various SDCPs. Any environment classifications are thus subjective to the choice of the two upper bounds.

Data streams in controlled environments are either artificially generated or extracted from real-world data stream, hence the number of patterns in these data streams is finite.

This allows the changes that occur in these data streams to be known and limited. The exact upper bounds of temporal and spatial severities can thus be determined. Severity measures can thus still provide insight into data streams in controlled environments.

6.2 Identifying environment instances

An environment instance of a dynamic classification problem is the set of patterns that represent the decision space of a classification problem for some period in time (refer to Section 2.2.2 on dynamic environments). Environment instances have four properties [25][33][98]:

1. Each environment instance contains one or more patterns.
2. A pattern can reoccur more than once in an environment instance.
3. The mappings between the inputs and targets, i.e. the *input–target pairs*, never change in the same environment instance, because then a new environment instance has been created.
4. The distribution of the targets for an environment instance must have changed when comparing an environment instance to the preceding environment instance.

Note that an environment change can result in the instances of a pattern disappearing, appearing, or their targets changing to reflect the the changes in the distribution of the targets.

Non-SDCPs explicitly indicate the patterns of an environment instance for the given problem. SDCPs, however, do not. Spatial and temporal severities, however, must be based on the environment instances in the SDCP [25]. Algorithm 4 is, therefore, proposed as a way of extracting environment instances from SDCPs based on data streams in controlled environments. The algorithm takes two parameters:

1. The sequentially-ordered set of patterns that occur in the data stream, D.
2. The bin width b_w, where $b_w \geq 0$. The bin width is used to bin the elements of the input vectors into interval groups with the size of b_w. Binning allows similar

input vectors to be seen as the same. This is essential for data streams where input vectors do not repeat or the input vectors are erroneous.

Algorithm 4 works in two modes: binning disabled, i.e. $b_w = 0$, and binning enabled, i.e. $b_w > 0$. Regardless of the mode, the main mechanics of the algorithm is the same: the algorithm scans sequentially through the ordered set of patterns D, and allocates each pattern to either the current or next environment instance. If the input vector has previously been recorded in the current environment instance, then the algorithm checks to see if the target has changed, otherwise the pattern is added to the current environment instance. If the target has changed, a new environment instance is added to the array of environment instances, and the pattern is added to the new environment instance.

Algorithm 4 SDCP environment instance extraction algorithm

Input: b_w, D
  Initialise an empty 2D array of patterns, named *Environment instances*
  Initialise an empty array of patterns, named *Environment instance*
  for $p = 0$ to $|D|$ **do**
    Set $\vec{z}_p$ as D_p's input vector
    Set $\vec{t}_p$ as D_p's target vector
    Set $\vec{b}_p$ as D_p's input vector
    if Binning mode is enabled, i.e. $b_w > 0$ **then**
      for $i = 0$ to I **do**
        Calculate a as $b_w \times \left\lfloor \frac{z_{p,i}}{b_w} \right\rfloor$.
        Calculate b as $a + b_w$
        Bin the i-th element of $\vec{z}_p$ to the bin with the interval $[a, b)$.
        Assign binned value for $z_{p,i}$ to $b_{p,i}$
      end for
    end if
    if If the binned input vector $\vec{b}_p$ matches any of the other binned input vectors in *Environment instance* **then**
      if $\vec{t}_p$ is not exactly the same as the target associated with the matched binned input vector in *Environment instance* **then**
        Append the array *Environment instance* to the array *Environment instances*
        Empty the array *Environment instance*
      end if
    end if
    Append the binned pattern $(\vec{b}_p, \vec{t}_p)$ to the array *Environment instance*
  end for
  return The array *Environment instances*

If binning is enabled, then the elements of the input vector of each pattern are binned

and used instead of the raw input vector by Algorithm 4. Without the binning mode, Algorithm 4 will fail in each of the following cases:

- If there is noise, i.e. erroneous patterns, in the data stream, then the algorithm will detect false environment changes and create non-existent environment instances. Algorithm 4, thus, over estimates the number environment instances.
- If an input vector only occurs in one environment instance in the data stream, then the algorithm will not be able to use the input vector to detect change and may leave out environment instances. Algorithm 4, thus, underestimates the number environment instances.

The binning procedure groups similar input vectors into a binned input vector. The patterns associated with each of the original input vectors are now all associated with the binned input vector. The likelihood of the targets associated with the binned input vector changing is thus higher than when the targets were associated with their original input vectors. Algorithm 4 controls the likelihood of targets changing by changing the size of the bins via the bin width parameter, b_w.

The larger b_w is, the less binned input vectors there are and the higher the likelihood of targets changing. If b_w is too large, then any two patterns whose targets differ, irrespective of their input vectors, will be seen as a change because all patterns will have the same binned input vector. If b_w is too small, then the number of pattern instances associated with the input vectors and their binned input vectors will remain unchanged.

The binning mode, therefore, transforms the original data stream before extracting the environment instances. To avoid significant alteration of the original data stream while overcoming the issues of noise and non-repetition of input vectors, b_w needs to be optimised. An optimal value for b_w will bin the input vectors such that:

- the number of binned input vectors is similar to the number of input vectors in the original data stream, while
- ensuring that binned input vectors have at least two patterns associated with it.

The former constraint prevents the data stream from being transformed significantly, while the latter constraint increases the chances for similar but not same input vectors to experience changes.

Two key advantages of Algorithm 4 is that the algorithm is deterministic and is agnostic of any classifier.

6.3 Spatial severity measure

Spatial severity describes the magnitude of change experienced when the environment changes [25]. A possible way to quantify spatial severity is to calculate the average change experienced by the environment instances with respect to the changes in targets, as follows:

$$\Theta_{b_w} = \frac{\sum_{e=1}^{|I_{b_w}|} \sum_{p=1}^{|I_{b_w,e}|} \Delta_\Theta(I_{b_w,e,p})}{|I_{b_w}|} \tag{6.1}$$

where Θ_{b_w} is the average spatial severity over all of the environment instances extracted by Algorithm 4 using a bin width b_w, I_{b_w} is the set of extracted environment instances, and $I_{b_w,e}$ is the set of binned patterns for the e-th environment instance in I_{b_w}. $I_{b_w,e,p}$ is the p-th pattern in $I_{b_w,e}$. $\Delta_\Theta(I_{b_w,e,p})$ is the spatial change for $I_{b_w,e,p}$, which is calculated as follows:

- If the binned input vector of $I_{b_w,e,p}$ occurred in a pattern earlier in $I_{b_w,e}$ or it did not occur in $I_{b_w,e-1}$, then $\Delta_\Theta(I_{b_w,e,p}) = 0$.
- Otherwise, $\Delta_\Theta(I_{b_w,e,p})$ is the Euclidean distance between the target of $I_{b_w,e,p}$ and the target of the first pattern with the same input vector found in $I_{b_w,e-1}$. The Euclidean distance is used because targets can have one or more floating-point values.

Furthermore, $\Theta_{b_w} \in [0, \Delta_\Theta^*]$ where Δ_Θ^* is the maximum change that can be experienced by a target. If $\Theta_{b_w} = 0$, then there are no changes. If $\Theta_{b_w} = 1$, then each environment instance experiences one unit of change on average. The higher Θ_{b_w} is, the more severe the average change across all the environment instances. Note that new input patterns are not considered as changes by Θ_{b_w} because there is no previous target vector to compare with.

6.4 Temporal severity measure

Temporal severity describes how long an environment instance exists [25]. In SDCPs a unit of time is one pattern, because streamed data classifiers process each pattern exactly once. A possible way to quantify temporal severity is to calculate the average number of patterns in the environment instances, i.e. the average size of the environment instances, extracted by Algorithm 4 using a bin width b_w, as follows:

$$\begin{aligned} \tau_{b_w} &= \frac{\sum_{e=0}^{|I_{b_w}|} |I_{b_w,e}|}{|I_{b_w}|} \\ &= \frac{|D|}{|I_{b_w}|} \end{aligned} \tag{6.2}$$

where $\tau_{b_w} \in [1, |D|]$. $\tau_{b_w} < 1$ is undefined, because each environment instance must at least have one pattern. The larger τ_{b_w} is, the longer it takes for changes to happen, i.e. lower temporal severity. Temporal severity therefore increases as τ_{b_w} decreases towards 1. If $\tau_{b_w} = |D|$, then there are no changes.

6.5 Normalising severity measures

The goal of the severity measures is to allow the classification of the dynamic environments of various SDCPs. Because the measures Θ_{b_w} and τ_{b_w} are not normalised, any classification of a dynamic environment of one SDCP is not necessarily equivalent to the exact same classification made for a different SDCP. This inconsistency in the classifications is due to the following three issues:

1. **Different units of measure issue**. The severity measures represent different units of measure. That is, spatial severity is based on Euclidean distance versus temporal severity that is based on the number of patterns. The environment classification scheme of Duhain and Engelbrecht [25] requires comparable values such as percentages or ratios in respect to the bounds of the measures.

2. **Same sized data stream issue**. The upper bounds of both severity measures are related to $|D|$. Hence, any percentage or ratio can only be compared to SDCPs with same sized data stream.

3. **Inverse issue**. τ_{b_w} associates smaller values with greater levels of temporal severity. The inverse, i.e. $\frac{1}{\tau_{b_w}}$, is required by the environment classification scheme of Duhain and Engelbrecht [25].

This section discusses the approaches used to normalise both measures. Section 6.5.1 discusses the normalisation of Θ_{b_w}. Section 6.5.2 discusses the normalisation of τ_{b_w}.

6.5.1 Normalising spatial severity

The different units of measure issue can be addressed by normalising the severity value to a ratio between the severity value and the bounds of the particular severity measures as follows:

$$\frac{\text{Severity value} - \text{Lower bound}}{\text{Upper bound} - \text{Lower bound}}$$

This approach, however, requires SDCP based on data streams in controlled environments, because of the need for upper bounds.

The same sized data stream issue can be addressed by using the maximum severity value found from the set of SDCPs that are being benchmarked, as the upper bound of the particular severity measure.

Using the solutions proposed above, spatial severity can be normalised as follows:

$$\Theta'_{b_w} = \frac{\Theta_{b_w}}{\max_{d=1}^{|S|} \Theta_{b_w}(S_d)} \tag{6.3}$$

where S is the set of data stream datasets and S_d is the d-th data stream in S. Θ'_{b_w} normalises Θ_{b_w} to a value with the range $[0, 1]$. If $\Theta'_{b_w} = 0$ then there is no change, i.e. the lowest level of spatial severity possible. If $\Theta'_{b_w} = 1$ then the particular SDCP has a spatial severity equal to the highest spatial severity found in S.

Note that if the unit of the target vector elements for one problem is different from those of another problem, e.g. binary, integer, or floating-point, then the units of the change magnitudes of the two problems will also be different. The spatial severity of the one problem will therefore not be comparable to the spatial severity of the other problem. It is thus important to ensure that the target vectors are encoded to the same unit. This thesis encodes the all target vectors into binary target vectors.

6.5.2 Normalising temporal severity

The different units of measure issue and same sized data stream issue for τ_{b_w} can be addressed in the same way as for Θ_{b_w} (refer to Section 6.5.1).

The inverse issue can be resolved by using the difference between a severity value and the upper bound of the value, as follows:

$$\text{Upper bound} - \text{Value}$$

.

Using the solutions proposed above, temporal severity can be normalised as follows:

$$\tau'_{b_w} = 1 - \frac{\tau_{b_w} - 1}{\max_{d=1}^{|S|} \tau_{b_w}(S_d) - 1} \tag{6.4}$$

τ'_{b_w} normalises τ_{b_w} to a linear value with the range $[0, 1]$. If $\tau'_{b_w} = 0$ then there is no change, i.e. lowest temporal severity possible. If $\tau'_{b_w} = 1$ then the particular SDCP has a temporal severity equal to the highest temporal severity in found in S.

Note that it is possible for $\max_{d=1}^{|S|} \tau_{b_w}(S_d)$ to be equal to 1, in which case τ'_{b_w} will be undefined. However, in such a case all of the SDCP in S are experiencing the highest degree of temporal severity possible and τ'_{b_w} can be said to be 1 for each of the SDCPs.

6.6 Dynamism of problem environments

Using Θ'_{b_w} and τ'_{b_w}, it is possible to classify the environments of SDCPs according to environment classification scheme of Duhain and Engelbrecht [25], as follows:

$$\Lambda_{b_w}(\Theta'_{b_w}, \tau'_{b_w}) = \begin{cases} \text{Quasi-Static}, & \text{if } \Theta'_{b_w} \leq 0.5 \text{ and } \tau'_{b_w} \leq 0.5 \\ \text{Abrupt}, & \text{else if } \tau'_{b_w} \leq 0.5 \\ \text{Progressive}, & \text{else if } \Theta'_{b_w} \leq 0.5 \\ \text{Chaotic}, & \text{otherwise} \end{cases} \tag{6.5}$$

where $\Lambda_{b_w}(\Theta'_{b_w}, \tau'_{b_w})$ is the environment classification for a SDCP with a normalised spatial severity of Θ'_{b_w} and normalised temporal severity of τ'_{b_w}.

The normalised severity measures, Θ'_{b_w} and τ'_{b_w}, form a 2-dimensional plane in continuous space. An environment of a SDCP thus exhibits the characteristics for each of the four dynamic environment types to some degree. The environment classification, Λ_{b_w}, and the two normalised severity measures, however, do not provide any information to indicate to what degree the given problem environment behaves like the class Λ_{b_w}. What would assist in this matter, is a singular value that indicates how an environment of a problem is *dominated* by Λ_{b_w} and the other three types of dynamic environments.

The degree by which the environment of a problem is dominated by each of the four types of dynamic environments can be calculated as follows:

$$\zeta_{Q,b_w} = 1 - \frac{\sqrt{(\tau'_{b_w})^2 + (\Theta'_{b_w})^2}}{\sqrt{2}} \tag{6.6}$$

$$\zeta_{A,b_w} = 1 - \frac{\sqrt{(\tau'_{b_w})^2 + (1 - \Theta'_{b_w})^2}}{\sqrt{2}} \tag{6.7}$$

$$\zeta_{P,b_w} = 1 - \frac{\sqrt{(1 - \tau'_{b_w})^2 + (\Theta'_{b_w})^2}}{\sqrt{2}} \tag{6.8}$$

$$\zeta_{C,b_w} = 1 - \frac{\sqrt{(1 - \tau'_{b_w})^2 + (1 - \Theta'_{b_w})^2}}{\sqrt{2}} \tag{6.9}$$

where ζ_{Q,b_w}, ζ_{A,b_w}, ζ_{P,b_w}, and ζ_{C,b_w} are the domination values of the quasi-static, abrupt, progressive and chaotic classifications, respectively. Equations (6.6) to (6.9) quantify domination as a ratio. The numerator of the ratio is the Euclidean distance of the point created by the severity measures from the point of complete domination. The denominator of the ratio is the maximum possible distance from complete domination in the 2-dimensional plane, i.e. $\sqrt{2}$. Each domination value is in the range $[0, 1]$. A domination value of 0 indicates *no domination*, i.e. the environment does not behave at all like the particular type of dynamic environment. On the other hand a domination value of 1 indicates *complete domination*, i.e. the environment only has characteristics of the particular type of dynamic environment.

The values ζ_{Q,b_w}, ζ_{A,b_w}, ζ_{P,b_w}, and ζ_{C,b_w}, however, only show how their respective dynamic environment type influences the environment of a problem, and not how all

four dynamic environment types influence the environment of the problem at the same time. The four domination values can be unified into a single value, as follows:

$$\zeta_{b_w} = max\{\zeta_{Q,b_w}, \zeta_{A,b_w}, \zeta_{P,b_w}, \zeta_{C,b_w}\} - min\{\zeta_{Q,b_w}, \zeta_{A,b_w}, \zeta_{P,b_w}, \zeta_{C,b_w}\} \tag{6.10}$$

where $\zeta_{b_w} \in [0, 1]$. Equation 6.10 uses the range of the four domination values, because it allows all four domination values to be interpreted via the difference between the most influential dynamic environment type, i.e. Λ_{b_w}, and least influential dynamic environment type. Values of ζ_{b_w} are interpreted, as follows:

- When $\zeta_{b_w} = 0$, then an environment is balanced amongst all four dynamic environment types. That is, all four dynamic environment types have an equal influence on the environment of the problem.
- When $\zeta_{b_w} = 1$, then the environment only exhibits the characteristics of the dynamic environment type identified by Λ_{b_w}, e.g. static if Λ_{b_w} is quasi-static.
- When $\zeta_{b_w} \approx 0.5$, then the environment of the problem primarily exhibits characteristics of both the dynamic environment type identified by Λ_{b_w} and an adjacent dynamic environment type.
- The larger ζ_{b_w} is, the more the environment of the problem behaves like the type of dynamic environment identified by Λ_{b_w}.
- The smaller ζ_{b_w} is, the more domination is distributed amongst the four dynamic environment types. That is, the less the environment of the problem exhibits characteristics of just one dynamic environment type.

To assist in visualising and understanding ζ_{b_w}, refer to Figure 6.1. Figure 6.1 provides a visualisation of ζ_{b_w} values with respect to Θ'_{b_w} and τ'_{b_w} using 20 contour intervals. By overlaying Figure 6.1 with Figure 2.5 it becomes clear that Equation (6.10) breaks each region of the four dynamic environment types, i.e. the four quadrants, into diagonal-like bands. The bands are orientated in such a way that they face the two extremes of each region, i.e. from the centre of the severity space to the relevant corner. Furthermore, the bands become more concave as ζ_{b_w} approaches a value of one. On the other hand,

the bands become straighter as ζ_{b_w} approaches a value of zero. Figure 6.1 indicates that a majority of problems have environments that display characteristics from two or more dynamic types of environments.

Using ζ_{b_w} together with Λ_{b_w} provides a summarised analysis of the dynamic environment of an SDCP, because Λ_{b_w} indicates the dominant dynamic environment type and ζ_{b_w} indicates the degree to which the environment of the SDCP conforms to the characteristics of Λ_{b_w}. Another benefit of summarising the environment with only Λ_{b_w} and ζ_{b_w} is that the location of Θ'_{b_w} and τ'_{b_w} can be estimated accurately.

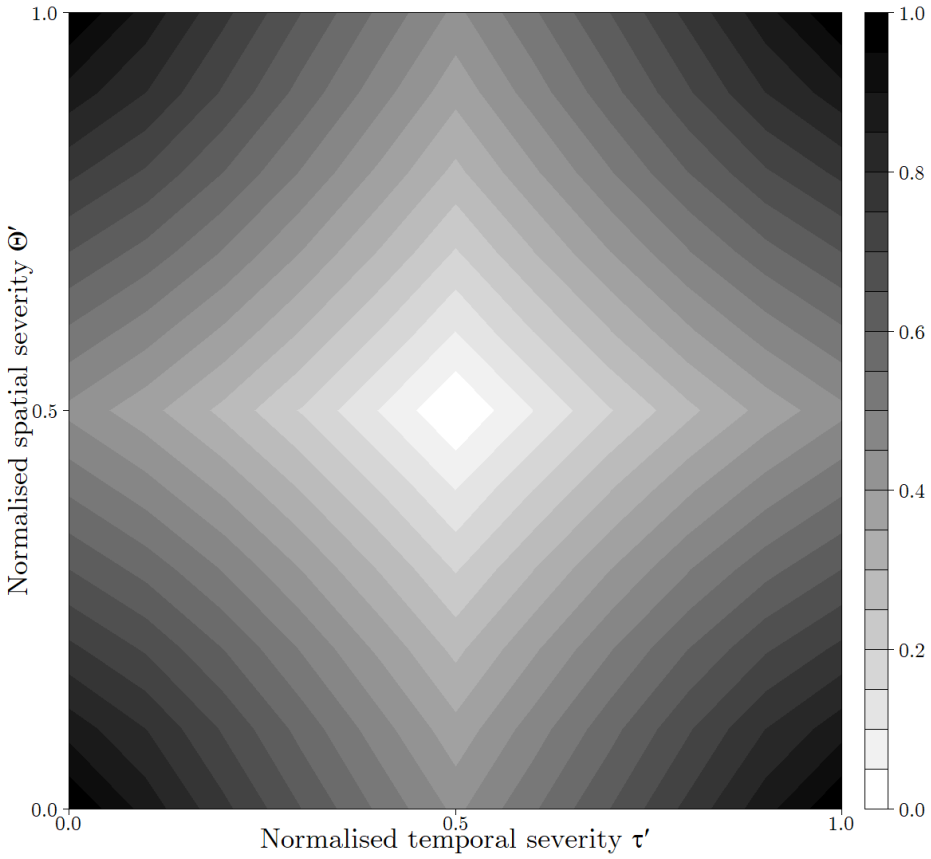


Figure 6.1: Contour plot of ζ_{b_w} with respect to Θ'_{b_w} and τ'_{b_w} using 20 levels

6.7 Summary

The chapter proposed a quantitative method of determining whether the overall environment of an SDCP is quasi-static, abrupt, progressive or chaotic. This included an algorithm for identifying environment instances in SDCPs, and a method for measuring both the spatial and temporal severity across the environment instances.

A measure for determining the degree by which the characteristics of the environment classification dominates the environment of a SDCP, was also proposed.

The quantitative method allows the environments of SDCPs to be classified automatically. Knowing the dynamic environment types of the SDCP provides a better understanding of the performance trends of streamed data classifiers, and another level for aggregating empirical results when comparing streamed data classifiers.

The next chapter introduces the regularisation-based FFNN streamed data classifiers investigated in this thesis.

Chapter 7

Regularised feed forward neural networks as streamed data classifiers

Everything should be made as simple as possible, but not simpler.

Albert Einstein (1879 – 1955)

Regularised 3-layer FFNNs have not been investigated thoroughly as streamed data classifiers, despite their ability to manage the complexity of the data. To address this gap in the research, this chapter proposes four regularised 3-layer FFNN classifiers for SDCPs. The term *regularised classifier* is used in this thesis to describe any classifier that uses regularisation.

The remainder of the chapter is organised as follows. Section 7.1 describes the FFNN architecture used for the purpose of this study. Section 7.2 describes two BP-based regularisation learning algorithms. Section 7.3 describes two QPSO-based regularisation learning algorithms. Section 7.4 presents the four streamed data classifiers. Section 7.5 justifies the use of the proposed classifiers as streamed data classifiers. Lastly, Section 7.6 summaries the chapter.

7.1 Proposed architecture

This thesis uses a 3-layer FFNN with the same structure as illustrated in Figure 4.1. The remainder of this thesis refers to this architecture as the *proposed architecture*.

The proposed architecture employs summation units (refer to Equation (4.2)). Instead of using traditional bounded activation functions, such as the sigmoid function, the proposed architecture uses the left-bounded ReLU activation function with $\lambda = 1$ and $\theta = 0$ (refer to Equation (4.4)). The ReLU activation function is favoured over the sigmoid function, because of its lower computational complexity and non-vanishing gradients [81].

All the synapse weights, including the bias neuron weights, are uniformly initialised within the range described by Equation (4.13).

The number of input neurons are fixed to the input vector size of the SDCP, i.e. I. The number of hidden neurons are chosen in such a way that the architecture is *over-parametrised*, i.e. the architecture has more hidden neurons than needed for the particular problem. Note, that this thesis investigates problems where the number of hidden neurons required have been empirically determined. Furthermore, note that any FFNN with an over-parametrised architecture has the ability to approximate at the same level of accuracy as FFNNs with a smaller architecture provided that the over-parametrised architecture does not overfit [31][33].

Both the input and the hidden layers contain a single bias neuron. The input signal of the bias neurons are fixed at -1.

The architecture assumes the use of binary encoded targets, e.g. each element in a target vector is either 0 or 1 [91]. If there are two classes, then *binary encoding* is used, otherwise, *one-hot encoding* is used [51][91]. Hence, the number of output neurons is dependent on the number of discrete classes a SDCP has. If there are two classes, then the FFNN uses one output neuron, otherwise the FFNN uses one output neuron per class [51].

The ReLU activation function results in output neurons values that are bounded below by 0. Output neuron values, however, need to be bounded to the range $[0, 1]$ for binary classification target vectors. A common approach to bounding the values of output neurons is to use thresholds [33][39][94]. The value of an output neuron, o_k, is therefore constrained to the range $[0, 1]$, as follows:

$$o_k = \begin{cases} 1, & \text{if } o_k \geq 1.0 \\ o_k, & \text{otherwise} \end{cases} \tag{7.1}$$

Note that the constraint on the output neuron values does not change the gradient of the output neurons, because the gradient of the ReLU activation function is considered to be 1 if $net_{o_k} > 0$, otherwise the gradient is 0.

In the case of two discrete classes, an output neuron value of 0 or 1 indicates if the pattern represents one class or the other. In the case of more than two discrete classes, each output indicates if the pattern represents the class or not. A value of 0 indicates that the pattern does not represent the class in question. A value of 1 indicates that the pattern represents the class in question. If more than one output neuron has a value of 1, then the classification is also considered to be incorrect.

7.2 Back propagation learning algorithms

This section describes a WD learning algorithm and a WE learning algorithm for streamed data classifiers that use stochastic BP to adjust the weights of the FFNN.

Section 7.2.1 discusses the implementation of the BP algorithm used. Section 7.2.2 discusses the BP-based WD learning algorithm. Lastly, Section 7.2.3 discusses the BP-based WE learning algorithm.

7.2.1 Back propagation weights adjustment algorithm

The proposed architecture requires the output-to-hidden layer and hidden-to-input layer weights to change.

Each output-to-hidden layer weight, w_{kh}, is changed as follows [33][115]:

$$w_{kh}(t+1) = w_{kh}(t) + \Delta w_{kh}(t) + \alpha \Delta w_{kh}(t-1) \tag{7.2}$$

where $\Delta w_{kh}(t)$ is the change that weight w_{kh} experiences at epoch t, calculated as

$$\Delta w_{kh}(t) = -\eta \delta_{o_k} h_h$$

where h_h is the activation value of the h-th hidden neuron, δ_{o_k} is the error signal of the k-th output neuron, calculated as

$$\delta_{o_k} = -(t_k - o_k) f'_{ReL}(\sum_{h=1}^{J+1} w_{kh} h_h) \tag{7.3}$$

and $-(t_k - o_k)$ is the partial derivative of Equation (4.5) for a single pattern, and J is the number of hidden neurons excluding bias neurons in the hidden layer. f'_{ReL} is the partial derivative of $f_{ReL}(net_{o_k})$ with respect to the net input signal, net_{o_k}, which is calculated as the weighted sum of inputs into the o_k. Because of the piecewise nature of f_{ReL}, the activation function is not differentiable at 0 [81][108][115]. In practice when training with BP, f'_{ReL} is considered to be 1 if $net_{o_k} > 0$, otherwise f'_{ReL} is 0 [71][81][115].

Each hidden-to-input layer weight, w_{hi}, is changed as follows [33][115]:

$$w_{hi}(t+1) = w_{hi}(t) + \Delta w_{hi}(t) + \alpha \Delta w_{hi}(t-1) \tag{7.4}$$

where $\Delta w_{hi}(t)$ is the change that weight w_{hi} experiences at epoch t, calculated as

$$\Delta w_{hi}(t) = -\eta \delta_{h_h} z_i$$

and δ_{h_h} is the error signal of the h-th hidden neuron, calculated as

$$\delta_{h_h} = (\sum_{k=1}^{K} \delta_{o_k} w_{kh}) f'_{ReL}(\sum_{i=1}^{I+1} w_{hi} z_i) \tag{7.5}$$

where z_i is the value of the i-th input neuron.

Algorithm 5 presents the pseudo code for the BP implementation used in this thesis. Algorithm 5 is derived from Algorithm 3. Algorithm 5 focuses on the training of a pattern for SDCPs, and not on FFNN initialisation and training set selection. Algorithm 5 considers each training set to consist out of one pattern, because of the one-pass requirement of SDCPs (see Section 5.2).

Algorithm 5 Supervised ReLU summation 3-layer FFNN BP weights adjustment algorithm for an environment instance

Input: η, α, the FFNN, pattern p
Feed forward pattern p's inputs into the FFNN
Calculate error signal for the output layer neurons, using Equation (7.3)
Calculate error signal for the hidden layer neurons, using Equation (7.5)
Adjust FFNN's output to hidden layer weights, using Equation (7.2)
Adjust FFNN's hidden to input layer weights, using Equation (7.4)
return FFNN

7.2.2 Weight decay learning algorithm

To incorporate WD regularisation into Algorithm 5 is a relatively straightforward process [10][118]. The regularisation Equation (4.14), with E_r as the WD Equation (4.15), must be used as the new training error function, instead of SSE_p. To do so, the partial derivative of the product between the regularisation coefficient (λ_r) and the WD penalty term (Equation (4.15)), calculated as

$$\frac{\partial}{\partial w_n}\left[\lambda_r \frac{1}{2}\sum_{n=1}^{n_s} w_n^2\right] = \lambda_r w_n \tag{7.6}$$

must be incorporated into the weight adjustment equations (7.2) and (7.4) as follows:

$$\Delta w_{kh}(t) = -\eta(\delta_{o_k} h_h + \lambda_r w_{kh}) \tag{7.7}$$

$$\Delta w_{hi}(t) = -\eta(\delta_{h_h} z_i + \lambda_r w_{hi}) \tag{7.8}$$

7.2.3 Weight elimination learning algorithm

WE regularisation can be incorporated into Algorithm 5 in the same way as was done with WD regularisation in Section 7.2.2. The partial derivative of the product between the regularisation coefficient (λ_r) and the WE penalty term (Equation (4.16)), calculated as

$$\frac{\partial}{\partial w_n}\left[\lambda_r \sum_{n=1}^{n_s} \frac{\frac{w_n^2}{w_0^2}}{1+\frac{w_n^2}{w_0^2}}\right] = \lambda_r \frac{\partial}{\partial w_n}\left[\sum_{n=1}^{n_s} \frac{w_n^2}{w_0^2 + w_n^2}\right] \tag{7.9}$$

$$= \lambda_r \frac{2 w_n w_0^2}{(w_0^2 + w_n^2)^2} \tag{7.10}$$

must be incorporated into the weight adjustment equations (7.2) and (7.4) as follows:

$$\Delta w_{kh}(t) = -\eta\left[\delta_{o_k} h_h + \lambda_r \frac{2 w_{kh} w_0^2}{(w_0^2 + w_{kh}^2)^2}\right] \tag{7.11}$$

$$\Delta w_{hi}(t) = -\eta\left[\delta_{h_h} z_i + \lambda_r \frac{2 w_{hi} w_0^2}{(w_0^2 + w_{hi}^2)^2}\right] \tag{7.12}$$

7.3 Quantum particle swarm optimisation learning algorithms

This section describes a WD learning algorithm and a WE learning algorithm for streamed data classifiers that use QPSO to adjust the weights of the FFNN.

Section 7.3.1 discusses the implementation of the QPSO weights adjustment algorithm. Section 7.3.2 discusses the QPSO-based WD learning algorithm. Lastly, Section 7.3.3 discusses the QPSO-based WE learning algorithm.

7.3.1 Quantum particle swarm optimisation weights adjustment algorithm

Algorithm 6 presents the pseudo code for the QPSO weights adjustment algorithm implementation used in this thesis. Algorithm 6 is derived from Algorithm 2 and the implementations discussed in Section 4.4. Algorithm 6 focuses only on training a swarm of particles for a pattern of a SDCPs, and not on swarm initialisation and training set selection. Algorithm 5 considers each training set to consist out of one pattern, because of the one-pass requirement of SDCPs (see Section 5.2).

7.3.2 Weight decay learning algorithm

Application of WD regularisation to Algorithm 6 requires even fewer modifications than needed with Algorithm 3. Regularisation is achieved by changing the training error function from MSE to the regularisation Equation (4.14), with E_r as the WD Equation (4.15).

7.3.3 Weight elimination learning algorithm

WE regularisation can be incorporated into Algorithm 6 in the same way as was done with WD regularisation in Section 7.3.2. Hence, the training error function is changed from MSE to the regularisation Equation (4.14), with E_r as WE Equation (4.16).

Algorithm 6 Supervised ReLU summation 3-layer FFNN QPSO weights adjustment algorithm for an environment instance

Input: ω, c_1, c_2, r, distribution d, swarm of n_p particles, pattern p
  for $i = 0$ to n_p **do**
    Calculate the quality of $\vec{y}_i$, using Equation (4.6) and pattern p
  end for
  for $i = 0$ to n_p **do**
    Set $\vec{y}_i$ as the best personal best position in the Von Neumann neighbourhood of particle i
    if (particle i is a quantum particle) **then**
      Update $\vec{x}_i$ using Equation (3.6)
    else
      Update $\vec{v}_i$ using Equation (3.1)
      Update $\vec{x}_i$ using Equation (3.2)
    end if
  end for
  for $i = 0$ to n_p **do**
    Calculate the quality of $\vec{x}_i$ using Equation (4.6) and pattern p
    if ($\vec{x}_i$ is better than $\vec{y}_i$) **then**
      $\vec{y}_i = \vec{x}_i$
    end if
  end for
  return Swarm of n_p particles

7.4 Proposed streamed data classifiers

By combining the proposed architecture and four regularisation learning algorithms, the following four streamed data classifiers are formed:

- *BP-WD*, which uses the WD BP-based learning algorithm (presented in Section 7.2.2) to train the FFNN. BP-WD has three tunable control parameters, i.e. α, η and λ_r.
- *BP-WE*, which uses the WE BP-based learning algorithm (presented in Section 7.2.3) to train the FFNN. BP-WE has four tunable control parameters, i.e. α, η, λ_r and w_0.
- *QPSO-WD*, which uses the WD QPSO-based learning algorithm (presented in Section 7.3.2) to train the FFNN. QPSO-WD has seven tunable control parameters, i.e. ω, c_1, c_2, n_p, distribution d, r and λ_r.

- *QPSO-WE*, which uses the WE QPSO-based learning algorithm (presented in Section 7.3.3) to train the FFNN. QPSO-WE has eight tunable control parameters, i.e. ω, c_1, c_2, n_p, distribution d, r, λ_r and w_0.

Because of the online learning requirement for SDCPs, no stopping conditions are used to terminate the learning algorithms of the four streamed data classifiers. However, each training set is only trained for one epoch. The maximum number of epochs stopping condition is, therefore, used to enforce the one-pass requirement.

Note that the term *proposed classifiers* is used to refer to BP-WD, BP-WE, QPSO-WD and QPSO-WE, for the remainder of this thesis.

7.5 Justification of proposed streamed data classifiers

This section discusses how the four streamed data classifiers (proposed in Section 7.4), address the issues of saturation, local optimum trapping, and the eleven SDCP requirements. Sections 7.5.1 to 7.5.13 present the discussions with reference to each of the thirteen issues, i.e. handling saturation, addressing local optima trapping, and the 11 design requirements of SDCPs. Lastly, Section 7.5.14 concludes the discussions.

7.5.1 Saturation issue

The left-bounded property of the ReLU activation function only allows saturation to occur on the left-side [97][115]. It was hypothesised by [115] that this feature of the ReLU function could potentially allow saturation to be reduced. No studies have, however, been carried out to evaluate the extent to which this is true. The ReLU activation function is employed by the four proposed classifiers due to its potential of reducing saturation. The use of ReLU activation functions as a means of reducing saturation will be investigated empirically by this thesis.

All four proposed classifiers make use of regularisation. Regularisation has been shown to reduce saturation when PSOs are used to adjust weights [10][95].

Rakitianskaia and Engelbrecht [96] suggested that by controlling the velocity and preventing it from exploding would reduce saturation, e.g. using velocity clamping. Velocity can be more effectively controlled by using the constriction coefficient [29]. The constriction coefficient does not explicitly set a hard limit on the velocity of particles, like velocity clamping, but rather guarantees that that the neutral particles in the swarm will reach first order and second order stability, i.e. the swarm will converge so that the expectation and variance of the positions are constant, under certain conditions [14][15][29]. These stability guarantees will ensure that velocity will not explode [14]. The use of the constriction coefficient can be emulated by setting the inertia parameter, ω, and positive acceleration coefficient parameters, c_1 and c_2 to the values discussed in Section 3.1.5. This approach is used by QPSO-WD and QPSO-WE, because it introduces no additional computational effort. Note that the quantum particles of QPSO do not make use of a velocity component, hence, their velocity cannot explode nor does the constriction coefficient affect them.

The proposed classifiers, therefore, were set up to reduce saturation.

7.5.2 Local optimum trapping issue

Both BP-WD and BP-WE allow the momentum and learning rate of the BP algorithm to be tuned. This reduces the susceptibility of BP-WD and BP-WE to local optimum trapping, because a suitable exploration-exploitation trade-off can be found for a SDCP (refer to Section 4.3.9). Note that in practical applications, the momentum and learning rate might not be an effective solution for preventing local optimum trapping, because the momentum and learning rate can not be adjusted as the environment changes.

On the other hand, QSO-WD and QPSO-WE are less susceptible to local optimum trapping, because their weight adjustment algorithms are population-based. Furthermore, the quantum particles in QSO-WD and QPSO-WE never converge fully, but continue to explore, therefore reducing the chance of local optimum trapping.

Lastly, the use of regularisation provides more information about the search space, i.e. model complexity, than learning algorithms that only account for accuracy [10]. BP-WD and BP-WE should thus stagnate less quickly, because gradients are generally larger in magnitude. This helps the search build momentum to escape from a local minimum.

On the other hand, QPSO-WD and QPSO-WE should converge slower, because the regularised search space will typically contain more optima which will increase swarm diversity. The slow down aids in reducing local optimum trapping, because if a particle gets trapped, the other particles take longer to also get trapped. The other particles, thus, have more time to search for better solutions.

The proposed classifiers, therefore, were set up to reduce local optimum trapping. QPSO-WD and QPSO-WE, however, are more capable to reduce local optimum trapping than BP-WD and BP-WE.

7.5.3 Bounded memory requirement

Three-layer FFNNs do not necessarily need to be excessively large to learn classification problems, as seen in [31], [94], [95] and [109]. Furthermore, regularisation is an architecture selection algorithm that does not alter the number of synapses or neurons. Hence, the number of neurons and synapses that are selected initially does not change during training, for either of the four proposed classifiers.

The bounded memory requirement can, therefore, be adhered to by each of the proposed classifiers provided that the initial architecture does not violate the bounded memory requirement.

7.5.4 Unbounded dataset requirement

Because all the proposed classifiers make use of regularisation, their ability to unlearn any patterns that were learned is enhanced, i.e., made faster. The proposed classifiers, therefore, have a greater chance of automatically replace old unnecessary information with new relevant information as the data stream progresses. Furthermore, the proposed classifiers also do not run the risk of building up excessive collections of patterns, because each pattern is only kept for one epoch.

The four proposed classifiers thus will be able to operate on an infinite number of patterns.

7.5.5 Concept drift requirement

Rakitianskaia [94] concluded that BP has the ability to handle certain types of dynamic environments, but can still become trapped by an optimum of the previous environment instance. The reason for this is because BP has no explicit change detection or change handling mechanisms.

On the other hand, QSO-WD and QPSO-WE make use of the dynamic QPSO algorithm to adjust their weights. The QPSO algorithm implicitly supports change detection through the revaluation of the quality of each particle every epoch, and change handling through quantum particles (refer to Section 3.2.2). Hence, QSO-WD and QPSO-WE are well-suited at handling concept drift in SDCPs [8][98].

Using regularisation with BP and PSO on static problems have been found to improve performance [50][95]. The main reason for this is that regularisation allows the classifiers to select the architecture during training and not just the weights [50]. The selection ability of regularisation should, thus, help to reduce the chances of the classifiers producing stale models.

Lastly, Bosman *et al.* [10] showed that regularisation penalisation terms, such as WE, can create search spaces with steeper gradients and more minimums. This can help Algorithm 5, which has a momentum term, to build up enough momentum to escape from local regions of previous environment instances.

All four proposed classifiers, therefore, have taken steps to address the concept drift requirement. QPSO-WD and QPSO-WE, however, employ more effective measures for handling concept drift than BP-WD and BP-WE.

7.5.6 Random dynamics requirement

BP-WD and BP-WE only partially adhere to the random dynamics requirement, because evidence in [94] suggests that BP can handle only some types of dynamic environments. Regularisation might also be able to assist in this regard.

The QPSO algorithm has been shown to be effective in dynamic environments with various levels of spatial and temporal severities [6][52][94]. Harrison *et al.* [52] shows that the effectiveness of the QPSO algorithm across quasi-static, progressive, abrupt

and chaotic environments is greater when using the linear decreasing distribution than when using the uniform distribution. The QPSO-WD and QPSO-WE implementations therefore make use of the linear decreasing distribution for the purpose of this study.

All four proposed classifiers have taken steps to address the random dynamics requirement. QPSO-WD and QPSO-WE, however, employ more effective measures for handling the random dynamics of SDCPs than BP-WD and BP-WE.

7.5.7 Online learning requirement

Each of the proposed classifiers learns from every pattern that comes through a data stream. The proposed classifiers, therefore, adhere to the online learning requirement.

7.5.8 High speed data streams requirement

The combination of the low complexity ReLU activation function and summation units makes the proposed classifiers significantly faster at processing patterns and learning than FFNNs classifiers using other combinations of bounded activation functions, e.g. sigmoid, and neuron unit types, e.g. product units [71][115].

The computational complexity of QPSO-WD and QPSO-WE can, however, exceed that of BP-WD and BP-WD, if too many particles are used. Various studies have suggested the use of 30 particles as a good swarm size that balances performance and computational complexity [33][52][98][115]. Hence, both QPSO-WD and QPSO-WE use 30 particles in this study.

Because of the concept drift requirement and low model complexity requirement, some form of architecture selection strategy is required for FFNN streamed data classifiers. Construction and pruning architecture selection algorithms add additional computational complexity to the learning algorithm, because of their explicit growing or shrinking operations [79]. Regularisation, however, requires no extra computational power to be spent on explicit growing and shrinking operations.

The proposed classifiers, therefore, have taken reasonable steps to keep computational complexity as low as possible. In extremely high speed environments these steps, however, may not be sufficient.

7.5.9 One-pass requirement

Each of the proposed classifiers processes every data point that comes through a data stream exactly once, because the maximum number of epochs per training set is one, $Max_{n_e} = 1$, and the size of the training set is always one, $|D_t| = 1$. Thus, the proposed classifiers adhere to the one-pass requirement.

7.5.10 Limited number of tunable control parameters requirement

BP-WD and BP-WE respectively have three and four tunable control parameters. On the other hand, QPSO-WD and QPSO-WE respectively have seven and eight tunable control parameters. The five control parameters ω, c_1, c_2, n_p and d, however, are fixed for the purpose of the study. QPSO-WD and QPSO-WE, thus, have only two and three tunable control parameters, respectively.

Self-adaptive learning algorithms, i.e. learning algorithms that tune their control parameters on-the-fly, would be the ultimate solution for this requirement. None of the proposed classifiers, however, considers self-adaptation for the following four reasons:

1. Self-adaptive approaches generally use additional optimisation algorithms to tune control parameters. These additional optimisation algorithms tend to have problem dependent control parameters. This creates an infinite, recursive need for self-adaptive algorithms. Self-adaptation that use optimisation algorithms is thus not an option for real-world streamed data problem (SDP), unless some creative way to deal with this recursive need is proposed.

2. Current self-adaptive approaches introduce additional computational and implementation complexity [33][53]. If too much computational complexity is added, then the high speed requirement would be violated.

3. Harrison *et al.* [53] argued that current self-adaptive algorithms for PSOs are inadequate, because they waste time by searching ranges that are ineffective or violate the theoretically derived convergence conditions for the PSO control parameters.

4. Pamparà and Engelbrecht [90] recently proposed an efficient self-adaptive QPSO, that adapts the radius control parameter by using the maximum swarm diversity of the quantum sub-swarm and standard sub-swarm. However, the algorithm does not adapt any other control parameters. The effect that regularisation will have on the performance of the self-adaptive algorithm is also unknown. Because the primary purpose of this thesis is to investigate the use of regularisation in FFNN stream data classifiers, the algorithm by Pamparà and Engelbrecht [90] is not considered by the study.

The four proposed classifiers, therefore, adhere to the limited number of tunable control parameters requirement. Under the consideration that five of the control parameters for both QPSO-WD and QPSO-WE are fixed in this study, QPSO-WD and QPSO-WE adhere more to the requirement than BP-WD and BP-WE.

7.5.11 Maintain low model complexity requirement

In the context of the proposed classifiers, model complexity refers to the structural complexity of their FFNN architectures. Activation functions used in FFNN allow decision boundaries to be mapped by a set of parametrised mathematical functions. The more similar the curve of the activation function is to the shape of the decision boundaries, the less activation functions are required. This allows FFNNs to have a high information capacity at relatively low levels of model complexity [2][33][71][74][96]. The proposed classifiers makes use of FFNNs, therefore, they do not necessarily require high amounts of model complexity to be able classify accurately.

Regularisation does not change the model complexity of the proposed classifiers, but it does change their effective model complexity, i.e. effective structural complexity (refer to Section 4.3.3). The mathematics behind summation units in the proposed classifiers allows regularisation to deactivate or activate individual synapses, because zero weights make the relevant input signals have no effect on net input signal. This is not the case for product units where a single zero weight would cause the net input signal to be zero. The proposed classifiers thus are able to maintain their effective model complexity at synapse level. Furthermore, the effective model complexity can be converted into actual model complexity through the use of a suitable pruning algorithm, if need be [31]. The

proposed classifiers, therefore, have taken reasonable steps to adhere to the requirement of maintaining low model complexity.

7.5.12 Robustness requirement

None of the proposed classifiers provides a way to prevent noisy patterns from effecting the model, such as pattern filtering using data-validation or cross-validation [75][110][124]. Regularisation, however, enables the four proposed classifiers to prevent the memorisation of noise by driving unnecessary weights to zero [50].

To be able to prevent the memorisation of noise, regularisation needs to know which patterns are valid and which are noise. To distinguish between the patterns, regularisation assumes that the majority of patterns are valid, because a weight required by a valid pattern will have a smaller training error on average, than the training error of weight of the same value but required by a noisy pattern [50][110][118]. Regularisation, therefore, fails to handle noise in the case where there are more erroneous patterns than valid patterns, because the erroneous patterns will be used to determine the training error in the majority of times [50][110].

The four proposed classifiers thus have the ability to tolerate, at least, low levels of noise in data streams.

7.5.13 Fault tolerance requirement

The proposed classifiers do not adhere to the fault tolerance requirement, because the design of the classifiers delegates this requirement to the underlying software and hardware. This is a reasonable expectation as most mature programmable computation systems provide mechanisms to error correct and handle faults occurring at a hardware level.

7.5.14 Conclusion

The above discussions show that the proposed classifiers are able to handle the issues of local optimum trapping and saturation that effect FFNNs (refer to Sections 4.3.9 and 4.4.2). The above discussions also show that the proposed classifiers meet the eleven

design requirements that need to be addressed by streamed data classifiers (refer to Section 5.2). Several important points were highlighted by the discussions, as follows:

1. QPSO-WD and QPSO-WE have a potential advantage over BP-WD and BP-WE, because their learning algorithms are dynamic.

2. Under the consideration that five of the control parameters for both QPSO-WD and QPSO-WE are fixed in this study, QPSO-WD has fewer tunable control parameters than BP-WD. The same is true for QPSO-WE and BP-WE.

3. BP-WD and BP-WE have a learning algorithm speed advantage over QPSO-WD and QPSO-WE per epoch, making BP-WD and BP-WE more suitable for high speed data streams.

4. The proposed classifiers have the ability to tolerate low levels of noise in data streams due to their passive approach to handling noisy patterns.

5. The proposed classifiers need to be implemented on hardware and software that can handle machine faults.

7.6 Summary

This chapter proposed four regularised FFNN streamed data classifiers, i.e. BP-WD, BP-WE, QPSO-WD and QPSO-WE. All four classifiers make use of an over-parametrised ReLU summation 3-layer FFNN architecture.

BP-WD uses a BP-based WD learning algorithm. BP-WE uses a BP-based WE learning algorithm. QPSO-WD uses a QPSO-based WD learning algorithm. QPSO-WE uses a QPSO-based WE learning algorithm. Discussions on how the proposed classifiers approach the issues of saturation, local optimum trapping, and the eleven SDCP requirements were also presented.

The next three chapters present the empirical analysis of the proposed algorithms, starting with Chapter 8. Chapter 8 discusses the procedure that was followed to empirically analyse the streamed data classifiers.

Chapter 8

Empirical process

What can be asserted without evidence can be dismissed without evidence.

Christopher Hitchens (1949 – 2011)

This chapter outlines the empirical process followed to investigate the four streamed data classifiers proposed in Section 7.4.

The remainder of the chapter is organised as follows. Section 8.1 presents the set of hypotheses that were empirically investigated. Section 8.2 summarises two additional non-regularised FFNN streamed data classifiers that were used as baselines to compare the proposed classifiers to. Section 8.3 presents the SDCPs that were used to evaluate the performance of the classifiers. Section 8.4 discusses the methodology used to measure the performance of the streamed data classifiers. Section 8.5 discusses the process used to tune the control parameters of the classifiers on the different SDCPs. Section 8.6 describes how each tuned classifier was benchmarked. Section 8.7 presents the methodology followed to analyse the obtained results. Lastly, Section 8.8 summarises the chapter.

8.1 Hypotheses about classifiers

Based on the discussions in Section 7.5 and the findings of the brief literature review in Section 5.3, the following hypotheses are proposed in this thesis:

1. The proposed classifiers will outperform their non-regularised counterparts on all the SDCPs, because the non-regularised counterparts do not make provision for architecture selection.

2. QPSO-WD will outperform BP-WD on most of the SDCPs, because the QPSO-WD is more suited for handling concept drift than BP-WD.

3. QPSO-WE will outperform BP-WE on most of the SDCPs, because the QPSO-WE is more suited for handling concept drift than BP-WE.

4. The BP-WE and QPSO-WE will outperform their WD counterparts on all the SDCPs, because WE accounts for weights relevancy during regularisation.

5. The proposed classifiers will have lower effective model complexity than their non-regularised counterparts on all the SDCPs, because the proposed classifiers implement regularisation to perform architecture selection.

6. The proposed classifiers will have lower levels of saturation than their non-regularised counterparts on all the SDCPs, because regularisation has been shown to reduce saturation.

7. The performance of the proposed classifiers will not scale well with an increase in noise, because none of the proposed classifiers provide any means of filtering noisy patterns. Regularisation, however, will help to unlearn some of the noise.

8. BP-WD and BP-WE will not be able to handle the dynamic environments of the SDCPs as effectively as QPSO-WD and QPSO-WE will, because BP is a static optimisation algorithm whereas the QPSO is a dynamic optimisation algorithm.

9. QPSO-WD and QPSO-WE will be able to maintain swarm diversity when dealing with the SDCPs.

These hypotheses are empirically investigated in Chapter 9 to determine their validity.

8.2 Baseline classifiers

In addition to the four classifiers proposed in Section 7.4, two non-regularised versions were used:

- *BP-N*, which uses Algorithm 5 (presented in Section 7.2.1) to train the proposed architecture (presented in Section 7.1). BP-N has two control parameters that need to be tuned, namely α, η.
- *QPSO-N*, which uses Algorithm 6 (presented in Section 7.3.1) to train the proposed architecture (presented in Section 7.1). QPSO-N has six control parameters that need to be tuned, namely ω, c_1, c_2, n_p, d, r. In the comparisons of QPSO-N with QPSO-WD and QPSO-WE, the control parameters ω, c_1, c_2, n_p and d were fixed in QPSO-N to the same values used for QPSO-WD and QPSO-WE (refer to Section 7.5).

These two classifiers represent the non-regularised classifiers for hypothesis one, four and five, and provide a baseline for the comparisons. Note that for the remainder of this thesis, the term *BP classifiers* refers to the BP-N, BP-WD and BP-WE classifiers. Likewise, the term *QPSO classifiers* refers to the QPSO-N, QPSO-WD and QPSO-WE classifiers.

8.3 Benchmark streamed data classification problems

The empirical investigation used the five classification *problem domains* used in [94] to construct the benchmark SDCPs. Sixteen SDCPs were constructed from each of the five problem domains, using the *sliding window algorithm*. These 80 SDCPs were used to benchmark the classifiers. Note that the 80 SDCPs are referred to as the *benchmark problems* for the remainder of this thesis.

Section 8.3.1 provides the reasoning behind using the five problem domains. Section 8.3.2 discusses each of the five problem domains. Section 8.3.3 discusses how the data

sets of the five problem domains were pre-processed. Section 8.3.4 discusses the sliding window algorithm that was used to generate the benchmark SDCPs. Section 8.3.5 presents an analysis of the environments of the benchmark problems using the method proposed in Chapter 6. Lastly, Section 8.3.6 presents an analysis of how difficult it is for a classifier to learn an environment instance of a benchmark problem. Note that the analyses conducted in sections 8.3.5 and 8.3.6 are done in an effort to supplement the empirical analysis of the classifiers by this thesis.

8.3.1 Reasons for using the five problem domains

These five problem domains were used for the reasons outlined below:

- Rakitianskaia [94] determined optimised 3-layer FFNNs architectures for each of the five problem domains. The optimal n_h found in [94] is, thus, a good basis for determining the over-parametrised architectures that were used in this thesis. This thesis used over-parametrised architectures that were obtained by multiplying the optimised n_h value with a factor of two. The number of hidden neurons used by a classifier in this thesis was two times that of the classifiers in [94]. This allowed an assessment of how effective regularisation is as an architecture selection technique for SDCPs.

- Rakitianskaia's [94] results provided a means to see whether saturation has been mitigated by the proposed classifiers, because [94] did not cater for saturation.

- The five problem domains represent a good mix of classification problems ranging from artificial to real-world classification problems. The domains further contained examples of noise, irrelevant inputs, and high and low dimensionality.

- Rakitianskaia's [94] investigation only required classifiers to handle concept drift and noise. Rakitianskaia's results could, thus, be compared with the results obtained by this thesis to see if the additional design requirements by SDCP, such as online learning, bounded memory, high speed data streams, one-pass, unbounded dataset, maintain low model complexity and random dynamics, had an effect on the performance of 3-layer FFNNs.

8.3.2 Problem domains

Five problem domains were used to generate the benchmark problems, namely the *moving hyperplane*, *dynamic sphere*, *sliding thresholds*, *SEA concepts* and *electricity pricing*. The remainder of this section elaborates on each of the five problem domains.

Moving hyperplane problem domain

The artificial moving hyperplane problem domain, or *hyperplane domain* for short, is based on the hyperplane equation, defined as

$$y = \sum_{i=1}^{I} \left(a_i z_i \right) + c \tag{8.1}$$

where a_i is the gradient of the i-th input variable, and c is an arbitrary user-selected constant [94].

The classification problem is formed by splitting y into class A and class B, as follows:

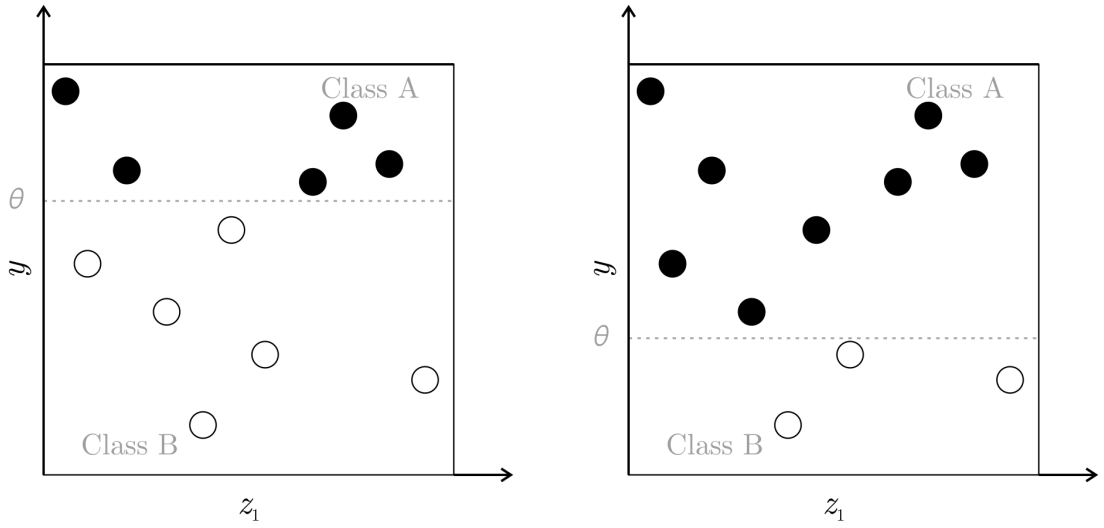
$$\text{target class} = \begin{cases} A, & \text{if } y > \theta \\ B, & \text{if } y \leq \theta \end{cases} \tag{8.2}$$

where θ is the class threshold [94]. The remainder of thesis refers to problems based on the hyperplane domain as the *hyperplane problems*. Figure 8.1 illustrates the hyperplane domain with one input variable and two classes. Figures 8.1a and 8.1b each represent an environment instance. The dashed lines represent the decision boundaries formed by θ, the black dots represent class A, and the white dots represent class B.

The hyperplane domain used in the empirical investigation was based on that used by [94]. The hyperplane problems consisted of 10 input variables and two classes.

Data for the hyperplane domain was generated as follows: 1000 input vectors were generated by uniformly sampling $z_i \in [0, 1]$. Ten environment instances, each consisting of 1000 patterns, were generated by using the 1000 input vectors and uniformly sampling $c \in [0, 1]$ and $\theta \in [0, 1]$ per environment instance. Hence, the data set consisted of 10000 patterns.

Rakitianskaia [94] used a 10-6-1 FFNN. Hence, the over-parametrised architecture used for the hyperplane problems in this study was a 10-12-1 FFNN.



(a) Environment instance 1

(b) Environment instance 2

Figure 8.1: Moving hyperplane problem domain illustration

Dynamic sphere problem domain

The artificial dynamic sphere problem domain, *sphere domain* for short, is based on the hypersphere equation, defined as

$$r^2 = \sum_{i=1}^{I} \left(z_i + c_i \right) \tag{8.3}$$

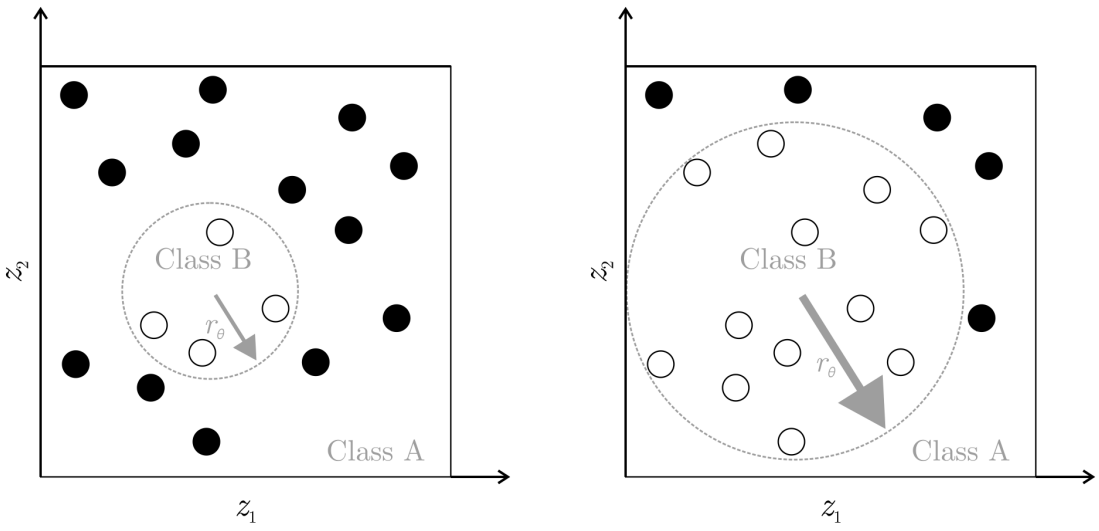
where r^2 is the squared radius of the hypersphere, and c_i is the i-th element of the hypersphere centre [94].

The classification problem is formed by splitting y into class A and class B, as follows:

$$\text{target class} = \begin{cases} A, & \text{if } r^2 > r_\theta^2 \\ B, & \text{if } r^2 \leq r_\theta^2 \end{cases} \tag{8.4}$$

where r_θ^2 is the squared radius of the threshold hypersphere [94]. The remainder of thesis refers to problems based on the sphere domain as the *sphere problems*. Figure 8.2 illustrates the sphere domain with two input variables and two classes. Figures 8.2a and 8.2b each represent an environment instance. The dashed circles represent the decision

boundaries formed by r_θ^2 and $\vec{c}$, the black dots represent class A, and the white dots represent class B.



(a) Environment instance 1

(b) Environment instance 2

Figure 8.2: Dynamic sphere problem domain illustration

The sphere domain, used in the empirical investigation, was based on that used by [94]. The sphere problems consisted of three input variables and two classes.

Data for the sphere domain was generated as follows: 1000 3-dimensional input vectors were generated by uniformly sampling $z_i \in [0, 1]$. Ten environment instances, each consisting of 1000 patterns, were generated by using the 1000 input vectors and uniformly sampling $c_i \in [0, 1]$ and $r_\theta^2 \in [0, 1]$ per environment instance. Hence, the data set consisted of 10000 patterns.

Rakitianskaia [94] used a 3-4-1 FFNN. Hence, the over-parametrised architecture used for the sphere problems in this study was a 3-8-1 FFNN.

Sliding thresholds problem domain

The artificial sliding thresholds problem domain, *thresholds domain* for short, is based on the two parallel lines, defined as

$$f_1(\vec{z}) = \theta_1 \tag{8.5}$$

$$f_2(\vec{z}) = \theta_2 \tag{8.6}$$

which are constrained by

$$\theta_1 < \theta_2 \tag{8.7}$$

where θ_1 and θ_2 are threshold constants that determine the placement of the lines f_1 and f_2, respectively [94].

The classification problem is formed by splitting the input space into three classes, as follows:

$$\text{target class} = \begin{cases} A, & \text{if } f_3(\vec{z}) \leq \theta_1 \\ B, & \text{if } f_3(\vec{z}) \geq \theta_2 \\ C & \text{otherwise} \end{cases} \tag{8.8}$$

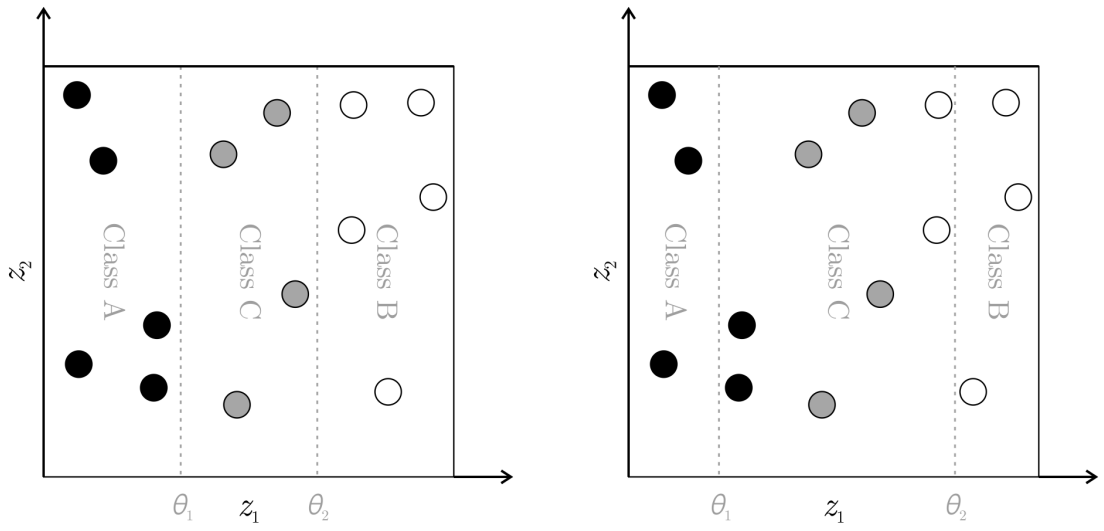
where $f_3(\vec{z})$ is a function for extracting the input variable of $\vec{z}$ that is constrained by the thresholds [94]. $f_3(\vec{z})$ causes the rest of the input variables to be irrelevant for all the patterns. That is, the sliding thresholds problem domain simulates problems with excess information, which can also be considered as a form of noise [94].

In the case of a 2-dimensional input vector, $f_3(\vec{z}) = z_1$. Therefore, the parallel lines are vertical thresholds. The remainder of thesis refers to problems based on the sliding thresholds problem domain as the *thresholds problems*.

The thresholds domain, used in the empirical investigation, was based on that used by [94]. The thresholds problems consisted of two input variables and three classes.

Data for the thresholds domain was generated as follows: 1000 2-dimensional input vectors were generated by uniformly sampling $x_i \in [0, 1]$. Ten environment instances, each consisting of 1000 patterns, were generated by using the 1000 input vectors and uniformly sampling $\theta_1 \in [0, 1]$ and $\theta_2 \in [0, 1]$, such that $\theta_1 < \theta_2$, per environment instance. Hence, the data set consisted of 10000 patterns. Figure 8.3 illustrates the thresholds domain used by this thesis. Figures 8.3a and 8.3b each represent an environment instance.

The vertical dashed lines represent the decision boundaries θ_1 and θ_2, the black dots represent class A, the white dots represent class B, and the grey dots represent class C.



(a) Environment instance 1

(b) Environment instance 2

Figure 8.3: Sliding thresholds problem domain illustration

Rakitianskaia [94] used a 2-3-3 FFNN. Hence, the over-parametrised architecture used for the thresholds problems in this study was a 2-6-3 FFNN.

SEA concepts problem domain

The artificial SEA concepts problem domain [94], *SEA domain* for short, is based on the 3-dimensional space defined as

$$y = z_1 + z_2 \tag{8.9}$$

The classification problem is formed by splitting y into class A and class B, as follows:

$$\text{target class} = \begin{cases} A, & \text{if } y \leq z_3 \\ B, & \text{if } y > z_3 \end{cases} \tag{8.10}$$

where the class threshold is the third input variable of the SEA problem [94]. The remainder of thesis refers to problems based on the SEA domain as the *SEA problems*.

The SEA domain, used in the empirical investigation, was based on that used by [94]. The SEA problems consisted of three input variables and two classes.

Data for the SEA domain was generated as follows: 10000 3-dimensional input vectors were generated by uniformly sampling $z_1 \in [0, 1]$ and $z_2 \in [0, 1]$. Four environment instances were generated by dividing the 10000 input vectors into four blocks, and assigning one of the values in $[7, 8, 9, 9.5]$ to z_3 of each block. Hence, the data set consisted of 10000 patterns. Figure 8.4 illustrates the SEA domain used by this thesis. Each block represents an environment instance. The dashed curves represent the decision boundary based on z_3, the black dots represent class A, and the white dots represent class B.

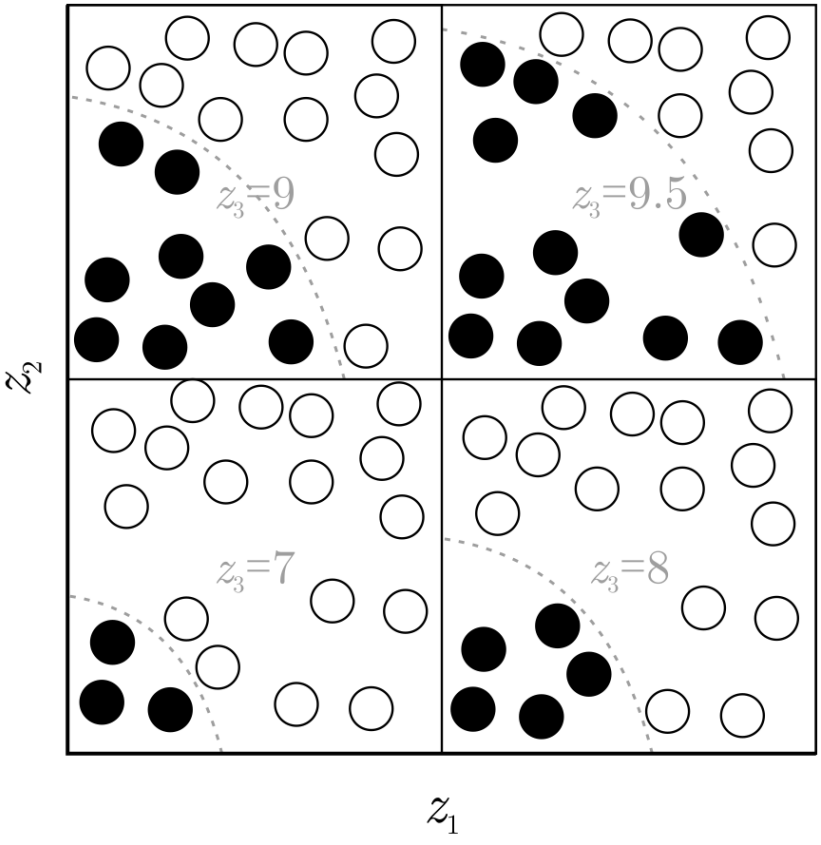


Figure 8.4: SEA concepts problem domain illustration

Furthermore, [94] injected 10% noise into each environment instance by randomly changing 10% of the pattern targets. SEA problems simulated noisy SDCPs, whose exact input vectors almost never repeated. This problem is included to test hypothesis seven.

Rakitianskaia [94] used a 3-4-1 FFNN. Hence, the over-parametrised architecture used for the SEA problems in this study was a 3-8-1 FFNN.

Electricity pricing problem domain

The real-world electricity pricing problem domain, *electricity domain* for short, is based on the electricity market in the state of New South Wales, Australia [94].

The electricity price at any particular moment is based on supply and demand, as is the case with any free-market. There are various factors that influence the supply of and demand for electricity ranging from number of end-users and weather, to production levels [94]. The problem domain consisted of six input variables and two classes [94].

The data set of the electricity pricing problem domain consisted of 27552 patterns. Each input vector represented six parameters that potentially influence the price of electricity [94]. The input vectors were generated by recording the six parameters every half an hour for the period 7 May 1996 to 5 December 1998 [94].

Each pattern has one of two classes as the target class. Class A indicates that the price at the time that the input vector was recorded is higher than the moving average price over the last 24 hours. On the other hand, class B indicates that the current price is lower than the 24 hour moving average price [94]. The aim of the problem is to predict whether the electricity price will go up or down based on the six parameters.

The remainder of the thesis refers to problems based on the electricity domain as the *electricity problems*. Rakitianskaia [94] used a 6-6-1 FFNN. Hence, the over-parametrised architecture used for the electricity problems in this study was a 6-12-1 FFNN.

8.3.3 Data preparation

The raw data sets of each of the five problem domains were pre-processed to ensure that each of the data sets were consistent and compatible with the classifiers used. Pre-processing was done using the three-step process outlined below.

Step 1: Encode non-numerical class labels to numerical class labels

The classifiers produce binary encoded outputs [91]. The class labels, however, are non-numerical. The labels, therefore, first encoded into to a numerical representation. Each class label was encoded to a positive integer that represented the order of occurrence in the dataset. For example, the class label that occurred in the first pattern was encoded to 0 for all of its instances in the remaining patterns. The next class label was encoded to 1 for all of its instances.

Step 2: Encode numerical class labels into binary classification target vectors

Next, the numerical class labels were encoded into binary classification target vectors. If there are two classes, then binary encoding is used [71][94]. In which case the target vector has one binary element that indicates which of the two classes the pattern belongs to [71].

If there are more than two classes, then the one-hot encoding scheme is used. In which case the target vector has one binary element per class [51]. Each binary element indicates whether or not the pattern belongs to a particular class [51][71]

Step 3: Scale input vector values

All input values were scaled to the range $[0, 1]$. Scaling was performed through the use of the linear min-max scaling technique, as follows [33][115]:

$$v_s = \left[\frac{v_u - v_{u,min}}{v_{u,max} - v_{u,min}}\right]\left[v_{s,max} - v_{s,min}\right] + v_{s,min} \tag{8.11}$$

where $v_s \in [v_{s,min}, v_{s,max}]$ is the scaled value of the original value $v_u \in [v_{u,min}, v_{u,max}]$.

Scaling was done for the following two reasons:

1. Van Wyk and Engelbrecht [115] concluded that scaled data sets resulted in better performance than unscaled data sets when using ReLU FFNNs trained by PSO weights adjustment algorithms.

2. Engelbrecht [33] recommended that the range of the inputs for a FFNN should be in the active range of the activation functions. Rectified linear activation functions,

however, have a left-bounded active range, i.e. $[0, \infty)$, which is not compatible with Equation (8.11). An upper bound of one was chosen to prevent the input values from causing premature saturation in the hidden layer.

8.3.4 Construction of benchmark problems

The SDCPs were constructed using the sliding window algorithm, presented in this section. The sliding window algorithm is commonly used to construct a dynamic classification problem from the data set of a problem domain [45][92]. Before explaining how the sliding window algorithm was used to create SDCPs, this section explains how the algorithm is used to construct dynamic classification problems.

Algorithm 7 presents the sliding window algorithm [94]. Algorithm 7 makes use of the parameters w_m, w_f and w_s, which are defined as follows:

- The *window size*, w_m, is a positive non-zero parameter that controls the number of patterns in each data window. The larger the value of w_m, the more patterns there are in each data window.
- The *window frequency*, w_f, is a positive non-zero parameter that controls the number of times each instance of the sliding window is repeated [94]. The larger the value of w_f, the more times each instance of sliding window is iterated over by the classifier.
- The *window step size*, w_s, is a positive non-zero parameter that controls the rate at which the sliding window moves [94]. The larger the value of w_s, the more patterns the algorithm skips during a step of the sliding window.

Algorithm 7 creates n_w sequential w_m-sized data windows by sliding a w_m-sized sliding window across the data set of the problem domain. The sliding window is slid from the start of the data set of problem domain, by moving the sliding window by w_s patterns, until there are no more patterns to fill the sliding window. Each data window created by the sliding window represents an environment instance. Each sliding window is repeated w_f times, before moving the sliding window foreword to simulate the time period an environment instance is available for.

Algorithm 7 Sliding windows algorithm

Input: D, w_m, w_f, w_s
  Initialise an empty 2D array of patterns, *data windows*
  $p = 0$
  while $p + w_m \leq |D|$ **do**
    Initialise empty array of patterns, *sliding window*
    for $i = 0$ to w_m **do**
      Append pattern D_{p+i} to *sliding window*
    end for
    for $i = 0$ to w_f **do**
      Append the *sliding window* to *data windows*
    end for
    $p = p + w_s$
  end while
  return *data windows*

Figure 8.5 illustrates the flattened sequence of data windows constructed by Algorithm 7 with the parameters $w_m = 2$, $w_s = 3$ and $w_f = 3$.

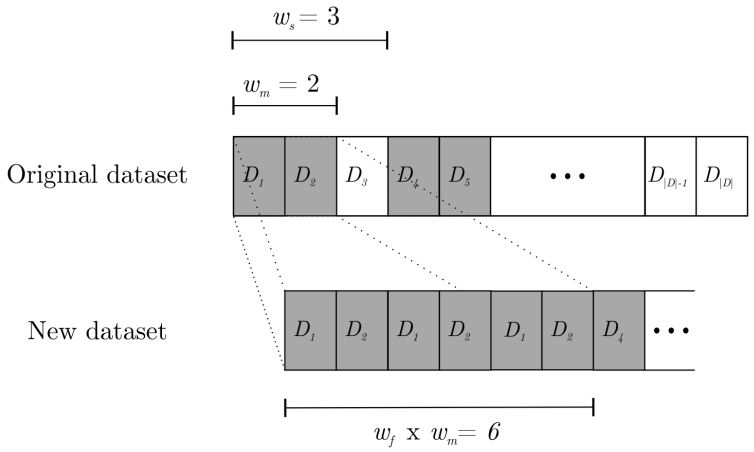


Figure 8.5: Illustration of sliding window algorithm

Furthermore, the total number of data windows, n_w, that can be created from an

arbitrary data set, D, is calculated as follows [94]:

$$n_w = w_f \left(1 + \frac{|D| - w_m}{w_s}\right) \tag{8.12}$$

SDCPs are a subclass of dynamic classification problems, where each data window consists of one pattern due to the one-pass requirement. Algorithm 7 can be used to construct SDCPs by simply setting the window size, w_m, to 1 and flattening the 2-dimensional array of data windows. Note that in this case each data window no longer represents an environment instance but just a pattern.

The sixteen benchmark problems for each benchmark problem domain were generated using this approach. The values for window frequency (w_f) and window step size (w_s) were selected in combinatorial fashion, using the value set $\{1, 2, 5, 10\}$.

These parameter choices caused the sliding window algorithm to construct data streams by duplicating and skipping patterns in the data set of the problem domain, as follows:

1. The smaller w_f, the less times the same pattern was duplicated and vice versa. If $w_f = 1$, then the patterns of the problem domain were not duplicated.

2. The smaller w_s, the larger the sample of patterns from the data set of the problem domain was and vice versa. If $w_s = 1$, then all the patterns of the problem domain were present in the data stream.

Table 8.1 labels the sixteen benchmark problems generated per problem domain according to their window frequency (w_f) and window step size (w_s) values. These labels were used throughout the remainder of this thesis. The four extreme benchmark problems were at the four corners of table, i.e A4, D1, D4 and A1.

According to Equation (8.12), A4 problems had the longest data stream and D1 problems had the shortest data stream. D4 problems skipped and duplicated the patterns most, and had the same number of patterns as the A1 problems. A1 problems were special cases in the sense that they converted the data set of the problem domain into a stream without altering the occurrence of patterns. An A1 problem is, thus, referred to as the *streamed benchmark problem* of its problem domain for the remainder of this thesis.

Table 8.1: Sliding window parameter configurations of the sixteen benchmark problems generated per problem domain.

		w_f			
		1	*2*	*5*	*10*
w_s	*1*	A1	A2	A3	A4
	2	B1	B2	B3	B4
	5	C1	C2	C3	C4
	10	D1	D2	D3	D4

8.3.5 Problem environment analysis

The environments of the 80 benchmark problems were pre-analysed using the method proposed in Chapter 6 to determine:

- The characteristics of the dynamic environments of the benchmark problems.
- The relationship between the parameters of the sliding window algorithm and the characteristics of the dynamic environment they generated.

The insight gained by this analysis was used to supplement the discussions of the empirical investigation of the proposed classifiers.

Analysis process

The following measures were recoded per benchmark problem by this analysis: the number of windows, n_w, the number of environment instances, $|I_{b_w}|$, the spatial severity, Θ_{b_w}, the temporal severity, τ_{b_w}, the normalised spatial severity, Θ'_{b_w}, the normalised temporal severity, τ'_{b_w}, the dynamic environment classification, Λ_{b_w}, and the dominance of the dynamic environment classification, ζ_{b_w}. Note that because w_m was 1 for the benchmark problems, the number of windows, n_w, is equal to the number of patterns, $|D|$.

Algorithm 4 presented in Chapter 6, was used to extract the actual environment instances from the benchmark problems. Algorithm 4 uses a binning width (b_w) to help cater for SDCPs containing noise and input vectors that do not reoccur (refer to Section

6.2). The binning width for the hyperplane, sphere and threshold problems was set to 0, because all the input vectors occurred more than once in each problem, and none of the problems contained noise.

The SEA problems, however, contained noise and not all the input vectors repeated. The electricity problems were assumed to contain noise, because of their real-world domain. Furthermore, a scan through the data set of the electricity domain revealed that input vectors sometimes only occurred once in the dataset. Hence, the SEA and electricity problems had their binning width optimised per benchmark problem. The binning width optimisation was done in the following manner:

- Because a change in the window frequency does not change the inputs vectors in the data set of the problem domain, the data sets of A1, B1, C1 and D1 were used to find the binning widths of all the other A, B, C and D series benchmark problems. That is, the binning width found for A1 of a particular problem was used for A2, A3, and A4 of the same problem domain.

- All values for the binning width in the range $(0, 1)$ were tested, using increments of 0.00001. The range $(0, 1)$ was used for the following reasons:

  - If $b_w < 0$, then Algorithm 4 would invert the signs of the pattern values, which is not desired.
  - If $b_w = 0$, then Algorithm 4 would use the exact patterns, which was not desired.
  - If $b_w \geq 1$, then all the patterns would be represented by one input vector due to all the values of the input vectors being in the range $[0, 1]$.

- The binning width value closest to zero, that resulted in each input vector repeating at least once during the data stream, was considered optimal, because it resulted in the most number of unique input vectors (refer to Section 6.2).

The normalised values for spatial and temporal severity, i.e. Θ'_{b_w} and τ'_{b_w}, were calculated on two levels:

- The *problem domain level*, which used the maximum of the particular severity found in the set of 16 benchmark problems per problem domain. This level was

used to analyse the effect of window step size and window frequency on the data sets of the problem domains.

- The *overall level*, which used the maximum of the particular severity found in the set of 80 benchmark problems. This level was used to compare the severities of all the benchmark problems against each other.

Analysis discussion

Tables 8.2 to 8.6 present the values of the environment analysis measures for each of the benchmark problems in each of the five problem domains. Note that values for Θ'_{b_w} and τ'_{b_w} were calculated on a problem domain level. To visualise the data in these tables the following figures were constructed:

Figures 8.6a to 8.6e plot the normalised temporal and spatial severities for each of the five problem domains. Each point is labelled according to the benchmark problem labels in Table 8.1.

Figure 8.7 illustrates the relationships between the window frequency and window step size parameters of Algorithm 7, and the spatial and temporal severities. The data points, in each of the four graphs, represent the average value of the severity measure for the particular values of the sliding window parameters in question.

The results in Tables 8.2 to 8.6 and Figures 8.6a to 8.7d show that both the spatial and temporal severity of the data set of a problem domain were generally affected by the choices of window frequency and window step size.

Window frequency (w_f) had an inverse linear relationship with the temporal severity, but had no effect on spatial severity. On the other hand, window step size (w_s) had an inverse exponential relationship with temporal severity, and a decreasing exponential relationship with spatial severity. The electricity problems mostly adhered to this observation, except for B1, B2, B3 and B4, which experienced a linear relationship between w_s and spatial severity. This was due to the environment instances decreasing while the spatial severity increased.

The temporal severity of non-streamed benchmark problems varied about the temporal severity of the streamed benchmark problem, whereas the spatial severity of non-streamed benchmark problems was always lower than the spatial severity of streamed

Table 8.2: Environment analysis of the hyperplane problems

Name	n_w	b_w	$\lvert I_{b_w}\rvert$	Θ_{b_w}	τ_{b_w}	Θ'_{b_w}	τ'_{b_w}	Λ_{b_w}	ζ_{b_w}
A1	10000	0	10	482.900	1000.000	1.000	0.900	Chaotic	0.881
A2	20000	0	10	482.900	2000.000	1.000	0.800	Chaotic	0.764
A3	50000	0	10	482.900	5000.000	1.000	0.500	Chaotic	0.437
A4	100000	0	10	482.900	10000.000	1.000	0	Abrupt	1.000
B1	5000	0	10	242.700	500.000	0.503	0.950	Chaotic	0.407
B2	10000	0	10	242.700	1000.000	0.503	0.900	Chaotic	0.370
B3	25000	0	10	242.700	2500.000	0.503	0.750	Chaotic	0.245
B4	50000	0	10	242.700	5000.000	0.503	0.500	Chaotic	0.003
C1	2000	0	10	92.700	200.000	0.192	0.980	Progressive	0.762
C2	4000	0	10	92.700	400.000	0.192	0.960	Progressive	0.749
C3	10000	0	10	92.700	1000.000	0.192	0.900	Progressive	0.702
C4	20000	0	10	92.700	2000.000	0.192	0.800	Progressive	0.608
D1	1000	0	10	47.600	100.000	0.099	0.990	Progressive	0.877
D2	2000	0	10	47.600	200.000	0.099	0.980	Progressive	0.870
D3	5000	0	10	47.600	500.000	0.099	0.950	Progressive	0.848
D4	10000	0	10	47.600	1000.000	0.099	0.900	Progressive	0.802

benchmark problems. The B series electricity problems were an exception to this observation.

Out of all 16 sliding window configurations, A4 experienced the lowest temporal severity, and resulted in the most abrupt environment out of the 16 sliding window configurations. D1 experienced the highest spatial severity, and resulted in the most progressive environment out of the 16 sliding window configurations. Note that in the electricity domain, D1 was classified as chaotic. The A1 problems were the most chaotic, except for the electricity domain where B1 was the most chaotic and A1 was the second most chaotic. Note that none of the benchmark problems were classified as quasi static.

In terms of environment dominance (ζ_{b_w}), artificial B4 problems, however, tended to have ζ_{b_w} values near zero. In the case of the electricity domain, C4 had a ζ_{b_w} value near zero. Furthermore, the distribution of ζ_{b_w} values indicates that almost 45% of the problems environments were dominated by mainly two types of environments, namely chaotic and progressive.

Figure 8.8 plots the temporal and spatial severity normalised on the overall level

Table 8.3: Environment analysis of the sphere problems

Name	n_w	b_w	$\lvert I_{b_w}\rvert$	Θ_{b_w}	τ_{b_w}	Θ'_{b_w}	τ'_{b_w}	Λ_{b_w}	ζ_{b_w}
A1	10000	0	10	477.800	1000.000	1.000	0.900	Chaotic	0.881
A2	20000	0	10	477.800	2000.000	1.000	0.800	Chaotic	0.764
A3	50000	0	10	477.800	5000.000	1.000	0.500	Chaotic	0.437
A4	100000	0	10	477.800	10000.000	1.000	0	Abrupt	1.000
B1	5000	0	10	240.700	500.000	0.504	0.950	Chaotic	0.408
B2	10000	0	10	240.700	1000.000	0.504	0.900	Chaotic	0.371
B3	25000	0	10	240.700	2500.000	0.504	0.750	Chaotic	0.246
B4	50000	0	10	240.700	5000.000	0.504	0.500	Chaotic	0.004
C1	2000	0	10	97.100	200.000	0.203	0.980	Progressive	0.749
C2	4000	0	10	97.100	400.000	0.203	0.960	Progressive	0.736
C3	10000	0	10	97.100	1000.000	0.203	0.900	Progressive	0.690
C4	20000	0	10	97.100	2000.000	0.203	0.800	Progressive	0.597
D1	1000	0	10	49.000	100.000	0.103	0.990	Progressive	0.872
D2	2000	0	10	49.000	200.000	0.103	0.980	Progressive	0.866
D3	5000	0	10	49.000	500.000	0.103	0.950	Progressive	0.843
D4	10000	0	10	49.000	1000.000	0.103	0.900	Progressive	0.798

for the 80 benchmark problems. The 80 benchmark problems did not provide for one problem that could be regarded as quasi-static. This should not be a problem for the investigation, because quasi-static environments will have the least impact on classifier performance (refer to Section 2.2.2). Furthermore, the benchmark suite consisted of very few abrupt problems. However, the benchmark suite consisted of many of the edge-case problems and progressive problems.

The hyperplane, sphere, and thresholds problems were mostly spread across the abrupt, progressive and chaotic regions of the dynamic environment classification scheme by [25]. Figure 8.8 clearly shows that the hyperplane and sphere problems exhibited very similar problem environments. On the other hand, the thresholds problems showed the highest levels of spatial severity and chaotic behaviour. Thresholds problem A4 was the most abrupt problem in the benchmark suite. Thresholds problem A1 was the most chaotic problem of the benchmark suite.

The SEA and electricity problems were clustered in the more extreme regions of the progressive classification. The SEA problems tended to exhibit higher spatial severity

Table 8.4: Environment analysis of the thresholds problems

Name	n_w	b_w	$\lvert I_{b_w}\rvert$	Θ_{b_w}	τ_{b_w}	Θ'_{b_w}	τ'_{b_w}	Λ_{b_w}	ζ_{b_w}
A1	10000	0	10	581.100	1000.000	1.000	0.900	Chaotic	0.881
A2	20000	0	10	581.100	2000.000	1.000	0.800	Chaotic	0.764
A3	50000	0	10	581.100	5000.000	1.000	0.500	Chaotic	0.437
A4	100000	0	10	581.100	10000.000	1.000	0	Abrupt	1.000
B1	5000	0	10	288.358	500.000	0.496	0.950	Progressive	0.408
B2	10000	0	10	288.358	1000.000	0.496	0.900	Progressive	0.371
B3	25000	0	10	288.358	2500.000	0.496	0.750	Progressive	0.246
B4	50000	0	10	288.358	5000.000	0.496	0.500	Progressive	0.004
C1	2000	0	10	115.117	200.000	0.198	0.980	Progressive	0.755
C2	4000	0	10	115.117	400.000	0.198	0.960	Progressive	0.742
C3	10000	0	10	115.117	1000.000	0.198	0.900	Progressive	0.696
C4	20000	0	10	115.117	2000.000	0.198	0.800	Progressive	0.602
D1	1000	0	10	56.993	100.000	0.098	0.990	Progressive	0.877
D2	2000	0	10	56.993	200.000	0.098	0.980	Progressive	0.871
D3	5000	0	10	56.993	500.000	0.098	0.950	Progressive	0.849
D4	10000	0	10	56.993	1000.000	0.098	0.900	Progressive	0.802

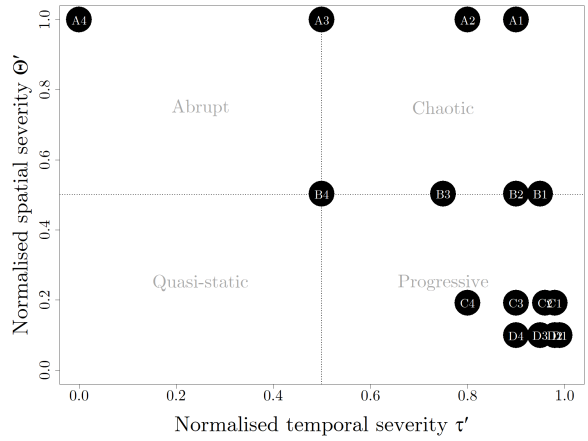
than the electricity benchmark problems. The electricity problems exhibited the highest levels of temporal severity. Electricity D1 was the most progressive problem in the benchmark suite. The main cause for Algorithm 4 detecting so many environment changes in the SEA and electricity problems was noise.

Table 8.5: Environment analysis of the SEA problems

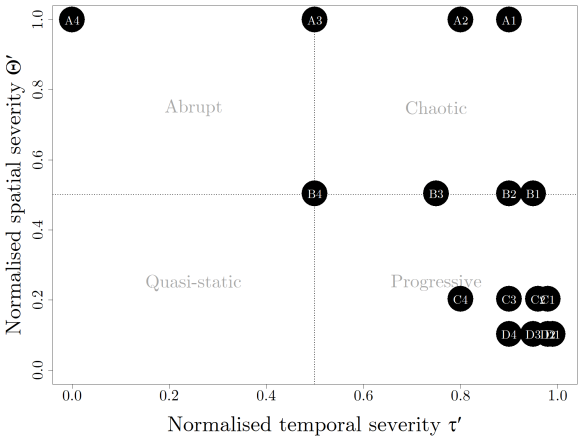
Name	n_w	b_w	$\lvert I_{b_w}\rvert$	Θ_{b_w}	τ_{b_w}	Θ'_{b_w}	τ'_{b_w}	Λ_{b_w}	ζ_{b_w}
A1	10000	0.09098	88	22.795	113.636	1.000	0.901	Chaotic	0.882
A2	20000	0.09098	88	22.795	227.273	1.000	0.801	Chaotic	0.765
A3	50000	0.09098	88	22.795	568.182	1.000	0.500	Chaotic	0.437
A4	100000	0.09098	88	22.795	1136.364	1.000	0	Abrupt	1.000
B1	5000	0.12501	82	12.951	60.976	0.568	0.947	Chaotic	0.473
B2	10000	0.12501	82	12.951	121.951	0.568	0.893	Chaotic	0.434
B3	25000	0.12501	82	12.951	304.878	0.568	0.732	Chaotic	0.296
B4	50000	0.12501	82	12.951	609.756	0.568	0.464	Abrupt	0.104
C1	2000	0.16664	49	8.980	40.816	0.394	0.965	Progressive	0.526
C2	4000	0.16664	49	8.980	81.633	0.394	0.929	Progressive	0.501
C3	10000	0.16664	49	8.980	204.082	0.394	0.821	Progressive	0.416
C4	20000	0.16664	49	8.980	408.163	0.394	0.641	Progressive	0.247
D1	1000	0.20074	35	6.229	28.571	0.273	0.976	Progressive	0.666
D2	2000	0.20074	35	6.229	57.143	0.273	0.951	Progressive	0.650
D3	5000	0.20074	35	6.229	142.857	0.273	0.875	Progressive	0.592
D4	10000	0.20074	35	6.229	285.714	0.273	0.749	Progressive	0.476

Table 8.6: Environment analysis of the electricity problems

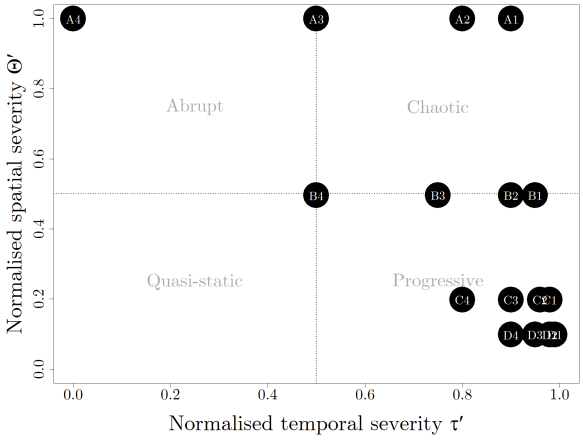
Name	n_w	b_w	$\lvert I_{b_w}\rvert$	Θ_{b_w}	τ_{b_w}	Θ'_{b_w}	τ'_{b_w}	Λ_{b_w}	ζ_{b_w}
A1	27552	0.50001	3161	1.610	8.716	0.894	0.910	Chaotic	0.804
A2	55104	0.50001	3161	1.610	17.432	0.894	0.809	Chaotic	0.698
A3	137760	0.50001	3161	1.610	43.581	0.894	0.506	Chaotic	0.369
A4	275520	0.50001	3161	1.610	87.162	0.894	0	Abrupt	0.873
B1	13776	0.50001	2000	1.802	6.888	1.000	0.932	Chaotic	0.918
B2	27552	0.50001	2000	1.802	13.776	1.000	0.852	Chaotic	0.824
B3	68880	0.50001	2000	1.802	34.440	1.000	0.612	Chaotic	0.555
B4	137760	0.50001	2000	1.802	68.880	1.000	0.212	Abrupt	0.750
C1	5511	0.98080	1776	1.069	3.103	0.593	0.976	Chaotic	0.519
C2	11022	0.98080	1776	1.069	6.206	0.593	0.940	Chaotic	0.495
C3	27555	0.98080	1776	1.069	15.515	0.593	0.832	Chaotic	0.411
C4	55110	0.98080	1776	1.069	31.030	0.593	0.651	Chaotic	0.244
D1	2756	0.98080	1174	1.058	2.348	0.587	0.984	Chaotic	0.518
D2	5512	0.98080	1174	1.058	4.695	0.587	0.957	Chaotic	0.501
D3	13780	0.98080	1174	1.058	11.738	0.587	0.875	Chaotic	0.440
D4	27560	0.98080	1174	1.058	23.475	0.587	0.739	Chaotic	0.322



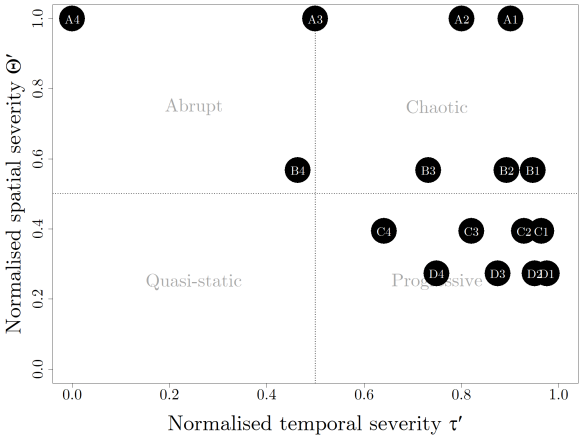
(a) Hyperplane problems



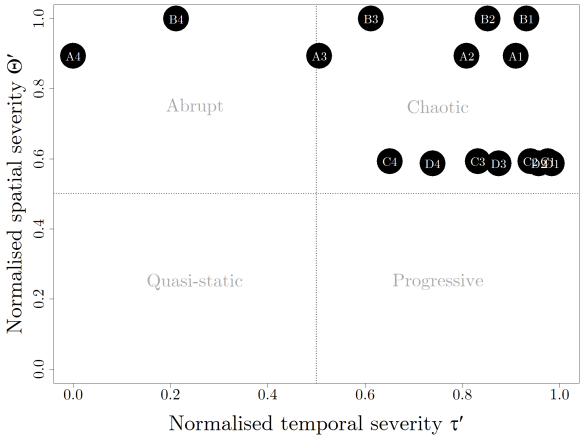
(b) Sphere problems



(c) Thresholds problems

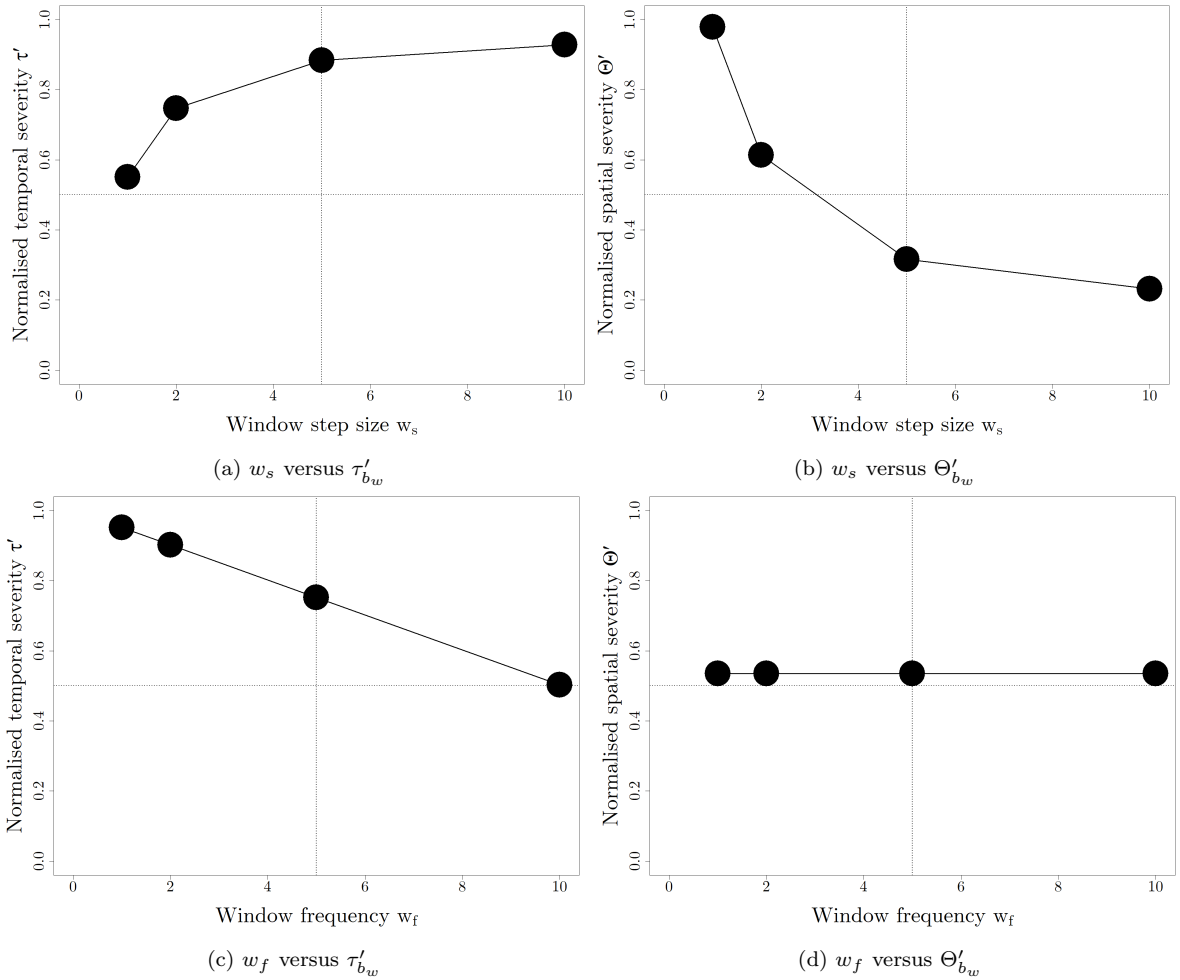


(d) SEA problems



(e) Electricity problems

Figure 8.6: Benchmark problems' environment severity levels



(a) w_s versus τ'_{b_w}

(b) w_s versus Θ'_{b_w}

(c) w_f versus τ'_{b_w}

(d) w_f versus Θ'_{b_w}

Figure 8.7: Severity trends for streamed data classification problems

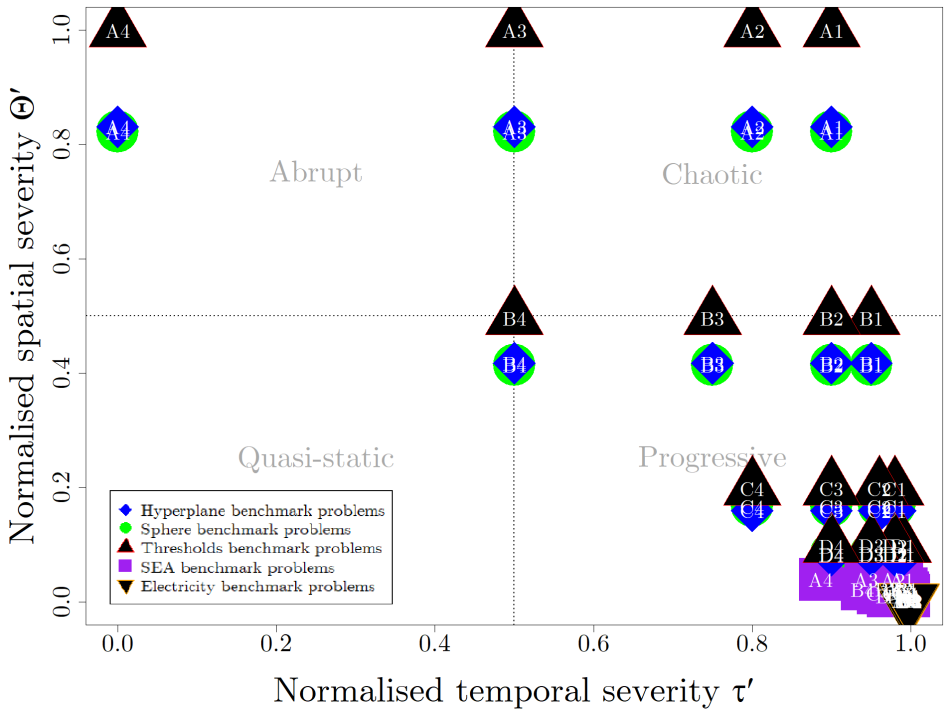


Figure 8.8: Overall benchmark problems' environment severity levels

8.3.6 Problem difficulty analysis

The problem environment classification scheme, i.e. quasi-static, abrupt, progressive and chaotic, qualifies the level of adaptability required by the classifier. That is, the classification scheme indicates the amount of adaptation required by the model and the time frame in which the adaptation must be performed by the classifier. The classification scheme, however, does not provide insight into how difficult it is for the classifier to learn the environment instances of a problem. This thesis refers to this concept as *problem difficulty*.

Classifying the problem difficulty of the benchmark problems

The difficulty of a SDCP can be quantified in two dimensions [2][39][69]:

1. **Data availability**, which describes the number of unique input–target pairs there are in an environment instance. The more unique input–target pairs there are, the more is known about the classification problem. Thus, the less the classifier has to generalise. The reverse is also true. Data availability, therefore, provides an indication of the generalisation ability required by classifiers to solve the problem.

2. **Pattern re-occurrence**, which describes the number of times patterns repeat during an environment instance. The more times a classifier sees an input–target pair, the more time the classifier has to learn from the example. The reverse is also true. Pattern re-occurrence, therefore, provides an indication of the rate at which classifiers need to learn examples.

Because the window step size and window frequency parameters control the skipping and repeating of input–target pairs for SDCPs, they can be used to define the problem difficulty, as follows:

- The window step size estimates the data availability dimension. The larger the value of w_s is, the less data is available and therefore the more difficult the problem is. The reverse is also true.

- The window frequency estimates the pattern re-occurrence dimension. The larger the value of w_f is, the more pattern instances of an input–target pair there are, and therefore the less difficult the problem is. The reverse is also true.

Four difficulty classes for SDCPs, based on data availability and pattern re-occurrence, are proposed as illustrated by Figure 8.9.

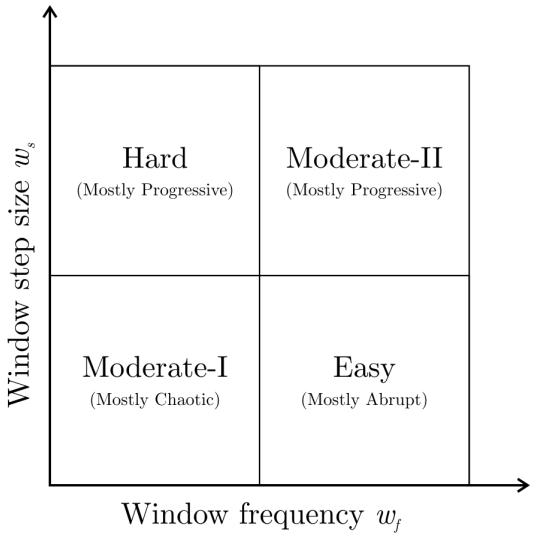


Figure 8.9: Streamed data classification problem difficulty: w_f versus w_s

- **Easy**: A low w_s and high w_f result in data streams that skip very few patterns in the data set of the problem domain, but repeats each pattern many times. Easy SDCPs, therefore, have high data availability and high pattern re-occurrence. Chances of underfitting and overfitting is relatively low for easy SDCPs. Hence, classifiers just need basic learning capabilities to handle this difficulty. Out of the 16 benchmark problems for a problem domain, A3, B3, A4 and B4 were considered easy.

- **Moderate-I**: A low w_s and low w_f result in data streams that skip very few patterns in the data set of the problem domain, and does not contain many repeats of the patterns. Moderate-I SDCPs, therefore, have high data availability and

low pattern re-occurrence. The risk of underfitting is high in moderate-I SDCPs, because the classifiers has a very short time frame in which to extract information from a pattern. The classifier needs to be able to learn patterns quickly, but does not necessarily need to generalise accurately with few examples to handle this difficulty. Out of the 16 benchmark problems for a problem domain, A1, B1, A2 and B2 were considered moderate-I.

- **Moderate-II**: A high w_s and high w_f result in data streams that skip many patterns in the data set of the problem domain, and provide many repeats of each pattern. Moderate-II SDCPs, therefore, have low data availability and high pattern re-occurrence. The risk of overfitting is high for moderate-II SDCPs, because the classifier will be exposed to many repeats of the same example. The classifier needs to be able to generalise accurately using few examples, but does not necessarily need to learn quickly to handle this difficulty. Out of the 16 benchmark problems for a problem domain, C3, D3, C4 and D4 were considered moderate-II.

- **Hard**: A high w_s and low w_f result in data streams that skip many patterns in the data set of the problem domain, and provide few repeats of each pattern. Hard SDCPs, therefore, have low data availability and low pattern re-occurrence. The risk of underfitting or overfitting is high in hard SDCPs, because the number of examples are limited and the time-frame to learn an example is very short. Classifiers need be able to learn patterns quickly and generalise accurately using a very limited number of examples, to handle this difficulty. Out of the 16 benchmark SDCPs for a problem domain, C1, D1, C2, and D2 were considered hard.

Note that the problem difficulty scheme is relative to the problem domain. Problems with the same problem difficulty class but with different domains are therefore not directly comparable, e.g., a hard problem in one domain might be easier than a easy problem in another problem domain.

Analysis discussion

Comparing the problem difficulty classifications to the environment classifications (refer to Section 8.3.5) of the benchmark problems, it was observed that hard problems

tended to be progressive most of the time. Easy problems on the other hand tended to be abrupt, and sometimes chaotic. Moderate-I problems tended to be chaotic, and moderate-II problems tended to be progressive most of the time. The difficulty of learning environment instances, therefore, does not coincide with the difficulty of adapting to environment changes.

A comparison of problem difficulty, and the temporal and spatial severities of SDCPs, it was observed that easy problems had a lower temporal severity than that of the streamed benchmark problems, but their spatial severity was similar to that of the streamed benchmark problems. Moderate-I problems were associated with spatial and temporal severities that were similar to those of the streamed benchmarks. On the other hand, moderate-II problems had lower spatial and temporal severities than that of the streamed benchmark problems. Lastly, hard problems had a lower spatial severity than that of the streamed benchmark problems, but their temporal severity was similar to that of the streamed benchmark problems. Streamed benchmark problems, therefore, represented the average case in terms of problem difficulty, and worst case in terms of problem environment.

The problem difficulty classification scheme and the findings in this analysis of the benchmark problems was used to supplement the empirical investigation of the proposed classifiers.

8.4 Performance measurement

Various aspects about the performance of the classifiers were measured in order to evaluate the hypothesis stated in Section 8.1. Section 8.4.1 discusses the approach used to measure the performance of streamed data classifiers. Sections 8.4.2 to 8.4.9 present the various performance aspects measured by this thesis.

8.4.1 Performance measuring methodology

Measuring the performance of classifiers in dynamic environments is different from measuring performance in a static environment, because the performance across multiple environments needs to be considered [98]. To address this concern, performance was

quantified using the *collective mean* approach [87]. The collective mean of a model error is defined as

$$\bar{C}_\epsilon = \frac{\sum_{t=1}^{T} \epsilon(t)}{T} \tag{8.13}$$

where T is the total number of epochs learned; and $\epsilon(t)$ is the performance measure, ϵ, for the best model found at either the end or start of epoch t, depending on the performance measure [87].

The three key advantages of the collective mean approach are [94][98]:

- The approach does not require any additional information about the environment changes, e.g. when changes occured, to interpret the value of the performance measure.
- The approach provides complete coverage of the performance measure over all environment instances. That is, the collective mean of a performance measure represents the best value average found across all environment instances.
- The approach allows classifiers using local searches, e.g. GD, to be compared to classifiers using population-based searches, e.g. PSO, because the approach takes the best model found for an epoch.

The primary disadvantage of the collective mean approach is that the approach uses averages, thus, exceptionally good performance in some epochs might cause the results to skew. This disadvantage can be mitigated if classification problems with a large number of epochs are used and/or if the trend of the performance measure is also analysed.

SDCPs introduce three additional complications for calculating performance measures during learning, as follows [45]:

1. *Sequential availability.* That is, patterns are only available one after the other. Therefore, there is no way of knowing what other patterns in the future form part of the current environment instance until they have arrived.
2. *One-pass requirement.* That is, each pattern can only be processed once. A window of an environment instance using the past patterns can, therefore, not be used to learn the classifier.

3. *Bounded memory requirement.* That is, patterns can not be stored by the learning algorithm for later use.

Performance measures for SDCPs, thus, need to be consider the following:

- Firstly, if the generalisation, validation and training sets are used to train a model, then the sets will always contain the same pattern per epoch.
- Secondly, the performance measures using these sets can only be measured before or after the model has been trained with a pattern.

One of the sets is, thus, rendered redundant.

The hold-out error method (refer to Section 4.3.2) used commonly in static and dynamic classification methods is, therefore, not applicable to SDCPs [67]. Gama *et al.* [45] suggested the use of the *predictive sequential error*, also known as the *prequential error* as an alternative to the hold-out error method. The prequential error can be calculated in two ways, namely using a the *sliding window* or a *fading-factor* [45].

The sliding window approach takes the last n patterns, and applies a performance measure to the current model using each of the n patterns [45]. The n calculated values are then averaged to form the prequential error for the epoch. The parameter n controls the smoothness of the error. The larger the value of n, the smoother, but the less reflective of the real error the performance curve becomes, and vice versa [45]. Because this approach does not adhere to the bounded memory and one-pass requirements, this approach can only be used for performance analysis of the classifiers.

The fading-factor approach accumulates the values of a performance measure into a single value by applying a weight, in the range $(0, 1)$, to the prequential sequential error value of the previous epoch. The weight is the rate at which the past error becomes less significant, i.e. fades. The weight needs to be optimised per problem [45]. The advantages of the fading-factor approach are:

- It does not violate the one-pass and bounded memory requirements [45]. This allows the fading-factor prequential error to be used to calculate generalisation and training errors during the learning of SDCPs.
- It converges to the same error value as determined by the hold-out error method [45].

The fading-factor, however, does come with its own disadvantages:

- The fading-factor approach is less interpretable than the sliding window approach from an analysis perspective, because it is not intuitive which old patterns effected the error value.
- In the context of generalisation and training errors, the fading factor approach introduces an additional problem dependent parameter when compared with the approach used by this thesis. Thus, making the classifiers less in line with the limited number of tunable control parameters requirement.

This thesis did not consider the fading factor approach due to the above disadvantages.

In light of the above discussions, this thesis adopted the following approach:

1. First, performance measures used to evaluate the training performance of a classifier were applied to the trained model(s) at the end of an epoch.
2. Second, performance measures used to evaluate the generalisation performance of a classifier were applied to the model(s) at the start of an epoch.
3. Third, it was decided that the validation set was only to be used for the purposes of performance analysis. The validation set was used to implement the sliding window prequential error. The validation set used the last 30 patterns. The value of 30 was chosen based on the idea that 30 is the minimum number of samples required to get a fair statistical sample, without excessive computational effort [56][83].

   Performance measures, based on the validation set, were applied to the trained model(s) at the end of an epoch. This, however, meant that performance measures using the validation set no longer represented an estimation of the generalisation performance, but rather how well the classifier preserved its performance over the last 30 patterns. This thesis, therefore, refers to this implementation of the validation error as the *memory error* or E_m.

   Note that, unlike Gama *et al.* [45], any performance measure was still applied to the validation set while it contained less than 30 patterns, which only occurred in the first 29 epochs.

4. Fourth, the performance analysis was done using the collective means of the performance measures.

8.4.2 Saturation performance measures

Equation (4.20) was used to measure saturation in the hidden neurons. Because the measure requires bounded activation functions, the measure had to be altered for it to work with ReLU activation functions.

An upper bounds parameter was introduced in order to bound the activation values. The value of the upper bound parameter estimates the level of activation that would cause ReLU hidden neurons to saturate the FFNN in a detrimental way. No research on what bounds would be appropriate for ReLU activation functions has been done, therefore, several upper bounds were investigated [97].

Rakitianskaia and Engelbrecht [96] argued that saturation in the output neurons was necessary for classification, but premature saturation in the hidden layers was unwanted. Thus, the smaller the activation values in the hidden neurons, the lower the chance of premature saturation [96]. Reasonable values for the upper bound are, therefore, values greater or equal to the upper bounds of the active ranges of the output neurons.

Three upper bounds, i.e. 1, 5 and 10, were chosen near to the upper bound of the active ranges of the output neurons, i.e. 1. These three parametrisations of φ_{b_w} were denoted as $\varphi_{b_w,1}$, $\varphi_{b_w,5}$, $\varphi_{b_w,10}$.

The minimum of $\varphi_{b_w,1}$, $\varphi_{b_w,5}$ and $\varphi_{b_w,10}$ was used to represent the saturation of the classifier. The minimum was used, for two reasons:

- The three measures, $\varphi_{b_w,1}$, $\varphi_{b_w,5}$ and $\varphi_{b_w,10}$, will not necessarily saturate at the same time, because of the different upper bounds.
- At the time of the investigation it was unclear which upper bound provided the most accurate measure of saturation in the hidden layers.

The binning width (b_w) was set to 0.1 in all three cases to ensure that the three different measures were consistent. The value of 0.1 also ensured that all three measures had 10 or more bins, as recommended by Rakitianskaia and Engelbrecht [97]. Equation

(4.20) also required a set of patterns to determine saturation. Thus, the following two saturation measures were used:

1. The saturation based on the generalisation set, φ_g, defined as

$$\min\{\varphi_{0.1,1}, \varphi_{0.1,5}, \varphi_{0.1,10}\}$$

   where $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$ is based on the generalisation set.

2. The saturation based on the validation set, φ_v, defined as

$$\min\{\varphi_{0.1,1}, \varphi_{0.1,5}, \varphi_{0.1,10}\}$$

   where $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$ is based on the validation set.

8.4.3 Accuracy performance measures

The accuracy of the classifiers was measured using the following five measures:

1. MSE based on the generalisation set, MSE_g (refer to Equation (4.6)).
2. MSE based on the validation set, MSE_m (refer to Equation (4.6)). MSE_m was measured to determine how well the classifiers remembered concepts learnt from the last 30 patterns in terms of MSE.
3. MSE based on the training set, MSE_t (refer to Equation (4.6)).
4. PCC based on the generalisation set, PCC_g (refer to Equation (4.7)).
5. PCC based on the validation set, PCC_m (refer to Equation (4.7)). PCC_m was measured to determine how well the classifiers remembered concepts learnt from the last 30 patterns in terms of PCC.

Note that PCC based on the training set was not measured, because PCC was not used to train the classifiers.

8.4.4 Structural complexity performance measures

Because of regularisation, the proposed classifiers change the structural complexity by reducing weight values towards zero. Thus, the optimal structural complexity and effective structural complexities need to be compared.

The optimal structural complexity was considered to be the structural complexity of the optimal architectures discussed in Section 8.3. The effective structural complexity, i.e. n_{s_e} and n_{n_e}, of a 3-layer FFNNs can be calculated, if a smaller architecture that has similar accuracy can be found (refer to Section 4.3.3). Pruning algorithms are well-suited for this purpose. The effective structural complexity was calculated by adapting the variance nullity testing pruning algorithm by Engelbrecht [31].

Variance nullity testing

Variance nullity testing is a statistical hypothesis testing technique, which tests whether the effect that a change in the weight of a synapse has on the outcomes of a ANN, i.e. the *sensitivity* of a synapse, varies statistically significantly across a set of patterns [31]. If the variance of the sensitivity is not significant, then the synapse is considered to be irrelevant and can be pruned [31]. Note that the validation set was used for these performance measures.

The sensitivity of a synapse can either be measured in terms of the objective function or output neuron values. The latter has the advantage of being independent of both the weights adjustment algorithm, and the objective function [31].

Variance nullity testing calculates sensitivity as the first-order derivative of the outputs with regards to the weight of the synapse [31]. The vector formed by the derivatives of a synapse and one or more outputs is known as the *output sensitivity vector* [31]. Each synapse forms one such vector per pattern [31]. Together, these vectors form the *output sensitivity matrix*, $S^{(p)}$, for pattern p in a data set D [31].

$S^{(p)}$ is used to calculate the *output sensitivity norm* of the j-th synapse for pattern p, as follows:

$$N_j^{(p)} = \sum_{k=1}^{K} S_{k,j}^{(p)} \tag{8.14}$$

where $S_{k,j}^{(p)}$ is the output sensitivity of the j-th synapse for the k-th output neuron, when using the pattern p [31].

Using $N_j^{(p)}$, the *average sensitivity norm* of the j-th synapse, across all patterns in the given data set D, is calculated as [31]:

$$\bar{N}_j = \frac{\sum_{p=1}^{|D|} N_j^{(p)}}{|D|} \tag{8.15}$$

$\bar{N}_j$ is used to determine the sample variance of the sensitivity norm for the j-th synapse, as follows [31]:

$$\sigma_j^2 = \frac{\sum_{p=1}^{|D|} (N_j^{(p)} - \bar{N}_j)^2}{|D| - 1} \tag{8.16}$$

Using σ_j^2, variance nullity testing constructs a Chi-square distribution test statistic for the variance in the sensitivity of the j-th synapse, as follows

$$\Upsilon_j = \frac{(|D| - 1)\sigma_j^2}{\sigma_0^2} \tag{8.17}$$

where σ_0^2 is the maximum amount of variance that an irrelevant synapse can display [31]. Hence, a larger σ_0^2 allows more pruning and vice versa [31]. Engelbrecht [31] suggested a value close to zero, e.g. 0.0001, in order to prevent haphazard pruning.

The significance of the variance is tested by using the Chi-squared distribution (χ^2) and null hypothesis, defined as $H_0 : \sigma_j^2 < \sigma_0^2$ [31]. The null hypothesis, H_0, thus states that any situation where $\sigma_j^2 \geq \sigma_0^2$ would reject H_0. On the other hand, any situation where $\sigma_j^2 < \sigma_0^2$ would accept the H_0.

Using Equation (8.17) and H_0, the critical value for the hypothesis test is defined as

$$\Upsilon_c = \chi^2_{|D|-1,1-\alpha_v} \tag{8.18}$$

where $\chi^2_{df,c}$ is the χ^2 distribution using df degrees of freedom and confidence interval of c; and α_v is the significance level, which controls the level of confidence that must be shown by the hypothesis test when determining the relevancy of a synapse [31]. A larger value for α_v will result in a larger chance of incorrectly pruning a synapse [31]. Engelbrecht [31] suggested that the test statistic should be very confident before pruning, i.e. $\alpha_v = 0.01$, in order to prevent false positives. The variance nullity algorithm considers any synapse with a test statistic below the critical value as irrelevant.

Adapted variance nullity pruning algorithm

This thesis used the variance nullity testing in the following way: The effective number of synapses (n_{s_e}) was calculated by counting the number of weights that do not have a test statistic below the critical value. The effective number of neurons (n_{n_e}) was determined by pruning the FFNN using the following two steps:

1. First, the irrelevant synapses were removed in such a way that the outputs of the FFNN will always produce values. In other words, each output must at a minimum be connected to either an input neuron or a bias neuron, directly or indirectly.

2. Second, all neurons that are not outputs, and do not have any incoming and outgoing synapses left after the first step, must be removed.

The number of remaining neurons is n_{n_e}. The measure, however, needs to be configured in terms of α_v and σ_0^2. Based on the recommendations made in [31], σ_0^2 was set to 0.0001 and α_p was set to 0.01. The algorithm made use of the validation set and was executed at the end of every epoch to determine n_{s_e} and n_{n_e}.

Structural complexity performance measures used

Structural complexity was measured by comparing the optimal architectures found by Rakitianskaia [94] to the effective architecture found by the classifiers. This was done by calculating the ratio between the total number of a particular structural component, e.g. neuron or synapse, in the optimal and the effective architectures, i.e. an oversize ratio. The following two oversize ratios were used as structural complexity performance measures:

1. The hidden neuron oversize ratio, $n_{h_{or}}$. This ratio was calculated by dividing n_{h_e} by the number of hidden neurons, including bias neurons, in the optimal architecture for the particular problem domain, where n_{h_e} is the number of neurons in the hidden layer obtained by the adapted variance nullity pruning algorithm. Because of the oversize factor of 2 (refer to Section 8.3), $n_{h_{or}} \in [0, 2]$. If $n_{h_{or}} = 1$, then the optimal and effective architecture contained the same number of hidden neurons.

2. The synapse oversize ratio, $n_{s_{or}}$. This ratio was calculated by dividing n_{s_e} by the number of synapses in the optimal architecture for the particular problem domain. Because the oversize factor of 2 also doubled the number of synapses, $n_{s_{or}} \in [0, 2]$. If $n_{s_{or}} = 1$, then the optimal and effective architecture contained the same number of synapses.

8.4.5 Computational complexity performance measures

The lower bound of the computational complexity of the classifiers to process one pattern was estimated, as follows

$$\Omega_{FFNN} = 2n_a + n_s \tag{8.19}$$

where n_s is the number of synapses, and n_a is the number of activated neurons, which is

$$n_a = n_n - (n_i + n_b)$$

where n_n is the number of neurons, n_i is the number of input neurons, and n_b is the number of bias neurons. Equation (8.19) makes the following three assumptions:

1. Each synapse will have at least one computational step, because each synapse weight will be used to calculate an input signal, i.e. $w_{is,n} \times v_{is,n}$.

2. Each activated neuron will at least have two computational steps, because each activated neuron requires the calculation of a net input signal and an activation value.

3. Input and bias neurons will have no computational steps, because no activation or net input signal is calculated for them during processing.

The lower bound of the effective computational complexity of the FFNNs, Ω_{FFNN_e}, was calculated by applying Equation (8.19) to the effective models produced by the adapted variance nullity pruning algorithm at the end of every epoch (refer to Section 8.4.4).

The effective reduction in computational complexity, due to the learning algorithm, was derived from Ω_{FFNN} and Ω_{FFNN_e}, as follows

$$\Omega_r = 100 \times \frac{\Omega_{FFNN} - \Omega_{FFNN_e}}{\Omega_{FFNN}} \tag{8.20}$$

where Ω_r represents the percentage by which the learning algorithm reduced the computational complexity of the classifier to process one pattern.

The remainder of this thesis refers to Ω_r as the *complexity reduction measure*.

8.4.6 Overfitting performance measures

Overfitting was measured using the following two measures:

1. The boolean result of the overfitting constraint Equation (4.10), O_ρ, at the end of every epoch. This measure returned 1 if a classifier was overfitting for an epoch, otherwise 0 was returned.

2. The boolean result of the overfitting constraint Equation (4.9), O_{MSE_g}, at the start of every epoch. This measure returned 1 if a classifier was overfitting for an epoch, otherwise 0 was returned. This measure used MSE_g instead of MSE_m, because MSE_m did not estimate the generalisation error but rather the classifiers ability to remember past patterns (refer to Section 8.4.1).

Both O_ρ and O_{MSE_g} make use of moving averages. To ensure that the moving average is smooth enough for the data stream in question, the moving average period should be chosen in accordance to the length of the data stream. A moving average period of 3% the length of the data stream was used. This allowed the measures to be comparable across different SDCPs, regardless of the length of the data streams.

8.4.7 Control parameter impact on performance measures

The more tunable control parameters there are, the more parameter configurations need to be evaluated during parameter tuning. Another factor that plays a part in the number of control parameter configurations is the size of the value set used for each control parameter. The larger a value set, the more parameter configurations there are that need to be tested. The number of control parameter configurations tested for a classifier ($|D_c|$) was calculated as follows:

$$|D_c| = \prod_{i=1}^{n_c}(|\mathcal{V}_i|) \tag{8.21}$$

where n_c is the number of tunable control parameters, and $\mathcal{V}_i$ is the set of potential values for the i-th control parameter.

The impact of the control parameters on saturation, accuracy, and complexity performance of the classifiers was analysed by comparing the differences in the performance measures to the differences in the number of control parameter configurations of the classifiers.

For example, consider a case with two classifiers, where their accuracy performance differed insignificantly, but the classifier with the slightly better accuracy had one more control parameter than the other classifier. The extra control parameter, therefore, had very little impact on the accuracy performance, but would increase the parameter configurations that needed to be tested. Hence, the additional parameter would not be worth the gain in accuracy performance.

8.4.8 Weight distribution performance measures

The regularisation algorithms maintain model complexity by altering the weight values. In the literature, especially architecture selection studies, it is often considered that the smaller the magnitudes of the weights are, the less complex an ANN is [10][31][33][118]. Hence, the distributions of both the weight values and weight magnitudes were calculated at the end of every epoch using the following measures:

1. The average magnitude of the weights, $\bar{w}$, defined as

$$\bar{w} = \frac{\sum_{j=1}^{n_s} |w_j|}{n_s} \tag{8.22}$$

   where $|w_j|$ is the magnitude, i.e. the absolute value, of the j-th weight, w_j, in the ANN.

2. The standard deviation of the weight magnitudes, σ_w, defined as

$$\sigma_w = \sqrt{\frac{\sum_{j=1}^{n_s} (|w_j| - \bar{w})^2}{n_s - 1}} \tag{8.23}$$

3. The weights frequency distribution, Ξ_w, was constructed by binning the weights using a binning width. The frequency of each bin was calculated by counting

the number of weights that fall into the range of that bin [95][97]. The binning width was set to 0.1 for the weights in the range $[-1, 1]$, and 1 for the weights in the ranges of $[-5, -1)$ and $(1, 5]$. Any weights larger or smaller than the covered ranges were counted under their nearest bin. These bin widths were used, because the values of weights in ANN with regularisation tended to be less than one, and saturated ANN tended to have weight values that were significantly more than one [10][68][95][96][97].

8.4.9 Swarm diversity performance measures

To evaluate the exploration and exploitation states of the QPSO classifiers, the swarm diversity was measured using Equation (3.3). The measure is denoted as $\mathcal{D}$.

8.5 Control parameter tuning process

All six of the classifiers used in the investigation had control parameters that required to be tuned. Various studies [10][29][52][72][95] have indicated good ranges for the control parameters. These ranges were used to construct the value sets for the purpose of control parameter tuning.

Furthermore, the control parameter tuning process assumed all the control parameters to be dependent on the problem domain and the other control parameters. The issue of problem domain dependence was addressed by using the streamed benchmark problem of each of the five problem domains. The issue of parameter dependence was addressed by testing all the control parameter configurations.

The control parameter tuning process trained each classifier, using each parameter configuration, 30 times per streamed benchmark problem. The collective mean of MSE_g, $\bar{C}_{MSE_g}$, was recorded after each of the 30 runs. The collective means were then averaged. Thirty samples were taken in order to get a valid statistical sample, because the six classifiers made use of stochastic learning algorithms [56].

The parameter configuration with the lowest average $\bar{C}_{MSE_g}$ for a classifier in a particular problem domain was considered to be the optimal parameter configuration.

Table 8.7 presents the various control parameter value sets used for tuning the six classifiers. The BP value sets were used by the BP classifiers. The QPSO value sets were used by the QPSO classifiers. The WD value sets were used by the WD-specific parameters in BP-WD and QPSO-WD. The WE value sets were used by the WE-specific parameters in BP-WE and QPSO-WE.

The control parameters w, c_1, c_2, d, and n_p were fixed because the values in Table 8.7 have been found to be optimal or *de facto* in ANN and PSO literature [29][33][52][99][115]. Discussion of these took place in Chapter 7.

The value sets of the remaining control parameters were chosen in such a way to ensure that the ranges of the control parameter values, suggested by literature, were uniformly covered.

BP literature suggests values in the range of $[0, 1]$ for both α and η [33][39][72]. Increments of 0.1 starting at 0.1 were used to cover α's and η's ranges.

QPSO literature suggests that small r values should be used when using the linear-decreasing distribution for sampling the positions of quantum particles. This prevents potentially very random position vectors from causing the swarm to search haphazardly [52]. The proposed range, thus, focused mostly on small values. Because Harrison *et al.* [52] investigated optimisation problems and not classification problems, some large values were included in the value set of r.

To allow comparison between the classifiers using WD and WE, the same regularisation coefficient (λ_r) value set was used for BP-WD, QPSO-WD, BP-WE and QPSO-WE. Bosman *et al.* [10] recommended that, when using weights elimination, λ_r should use the range $[0.001, 0.1]$. Hence, the value set chosen for λ_r focused more around the range $[0.001, 0.1]$, but allowed for values outside the range also to be considered, because WD might require such values for SDCPs. Furthermore, Bosman *et al.* [10] recommended that w_0 should be set to 0.01. Hence, the value set chosen for w_0 focused on values around 0.01.

The optimal parameter configurations found for BP-N, BP-WD, BP-WE, QPSO-N, QPSO-WD and QPSO-WE, per problem domain, are listed in Tables 8.8, 8.9, 8.10, 8.11, 8.12 and 8.13, respectively.

Table 8.7: Parameter sets for all classifier parameter tuning simulations

Parameter	Value sets
General parameters	
Number of runs	30
Max_{n_e}	1
n_w	Set per benchmark problem as discussed in section 8.3
n_i	Set per problem domain as discussed in section 8.3
n_h	Set per problem domain as discussed in section 8.3
n_o	Set per problem domain as discussed in section 8.3
BP parameters	
α	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
η	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
QPSO parameters	
n_p	30
U	Linear decreasing
r	$[0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.5, 5.0]$
r_1 and r_2 range	$[0, 1]$
c_1 and c_2	1.496180
ω	0.729844
WD parameters	
λ_r	$[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$
WE parameters	
λ_r	$[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$
w_0	$[0.005, 0.0075, 0.01, 0.025, 0.05, 0.1, 0.5, 1.0]$

Table 8.8: Benchmark parameters for BP-N

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.3

Table 8.9: Benchmark parameters for BP-WD

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.3
λ_r	0.0001	0.0001	0.0001	0.0001	0.0001

Table 8.10: Benchmark parameters for BP-WE

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.1
λ_r	0.01	0.0001	0.0005	0.0001	0.01
w_0	0.005	0.05	0.5	0.05	0.05

Table 8.11: Benchmark parameters for QPSO-N

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
r	5.0	5.0	0.1	5.0	5.0

Table 8.12: Benchmark parameters for QPSO-WD

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
r	0.1	0.1	0.1	0.1	0.25
λ_r	0.5	0.01	0.01	0.001	0.01

Table 8.13: Benchmark parameters for QPSO-WE

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
r	0.1	0.1	0.25	0.1	0.25
λ_r	0.05	0.01	0.01	0.01	0.1
w_0	0.5	0.5	0.5	0.1	1.0

8.6 Benchmarking process

After the control parameter tuning process was completed, the optimal parameter configurations in Tables 8.8 to 8.13 were used to carry out the benchmarking of the six classifiers. The benchmarking simulations measured the performance of the six classifiers on the 80 benchmark problems, using the performance measures discussed in Section 8.4.

Each performance measure was sampled 30 times per *classifier-benchmark pair*, i.e. the combination of a classifier and a benchmark problem. Performance measures were calculated for each epoch in each run, for each classifier-benchmark pair. The classifier-benchmark pairs each used the optimal parameter configuration that corresponded to the problem domain of the benchmark problem.

8.7 Result analysis methodology

The benchmarking results were analysed using a top-down approach, which used descriptive statistics, Mann-Whitney-U-based ranking, and performance trends. Each of these analysis techniques are discussed in sections 8.7.1, 8.7.2, and 8.7.3 respectively.

8.7.1 Descriptive statistics

The performance measures of each benchmark run was summarised using the collective mean approach. In the case of the weights frequency distribution (Ξ_w), each frequency bin was summarised using the collective mean approach to get the collective mean of the weights frequency distribution, $\bar{C}_{\Xi_w}$, for each run.

Afterwards, the collective means for each performance measure were aggregated over the 30 runs for each classifier-benchmark pair using the descriptive statistical measures mean and standard deviation. The thesis uses the notation $\bar{x} \pm \sigma$ to represent the two descriptive statistical measures, where $\bar{x}$ refers to the mean and σ refers to the standard deviation.

The benchmark results were further aggregated on four additional levels:

- *Classifier-domain* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the benchmark problem domains, i.e. the hyperplane, sphere, thresholds, SEA, and electricity domains.

- *Classifier-difficulty* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the benchmark problem difficulties, i.e. easy, moderate-I, moderate-II, and hard (refer to Section 8.3.6).

- *Classifier-environment* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the problem environments, i.e. abrupt, progressive, and chaotic. Quasi-static was left out as there were no quasi-static benchmark problems in the benchmark problem suite that was used (refer to Section 8.3.5).

- *Classifier-measure* level per classifier. This aggregation level grouped results of the classifiers by performance measure over all the benchmark problems.

8.7.2 Mann-Whitney-U-based ranking

Helbig and Engelbrecht [56] suggested a ranking approach for optimisers based on the number of favourable and unfavourable statistical differences when comparing optimisers on dynamic multi-objective optimisation problems. Since the ANN learning problem is a dynamic multi-objective optimisation problem (refer to Section 4.3) a similar approach was adopted for comparing the classifiers.

A series of Mann-Whitney U (MWU) pair-wise comparisons were done to see if the performance of one classifier was significantly different from another classifier on the same benchmark problem. If there was a significant difference between the two classifiers, then the classifier whose performance measure had a more favourable median was considered the winner. If there was no significant difference, the classifiers were considered tied.

A two-tailed MWU test was used, because the hypothesis test was based on weather or not their was a statistical difference in the performance of the two classifiers. The MWU tests were performed using a confidence interval of 0.95, i.e., a significance level of 0.05. Any p-value below 0.05 indicated a significant difference.

The total number of wins, ties and losses for each classifier were tallied per performance measure. The classifier with the least number of losses for the performance measure was the winner and was assigned the highest rank, i.e. 1. If there was a tie between the number of losses, then the classifier with more wins was considered better.

The notation *winning percentage/drawing percentage/losing percentage* is used in this thesis to represent the MWU-based ranking results. The percentage of wins, ties and losses were each calculated as follows:

$$\frac{numberofwins/ties/losses}{wins + ties + losses} \times 100$$

In the case where a rank was needed, the notation was extended as follows: *rank* (*winning percentage/drawing percentage/losing percentage*). If a classifier, for example, had the result 2(40/50/10), then the classifier was ranked second best out of the pool of classifiers. Furthermore, the result indicated that the classifier won 40%, tied 50%, and

lost 10% of the time in all the pair-wise comparisons between the classifier and the other classifiers in the pool. The MWU ranks should be compared to the ranks in the same row of a results table.

8.7.3 Performance trends

Trend analysis was carried out on the various performance measures. The 30 run values of a performance measure per epoch were averaged. It is possible that a epoch average could have infinitely large values, because of its performance measure, e.g. swarm diversity, and weight magnitude. In the case where epoch averages had a potentially broad range of values, e.g. $[10^{-300}, 10^{300}]$, a logarithmic scale was used.

Next, all trend lines were smoothed using a moving average with a period of 3% of the number of patterns in the SDCP. This allowed the performance trend of the measure to be extracted regardless of the number of patterns in the SDCP.

The sample standard deviation of the moving average was also determined for $\mathcal{D}$ and MSE_g. Two bands were formed along each moving average trend. The *positive band* represents the moving average plus the standard deviation. The *negative band* represents the moving average minus the standard deviation. These bands represented the volatility of the performance trend. A larger channel meant more volatility in the performance trend, and vice versa.

Lastly, the Pearson correlation coefficient was used to quantify the level of linear correlation between the performance trends of the measures, where needed. The Pearson correlation coefficient is defined as

$$\rho_{x,y} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}} \tag{8.24}$$

where x_i and y_i are the i-th values of the two data series being compared, and n is the size of the two data series. The sign of the Pearson correlation coefficient indicates the direction of the linear correlation. That is, a negative value shows an inverse relationship, and a positive value shows a linear relationship. The magnitude of the coefficient indicates the strength of the linear correlation. That is, a value of 0 means no correlation. On the other hand, a magnitude of 1 indicates a one to one correlation. A magnitude of 0.5 indicates a mild correlation. A magnitude of 0.8 indicates a strong correlation.

8.8 Summary

This chapter detailed the nine hypotheses made about the use of the proposed classifiers for SDCPs, and the procedure used to investigate these hypothesis. To assist with the investigation of the hypotheses, two baseline streamed data classifiers were introduced.

The procedure utilised a benchmark suite of 80 SDCPs to tune the control parameters of the classifiers. The tuned classifiers were then benchmarked using the benchmark suite. Performance was calculated using several performance measures, covering a broad range of performance areas. The procedure employed statistical analysis, MWU-ranking, and performance trend analysis to analysis the benchmark results.

Chapter 9 presents the analysis of the benchmarking results for the six classifiers, namely BP-N, BP-WD, BP-WE, QPSO-N, QPSO-WD, and QPSO-WD.

Chapter 9

Empirical analysis

This chapter presents the empirical analysis of the benchmark results for the six classifiers. The results were calculated using the performance measures discussed in Section 8.4, and the procedures in sections 8.5 and 8.6. The results are analysed using the methodology in Section 8.7.

The primary objective of this empirical analysis is to empirically determine whether or not the hypotheses in Section 8.1 were valid. The secondary objectives of the empirical analysis are to gain a better understanding of how the classifiers dealt with SDCPs, and to determine what improvements could be made to the proposed classifiers.

The remainder of this chapter is organised as follows. Sections 9.1, 9.2, and 9.3 present a statistical analysis of the collective mean results of the accuracy performance measures, saturation measures, and complexity performance measures, respectively. Note that complexity performance measures include both structural and computational complexity measures. Section 9.4 compares the saturation, accuracy and complexity performance trends to see how the classifiers behaved over time. Section 9.5 analyses the overfitting behaviour of the classifiers. Section 9.6 summaries the above discussions by presenting the overall MWU ranking of the classifiers based on their saturation, accuracy and complexity performance.

The remaining sections of the chapter analyse the auxiliary performance measures to better understand the findings made by the above discussions as follows. Section 9.7 analyses the control parameters of the classifiers. Section 9.8 analyses the weight distri-

butions of the classifiers. Section 9.9 analyses the swarm diversity of the QPSO classifiers. Section 9.10 presents the overall conclusions drawn from the empirical analysis. Lastly, Section 9.11 summarises the chapter.

Note that any value in a result table that is in **bold** indicates the best value in the set. Conversely, a value in *italics* indicates the worst value in the set. The ∞ symbol indicates values that was the maximum value of $10^{308.25}$. Furthermore, this chapter employs the notations and terminology presented in Chapter 8.

9.1 Accuracy performance analysis

This section analyses the collective means of the five accuracy performance measures MSE_t, MSE_m, MSE_g, PCC_m and PCC_g (refer to Section 8.4.3). The key objectives of the accuracy performance analysis are to determine:

1. How good the classifiers were at learning patterns. The MSE_t was used as an indicator of how well the classifiers learned patterns.

2. How good the classifiers were at remembering previously learned patterns. The MSE_m and PCC_m were used as indicators of well the classifiers remembered previously learned patterns.

3. How good the classifiers were at generalising. The MSE_g and PCC_g were used as indicators of how well the classifiers generalised.

4. If the regularised classifiers were more accurate than their non-regularised counterparts.

5. If the regularised QPSO classifiers were more accurate than their regularised BP counterparts.

6. The effects of problem difficulty, environment, dimensionality, and noise on the accuracy performance of the classifiers.

7. If the categorisation of problem difficulty was reflected in the accuracy results. That is, does decreasing problem difficulty, increase the accuracy of the classifiers, and vice versa.

8. If there were signs of overfitting.

Table 9.1 presents the statistical and MWU-based ranking analysis of the accuracy performance with regards to the classifiers for each of the problem domains. Table 9.2 presents the statistical and MWU-based ranking analysis of the accuracy performance with regards to the classifiers for each of the problem difficulties. Table 9.3 presents the statistical and MWU-based ranking analysis of the accuracy performance with regards to the classifiers for each of the problem environments. Table 9.4 presents the overall MWU-based ranking results of how the classifiers faired against each other based on their accuracy performance. The accuracy results in Tables 9.1 to 9.4 show the following:

BP-N was the best overall at generalising and remembering patterns in SDCPs. Furthermore, BP-N was capable of good levels of accuracy in various dynamic environments. This went against the expectations of the study.

As expected, the MSE measures showed QPSO-N to have the worst accuracy on all the problem domains. On the contrary, the PCC measures showed that QPSO-N performed very well, at times surpassing the PCC results of the regularised QPSO classifiers. These contrasting observations were most likely the result of complete saturation.

The idea of complete saturation is supported by the fact that PCC results were near to 12.5% in the thresholds domain, whereas the PCC results for the other four problem domains were near to 50%. This is because the thresholds domain is the only problem domain among the five problem domains that had three target classes, while the rest had two target classes. Consider the following example:

If a two target class classifier is completely saturated, then the outputs will always be one or the other class. Thus, a saturated two target class classifier has a 50% chance of being correct. Saturation, however, can only occur after some period of training. The probability of the classifier classifying a pattern correctly should, therefore, be higher than 50%. The same idea can be applied to a three target class classifier, however, the classifier should be able to classify more than 12.5% of the patterns correctly. The 12.5%

Table 9.1: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for accuracy performance measures

(a) Training mean square error (MSE_t)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.0027±0.0035** **1(70.00/27.50/2.50)**	**0.0027±0.0033** 2(70.00/25.00/5.00)	0.0029±0.0036 3(63.75/20.00/16.25)	*0.3537±0.0249* *6(0.00/0.00/100.00)*	0.0664±0.0144 5(20.00/0.00/80.00)	0.0404±0.0173 4(40.00/0.00/60.00)
Sphere	0.0216±0.0153 4(32.50/26.25/41.25)	0.0221±0.0157 5(21.25/18.75/60.00)	0.0216±0.0156 3(36.25/27.50/36.25)	*0.3245±0.1101* *6(0.00/5.00/95.00)*	**0.0061±0.0026** 2(70.00/5.00/25.00)	0.0142±0.0081 **1(97.50/2.50/0.00)**
Thresholds	**0.0103±0.0056** **1(86.25/13.75/0.00)**	0.0107±0.0056 2(68.75/20.00/11.25)	0.0113±0.0056 3(55.00/12.50/32.50)	*0.2464±0.0451* *6(0.00/0.00/100.00)*	0.0230±0.0058 5(20.00/0.00/80.00)	0.0143±0.0051 4(40.00/13.75/46.25)
SEA	0.0598±0.0234 2(36.25/32.50/31.25)	0.0606±0.0232 4(25.00/21.25/53.75)	0.0598±0.0235 2(36.25/32.50/31.25)	*0.3183±0.0628* *5(0.00/0.00/100.00)*	**0.0043±0.0033** **1(100.00/0.00/0.00)**	0.0949±0.0400 3(53.75/11.25/35.00)
Electricity	0.0260±0.0182 2(67.50/1.25/31.25)	0.0285±0.0180 5(43.75/1.25/55.00)	0.0536±0.0651 3(60.00/0.00/40.00)	*0.4993±0.0920* *6(0.00/0.00/100.00)*	**0.0178±0.0056** **1(80.00/0.00/20.00)**	0.0279±0.0086 4(46.25/2.50/51.25)

(b) Memory mean square error (MSE_m)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	0.0443±0.0274 2(77.50/11.25/11.25)	0.0455±0.0265 3(60.00/7.50/32.50)	**0.0430±0.0254** **1(91.25/3.75/5.00)**	*0.3845±0.0436* *6(0.00/0.00/100.00)*	0.1448±0.0175 5(20.00/6.25/73.75)	0.1405±0.0186 4(33.75/6.25/60.00)
Sphere	**0.0702±0.0335** **1(73.75/26.25/0.00)**	0.0715±0.0334 3(60.00/26.25/13.75)	0.0710±0.0337 2(65.00/30.00/5.00)	*0.3988±0.0476* *6(0.00/0.00/100.00)*	0.1961±0.0225 5(20.00/1.25/78.75)	0.1813±0.0304 4(38.75/1.25/60.00)
Thresholds	**0.0331±0.0174** **1(76.25/23.75/0.00)**	0.0335±0.0173 2(61.25/30.00/8.75)	0.0337±0.0175 3(61.25/28.75/10.00)	*0.2846±0.0163* *6(0.00/0.00/100.00)*	0.1275±0.0108 5(20.00/5.00/75.00)	0.1098±0.0164 4(35.00/5.00/60.00)
SEA	**0.1282±0.0258** **1(72.50/27.50/0.00)**	0.1291±0.0246 3(61.25/22.50/16.25)	0.1286±0.0257 2(67.50/27.50/5.00)	*0.3708±0.0158* *6(2.50/5.00/92.50)*	0.3603±0.0302 5(12.50/5.00/82.50)	0.2282±0.0286 4(40.00/0.00/60.00)
Electricity	**0.2203±0.0436** **1(92.50/1.25/6.25)**	0.2264±0.0424 2(75.00/1.25/23.75)	0.2901±0.0445 4(37.50/0.00/62.50)	*0.5398±0.0409* *6(0.00/0.00/100.00)*	0.2864±0.0697 5(33.75/0.00/66.25)	0.2714±0.0668 3(60.00/0.00/40.00)

(c) Memory percentage correct classification error (PCC_m)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**80.5178±10.2529** **1(85.00/11.25/3.75)**	79.5854±9.5886 3(62.50/7.50/30.00)	79.9934±9.4008 2(78.75/8.75/12.50)	61.5537±4.3628 4(40.00/0.00/60.00)	*1.5456±0.9681* *6(0.00/0.00/100.00)*	13.7709±8.1818 5(20.00/0.00/80.00)
Sphere	**67.5005±11.7622** **1(73.75/16.25/10.00)**	66.5257±11.2983 3(46.25/13.75/40.00)	67.0630±11.6111 2(60.00/16.25/23.75)	60.1143±4.7564 5(32.50/11.25/56.25)	*55.2463±2.4030* *6(15.00/1.25/83.75)*	58.4173±2.7255 4(41.25/3.75/55.00)
Thresholds	**67.2708±10.9450** **1(93.75/6.25/0.00)**	66.2988±10.5605 2(75.00/10.00/15.00)	65.1331±10.5987 3(58.75/3.75/37.50)	26.8981±13.7610 5(11.25/11.25/77.50)	*24.7291±1.1552* *6(7.50/5.00/87.50)*	42.4968±6.3503 4(33.75/3.75/62.50)
SEA	35.8517±9.9392 4(20.00/16.25/63.75)	*35.2188±9.6418* *6(0.00/13.75/86.25)*	35.7408±10.0975 5(16.25/17.50/66.25)	**62.9222±1.5763** **1(86.25/8.75/5.00)**	48.3348±3.1967 3(60.00/0.00/40.00)	62.0699±3.5445 2(85.00/8.75/6.25)
Electricity	40.0129±16.3249 **1(87.50/1.25/11.25)**	35.4457±12.6871 3(67.50/1.25/31.25)	*6.6181±7.9148* *6(6.25/1.25/92.50)*	**46.0194±4.0921** 2(83.75/0.00/16.25)	14.7658±2.4112 4(37.50/1.25/61.25)	7.4306±1.4391 5(15.00/0.00/85.00)

(d) Generalisation mean square error (MSE_g)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	0.0296±0.0260 **1(82.50/15.00/2.50)**	0.0299±0.0256 3(62.50/16.25/21.25)	**0.0294±0.0249** 2(71.25/16.25/12.50)	*0.3804±0.0503* *6(0.00/0.00/100.00)*	0.1081±0.0399 5(20.00/1.25/78.75)	0.0921±0.0490 4(38.75/1.25/60.00)
Sphere	**0.0551±0.0377** 2(71.25/26.25/2.50)	0.0557±0.0381 3(60.00/21.25/18.75)	0.0553±0.0379 **1(71.25/27.50/1.25)**	*0.3752±0.0603* *6(0.00/0.00/100.00)*	0.0985±0.0688 5(20.00/2.50/77.50)	0.0973±0.0657 4(37.50/2.50/60.00)
Thresholds	**0.0273±0.0172** **1(83.75/16.25/0.00)**	0.0276±0.0171 2(70.00/18.75/11.25)	0.0280±0.0171 3(62.50/12.50/25.00)	*0.2738±0.0202* *6(0.00/0.00/100.00)*	0.0750±0.0391 5(20.00/0.00/80.00)	0.0598±0.0361 4(40.00/0.00/60.00)
SEA	**0.1114±0.0478** **1(57.50/37.50/5.00)**	0.1119±0.0472 3(43.75/31.25/25.00)	0.1115±0.0477 2(55.00/40.00/5.00)	*0.3568±0.0282* *6(0.00/5.00/95.00)*	0.1831±0.1363 5(35.00/12.50/52.50)	0.1751±0.0526 4(30.00/31.25/38.75)
Electricity	**0.0998±0.0699** **1(81.25/2.50/16.25)**	0.1023±0.0690 3(58.75/1.25/40.00)	0.1210±0.0817 2(61.25/1.25/37.50)	*0.5255±0.0476* *6(0.00/0.00/100.00)*	0.1170±0.0942 4(53.75/0.00/46.25)	0.1173±0.0868 5(42.50/0.00/57.50)

(e) Generalisation percentage correct classification error (PCC_g)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**81.8334±11.9262** **1(85.00/12.50/2.50)**	81.2043±11.4578 3(61.25/13.75/25.00)	81.5766±11.3502 2(73.75/13.75/12.50)	61.9385±5.0492 4(40.00/0.00/60.00)	*1.4871±0.9773* *6(0.00/0.00/100.00)*	13.5273±8.7044 5(20.00/0.00/80.00)
Sphere	67.5769±12.1462 2(52.50/25.00/22.50)	66.7762±11.9000 4(30.00/20.00/50.00)	67.2979±12.1965 3(42.50/26.25/31.25)	*62.4632±6.0127* 5(26.25/13.75/60.00)	64.5992±7.9827 *6(30.00/2.50/67.50)*	**69.3535±9.2659** **1(68.75/12.50/18.75)**
Thresholds	**66.5919±10.9518** **1(91.25/7.50/1.25)**	65.6423±10.5532 2(75.00/11.25/13.75)	64.4263±10.5758 3(58.75/3.75/37.50)	28.2938±14.3404 5(13.75/8.75/77.50)	*25.7685±1.7229* *6(7.50/5.00/87.50)*	48.9703±10.7458 4(33.75/3.75/62.50)
SEA	35.0198±9.7107 4(17.50/22.50/60.00)	*34.4524±9.4374* *6(0.00/20.00/80.00)*	34.8365±9.6887 5(8.75/25.00/66.25)	64.3005±2.8193 3(67.50/6.25/26.25)	**69.6214±16.6769** 2(82.50/2.50/15.00)	68.8185±7.0830 **1(83.75/3.75/12.50)**
Electricity	45.1450±17.8577 **1(88.75/1.25/10.00)**	39.0552±15.3828 3(66.25/2.50/31.25)	10.1391±11.4931 5(12.50/1.25/86.25)	**47.4473±4.7624** 2(82.50/0.00/17.50)	17.8350±2.4587 4(35.00/0.00/65.00)	*8.3214±1.5849* *6(12.50/0.00/87.50)*

is derived by multiplying the 50% probabilities of the three mutually exclusive outputs together, i.e. $50\% \times 50\% \times 50\%$.

The above example also explains why the MSE values showed poor performance for QPSO-N, because these values were at the extreme values of zero and one most of the time. If the QPSO-N did suffer from complete saturation then, the QPSO-N would be unsuitable for SDCPs. Thus this needs to be confirmed by analysing the saturation measures.

Regularisation degraded the accuracy performance of the BP weights adjustment algorithm. However, regularisation, especially WE, improved the MSE performance of the QPSO weights adjustment algorithm for SDCPs.

The low MSE_t values of all the classifiers, except QPSO-N, showed that the classifiers

Table 9.2: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for accuracy performance measures

(a) Training mean square error (MSE_t)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	**0.0134±0.0163** **1(68.00/13.00/19.00)**	0.0145±0.0168 5(44.00/9.00/47.00)	0.0232±0.0513 2(64.00/16.00/20.00)	*0.4090±0.0940* *6(0.00/0.00/100.00)*	0.0188±0.0183 4(54.00/0.00/46.00)	0.0400±0.0542 3(46.00/10.00/44.00)
Moderate-I	0.0332±0.0308 **1(50.00/21.00/29.00)**	0.0341±0.0311 5(43.00/15.00/42.00)	0.0336±0.0320 3(51.00/16.00/33.00)	*0.3686±0.1016* *6(0.00/0.00/100.00)*	**0.0240±0.0250** 4(63.00/0.00/37.00)	0.0398±0.0273 2(67.00/0.00/33.00)
Moderate-II	**0.0130±0.0139** **1(66.00/19.00/15.00)**	0.0136±0.0140 2(54.00/15.00/31.00)	0.0135±0.0142 2(54.00/15.00/31.00)	*0.3521±0.0956* *5(0.00/0.00/100.00)*	0.0216±0.0204 3(52.00/1.00/47.00)	0.0376±0.0373 4(47.00/4.00/49.00)
Hard	0.0367±0.0250 **1(50.00/28.00/22.00)**	0.0374±0.0254 3(42.00/30.00/28.00)	0.0491±0.0401 5(32.00/27.00/41.00)	*0.2640±0.1027* *6(0.00/4.00/96.00)*	**0.0295±0.0304** 4(63.00/3.00/34.00)	0.0360±0.0175 2(62.00/10.00/28.00)

(b) Memory mean square error (MSE_m)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	**0.0672±0.0639** **1(93.00/3.00/4.00)**	0.0701±0.0661 3(61.00/4.00/35.00)	0.0849±0.0961 2(64.00/7.00/29.00)	*0.4165±0.0930* *6(0.00/0.00/100.00)*	0.1918±0.0842 5(26.00/1.00/73.00)	0.1590±0.0515 4(48.00/1.00/51.00)
Moderate-I	**0.1009±0.0862** **1(77.00/22.00/1.00)**	0.1028±0.0886 3(64.00/18.00/18.00)	0.1103±0.1039 2(63.00/24.00/13.00)	*0.3997±0.0965* *6(0.00/1.00/99.00)*	0.2283±0.1021 5(20.00/4.00/76.00)	0.1809±0.0713 4(40.00/3.00/57.00)
Moderate-II	**0.0957±0.0661** **1(77.00/19.00/4.00)**	0.0984±0.0689 2(66.00/19.00/15.00)	0.1236±0.1174 3(61.00/18.00/21.00)	*0.3873±0.0924* *6(0.00/1.00/99.00)*	0.2185±0.0908 5(23.00/4.00/73.00)	0.1861±0.0649 4(41.00/3.00/56.00)
Hard	**0.1330±0.0744** **1(67.00/28.00/5.00)**	0.1336±0.0745 3(63.00/29.00/8.00)	0.1343±0.0775 2(70.00/23.00/7.00)	*0.3792±0.0788* *6(2.00/2.00/96.00)*	0.2534±0.1009 5(16.00/5.00/79.00)	0.2189±0.0760 4(37.00/3.00/60.00)

(c) Memory percentage correct classification error (PCC_m)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	**71.1375±17.3775** **1(86.00/0.00/14.00)**	67.5381±18.8088 2(56.00/1.00/43.00)	61.4998±28.4872 3(52.00/2.00/46.00)	46.4666±19.9619 5(38.00/1.00/61.00)	*30.5553±21.3814* *6(25.00/1.00/74.00)*	41.3074±21.8379 4(40.00/1.00/59.00)
Moderate-I	**54.2860±23.0222** **1(71.00/12.00/17.00)**	53.4626±23.4110 2(51.00/7.00/42.00)	48.5740±30.6027 4(47.00/8.00/45.00)	50.9033±14.5641 3(49.00/9.00/42.00)	*28.1327±20.0461* *6(21.00/2.00/77.00)*	35.9494±24.7014 5(40.00/4.00/56.00)
Moderate-II	**63.1757±15.6818** **1(81.00/5.00/14.00)**	61.6657±15.8855 2(55.00/5.00/40.00)	53.7555±27.3672 3(46.00/8.00/46.00)	51.9559±15.4558 4(45.00/7.00/48.00)	*29.8271±21.6721* *6(22.00/2.00/76.00)*	37.8279±23.6618 5(35.00/5.00/60.00)
Hard	44.3238±18.3547 2(50.00/24.00/26.00)	43.7930±18.4561 3(39.00/24.00/37.00)	39.8093±24.1654 4(31.00/20.00/49.00)	**56.6804±9.2216** **1(71.00/8.00/21.00)**	*27.1823±19.9601* *6(28.00/1.00/71.00)*	32.2637±23.2768 5(41.00/3.00/56.00)

were able to learn the decision boundaries from patterns in SDCPs accurately. However, the training and generalisation accuracies of these classifiers showed signs of overfitting.

Furthermore, the memory errors, i.e. MSE_m and PCC_m tended to be worse than the generalisation errors, i.e. MSE_g and $PCCg$. This was more prominent in the

(d) Generalisation mean square error (MSE_g)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	**0.0291±0.0260**	0.0305±0.0265	0.0389±0.0531	*0.4109±0.0936*	0.0515±0.0202	0.0628±0.0535
	1(82.00/7.00/11.00)	3(53.00/4.00/43.00)	2(73.00/9.00/18.00)	*6(0.00/0.00/100.00)*	4(40.00/4.00/56.00)	5(36.00/8.00/56.00)
Moderate-I	**0.0790±0.0486**	0.0798±0.0488	0.0791±0.0496	*0.3960±0.0932*	0.1593±0.0870	0.1298±0.0458
	1(71.00/22.00/7.00)	3(57.00/19.00/24.00)	2(66.00/23.00/11.00)	*6(0.00/2.00/98.00)*	5(26.00/3.00/71.00)	4(42.00/7.00/51.00)
Moderate-II	**0.0358±0.0258**	0.0366±0.0261	0.0386±0.0290	*0.3606±0.0939*	0.0595±0.0206	0.0678±0.0383
	1(79.00/19.00/2.00)	2(63.00/16.00/21.00)	3(60.00/19.00/21.00)	*6(0.00/0.00/100.00)*	5(33.00/2.00/65.00)	4(33.00/8.00/59.00)
Hard	**0.1146±0.0630**	0.1151±0.0631	0.1196±0.0689	*0.3619±0.0844*	0.1950±0.0935	0.1728±0.0735
	1(69.00/30.00/1.00)	2(63.00/32.00/5.00)	3(58.00/27.00/15.00)	*6(0.00/2.00/98.00)*	5(20.00/4.00/76.00)	4(40.00/5.00/55.00)

(e) Generalisation percentage correct classification error (PCC_g)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	**70.6192±18.8573**	66.6190±21.6140	62.7592±27.7451	47.0649±20.0484	*40.2676±32.1852*	49.2104±27.8631
	1(83.00/5.00/12.00)	3(49.00/4.00/47.00)	2(51.00/7.00/42.00)	*5(34.00/0.00/66.00)*	*5(34.00/0.00/66.00)*	4(39.00/4.00/57.00)
Moderate-I	**54.2417±21.3290**	53.1408±21.9535	47.6658±29.8415	51.2764±14.4195	*31.6236±23.4974*	37.9179±26.4395
	1(69.00/12.00/19.00)	2(49.00/10.00/41.00)	4(43.00/9.00/48.00)	3(46.00/9.00/45.00)	*6(25.00/2.00/73.00)*	5(45.00/4.00/51.00)
Moderate-II	**67.8696±15.6180**	66.2484±15.6093	56.9481±26.6778	54.7571±15.8841	*40.3784±32.2936*	45.1703±29.9416
	1(69.00/15.00/16.00)	2(51.00/15.00/34.00)	3(37.00/18.00/45.00)	5(33.00/7.00/60.00)	*6(33.00/3.00/64.00)*	4(46.00/4.00/50.00)
Hard	44.2032±17.6472	43.6963±17.7294	39.2478±23.7219	**58.4563±9.4052**	*31.1794±23.9303*	34.8941±25.8256
	2(47.00/23.00/30.00)	3(37.00/25.00/38.00)	5(26.00/22.00/52.00)	**1(71.00/7.00/22.00)**	*6(32.00/3.00/65.00)*	4(45.00/4.00/51.00)

regularised QPSO classifiers than the BP classifiers. The regularised QPSO classifiers, therefore, forgot patterns significantly faster than the regularised BP classifiers. This explained the lower than expected accuracy performance results of the regularised QPSO classifiers.

The MSE and PCC performance measures provided conflicting conclusions. This was in-line with the findings by Twomey and Smith [113] for static classification problems. Both measures should, therefore, be considered in studies that look at streamed data stream classifiers.

The regularised QPSO classifiers showed potential at handling noisy problems, while the BP classifiers did not. This was evident from the SEA domain, where the regularised QPSO classifiers significantly outperformed the BP classifiers according to the PCC measures, with QPSO-WD and QPSO-WE achieving very similar results. Note that the regularised QPSO classifiers did not fair so well on the real-world electricity problem.

Furthermore, high dimensional SDCPs had a detrimental effect on the accuracy of

Table 9.3: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for accuracy performance measures

(a) Training mean square error (MSE_t)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**0.0226±0.0230**	0.0230±0.0233	0.0231±0.0229	*0.2722±0.0748*	0.0264±0.0272	0.0353±0.0311
	1(57.22/26.67/16.11)	2(50.00/25.56/24.44)	3(43.33/23.33/33.33)	*6(0.00/2.22/97.78)*	5(50.00/2.22/47.78)	4(56.11/6.67/37.22)
Abrupt	**0.0122±0.0124**	0.0136±0.0128	0.0430±0.0832	*0.4392±0.0990*	0.0160±0.0161	0.0473±0.0641
	1(74.29/8.57/17.14)	4(45.71/2.86/51.43)	2(65.71/11.43/22.86)	*6(0.00/0.00/100.00)*	3(60.00/0.00/40.00)	5(40.00/5.71/54.29)
Chaotic	0.0278±0.0275	0.0289±0.0276	0.0339±0.0389	*0.4054±0.0945*	**0.0221±0.0216**	0.0396±0.0346
	1(56.76/16.22/27.03)	5(41.62/11.89/46.49)	2(54.05/15.14/30.81)	*6(0.00/0.00/100.00)*	3(65.41/0.00/34.59)	4(57.84/5.41/36.76)

(b) Memory mean square error (MSE_m)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	0.0789±0.0462	0.0795±0.0461	**0.0788±0.0463**	*0.3431±0.0571*	0.2041±0.0940	0.1680±0.0548
	2(67.78/27.78/4.44)	3(61.11/28.89/10.00)	**1(69.44/26.67/3.89)**	*6(1.11/1.67/97.22)*	5(17.22/5.56/77.22)	4(36.11/3.89/60.00)
Abrupt	**0.0721±0.0594**	0.0757±0.0612	0.1069±0.1149	*0.4430±0.0981*	0.1911±0.0875	0.1567±0.0536
	1(91.43/0.00/8.57)	3(60.00/0.00/40.00)	2(62.86/0.00/37.14)	*6(0.00/0.00/100.00)*	5(31.43/0.00/68.57)	4(54.29/0.00/45.71)
Chaotic	**0.1241±0.0931**	0.1272±0.0950	0.1481±0.1227	*0.4379±0.0889*	0.2474±0.0949	0.2095±0.0769
	1(86.49/11.89/1.62)	2(66.49/9.73/23.78)	3(60.00/12.97/27.03)	*6(0.00/0.54/99.46)*	5(23.24/2.16/74.59)	4(44.32/1.62/54.05)

(c) Memory percentage correct classification error (PCC_m)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**59.6057±17.4275**	58.8505±17.2130	58.6347±17.0091	52.0496±17.1994	*31.3512±20.3961*	42.2506±19.5774
	1(63.89/16.67/19.44)	3(46.67/16.67/36.67)	2(48.33/15.56/36.11)	4(45.56/9.44/45.00)	*6(20.00/2.22/77.78)*	5(42.78/5.00/52.22)
Abrupt	**69.7218±17.8179**	63.9425±19.8207	57.3227±28.4463	46.6844±20.0509	*32.7962±22.0601*	41.5980±24.1980
	1(77.14/0.00/22.86)	3(48.57/2.86/48.57)	2(51.43/5.71/42.86)	4(45.71/2.86/51.43)	*6(28.57/2.86/68.57)*	5(40.00/2.86/57.14)
Chaotic	**54.7189±24.0935**	53.0533±24.1710	42.1802±34.6732	51.8796±12.7650	*25.8306±20.2804*	30.6692±25.2160
	1(78.92/5.95/15.14)	3(54.05/3.24/42.70)	4(38.38/4.32/57.30)	2(56.76/3.78/39.46)	*6(27.03/0.54/72.43)*	5(35.14/1.62/63.24)

the regularised QPSO classifiers. This was not the case for the BP classifiers.

The problem difficulty classification scheme proposed in Section 8.3.6 was shown to be valid for SDCPs at a high-level. However, QPSO-N did not follow the classification scheme. This was due to the complete saturation of QPSO-N (refer to Section 9.2). Furthermore, according to the accuracy results moderate-I problems generally were more difficult than moderate-II problems. Thus, window frequency (w_f) has a more severe effect on the accuracy performance of a stream data classifier than window step size (w_s). Figure 9.1 illustrates the above by plotting the mean of the PCC_g in Table 9.2e against

(d) Generalisation mean square error (MSE_g)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**0.0592±0.0500**	0.0596±0.0499	0.0594±0.0497	*0.3231±0.0601*	0.1149±0.0887	0.1057±0.0702
	1(70.00/28.33/1.67)	2(61.11/27.78/11.11)	3(61.67/26.67/11.67)	*6(0.00/1.11/98.89)*	5(23.33/3.33/73.33)	4(36.67/7.22/56.11)
Abrupt	**0.0252±0.0192**	0.0269±0.0196	0.0555±0.0825	*0.4398±0.0988*	0.0403±0.0161	0.0629±0.0633
	1(85.71/5.71/8.57)	3(51.43/2.86/45.71)	2(77.14/5.71/17.14)	*6(0.00/0.00/100.00)*	4(45.71/2.86/51.43)	5(31.43/0.00/68.57)
Chaotic	**0.0773±0.0610**	0.0785±0.0610	0.0809±0.0665	*0.4291±0.0845*	0.1321±0.0925	0.1195±0.0698
	1(78.38/13.51/8.11)	3(58.38/10.81/30.81)	2(64.32/15.14/20.54)	*6(0.00/1.08/98.92)*	5(32.97/3.24/63.78)	4(40.00/8.11/51.89)

(e) Generalisation percentage correct classification error (PCC_g)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**60.1007±18.5690**	59.4296±18.4118	59.1302±18.2265	54.1608±17.7076	*38.7415±28.7047*	48.2806±24.3954
	1(55.00/21.67/23.33)	2(42.22/22.78/35.00)	3(39.44/22.22/38.33)	5(40.56/8.89/50.56)	*6(29.44/3.89/66.67)*	4(51.11/5.00/43.89)
Abrupt	**68.4628±19.4940**	61.9730±23.3047	58.9761±26.9168	47.0126±20.1263	*44.2696±35.1474*	49.2819±30.1378
	1(80.00/2.86/17.14)	3(40.00/8.57/51.43)	2(48.57/8.57/42.86)	4(42.86/0.00/57.14)	5(40.00/0.00/60.00)	*6(37.14/2.86/60.00)*
Chaotic	**56.6435±23.4713**	54.6165±23.7548	42.9973±33.7676	52.7626±12.6670	*31.4702±26.1957*	34.0751±28.8328
	1(76.22/8.11/15.68)	2(51.89/5.41/42.70)	4(37.30/7.03/55.68)	3(51.89/3.78/44.32)	*6(30.81/0.54/68.65)*	5(37.84/3.24/58.92)

Table 9.4: MWU-based pairwise comparison of the classifiers for accuracy performance measures (Wins/Ties/Losses percentages)

(a) Training mean square error (MSE_t)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	53.75/45.00/1.25	36.25/48.75/15.00	98.75/1.25/0.00	48.75/0.00/51.25	55.00/6.25/38.75
BP-WD	1.25/45.00/53.75	—	31.25/30.00/38.75	98.75/1.25/0.00	47.50/0.00/52.50	50.00/10.00/40.00
BP-WE	15.00/48.75/36.25	38.75/30.00/31.25	—	98.75/1.25/0.00	51.25/1.25/47.50	47.50/11.25/41.25
QPSO-N	0.00/1.25/98.75	0.00/1.25/98.75	0.00/1.25/98.75	—	0.00/1.25/98.75	0.00/0.00/100.00
QPSO-WD	51.25/0.00/48.75	52.50/0.00/47.50	47.50/1.25/51.25	98.75/1.25/0.00	—	40.00/2.50/57.50
QPSO-WE	38.75/6.25/55.00	40.00/10.00/50.00	41.25/11.25/47.50	100.00/0.00/0.00	57.50/2.50/40.00	—
Overall rank	**1(58.50/20.25/21.25)**	3(45.75/17.25/37.00)	2(50.25/18.50/31.25)	*6(0.00/1.00/99.00)*	5(58.00/1.00/41.00)	4(55.50/6.00/38.50)

(b) Memory mean square error (MSE_m)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	55.00/43.75/1.25	41.25/46.25/12.50	100.00/0.00/0.00	98.75/0.00/1.25	97.50/0.00/2.50
BP-WD	1.25/43.75/55.00	—	21.25/43.75/35.00	100.00/0.00/0.00	98.75/0.00/1.25	96.25/0.00/3.75
BP-WE	12.50/46.25/41.25	35.00/43.75/21.25	—	100.00/0.00/0.00	88.75/0.00/11.25	86.25/0.00/13.75
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	2.50/5.00/92.50	0.00/0.00/100.00
QPSO-WD	1.25/0.00/98.75	1.25/0.00/98.75	11.25/0.00/88.75	92.50/5.00/2.50	—	0.00/12.50/87.50
QPSO-WE	2.50/0.00/97.50	3.75/0.00/96.25	13.75/0.00/86.25	100.00/0.00/0.00	87.50/12.50/0.00	—
Overall rank	**1(78.50/18.00/3.50)**	3(63.50/17.50/19.00)	2(64.50/18.00/17.50)	*6(0.50/1.00/98.50)*	5(21.25/3.50/75.25)	4(41.50/2.50/56.00)

(c) Memory percentage correct classification error (PCC_m)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	77.50/22.50/0.00	68.75/23.75/7.50	61.25/3.75/35.00	76.25/1.25/22.50	76.25/0.00/23.75
BP-WD	0.00/22.50/77.50	—	38.75/20.00/41.25	60.00/3.75/36.25	76.25/0.00/23.75	76.25/0.00/23.75
BP-WE	7.50/23.75/68.75	41.25/20.00/38.75	—	52.50/2.50/45.00	57.50/1.25/41.25	61.25/0.00/38.75
QPSO-N	35.00/3.75/61.25	36.25/3.75/60.00	45.00/2.50/52.50	—	82.50/5.00/12.50	55.00/16.25/28.75
QPSO-WD	22.50/1.25/76.25	23.75/0.00/76.25	41.25/1.25/57.50	12.50/5.00/82.50	—	20.00/0.00/80.00
QPSO-WE	23.75/0.00/76.25	23.75/0.00/76.25	38.75/0.00/61.25	28.75/16.25/55.00	80.00/0.00/20.00	—
Overall rank	**1(72.00/10.25/17.75)**	2(50.25/9.25/40.50)	4(44.00/9.50/46.50)	3(50.75/6.25/43.00)	*6(24.00/1.50/74.50)*	5(39.00/3.25/57.75)

(d) Generalisation mean square error (MSE_g)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	60.00/40.00/0.00	40.00/48.75/11.25	100.00/0.00/0.00	88.75/0.00/11.25	87.50/8.75/3.75
BP-WD	0.00/40.00/60.00	—	23.75/38.75/37.50	100.00/0.00/0.00	85.00/1.25/13.75	86.25/8.75/5.00
BP-WE	11.25/48.75/40.00	37.50/38.75/23.75	—	100.00/0.00/0.00	87.50/1.25/11.25	85.00/8.75/6.25
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	0.00/5.00/95.00	0.00/0.00/100.00
QPSO-WD	11.25/0.00/88.75	13.75/1.25/85.00	11.25/1.25/87.50	95.00/5.00/0.00	—	17.50/8.75/73.75
QPSO-WE	3.75/8.75/87.50	5.00/8.75/86.25	6.25/8.75/85.00	100.00/0.00/0.00	73.75/8.75/17.50	—
Overall rank	**1(75.25/19.50/5.25)**	3(59.00/17.75/23.25)	2(64.25/19.50/16.25)	*6(0.00/1.00/99.00)*	5(29.75/3.25/67.00)	4(37.75/7.00/55.25)

(e) Generalisation percentage correct classification error (PCC_g)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	70.00/30.00/0.00	65.00/32.50/2.50	61.25/2.50/36.25	73.75/0.00/26.25	65.00/3.75/31.25
BP-WD	0.00/30.00/70.00	—	36.25/32.50/31.25	58.75/2.50/38.75	73.75/0.00/26.25	63.75/2.50/33.75
BP-WE	2.50/32.50/65.00	31.25/32.50/36.25	—	51.25/2.50/46.25	58.75/0.00/41.25	52.50/2.50/45.00
QPSO-N	36.25/2.50/61.25	38.75/2.50/58.75	46.25/2.50/51.25	—	58.75/10.00/31.25	50.00/11.25/38.75
QPSO-WD	26.25/0.00/73.75	26.25/0.00/73.75	41.25/0.00/58.75	31.25/10.00/58.75	—	30.00/0.00/70.00
QPSO-WE	31.25/3.75/65.00	33.75/2.50/63.75	45.00/2.50/52.50	38.75/11.25/50.00	70.00/0.00/30.00	—
Overall rank	**1(67.00/13.75/19.25)**	2(46.50/13.50/40.00)	3(39.25/14.00/46.75)	4(46.00/5.75/48.25)	*6(31.00/2.00/67.00)*	5(43.75/4.00/52.25)

the problem difficulties for each classifier.

Lastly, SDCPs that have low temporal severity, e.g. abrupt, allowed classifiers to achieve the best levels of accuracy. Thus, longer environment instances allowed for better accuracy to be achieved, because there is more time to learn.

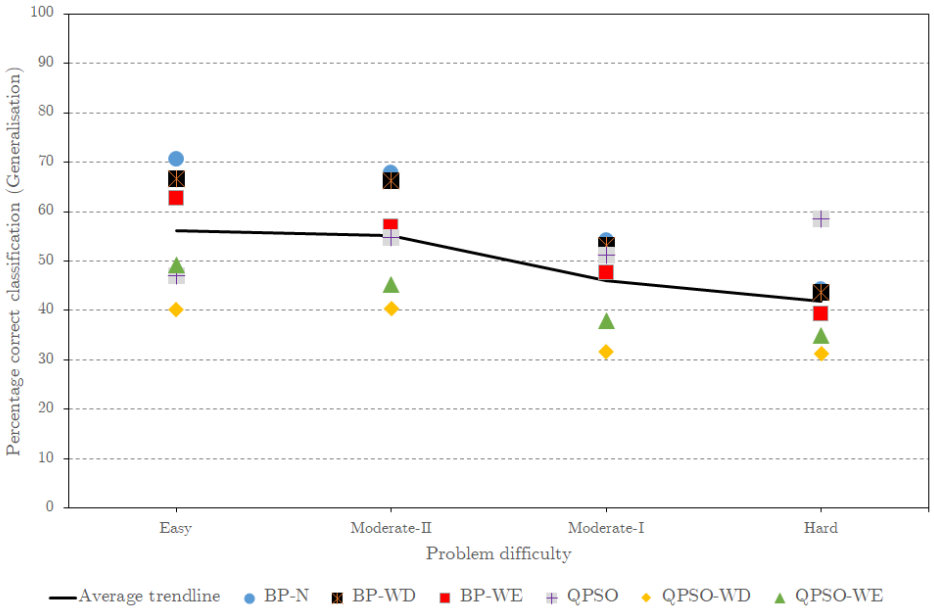


Figure 9.1: PCC_g versus problem difficulty scatter plot

9.2 Saturation analysis

This section analyses the collective means of the two saturation performance measures φ_g and φ_v, including their components (refer to Section 8.4.2). Note that the saturation measures calculate saturation levels for the hidden neurons only. The key objectives of the saturation analysis are to determine:

1. If the level of saturation experienced by the BP-based classifiers differed from the level experienced by the QPSO-based classifiers.
2. If the regularised classifiers saturated less than their non-regularised counterparts.
3. If ReLU activation functions helped to reduce saturation.
4. The effects of problem difficulty, environment, dimensionality, and noise on the saturation levels of the classifiers.
5. If the saturation measures were suitable for ReLU activation functions.
6. If the saturation measures could be simplified.

7. Confirm the presences of complete saturation in QPSO-N.

Table 9.5 presents the statistical and MWU-based ranking analysis of the saturation levels with regards to the classifiers for each of the problem domains. Table 9.6 presents the statistical and MWU-based ranking analysis of the saturation levels with regards to the classifiers for each of the problem difficulties. Table 9.7 presents the statistical and MWU-based ranking analysis of the saturation levels with regards to the classifiers for each of the problem environments. Table 9.8 presents the MWU-based ranking results of how the classifiers faired against each other over all the problems based on saturation levels. The saturation results in Tables 9.5 to 9.8 show the following:

The saturation measures, i.e. φ_v and φ_g, very closely reflected the same outcomes for most of the problem domains. The SEA problem domain, however, exhibited some minor, but negligible variations between φ_v and φ_g. The average Pearson correlation coefficient for the two measures was 0.995 ± 0.0112. The correlation between the two measures indicates that only one of the two measures need to be measured. Because φ_g does not violate the one-pass requirement of SDCPs, and is computationally less complex than φ_v, it should be preferred over φ_v.

Of the three saturation components for φ_v and φ_g, the components using an upper bound of one, i.e. $\varphi_{0.1,1_v}$ and $\varphi_{0.1,1_g}$, correlated the most with their corresponding saturation measures for the problem domains. The average Pearson correlation coefficients for φ_v and $\varphi_{0.1,1_v}$ was 0.9302 ± 0.2506. The average Pearson correlation coefficients for φ_g and $\varphi_{0.1,1_g}$ was 0.9118 ± 0.2652. The other two components, however, correlated significantly less with a maximum average Pearson correlation coefficient of 0.7317 ± 0.3506. This meant that the activation values of the classifiers were mostly in the range $[0, 1]$ most often regardless of classifier or problem domain. Using only the $\varphi_{0.1,1}$ component as a measure of saturation in the hidden neurons should, therefore, be sufficient for streamed data classifiers using ReLU activation functions.

The complete saturation of QPSO-N was confirmed by the results of the saturation measures. This supported the explanation in Section 9.1 as to why the QPSO-N showed abnormal accuracy performance.

The BP classifiers saturated significantly less than the QPSO classifiers the majority of the time. Thus the BP weights adjustment algorithm was less prone to saturation than

Table 9.5: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for saturation performance measures

(a) Saturation in hidden neurons based on generalisation set (φ_g)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.6774±0.0707**	0.6996±0.0860	0.6787±0.0519	*0.9978±0.0022*	0.7872±0.0095	0.7924±0.0189
	1(88.75/3.75/7.50)	3(61.25/7.50/31.25)	2(76.25/6.25/17.50)	*6(0.00/0.00/100.00)*	4(31.25/8.75/60.00)	5(25.00/8.75/66.25)
Sphere	**0.6843±0.0667**	0.6947±0.0716	0.6864±0.0708	*0.9988±0.0016*	0.7815±0.0107	0.8524±0.0401
	1(73.75/23.75/2.50)	3(58.75/21.25/20.00)	2(72.50/23.75/3.75)	*6(0.00/0.00/100.00)*	4(40.00/1.25/58.75)	5(20.00/0.00/80.00)
Thresholds	**0.6548±0.0898**	0.6588±0.0930	0.7046±0.1173	*0.9967±0.0036*	0.7135±0.0063	0.8026±0.0179
	1(72.50/21.25/6.25)	2(68.75/20.00/11.25)	4(45.00/8.75/46.25)	*6(0.00/0.00/100.00)*	3(62.50/3.75/33.75)	5(23.75/1.25/75.00)
SEA	**0.7072±0.0801**	0.7137±0.0799	0.7105±0.0869	*0.9987±0.0015*	0.7542±0.0037	0.7565±0.0111
	1(55.00/28.75/16.25)	4(43.75/25.00/31.25)	2(58.75/21.25/20.00)	*6(0.00/0.00/100.00)*	5(46.25/20.00/33.75)	3(27.50/42.50/30.00)
Electricity	0.8389±0.0339	0.8615±0.0472	**0.4093±0.2423**	*0.9996±0.0005*	0.7236±0.0049	0.7376±0.0067
	4(36.25/3.75/60.00)	5(20.00/3.75/76.25)	**1(97.50/0.00/2.50)**	*6(0.00/0.00/100.00)*	2(81.25/0.00/18.75)	3(61.25/0.00/38.75)

(b) Saturation in hidden neurons based on validation set (φ_v)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.6777±0.0707**	0.6999±0.0859	0.6789±0.0518	*0.9980±0.0021*	0.7872±0.0096	0.7918±0.0189
	1(87.50/5.00/7.50)	3(61.25/8.75/30.00)	2(76.25/6.25/17.50)	*6(0.00/0.00/100.00)*	4(31.25/8.75/60.00)	5(25.00/8.75/66.25)
Sphere	**0.6853±0.0671**	0.6953±0.0716	0.6877±0.0716	*0.9989±0.0014*	0.7823±0.0096	0.8558±0.0385
	1(73.75/23.75/2.50)	3(58.75/21.25/20.00)	2(72.50/23.75/3.75)	*6(0.00/0.00/100.00)*	4(40.00/1.25/58.75)	5(20.00/0.00/80.00)
Thresholds	**0.6679±0.0970**	0.6700±0.0990	0.7162±0.1236	*0.9969±0.0033*	0.7147±0.0061	0.8098±0.0193
	1(70.00/23.75/6.25)	2(66.25/22.50/11.25)	4(45.00/11.25/43.75)	*6(0.00/0.00/100.00)*	3(62.50/6.25/31.25)	5(23.75/1.25/75.00)
SEA	**0.7079±0.0794**	0.7144±0.0792	0.7113±0.0864	*0.9989±0.0013*	0.7582±0.0053	0.7683±0.0107
	1(56.25/27.50/16.25)	3(45.00/25.00/30.00)	2(60.00/20.00/20.00)	*6(0.00/0.00/100.00)*	4(53.75/12.50/33.75)	5(26.25/32.50/41.25)
Electricity	0.8466±0.0329	0.8689±0.0452	**0.4093±0.2424**	*0.9996±0.0004*	0.7237±0.0048	0.7378±0.0067
	4(36.25/3.75/60.00)	5(20.00/3.75/76.25)	**1(97.50/0.00/2.50)**	*6(0.00/0.00/100.00)*	2(80.00/1.25/18.75)	3(61.25/1.25/37.50)

(c) Saturation in hidden neurons based on generalisation set and upper bound of 1 ($\varphi_{0.1,1_g}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.6806±0.0676**	0.7028±0.0830	0.6814±0.0496	*0.9996±0.0004*	0.7872±0.0095	0.7924±0.0189
	1(87.50/5.00/7.50)	3(61.25/8.75/30.00)	2(76.25/6.25/17.50)	*6(0.00/0.00/100.00)*	4(31.25/8.75/60.00)	5(25.00/8.75/66.25)
Sphere	**0.6868±0.0659**	0.6968±0.0704	0.6894±0.0706	*0.9997±0.0003*	0.7846±0.0098	0.8639±0.0364
	1(72.50/25.00/2.50)	3(58.75/23.75/17.50)	2(71.25/25.00/3.75)	*6(0.00/0.00/100.00)*	4(40.00/1.25/58.75)	5(20.00/0.00/80.00)
Thresholds	**0.6756±0.0989**	0.6762±0.0994	0.7273±0.1287	*0.9989±0.0012*	0.7186±0.0061	0.8394±0.0235
	1(67.50/23.75/8.75)	**1(67.50/23.75/8.75)**	3(43.75/13.75/42.50)	*5(0.00/0.00/100.00)*	2(65.00/3.75/31.25)	4(22.50/2.50/75.00)
SEA	**0.7103±0.0780**	0.7165±0.0780	0.7136±0.0852	*0.9997±0.0003*	0.7665±0.0088	0.8764±0.0143
	1(65.00/22.50/12.50)	3(53.75/18.75/27.50)	2(67.50/18.75/13.75)	*6(0.00/0.00/100.00)*	4(61.25/5.00/33.75)	5(20.00/0.00/80.00)
Electricity	0.8390±0.0339	0.8616±0.0472	**0.4603±0.3042**	*0.9999±0.0001*	0.7237±0.0049	0.7376±0.0067
	4(41.25/3.75/55.00)	5(22.50/5.00/72.50)	2(81.25/1.25/17.50)	*6(0.00/0.00/100.00)*	**1(85.00/0.00/15.00)**	3(65.00/0.00/35.00)

the QPSO weights adjustment algorithm. Using ReLU activation functions with the BP weights adjustment algorithm, therefore, is effective in handling saturation. However, the same was not true for the QPSO weights adjustment algorithm. This was evident from the fact that QPSO-N saturated almost completely all the time.

Regularisation helped to reduce saturation in both the regularised BP classifiers and the regularised QPSO classifiers. Regularisation, however, was more successful in reducing saturation for the regularised QPSO classifiers than for the regularised BP classifiers.

BP-WE had the lowest levels of saturation for the BP classifiers. QPSO-WD had the lowest levels of saturation for the QPSO classifiers. However, WD caused the BP weights adjustment algorithm to become less effective at handling saturation. Hence, the behaviour of the regularisation approaches differed over different weights adjustment algorithms.

The regularised QPSO classifiers had significantly more consistent saturation levels than the BP classifiers across the problem domains, difficulties, and environments. Hence, the regularised QPSO classifiers were better at controlling saturation in the hidden neurons, than the BP classifiers.

Noise caused the BP classifiers to saturate more. On the other hand, noise caused the QPSO classifiers to saturate less. On the other hand, problem dimensionality did not have any significant effect on the saturation levels.

Furthermore, the problem difficulty results show that the more patterns there were in the SDCP, the more saturated the classifiers became. This relationship, however, was much more prevalent for the BP classifiers, than for the QPSO classifiers. On the other hand, the problem environment results showed that as temporal severity decreased and spatial severity increased, the more saturated the classifiers became. The BP classifiers were more susceptible to this phenomenon than the QPSO classifiers.

(d) Saturation in hidden neurons based on validation set and upper bound of 1 ($\varphi_{0.1,1_v}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.6805±0.0679**	0.7027±0.0832	0.6813±0.0497	*0.9997±0.0003*	0.7872±0.0096	0.7918±0.0189
	1(87.50/5.00/7.50)	3(61.25/8.75/30.00)	2(76.25/6.25/17.50)	*6(0.00/0.00/100.00)*	4(31.25/8.75/60.00)	5(25.00/8.75/66.25)
Sphere	**0.6867±0.0661**	0.6967±0.0706	0.6893±0.0707	*0.9998±0.0003*	0.7837±0.0092	0.8638±0.0364
	1(72.50/25.00/2.50)	3(58.75/23.75/17.50)	2(71.25/25.00/3.75)	*6(0.00/0.00/100.00)*	4(40.00/1.25/58.75)	5(20.00/0.00/80.00)
Thresholds	**0.6751±0.0992**	0.6757±0.0998	0.7269±0.1291	*0.9990±0.0011*	0.7176±0.0059	0.8390±0.0235
	1(67.50/25.00/7.50)	**1(67.50/25.00/7.50)**	3(43.75/13.75/42.50)	*5(0.00/0.00/100.00)*	2(62.50/6.25/31.25)	4(22.50/2.50/75.00)
SEA	**0.7105±0.0775**	0.7168±0.0776	0.7137±0.0847	*0.9998±0.0003*	0.7664±0.0088	0.8765±0.0143
	1(65.00/22.50/12.50)	3(53.75/18.75/27.50)	2(67.50/17.50/15.00)	*6(0.00/0.00/100.00)*	4(62.50/3.75/33.75)	5(20.00/0.00/80.00)
Electricity	0.8467±0.0329	0.8690±0.0452	**0.4603±0.3043**	*0.9999±0.0001*	0.7238±0.0048	0.7378±0.0067
	4(41.25/3.75/55.00)	5(21.25/6.25/72.50)	2(81.25/2.50/16.25)	*6(0.00/0.00/100.00)*	**1(83.75/1.25/15.00)**	3(65.00/1.25/33.75)

(e) Saturation in hidden neurons based on generalisation set and upper bound of 5 ($\varphi_{0.1,5_g}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.9098±0.0265**	0.9167±0.0312	0.9103±0.0210	*0.9986±0.0014*	0.9547±0.0028	0.9536±0.0048
	1(87.50/5.00/7.50)	3(62.50/10.00/27.50)	2(78.75/2.50/18.75)	*6(0.00/0.00/100.00)*	5(25.00/7.50/67.50)	4(30.00/7.50/62.50)
Sphere	**0.9012±0.0212**	0.9082±0.0243	0.9013±0.0208	*0.9993±0.0009*	0.9247±0.0074	0.9381±0.0284
	2(72.50/25.00/2.50)	3(52.50/16.25/31.25)	**1(75.00/22.50/2.50)**	*6(0.00/0.00/100.00)*	5(32.50/7.50/60.00)	4(22.50/18.75/58.75)
Thresholds	**0.8488±0.0247**	0.8547±0.0267	0.8715±0.0345	*0.9976±0.0025*	0.9120±0.0054	0.8948±0.0113
	1(83.75/16.25/0.00)	2(76.25/16.25/7.50)	3(51.25/16.25/32.50)	*6(0.00/0.00/100.00)*	5(20.00/10.00/70.00)	4(31.25/16.25/52.50)
SEA	0.9102±0.0339	0.9138±0.0333	0.9111±0.0340	*0.9992±0.0009*	0.8907±0.0136	**0.7939±0.0114**
	2(45.00/23.75/31.25)	5(25.00/20.00/55.00)	3(36.25/27.50/36.25)	*6(0.00/0.00/100.00)*	4(57.50/5.00/37.50)	**1(96.25/3.75/0.00)**
Electricity	0.9647±0.0085	0.9699±0.0108	**0.7677±0.0754**	*0.9998±0.0003*	0.9235±0.0027	0.9339±0.0034
	4(38.75/1.25/60.00)	5(20.00/1.25/78.75)	**1(100.00/0.00/0.00)**	*6(0.00/0.00/100.00)*	2(80.00/0.00/20.00)	3(60.00/0.00/40.00)

(f) Saturation in hidden neurons based on validation set and upper bound of 5 ($\varphi_{0.1,5_v}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.9099±0.0264**	0.9167±0.0311	0.9104±0.0209	*0.9986±0.0014*	0.9549±0.0027	0.9537±0.0048
	1(87.50/5.00/7.50)	3(62.50/10.00/27.50)	2(78.75/2.50/18.75)	*6(0.00/0.00/100.00)*	5(25.00/7.50/67.50)	4(30.00/7.50/62.50)
Sphere	**0.9013±0.0212**	0.9083±0.0242	0.9014±0.0208	*0.9993±0.0009*	0.9248±0.0074	0.9381±0.0284
	2(72.50/25.00/2.50)	3(52.50/16.25/31.25)	**1(75.00/22.50/2.50)**	*6(0.00/0.00/100.00)*	5(32.50/7.50/60.00)	4(22.50/18.75/58.75)
Thresholds	**0.8489±0.0246**	0.8547±0.0266	0.8716±0.0344	*0.9976±0.0025*	0.9123±0.0054	0.8948±0.0112
	1(83.75/16.25/0.00)	2(76.25/16.25/7.50)	3(51.25/16.25/32.50)	*6(0.00/0.00/100.00)*	5(20.00/10.00/70.00)	4(31.25/16.25/52.50)
SEA	0.9104±0.0338	0.9140±0.0332	0.9113±0.0339	*0.9992±0.0009*	0.8905±0.0135	**0.7939±0.0113**
	2(45.00/23.75/31.25)	5(22.50/21.25/56.25)	4(36.25/27.50/36.25)	*6(0.00/0.00/100.00)*	3(58.75/6.25/35.00)	**1(96.25/3.75/0.00)**
Electricity	0.9665±0.0081	0.9716±0.0103	**0.7677±0.0754**	*0.9997±0.0003*	0.9235±0.0027	0.9339±0.0034
	4(38.75/1.25/60.00)	5(20.00/1.25/78.75)	**1(100.00/0.00/0.00)**	*6(0.00/0.00/100.00)*	2(80.00/0.00/20.00)	3(60.00/0.00/40.00)

(g) Saturation in hidden neurons based on generalisation set and upper bound of 10 ($\varphi_{0.1,10_g}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.9549±0.0133**	0.9583±0.0156	0.9552±0.0105	*0.9980±0.0020*	0.9773±0.0014	0.9768±0.0024
	1(87.50/5.00/7.50)	3(62.50/10.00/27.50)	2(78.75/2.50/18.75)	*6(0.00/0.00/100.00)*	5(25.00/7.50/67.50)	4(30.00/7.50/62.50)
Sphere	**0.9506±0.0106**	0.9541±0.0122	0.9507±0.0104	*0.9990±0.0013*	0.9623±0.0037	0.9632±0.0147
	2(72.50/25.00/2.50)	3(52.50/16.25/31.25)	**1(75.00/22.50/2.50)**	*6(0.00/0.00/100.00)*	5(32.50/7.50/60.00)	4(22.50/18.75/58.75)
Thresholds	**0.9241±0.0121**	0.9272±0.0133	0.9348±0.0163	*0.9977±0.0025*	0.9559±0.0027	0.9265±0.0197
	1(73.75/20.00/6.25)	2(65.00/21.25/13.75)	4(48.75/11.25/40.00)	*6(0.00/0.00/100.00)*	5(20.00/5.00/75.00)	3(55.00/17.50/27.50)
SEA	0.9551±0.0170	0.9569±0.0167	0.9556±0.0170	*0.9990±0.0012*	0.9449±0.0071	**0.8586±0.0194**
	2(45.00/22.50/32.50)	5(22.50/20.00/57.50)	4(36.25/26.25/37.50)	*6(0.00/0.00/100.00)*	3(58.75/6.25/35.00)	**1(100.00/0.00/0.00)**
Electricity	0.9824±0.0043	0.9850±0.0054	**0.8838±0.0377**	*0.9997±0.0004*	0.9617±0.0014	0.9669±0.0017
	4(38.75/1.25/60.00)	5(20.00/1.25/78.75)	**1(100.00/0.00/0.00)**	*6(0.00/0.00/100.00)*	2(80.00/0.00/20.00)	3(60.00/0.00/40.00)

(h) Saturation in hidden neurons based on validation set and upper bound of 10 ($\varphi_{0.1,10_v}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	**0.9550±0.0132**	0.9584±0.0155	0.9552±0.0104	*0.9980±0.0020*	0.9774±0.0013	0.9769±0.0024
	1(87.50/5.00/7.50)	3(62.50/10.00/27.50)	2(78.75/2.50/18.75)	*6(0.00/0.00/100.00)*	5(25.00/7.50/67.50)	4(30.00/7.50/62.50)
Sphere	**0.9506±0.0106**	0.9541±0.0121	0.9507±0.0104	*0.9990±0.0013*	0.9624±0.0037	0.9632±0.0147
	2(72.50/25.00/2.50)	3(52.50/16.25/31.25)	**1(75.00/22.50/2.50)**	*6(0.00/0.00/100.00)*	5(32.50/7.50/60.00)	4(22.50/18.75/58.75)
Thresholds	**0.9242±0.0121**	0.9273±0.0132	0.9349±0.0163	*0.9977±0.0025*	0.9561±0.0027	0.9266±0.0197
	1(73.75/20.00/6.25)	2(65.00/21.25/13.75)	4(48.75/11.25/40.00)	*6(0.00/0.00/100.00)*	5(20.00/5.00/75.00)	3(55.00/17.50/27.50)
SEA	0.9552±0.0169	0.9570±0.0166	0.9557±0.0169	*0.9990±0.0012*	0.9447±0.0070	**0.8585±0.0194**
	2(45.00/22.50/32.50)	5(22.50/20.00/57.50)	4(36.25/26.25/37.50)	*6(0.00/0.00/100.00)*	3(58.75/6.25/35.00)	**1(100.00/0.00/0.00)**
Electricity	0.9832±0.0041	0.9858±0.0051	**0.8839±0.0377**	*0.9996±0.0004*	0.9618±0.0014	0.9670±0.0017
	4(38.75/1.25/60.00)	5(20.00/1.25/78.75)	**1(100.00/0.00/0.00)**	*6(0.00/0.00/100.00)*	2(80.00/0.00/20.00)	3(60.00/0.00/40.00)

Table 9.6: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for saturation performance measures

(a) Saturation in hidden neurons based on generalisation set (φ_g)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	0.7753±0.0546 2(67.00/9.00/24.00)	0.8041±0.0664 4(41.00/12.00/47.00)	0.7538±0.0695 3(59.00/10.00/31.00)	*0.9998±0.0001* *6(0.00/0.00/100.00)*	**0.7510±0.0307** **1(72.00/5.00/23.00)**	0.8048±0.0595 5(36.00/14.00/50.00)
Moderate-I	0.7564±0.0619 2(60.00/21.00/19.00)	0.7644±0.0666 3(47.00/21.00/32.00)	**0.6288±0.2396** **1(65.00/17.00/18.00)**	*0.9992±0.0006* *6(0.00/0.00/100.00)*	0.7563±0.0323 4(55.00/9.00/36.00)	0.7916±0.0445 5(31.00/16.00/53.00)
Moderate-II	0.6813±0.0735 2(73.00/10.00/17.00)	0.6939±0.0768 3(57.00/8.00/35.00)	**0.6245±0.1326** **1(80.00/7.00/13.00)**	*0.9983±0.0017* *6(0.00/0.00/100.00)*	0.7502±0.0318 4(42.00/7.00/51.00)	0.7851±0.0446 5(29.00/6.00/65.00)
Hard	0.6371±0.1113 2(61.00/25.00/14.00)	0.6401±0.1128 3(57.00/21.00/22.00)	**0.5444±0.1464** **1(76.00/14.00/10.00)**	*0.9960±0.0031* *6(0.00/0.00/100.00)*	0.7505±0.0300 4(40.00/6.00/54.00)	0.7716±0.0222 5(30.00/6.00/64.00)

(b) Saturation in hidden neurons based on validation set (φ_v)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	0.7807±0.0548 2(67.00/9.00/24.00)	0.8081±0.0655 4(41.00/13.00/46.00)	0.7581±0.0740 3(59.00/10.00/31.00)	*0.9998±0.0001* *6(0.00/0.00/100.00)*	**0.7514±0.0304** **1(74.00/3.00/23.00)**	0.8089±0.0587 5(35.00/13.00/52.00)
Moderate-I	0.7627±0.0635 2(60.00/21.00/19.00)	0.7703±0.0682 3(47.00/21.00/32.00)	**0.6326±0.2418** **1(66.00/16.00/18.00)**	*0.9993±0.0006* *6(0.00/0.00/100.00)*	0.7581±0.0323 4(55.00/11.00/34.00)	0.7965±0.0444 5(31.00/13.00/56.00)
Moderate-II	0.6852±0.0753 2(73.00/10.00/17.00)	0.6978±0.0790 3(57.00/8.00/35.00)	**0.6264±0.1337** **1(80.00/7.00/13.00)**	*0.9983±0.0017* *6(0.00/0.00/100.00)*	0.7507±0.0314 4(45.00/4.00/51.00)	0.7889±0.0445 5(29.00/3.00/68.00)
Hard	0.6397±0.1123 2(59.00/27.00/14.00)	0.6425±0.1139 3(56.00/23.00/21.00)	**0.5455±0.1467** **1(76.00/16.00/8.00)**	*0.9963±0.0029* *6(0.00/0.00/100.00)*	0.7526±0.0301 4(40.00/6.00/54.00)	0.7764±0.0230 5(30.00/6.00/64.00)

Table 9.7: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for saturation performance measures

(a) Saturation in hidden neurons based on generalisation set (φ_g)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**0.6303±0.0655**	0.6369±0.0682	0.6481±0.0773	*0.9968±0.0027*	0.7524±0.0317	0.7908±0.0319
	1(75.56/19.44/5.00)	2(66.67/16.11/17.22)	3(68.33/12.22/19.44)	*6(0.00/0.00/100.00)*	4(36.11/7.22/56.67)	5(22.22/7.22/70.56)
Abrupt	0.7932±0.0566	0.8320±0.0724	0.7753±0.0605	*0.9999±0.0000*	**0.7475±0.0289**	0.7958±0.0665
	2(65.71/5.71/28.57)	5(31.43/11.43/57.14)	3(57.14/5.71/37.14)	*6(0.00/0.00/100.00)*	**1(77.14/8.57/14.29)**	4(42.86/20.00/37.14)
Chaotic	0.7773±0.0539	0.7918±0.0584	**0.6020±0.2332**	*0.9995±0.0004*	0.7525±0.0309	0.7844±0.0528
	2(55.14/15.14/29.73)	4(38.38/15.68/45.95)	**1(74.05/12.97/12.97)**	*6(0.00/0.00/100.00)*	3(63.24/5.95/30.81)	5(38.38/11.89/49.73)

(b) Saturation in hidden neurons based on validation set (φ_v)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	**0.6340±0.0672**	0.6402±0.0696	0.6514±0.0802	*0.9970±0.0025*	0.7539±0.0313	0.7965±0.0306
	1(73.89/21.11/5.00)	2(65.56/17.78/16.67)	3(68.33/13.33/18.33)	*6(0.00/0.00/100.00)*	4(38.33/6.11/55.56)	5(22.22/5.00/72.78)
Abrupt	0.7976±0.0562	0.8348±0.0715	0.7783±0.0643	*0.9999±0.0000*	**0.7478±0.0287**	0.7996±0.0658
	2(65.71/5.71/28.57)	5(31.43/11.43/57.14)	3(57.14/5.71/37.14)	*6(0.00/0.00/100.00)*	**1(80.00/5.71/14.29)**	4(42.86/17.14/40.00)
Chaotic	0.7826±0.0562	0.7968±0.0602	**0.6042±0.2351**	*0.9995±0.0004*	0.7535±0.0310	0.7877±0.0530
	2(55.68/14.59/29.73)	4(38.92/15.68/45.41)	**1(74.59/12.43/12.97)**	*6(0.00/0.00/100.00)*	3(63.24/5.95/30.81)	5(37.84/10.81/51.35)

Table 9.8: MWU-based pairwise comparison of the classifiers for saturation performance measures (Wins/Ties/Losses percentages)

(a) Saturation in hidden neurons based on generalisation set (φ_g)

Classifier	**Classifier**					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	56.25/43.75/0.00	36.25/27.50/36.25	100.00/0.00/0.00	63.75/1.25/35.00	70.00/8.75/21.25
BP-WD	0.00/43.75/56.25	—	25.00/23.75/51.25	100.00/0.00/0.00	58.75/2.50/38.75	68.75/7.50/23.75
BP-WE	36.25/27.50/36.25	51.25/23.75/25.00	—	100.00/0.00/0.00	78.75/1.25/20.00	83.75/7.50/8.75
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	0.00/0.00/100.00	0.00/0.00/100.00
QPSO-WD	35.00/1.25/63.75	38.75/2.50/58.75	20.00/1.25/78.75	100.00/0.00/0.00	—	67.50/28.75/3.75
QPSO-WE	21.25/8.75/70.00	23.75/7.50/68.75	8.75/7.50/83.75	100.00/0.00/0.00	3.75/28.75/67.50	—
Overall rank	2(65.25/16.25/18.50)	3(50.50/15.50/34.00)	**1(70.00/12.00/18.00)**	*6(0.00/0.00/100.00)*	4(52.25/6.75/41.00)	5(31.50/10.50/58.00)

(b) Saturation in hidden neurons based on validation set (φ_v)

Classifier	**Classifier**					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	55.00/45.00/0.00	35.00/28.75/36.25	100.00/0.00/0.00	62.50/2.50/35.00	71.25/7.50/21.25
BP-WD	0.00/45.00/55.00	—	23.75/25.00/51.25	100.00/0.00/0.00	57.50/3.75/38.75	70.00/7.50/22.50
BP-WE	36.25/28.75/35.00	51.25/25.00/23.75	—	100.00/0.00/0.00	78.75/1.25/20.00	85.00/6.25/8.75
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	0.00/0.00/100.00	0.00/0.00/100.00
QPSO-WD	35.00/2.50/62.50	38.75/3.75/57.50	20.00/1.25/78.75	100.00/0.00/0.00	—	73.75/22.50/3.75
QPSO-WE	21.25/7.50/71.25	22.50/7.50/70.00	8.75/6.25/85.00	100.00/0.00/0.00	3.75/22.50/73.75	—
Overall rank	2(64.75/16.75/18.50)	3(50.25/16.25/33.50)	**1(70.25/12.25/17.50)**	*6(0.00/0.00/100.00)*	4(53.50/6.00/40.50)	5(31.25/8.75/60.00)

9.3 Complexity performance analysis

This section analyses the collective means of the three complexity performance measures $n_{s_{or}}$, $n_{h_{or}}$ and Ω_r (refer to sections 8.4.4 and 8.4.5). The key objectives of the complexity performance analysis are to determine:

1. How well the classifiers reduced their computational complexity. The complexity reduction measure, Ω_r, is used as an indicator for the reduction in computational complexity.

2. How well the classifiers maintained their structural complexity. The oversize ratios for synapses ($n_{s_{or}}$) and hidden neurons ($n_{h_{or}}$) are used as indicators for the level of structural complexity.

3. Whether the regularised classifiers were less complex than their non-regularised counterparts.

4. Whether the regularised QPSO classifiers were less complex than their regularised BP counterparts.

5. Whether or not WE was better than WD, at reducing the complexity of the models.

6. Whether or not regularisation managed to find the optimal architectures for the various problem domains.

7. How the computational and structural complexity measures compared.

8. The effects of problem difficulty, environment, dimensionality, noise and saturation on the complexity performance of the classifiers.

9. The relation between the accuracy and complexity of the classifiers.

Table 9.9 presents the statistical and MWU-based ranking analysis of the complexity performance with regards to the classifiers for each of the problem domains. Table 9.10 presents the statistical and MWU-based ranking analysis of the complexity performance with regards to the classifiers for each of the problem difficulties. Table 9.11 presents the statistical and MWU-based ranking analysis of the complexity performance with regards

to the classifiers for each of the problem environments. Table 9.12 presents the overall MWU-based ranking results of how the classifiers faired against each other based on their complexity performance. The complexity performance results in Tables 9.9 to 9.12 show the following:

The synapse oversize ratio ($n_{s_{or}}$) and the complexity reduction measure (Ω_r) produced the same rankings for the problem domains, problem difficulties, and problem environments. Therefore, either measure can be used to estimate the lower bound computational complexity of a FFNN. Furthermore, The values for $n_{s_{or}}$ never exceeded 1.67. Architecture selection should therefore be employed by FFNN streamed data classifiers.

The mean values for $n_{s_{or}}$ were significantly lower than the mean values of hidden neuron oversize ratio ($n_{h_{or}}$) for all the problem domains, problem difficulties, and problem environments. Thus, as expected, synapses have a higher chance of being irrelevant than hidden neurons. Architecture selection algorithms for SDCP should, therefore, function at a synapse level rather than at a neuron level.

The above observations also provided evidence that the pruning algorithm proposed by Engelbrecht [31] was able to determine the effective model for the FFNN classifiers.

The BP classifiers had the worst complexity performance results, therefore, the BP weights adjustment algorithm was not naturally proficient at maintaining complexity. On the other hand, QPSO-N had the best complexity performance, however, QPSO-N was completely saturated. Complete saturation thus improves complexity performance, because complete saturation degrades information capacity thereby rendering various hidden neurons irrelevant and prune-able.

WE helped both weight adjustment algorithms, i.e. BP and QPSO, to achieve better complexity performance, than WD. However, not all regularisation approaches aided the complexity performance of the BP weights adjustment algorithm, for example WD.

QPSO-WD and QPSO-WE were outperformed by QPSO-N with regards to complexity performance. However, both classifiers maintained reasonable saturation levels (refer to Section 9.2). Thus, regularisation helped to boost the complexity performance of the QPSO weights adjustment algorithm without allowing unwanted saturation in the hidden neurons.

Generally, complexity and saturation levels correlated to the extent, that the more

Table 9.9: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for complexity performance measures

(a) Synapse oversize ratio ($n_{s_{or}}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	1.0353±0.4076 4(32.50/11.25/56.25)	0.9993±0.4344 2(48.75/12.50/38.75)	*1.0882±0.3281* *6(28.75/10.00/61.25)*	**0.4831±0.4538** **1(92.50/1.25/6.25)**	1.0830±0.0758 5(27.50/13.75/58.75)	1.0455±0.1257 3(38.75/13.75/47.50)
Sphere	*1.3240±0.1977* *6(0.00/17.50/82.50)*	1.3182±0.2016 5(3.75/18.75/77.50)	1.2483±0.2260 4(36.25/3.75/60.00)	**0.4353±0.3242** **1(92.50/5.00/2.50)**	0.8249±0.0937 3(61.25/7.50/31.25)	0.7404±0.1358 2(76.25/7.50/16.25)
Thresholds	*1.4926±0.1906* 5(2.50/23.75/73.75)	1.4923±0.1923 *6(1.25/23.75/75.00)*	1.3266±0.3675 4(31.25/13.75/55.00)	**0.5722±0.2152** **1(95.00/5.00/0.00)**	0.9077±0.0542 3(56.25/15.00/28.75)	0.8847±0.0706 2(63.75/18.75/17.50)
SEA	1.6581±0.1540 5(2.50/18.75/78.75)	*1.6638±0.1553* *6(1.25/17.50/81.25)*	1.5526±0.1951 4(36.25/3.75/60.00)	**0.3129±0.2499** **1(97.50/2.50/0.00)**	0.8123±0.1015 2(67.50/13.75/18.75)	0.8336±0.1582 3(60.00/13.75/26.25)
Electricity	*1.0709±0.3756* *6(0.00/6.25/93.75)*	0.9983±0.4437 5(21.25/7.50/71.25)	0.2669±0.1122 2(83.75/0.00/16.25)	**0.2473±0.2809** **1(93.75/2.50/3.75)**	0.7608±0.2385 4(35.00/3.75/61.25)	0.7394±0.2357 3(55.00/2.50/42.50)

(b) Hidden neuron oversize ratio ($n_{h_{or}}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	1.8140±0.1184 4(21.25/16.25/62.50)	1.8151±0.1176 5(12.50/15.00/72.50)	*1.8201±0.1115* *6(6.25/8.75/85.00)*	**1.2011±0.2073** **1(100.00/0.00/0.00)**	1.6515±0.0518 3(60.00/7.50/32.50)	1.6022±0.0859 2(72.50/7.50/20.00)
Sphere	1.8054±0.0871 5(6.25/27.50/66.25)	*1.8057±0.0861* *6(5.00/27.50/67.50)*	1.7788±0.1111 4(5.00/32.50/62.50)	**1.1468±0.1307** **1(97.50/2.50/0.00)**	1.3932±0.0423 3(60.00/10.00/30.00)	1.3627±0.0579 2(71.25/10.00/18.75)
Thresholds	*1.9997±0.0003* *6(0.00/26.25/73.75)*	1.9996±0.0004 5(1.25/26.25/72.50)	1.8337±0.2273 4(31.25/16.25/52.50)	**1.2406±0.1687** **1(100.00/0.00/0.00)**	1.6366±0.0277 3(56.25/0.00/43.75)	1.5513±0.0292 2(76.25/1.25/22.50)
SEA	1.9765±0.0282 5(2.50/25.00/72.50)	*1.9776±0.0264* *6(0.00/23.75/76.25)*	1.8996±0.0946 4(26.25/13.75/60.00)	**1.0949±0.0925** **1(100.00/0.00/0.00)**	1.3500±0.0442 2(65.00/15.00/20.00)	1.3749±0.0749 3(60.00/15.00/25.00)
Electricity	*1.9237±0.1198* *6(3.75/7.50/88.75)*	1.8465±0.2632 5(11.25/7.50/81.25)	1.3621±0.1805 2(66.25/7.50/26.25)	**1.0792±0.1110** **1(100.00/0.00/0.00)**	1.4457±0.0934 3(48.75/16.25/35.00)	1.4529±0.0979 4(42.50/16.25/41.25)

(c) Complexity reduction (Ω_r)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	38.2707±19.5678 4(32.50/10.00/57.50)	39.9280±20.7881 2(47.50/12.50/40.00)	*35.7858±15.8544* *6(27.50/10.00/62.50)*	**68.0819±22.4192** **1(92.50/2.50/5.00)**	37.2063±3.8543 5(28.75/11.25/60.00)	39.2882±6.4000 3(41.25/13.75/45.00)
Sphere	*25.0045±8.1887* *6(1.25/16.25/82.50)*	25.2092±8.3043 5(3.75/17.50/78.75)	28.0618±9.5303 4(36.25/3.75/60.00)	**65.5667±13.3104** **1(92.50/7.50/0.00)**	48.3571±3.9048 3(60.00/8.75/31.25)	51.7793±5.6133 2(75.00/8.75/16.25)
Thresholds	*13.4331±7.0227* 5(2.50/23.75/73.75)	13.4445±7.0883 *6(1.25/23.75/75.00)*	21.2988±15.8277 4(32.50/12.50/55.00)	**55.3341±9.6979** **1(96.25/3.75/0.00)**	38.8067±2.2554 3(56.25/7.50/36.25)	40.5509±2.8741 2(70.00/11.25/18.75)
SEA	10.7911±5.8345 5(3.75/17.50/78.75)	*10.5770±5.8643* *6(1.25/16.25/82.50)*	15.5920±8.1741 4(36.25/3.75/60.00)	**70.6273±10.1498** **1(97.50/2.50/0.00)**	49.3925±4.2119 2(68.75/12.50/18.75)	48.2965±6.6059 3(60.00/12.50/27.50)
Electricity	*36.9424±16.0253* *6(0.00/6.25/93.75)*	40.5884±20.0353 5(18.75/6.25/75.00)	74.4535±6.1764 2(82.50/1.25/16.25)	**77.9930±12.2729** **1(93.75/3.75/2.50)**	53.9616±10.4044 4(37.50/5.00/57.50)	54.7438±10.3306 3(53.75/5.00/41.25)

Table 9.10: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for complexity performance measures

(a) Synapse oversize ratio ($n_{s_{or}}$)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	*1.0001±0.4229* *6(12.00/10.00/78.00)*	0.9489±0.4850 5(26.00/11.00/63.00)	0.8042±0.4267 3(51.00/3.00/46.00)	**0.1467±0.1156** **1(100.00/0.00/0.00)**	0.7546±0.2130 4(38.00/8.00/54.00)	0.6771±0.2024 2(53.00/8.00/39.00)
Moderate-I	*1.4224±0.2995* *6(4.00/14.00/82.00)*	1.4041±0.3392 5(14.00/15.00/71.00)	1.1824±0.5020 4(48.00/3.00/49.00)	**0.3447±0.1930** **1(100.00/0.00/0.00)**	0.9392±0.1134 3(48.00/13.00/39.00)	0.9225±0.1199 2(58.00/11.00/31.00)
Moderate-II	*1.2433±0.2548* *6(12.00/12.00/76.00)*	1.2321±0.2663 5(17.00/12.00/71.00)	1.0308±0.4299 4(43.00/3.00/54.00)	**0.2972±0.1617** **1(100.00/0.00/0.00)**	0.8281±0.1232 3(49.00/5.00/46.00)	0.7993±0.1365 2(60.00/6.00/34.00)
Hard	*1.5991±0.1493* *6(2.00/26.00/72.00)*	1.5924±0.1520 5(4.00/26.00/70.00)	1.3685±0.5401 4(31.00/16.00/53.00)	**0.8520±0.2791** **1(77.00/13.00/10.00)**	0.9891±0.1074 3(63.00/17.00/20.00)	0.9960±0.1106 2(64.00/20.00/16.00)

(b) Hidden neuron oversize ratio ($n_{h_{or}}$)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	*1.8347±0.1388* *6(9.00/12.00/79.00)*	1.7778±0.2310 5(11.00/12.00/77.00)	1.5523±0.2595 4(41.00/5.00/54.00)	**1.0293±0.0276** **1(100.00/0.00/0.00)**	1.4437±0.1495 3(53.00/6.00/41.00)	1.3910±0.1216 2(65.00/7.00/28.00)
Moderate-I	*1.9454±0.0693* *6(3.00/22.00/75.00)*	1.9407±0.0715 5(6.00/19.00/75.00)	1.8067±0.2033 4(27.00/19.00/54.00)	**1.1235±0.0923** **1(100.00/0.00/0.00)**	1.5309±0.1282 3(60.00/9.00/31.00)	1.5105±0.0923 2(65.00/9.00/26.00)
Moderate-II	1.8673±0.1318 5(13.00/17.00/70.00)	*1.8688±0.1309* *6(4.00/18.00/78.00)*	1.7255±0.2082 4(26.00/13.00/61.00)	**1.0999±0.0767** **1(100.00/0.00/0.00)**	1.4670±0.1288 3(60.00/7.00/33.00)	1.4425±0.1023 2(66.00/7.00/27.00)
Hard	1.9681±0.0492 *6(2.00/31.00/67.00)*	*1.9683±0.0492* 5(3.00/31.00/66.00)	1.8708±0.1921 4(14.00/26.00/60.00)	**1.3574±0.1520** **1(98.00/2.00/0.00)**	1.5399±0.1269 3(59.00/17.00/24.00)	1.5312±0.1104 2(62.00/17.00/21.00)

(c) Complexity reduction (Ω_r)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	*38.3735±20.2503* *6(12.00/10.00/78.00)*	41.0578±23.6296 5(26.00/10.00/64.00)	48.4252±20.5454 3(52.00/2.00/46.00)	**79.2003±8.1788** **1(100.00/0.00/0.00)**	50.8954±11.0611 4(40.00/3.00/57.00)	54.4435±10.4101 2(55.00/5.00/40.00)
Moderate-I	*20.3844±13.4807* *6(5.00/12.00/83.00)*	21.2301±15.1635 5(11.00/14.00/75.00)	31.1449±23.3110 4(47.00/3.00/50.00)	**70.4918±10.7328** **1(100.00/0.00/0.00)**	42.8294±6.4122 3(51.00/8.00/41.00)	43.6601±6.3657 2(63.00/9.00/28.00)
Moderate-II	*28.0433±12.4052* *6(13.00/11.00/76.00)*	28.5030±12.9329 5(17.00/11.00/72.00)	37.7471±20.6881 4(43.00/3.00/54.00)	**72.5754±9.5302** **1(100.00/0.00/0.00)**	47.7451±6.4639 3(49.00/5.00/46.00)	49.0907±6.5124 2(60.00/6.00/34.00)
Hard	*12.7523±7.1299* *6(2.00/26.00/72.00)*	13.0067±7.2815 5(4.00/26.00/70.00)	22.8363±24.8065 4(30.00/17.00/53.00)	**47.8150±13.3698** **1(78.00/16.00/6.00)**	40.7094±5.4741 3(61.00/20.00/19.00)	40.5326±5.3570 2(62.00/21.00/17.00)

saturated the hidden neurons of the classifiers were, the less complexity they required. For instance, the regularised QPSO classifiers had an average Pearson correlation coefficient between φ_g and Ω_r of 0.6616±0.3112.

Noise affected complexity performance. That is, the more a classifier captured noise,

Table 9.11: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for complexity performance measures

(a) Synapse oversize ratio ($n_{s_{or}}$)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	*1.4498±0.2815*	1.4457±0.2863	1.3981±0.3051	**0.6064±0.3403**	0.9098±0.1353	0.9010±0.1566
	6(7.78/21.11/71.11)	5(9.44/21.11/69.44)	4(26.67/11.11/62.22)	**1(90.00/6.11/3.89)**	3(59.44/12.78/27.78)	2(62.78/15.56/21.67)
Abrupt	*0.8959±0.4648*	0.8327±0.5350	0.6786±0.4619	**0.1008±0.0971**	0.6469±0.2452	0.5615±0.2032
	6(8.57/8.57/82.86)	5(25.71/11.43/62.86)	3(57.14/0.00/42.86)	**1(100.00/0.00/0.00)**	4(37.14/8.57/54.29)	2(54.29/5.71/40.00)
Chaotic	*1.2658±0.3593*	1.2345±0.4045	0.8821±0.5267	**0.2778±0.2210**	0.8902±0.1557	0.8522±0.1697
	6(7.03/11.35/81.62)	5(18.92/11.89/69.19)	3(56.76/2.70/40.54)	**1(97.30/1.08/1.62)**	4(42.16/9.19/48.65)	2(55.68/8.11/36.22)

(b) Hidden neuron oversize ratio ($n_{h_{or}}$)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	1.9138±0.1156	*1.9147±0.1147*	1.8729±0.1381	**1.2444±0.1748**	1.5200±0.1437	1.4892±0.1199
	5(7.22/26.67/66.11)	*6(3.33/27.22/69.44)*	4(12.78/21.11/66.11)	**1(98.89/1.11/0.00)**	3(61.11/8.89/30.00)	2(69.44/9.44/21.11)
Abrupt	*1.8063±0.1611*	1.7128±0.3011	1.4687±0.3064	**1.0133±0.0134**	1.3793±0.1532	1.3276±0.1179
	6(5.71/11.43/82.86)	5(17.14/11.43/71.43)	4(45.71/5.71/48.57)	**1(100.00/0.00/0.00)**	3(48.57/8.57/42.86)	2(60.00/8.57/31.43)
Chaotic	*1.9127±0.1023*	1.8971±0.1364	1.6595±0.2419	**1.0895±0.0891**	1.4935±0.1193	1.4756±0.1024
	5(6.49/16.22/77.30)	*6(6.49/14.59/78.92)*	4(37.30/12.43/50.27)	**1(100.00/0.00/0.00)**	3(56.76/10.81/32.43)	2(60.54/10.81/28.65)

(c) Complexity reduction (Ω_r)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	*18.3308±12.8432*	18.4948±13.0873	20.6277±13.5472	**57.4849±15.6555**	42.8254±6.6542	43.4609±7.0773
	6(8.33/20.56/71.11)	5(9.44/20.56/70.00)	4(26.67/11.11/62.22)	**1(90.56/7.22/2.22)**	3(58.33/12.78/28.89)	2(63.33/14.44/22.22)
Abrupt	*42.9507±22.4625*	46.4366±26.5322	54.4125±22.7194	**81.3608±7.3639**	56.2029±12.7666	60.0649±10.8693
	6(8.57/8.57/82.86)	5(25.71/8.57/65.71)	3(57.14/0.00/42.86)	**1(100.00/0.00/0.00)**	4(40.00/5.71/54.29)	2(54.29/5.71/40.00)
Chaotic	*27.8515±16.1282*	29.3267±18.3145	45.3942±24.4885	**74.6667±10.5461**	46.1744±7.7276	47.8241±7.9971
	6(7.57/10.27/82.16)	5(17.30/11.35/71.35)	3(56.22/2.70/41.08)	**1(97.30/1.62/1.08)**	4(44.32/5.95/49.73)	2(57.84/7.03/35.14)

the less the classifier could reduce model complexity. This could be seen in the complexity results of the BP classifiers for the SEA problems begin worse than for other problems. The complexity performance of the regularised QPSO classifiers did not degrade when faced with noise in the SEA problems, because regularisation managed to unlearn the noise as per their accuracy performance (refer to Section 9.1).

An increase in the problem difficulty resulted in a decrease in the complexity performance of the classifiers, and vice versa. Thus, the problem difficulty classification scheme in Section 8.3.6 could also be used to determine the complexity performance of

Table 9.12: MWU-based pairwise comparison of the classifiers for complexity performance measures (Wins/Ties/Losses percentages)

(a) Synapse oversize ratio ($n_{s_{or}}$)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	3.75/63.75/32.50	13.75/11.25/75.00	0.00/0.00/100.00	10.00/1.25/88.75	10.00/1.25/88.75
BP-WD	32.50/63.75/3.75	—	13.75/13.75/72.50	0.00/0.00/100.00	15.00/2.50/82.50	15.00/0.00/85.00
BP-WE	75.00/11.25/13.75	72.50/13.75/13.75	—	3.75/0.00/96.25	33.75/2.50/63.75	31.25/3.75/65.00
QPSO-N	100.00/0.00/0.00	100.00/0.00/0.00	96.25/0.00/3.75	—	88.75/6.25/5.00	86.25/10.00/3.75
QPSO-WD	88.75/1.25/10.00	82.50/2.50/15.00	63.75/2.50/33.75	5.00/6.25/88.75	—	7.50/41.25/51.25
QPSO-WE	88.75/1.25/10.00	85.00/0.00/15.00	65.00/3.75/31.25	3.75/10.00/86.25	51.25/41.25/7.50	—
Overall rank	*6(7.50/15.50/77.00)*	5(15.25/16.00/68.75)	4(43.25/6.25/50.50)	**1(94.25/3.25/2.50)**	3(49.50/10.75/39.75)	2(58.75/11.25/30.00)

(b) Hidden neuron oversize ratio ($n_{h_{or}}$)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	20.00/66.25/13.75	13.75/36.25/50.00	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00
BP-WD	13.75/66.25/20.00	—	13.75/33.75/52.50	0.00/0.00/100.00	1.25/0.00/98.75	1.25/0.00/98.75
BP-WE	50.00/36.25/13.75	52.50/33.75/13.75	—	0.00/0.00/100.00	16.25/3.75/80.00	16.25/5.00/78.75
QPSO-N	100.00/0.00/0.00	100.00/0.00/0.00	100.00/0.00/0.00	—	98.75/1.25/0.00	98.75/1.25/0.00
QPSO-WD	100.00/0.00/0.00	98.75/0.00/1.25	80.00/3.75/16.25	0.00/1.25/98.75	—	11.25/43.75/45.00
QPSO-WE	100.00/0.00/0.00	98.75/0.00/1.25	78.75/5.00/16.25	0.00/1.25/98.75	45.00/43.75/11.25	—
Overall rank	5(6.75/20.50/72.75)	*6(6.00/20.00/74.00)*	4(27.00/15.75/57.25)	**1(99.50/0.50/0.00)**	3(58.00/9.75/32.25)	2(64.50/10.00/25.50)

(c) Complexity reduction (Ω_r)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	6.25/61.25/32.50	13.75/11.25/75.00	0.00/0.00/100.00	10.00/1.25/88.75	10.00/0.00/90.00
BP-WD	32.50/61.25/6.25	—	13.75/13.75/72.50	0.00/0.00/100.00	13.75/0.00/86.25	12.50/1.25/86.25
BP-WE	75.00/11.25/13.75	72.50/13.75/13.75	—	2.50/1.25/96.25	35.00/0.00/65.00	30.00/5.00/65.00
QPSO-N	100.00/0.00/0.00	100.00/0.00/0.00	96.25/1.25/2.50	—	88.75/8.75/2.50	87.50/10.00/2.50
QPSO-WD	88.75/1.25/10.00	86.25/0.00/13.75	65.00/0.00/35.00	2.50/8.75/88.75	—	8.75/35.00/56.25
QPSO-WE	90.00/0.00/10.00	86.25/1.25/12.50	65.00/5.00/30.00	2.50/10.00/87.50	56.25/35.00/8.75	—
Overall rank	*6(8.00/14.75/77.25)*	5(14.50/15.25/70.25)	4(43.00/6.25/50.75)	**1(94.50/4.00/1.50)**	3(50.25/9.00/40.75)	2(60.00/10.25/29.75)

classifier for SDCPs at a high-level.

Furthermore, the more times a unique input–target pair was repeated, the more fitted the classifiers became, and accordingly, the more complexity was reduced. The degree of fitting of the classifiers was also significantly influenced by temporal severity, but not significantly by spatial severity.

Abrupt environments resulted in the best complexity performance, and progressive

environments the worst.

The dimensionality of a problem had inconsistent effects on complexity performance between the BP classifiers and the QPSO classifiers. That is, higher dimensional problems, i.e. the hyperplane and electricity domains, lead to better complexity performance for the BP classifiers. On the other hand, higher dimensional problems, lead to worse complexity performance for the QPSO classifiers.

Lastly, the complexity results and correlation coefficients showed that there was no consistent relationship between accuracy and complexity performance.

9.4 Saturation, accuracy and complexity performance trends analysis

The statistical analysis of the saturation levels, accuracy performance and complexity performance in the previous sections, resulted in various findings, including several relationships between the performance measures. The statistical analyses were, however, performed on the collective means of the performance measures. This only gave a high-level view of the classifiers' performance.

A low-level analysis that considers the performance trends of the classifiers over time for each benchmark problem is, therefore, carried out in this section to verify and confirm the findings of the previous sections. Furthermore, investigation of the performance trends of the classifiers allowed the time-based behaviours of the classifiers to be identified.

This section compares the performance trends of φ_g, MSE_g, MSE_m, MSE_t, PCC_g, PCC_m, and Ω_r on each of the benchmark problems. Because PCC_g, PCC_m, and Ω_r were in the range $[0, 100]$, the values of φ_g, MSE_g, MSE_m, and MSE_t were linearly scaled to the same range by multiplying their values by 100. Furthermore, φ_v was not used because it was shown in the saturation analysis (refer to Section 9.2) that φ_v was practically identical to φ_g.

Note that, due to the sheer volume of performance trend graphs used by this analysis, the graphs have been omitted from this section. However, the graphs are presented in appendix A for the benefit of the reader.

Figure A.1 in appendix A provides the legend for all the performance trend graphs analysed in this section.

The performance trends of the classifiers for the five problem domains are presented in figures A.2 to A.41. The figures are grouped according to the problem domain. These figures are then further grouped and sorted according to the window step size (w_s) series, i.e. series A to series D, and the weights adjustment algorithm, i.e. BP and QPSO. Figure A.2, for example, presents the performance trend graphs of the BP classifiers for the A1 to A4 hyperplane problems. Analysis of the performance trends in figures A.2 to A.41 revealed the following:

The performance trends of the BP classifiers were very similar to each other per benchmark problem. Thus, using regularisation with the BP weights adjustment algorithm to learn a ReLU FFNN SDCP did not improve performance trends. WE did manage to slightly improve the complexity reduction trends of the BP weights adjustment algorithm, but at the expense of the accuracy performance trends.

Furthermore, the sub-par accuracy trends of the BP classifiers for the SEA and electricity problems supported the idea that a static weights adjustment algorithm like BP, regardless of regularisation, was not suitable for noisy SDCPs.

The saturation trends of the BP classifiers generally showed a progressive or rapid increase to high levels. Furthermore, the BP-WE completely saturated for the electricity A4 problem. Thus, the BP classifiers stood a high chance of complete saturation for SDCPs with longer data streams. Investigation into problems with longer data streams should be carried out in future work.

The performance trends of QPSO-N oscillated widely and showed an early onset of complete saturation for all benchmark problems. The complete saturation caused the trends of the PCC measures to be very similar to each other, the same was true for the trends of the MSE measures. Furthermore, complete saturation caused the complexity reduction trends for the QPSO-N to rise rapidly and remain stable around a high level, e.g. 90%. This supports findings that complete saturation occurs after some learning has taken place and leads to irrevocable overfitting, because of a significant loss of information capacity. The performance trends of QPSO-N therefore confirmed that the classifier completely saturated on all SDCPs.

The accuracy performance trends of the classifiers were effected more severely by complete saturation when the classifiers dealt with SDCPs that had more than two target classes than when the classifiers dealt with SDCPs that only had two target classes. This supported the explanation given in the accuracy performance analysis (refer to Section 9.1).

WD and WE regularisation significantly improved the performance trends of the QPSO weights adjustment algorithm. Regularisation stabilised saturation trends for the QPSO weights adjustment algorithm through the stabilisation of the complexity reduction trend. The performance trends suggested that only saturation in the hidden neurons that was necessary for accuracy performance was preserved by regularised QPSO classifiers. QPSO-WE was the best at distinguishing between necessary and unnecessary saturation for most problems. QPSO-WD, however, reduced all saturation regardless. The main reason for this is the difference in the aggressiveness of the weight penalisation for the two regularisation terms.

QPSO-WE and QPSO-WD handled noisy SDCPs differently. QPSO-WE showed possible signs that it might be susceptible to overfitting for very long data streams when noise is present. The aggressive weight penalisation by QPSO-WD prevented any overfitting. Thus the aggressive weight penalisation is not necessarily detrimental for its accuracy performance on noisy SDCPs. Regardless, the performance trends suggested that QPSO-WE was better than QPSO-WD for the noisy problems, because not only could QPSO-WE learn to generalise effectively and maintain complexity, but QPSO-WE could also remember effectively.

The regularised QPSO classifiers were not good at remembering, whereas the BP classifiers were only able to forget some of what they had learnt. This was evident in the memory accuracy trends, i.e. MSE_m and PCC_m, that were usually worse than the generalisation accuracy trends, i.e. MSE_g and PCC_g. Furthermore, the MSE_m trends of the regularised QPSO classifiers were most of the time significantly worse than their MSE_g and MSE_t trends. Note that the MSE_t trends came close to zero in most problems, and in some cases stayed close to zero for the duration of the problem. Both BP and QPSO weight adjustment algorithms were unable to adjust the rates at which they learnt and unlearned information as needed. A possible solution would be to

dynamically adjust η for the BP classifiers and λ_r for all the classifiers.

The PCC trends and their MSE counterparts shared an inverted relationship for the majority of the benchmark problems, as expected. The PCC trends, however, were significantly more sensitive to environment changes than the MSE trends. Thus the classifiers stayed close to the correct outputs even when they classified incorrectly. Note that PCC operates in a discrete space, i.e. 0% or 100%, while MSE operates in a continuous space, i.e. $[0, 1]$.

The BP classifiers were able to delay saturation longer for SDCPs that had a high temporal severity and a low amount of information, i.e. large window step size. On the other hand, high temporal severity appeared to prevent the regularised QPSO classifiers from learning, because the rate at which they unlearned information were too large.

The BP classifiers were not effective at forgetting, whereas the regularised QPSO classifiers were. This was supported by the decreasing complexity reduction trends (Ω_r) and increasing saturation trends of the BP classifiers for most benchmark problems. On the other hand, QPSO classifiers had more constant and stable trends in terms of complexity reduction and saturation.

The BP classifiers showed signs of overfitting to early environment instances, because of their very high learning rate. Further investigation into overfitting and optimising the learning rate is required. Additionally, the benchmark suite should also generate longer data streams from the problem domains to support this further investigation.

The regularised QPSO classifiers were able to get near to the optimal complexity levels consistently, whereas the BP classifiers did not. This supported the idea that architecture selection should be used with a dynamic weights adjustment algorithm for SDCPs.

Furthermore, complexity performance trends generally varied significantly between environment changes and between problems. To avoid a situation were a FFNN is not complex enough, a fully connected FFNN should be used for streamed data classifiers.

The accuracy and complexity performance trends showed that the BP classifiers were able to allocate information efficiently to the neurons during periods of high accuracy. That is complexity reduction was high during times of high accuracy.

The performance trends of the BP classifiers also showed that problem dimensionality

effects saturation. That is, increases in dimensionality leads to saturation trends rising faster to higher levels, and becoming more volatile. Furthermore, lower dimensionality SDCPs were better for the regularised QPSO classifiers, than for the BP classifiers, and allowed the regularised QPSO classifiers to perform closer to the BP classifiers without the threat of rising saturation.

The presence of noise or irrelevant information increased the rate of saturation for the BP classifiers, especially if there was prolonged exposure to patterns with these characteristics. This was not the case for the QPSO classifiers.

The problem difficulty classification scheme proposed in Section 8.3.6 was supported by the performance trends. Furthermore, increases in the window step size appeared to consistently increase the intra-environment volatility of the performance trends, i.e. the volatility of the trends during an environment instance. On the other hand, increases in the window frequency appeared to consistently decrease intra-environment volatility in performance trends and improve accuracy performance.

The performance trends were more sensitive to temporal than spatial severity. Furthermore, high spatial severity generally had a counter effect on high temporal severity. This was evident from the observations that abrupt environments did not have a significant impact on the performance trends, whereas progressive environments had a significant detrimental effect on the performance trends, and chaotic environments only worsened performance trends slightly when compared to abrupt environments.

All the classifiers had significantly worse performance trends for the electricity problems than for the other problems. This suggests that a real-world domain could be more challenging for the classifiers than expected. The benchmark suite should acquire more real-world problem domains to further investigate this. Additionally, the benchmark suite should also acquire artificial problems with extremely high temporal severity, because this appeared to be a key characteristic of the real-world electricity domain (refer to Section 8.3.5)

9.5 Overfitting analysis

The discussions in sections 9.2 to 9.4 suggested that overfitting was present in the classifiers. This section analyses the two overfitting measures O_ρ and O_{MSE_g} (refer to Section 8.4.6).

Figures 9.2 and 9.3 present a comparison between the raw MSE_g and MSE_t trends, and the 3% moving average MSE_g and MSE_t trends for the BP classifiers and QPSO classifiers, respectively. The graphs on the left-hand side represent the raw trends. The graphs on the right-hand side represent the moving average trends. The moving average trend graphs are also overlaid with the detection results of O_{MSE_g}. Note that figures 9.2 and 9.3 include the plus and minus moving average standard deviation bands for the MSE_g trend to provide insight into the stability of the MSE_g trend.

Figures 9.2 and 9.3 represent the typical behaviour found for the classifiers for the benchmark problems. The raw MSE performance trends revealed extremely volatile trends that would either bounce between zero and very high levels, or remain at zero for several epochs. This was a side effect of the classifiers learning each pattern for only one epoch, i.e. the one-pass requirement.

The overfitting measure O_ρ produced meaningless values, because of the division by zero that occurred due to the presence of zeros in the raw MSE_t trend. The overfitting measure O_ρ, therefore, cannot work with SDCPs. An alternative overfitting measure that considers MSE_t and MSE_g needs to be developed. A possible starting point can be looking at the difference between MSE_t and MSE_g. Another starting point could be to add a very small negligible value to MSE_t, e.g. 1×10^{-16}.

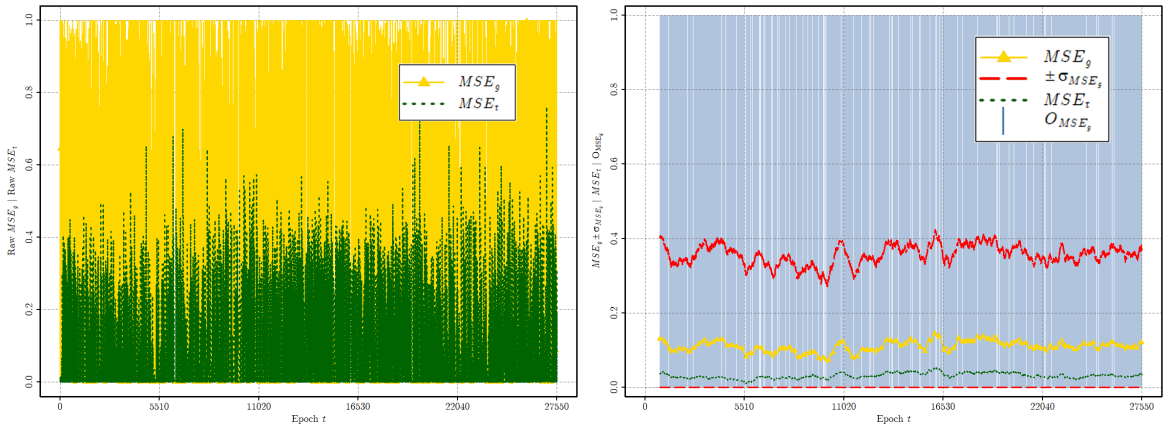
The zero MSE_t values did not present problems for the overfitting measure, O_{MSE_g}. The O_{MSE_g} results, however, left much to be desired. The biggest concern is that O_{MSE_g} suffered from many false positives. The main reason for the false positives was the high volatility of the MSE_g values. This can be seen in the large standard deviation bands in the moving average trend graphs.

Another issue with O_{MSE_g} was that it only relied on the moving average of MSE_g and did not consider MSE_t. This lead to false positives in the cases were the trend volatility was high but the directions in which MSE_g and MSE_t moved were the same, i.e. correlated.

The average Pearson correlation coefficient for MSE_g and MSE_t was 0.6137±0.3711. Thus when MSE_g increased or decreased so too did MSE_t most of the time. Hence, the classifiers were generally not overfitting, but O_{MSE_g} indicated that the classifiers were overfitting. A possible solution to this problem is to compare the direction of the trends of both MSE_g and MSE_t.

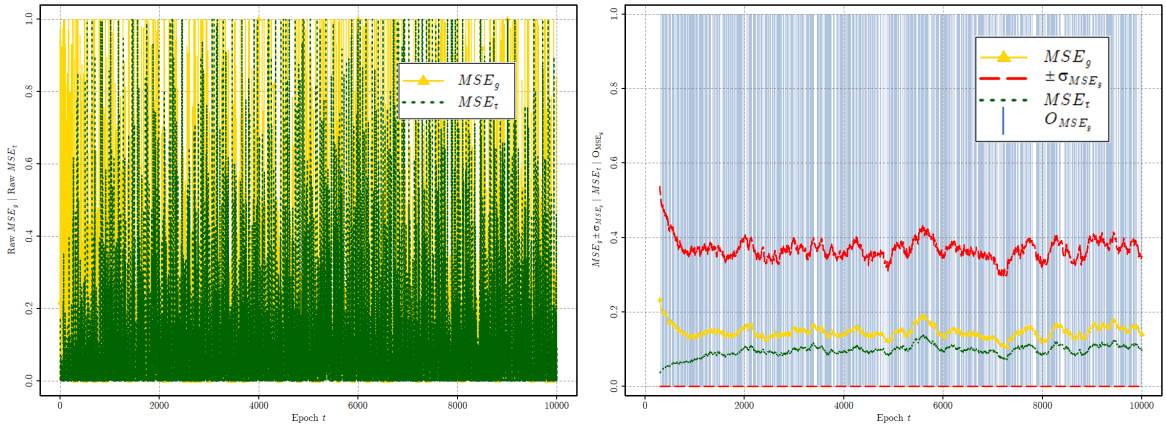
O_{MSE_g} trends for the completely saturated QPSO-N, which experienced overfitting early on, revealed a further issue with O_{MSE_g}. That is, O_{MSE_g} did not consider if overfitting happened in prior environments, which lead to false negatives. A possible solution could be to integrate the saturation measure into the overfitting measure. Another possible solution is to convert the overfitting measure into a flip-flop operator, i.e. stays in a state until a certain event occurs.

The findings above indicate that an alternative overfitting measure that mitigates problems of division by zero, only considering MSE_g, and dynamic environments is required in order to detect overfitting in SDCPs successfully. Several suggestions to solve these problems were made.



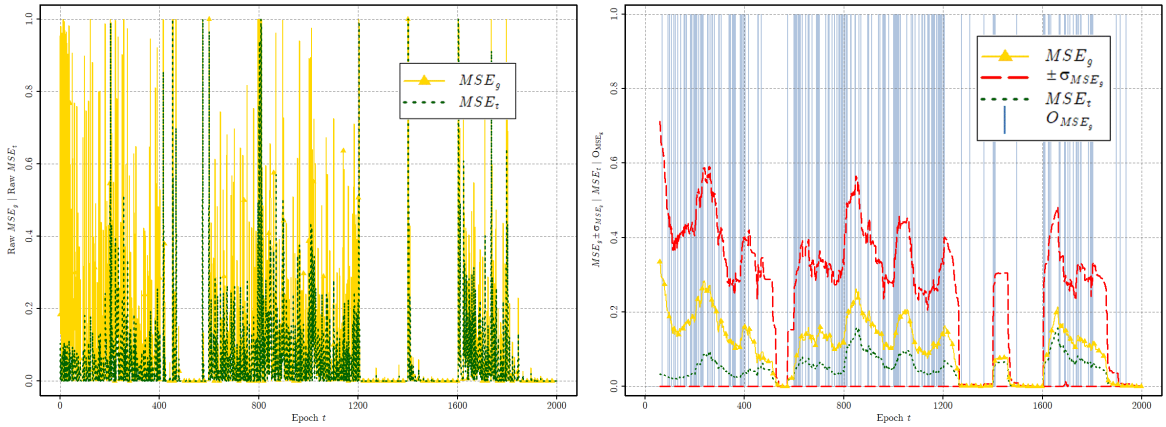
(a) Raw trends – BP-N – Electricity B2

(b) Moving average trends – BP-N – Electricity B2



(c) Raw trends – BP-WD – SEA A1

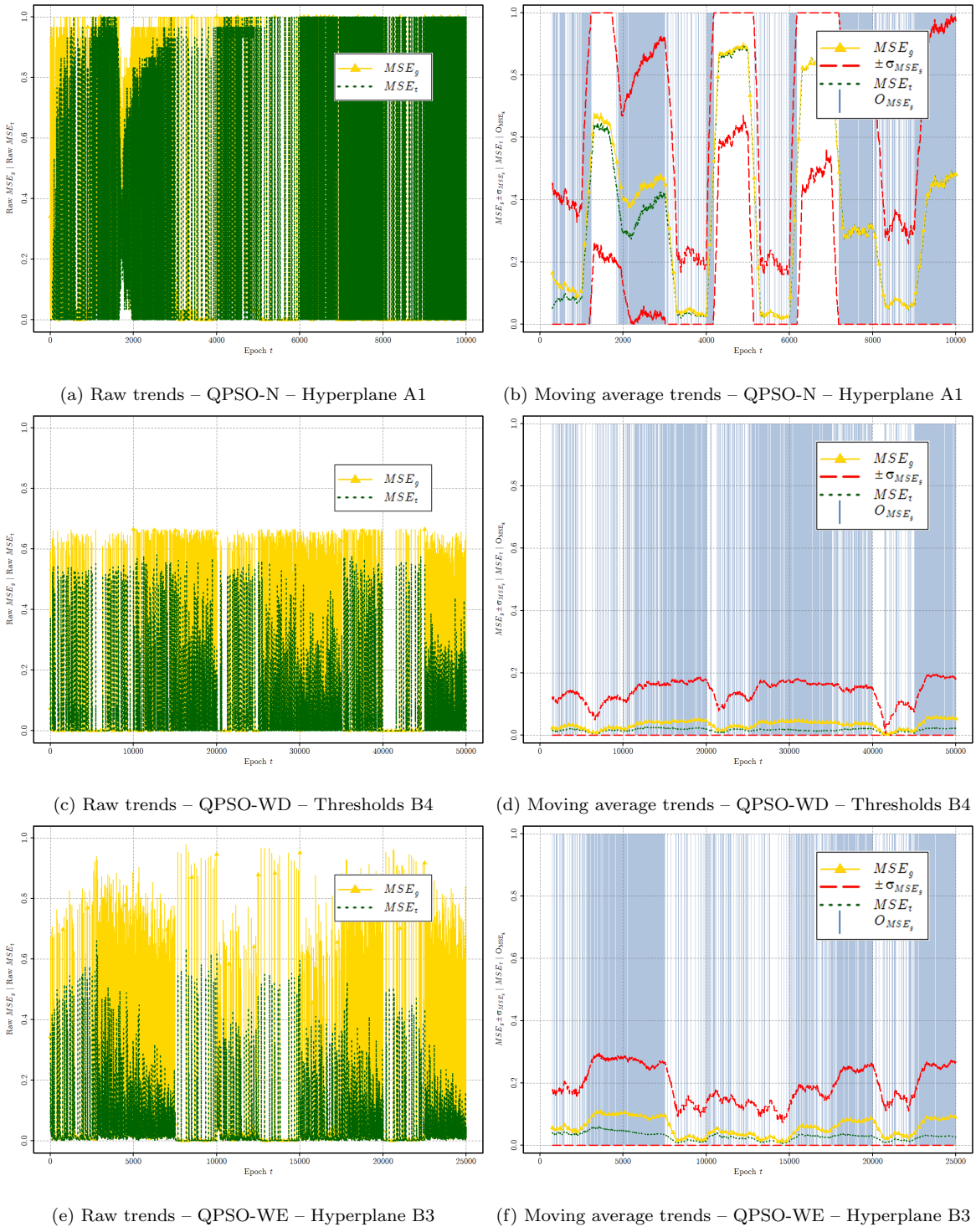
(d) Moving average trends – BP-WD – SEA A1



(e) Raw trends – BP-WE – Sphere C1

(f) Moving average trends – BP-WE – Sphere C1

Figure 9.2: Raw MSE_g and MSE_t trends versus moving average $MSE_g \pm \sigma_{MSE_g}$ and MSE_t trends, and O_{MSE_g} for the BP classifiers



(a) Raw trends – QPSO-N – Hyperplane A1

(b) Moving average trends – QPSO-N – Hyperplane A1

(c) Raw trends – QPSO-WD – Thresholds B4

(d) Moving average trends – QPSO-WD – Thresholds B4

(e) Raw trends – QPSO-WE – Hyperplane B3

(f) Moving average trends – QPSO-WE – Hyperplane B3

Figure 9.3: Raw MSE_g and MSE_t trends versus moving average $MSE_g \pm \sigma_{MSE_g}$ and MSE_t trends, and O_{MSE_g} for the QPSO classifiers

9.6 Overall statistical rank analysis

Table 9.13 presents the overall rankings for the classifiers based on the results of the MWU pairwise comparisons of the saturation levels, accuracy performance, and complexity performance in sections 9.1, 9.2, and 9.3. The rank of a classifier for each of the three performance categories, i.e. saturation, accuracy and complexity, was determined using the average wins-ties-loses percentages of the classifier in each category. The accuracy-complexity rank of the classifier was determined by averaging the average wins-ties-loses percentages for the accuracy and complexity performance categories. The overall rank of the classifier was determined by averaging the average wins-ties-loses percentages for the three performance categories.

Table 9.13: MWU-based ranking of the classifiers with regards to the performance measures (Wins/Ties/Losses percentages)

Measure	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
φ_g	2(65.25/16.25/18.50)	3(50.50/15.50/34.00)	**1(70.00/12.00/18.00)**	*6(0.00/0.00/100.00)*	4(52.25/6.75/41.00)	5(31.50/10.50/58.00)
φ_v	2(64.75/16.75/18.50)	3(50.25/16.25/33.50)	**1(70.25/12.25/17.50)**	*6(0.00/0.00/100.00)*	4(53.50/6.00/40.50)	5(31.25/8.75/60.00)
Saturation rank	2(65.00/16.50/18.50)	3(50.38/15.88/33.75)	**1(70.13/12.13/17.75)**	*6(0.00/0.00/100.00)*	4(52.88/6.38/40.75)	5(31.37/9.63/59.00)
MSE_g	**1(75.25/19.50/5.25)**	3(59.00/17.75/23.25)	2(64.25/19.50/16.25)	*6(0.00/1.00/99.00)*	5(29.75/3.25/67.00)	4(37.75/7.00/55.25)
MSE_m	**1(78.50/18.00/3.50)**	3(63.50/17.50/19.00)	2(64.50/18.00/17.50)	*6(0.50/1.00/98.50)*	5(21.25/3.50/75.25)	4(41.50/2.50/56.00)
PCC_g	**1(67.00/13.75/19.25)**	2(46.50/13.50/40.00)	3(39.25/14.00/46.75)	4(46.00/5.75/48.25)	*6(31.00/2.00/67.00)*	5(43.75/4.00/52.25)
PCC_m	**1(72.00/10.25/17.75)**	2(50.25/9.25/40.50)	4(44.00/9.50/46.50)	3(50.75/6.25/43.00)	*6(24.00/1.50/74.50)*	5(39.00/3.25/57.75)
Accuracy rank	**1(73.19/15.38/11.44)**	2(54.81/14.50/30.69)	3(53.00/15.25/31.75)	*6(24.31/3.50/72.19)*	5(26.50/2.56/70.94)	4(40.50/4.19/55.31)
n_{sor}	*6(7.50/15.50/77.00)*	5(15.25/16.00/68.75)	4(43.25/6.25/50.50)	**1(94.25/3.25/2.50)**	3(49.50/10.75/39.75)	2(58.75/11.25/30.00)
n_{hor}	5(6.75/20.50/72.75)	*6(6.00/20.00/74.00)*	4(27.00/15.75/57.25)	**1(99.50/0.50/0.00)**	3(58.00/9.75/32.25)	2(64.50/10.00/25.50)
Ω_r	*6(8.00/14.75/77.25)*	5(14.50/15.25/70.25)	4(43.00/6.25/50.75)	**1(94.50/4.00/1.50)**	3(50.25/9.00/40.75)	2(60.00/10.25/29.75)
Complexity rank	*6(7.42/16.92/75.67)*	5(11.92/17.08/71.00)	4(37.75/9.42/52.83)	**1(96.08/2.58/1.33)**	3(52.58/9.83/37.58)	2(61.08/10.50/28.42)
Accuracy-complexity rank	4(40.30/16.15/43.55)	5(33.36/15.79/50.84)	3(45.38/12.33/42.29)	**1(60.20/3.04/36.76)**	*6(39.54/6.20/54.26)*	2(50.79/7.34/41.86)
Overall rank	2(48.53/16.26/35.20)	3(39.03/15.82/45.15)	**1(53.63/12.26/34.11)**	*6(40.13/2.03/57.84)*	5(43.99/6.26/49.76)	4(44.32/8.10/47.58)

Figures 9.4 and 9.5 illustrate the rank categories, i.e. the saturation, accuracy, complexity, accuracy-complexity, and overall results of Table 9.13. Figure 9.4 plots the inverted rank, i.e. reserved rank number so that higher values indicate better performance, for the five rank categories against the classifiers. Figure 9.5 plots the MWU-comparison winning percentages for the five rank categories against the classifiers.

The overall saturation rankings confirmed that QPSO-N was significantly more saturated than any other classifier. According to the saturation rank, the BP classifiers were

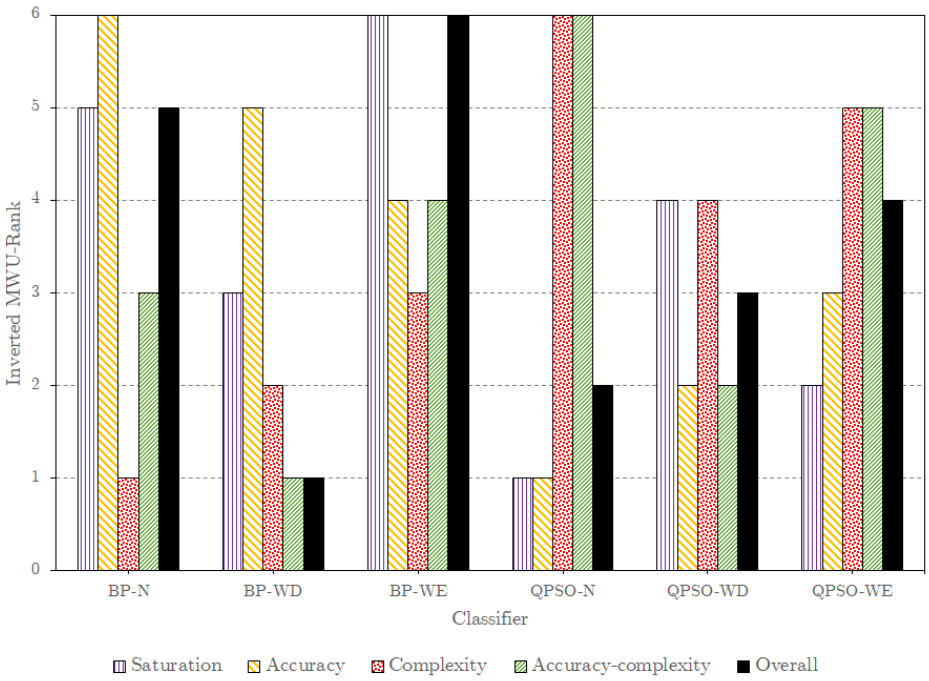


Figure 9.4: Classifier versus overall inverted MWU-Rank

significantly less saturated than the QPSO classifiers, with BP-WE being the least saturated. The difference between BP-N and BP-WE saturation rank losses was negligible. Whereas, the saturation rank losses of BP-WD was significantly worse than both BP-N and BP-WE. This confirmed the findings that the BP weights adjustment algorithm had some ability to handle saturation, and that only certain regularisation algorithms could improve this ability. Note that the saturation ranks results did not reveal the increasing saturation trends observed in the BP classifiers (refer to Section 9.4). This supports the idea that SDCPs did not allow the BP classifiers to learn long enough to see the trend affect the collective means.

The saturation rank for QPSO-WD and QPSO-WE support the finding that QPSO-WD was better at reducing saturation than QPSO-WE. However, because the accuracy rank of QPSO-WE is better than that of QPSO-WD, the rankings also support the finding that QPSO-WE was able to identify necessary saturation from unnecessary saturation.

According to the accuracy ranks, the BP classifiers outperformed the QPSO classifiers

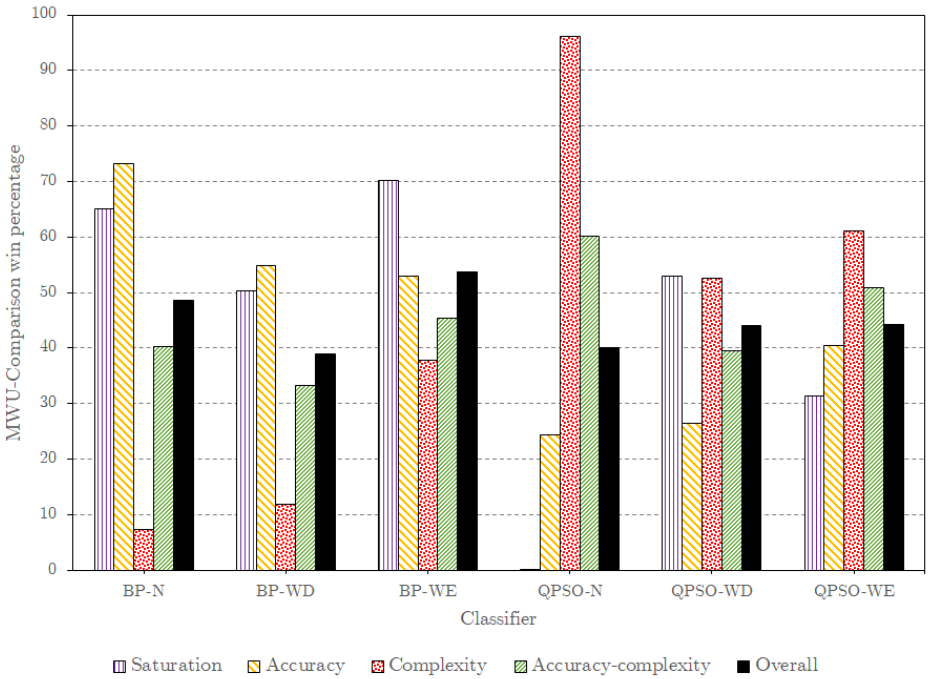


Figure 9.5: Classifier versus overall MWU-Comparison winning percentages

significantly. The accuracy ranks supported the findings that regularisation degraded the accuracy of the BP weights adjustment algorithm significantly, whereas, the QPSO weights adjustment algorithm benefited significantly from regularisation with regards to accuracy, with QPSO-WE boosting accuracy performance the most. The saturation ranks support the finding that the boost in accuracy for the QPSO-WD and QPSO-WE was due to better saturation control.

The complexity rankings showed the QPSO classifiers to be significantly better than the BP classifiers at reducing and maintaining complexity at those reduced levels. QPSO-N reduced complexity the most and BP-N the least. The saturation ranks in conjunction with the complexity rank of QPSO-N confirm the finding that the reduction in information capacity allowed the QPSO-N to achieve significantly high levels of complexity reduction.

The complexity ranks provide further support for the finding that WE did help the BP weights algorithm to maintain complexity, whereas WD did not assist in maintaining complexity of the classifiers. WE also provided better complexity results than WD for

the QPSO weights adjustment algorithm. QPSO-WD, however, performed very close to QPSO-WE.

QPSO-N had the best accuracy-complexity rank. The result, however, can be safely ignored as an anomaly caused by the complete saturation of the QPSO-N on problems with only two target classes. Under this consideration, QPSO-WE was ranked as the most proficient at managing the complexity and accuracy performance, followed by the BP classifiers. This suggests that using a dynamic weights adjustment algorithm and regularisation for SDCPs has potential, and should be researched further.

The overall ranks showed the BP-WE to be the best classifier. However, the ability for BP-WE to handle saturation and complexity came at a significant expense to accuracy performance. BP-N should, therefore, rather be considered the better BP classifier. The overall ranks showed that the QPSO classifiers were worse than the BP classifiers. QPSO-WE, however, competed with BP-WD in the accuracy and complexity rankings.

9.7 Control parameters analysis

This section analyses the control parameters of the classifiers. The key objectives for the control parameter analysis are to determine:

1. The impact of the control parameters on the performance of the classifiers. The approach documented in Section 8.4.7 was used to determine the impact of the control parameters.

2. Whether the performance gains justify the time spent tuning the control parameters. Note that time to test a parameter configuration was assumed to be constant for all classifiers. Thus the time spent on tuning the control parameters of a classifier spent was quantified as the number control parameter configurations ($|D_c|$) that were tested (refer to Equation 8.21).

3. If it is possible to reduce the time taken to tune control parameters for the classifiers by reducing the ranges for optimal parameter values. The optimal values of the control parameters are listed in Tables 8.8 to 8.13 in Section 8.5 of Chapter 8.

Table 9.14 presents the impact of the control parameters for each classifier on the control parameter tuning process. The table included the number of control parameters of each classifier (n_c), and the number control parameter configurations tested ($|D_c|$). To determine the impact made by these control parameter numbers, the table also includes the five MWU-based rank categories that were presented in Table 9.13 (refer to Section 9.6).

Table 9.14: Comparison of the control parameters of the classifier and the overall performance of the classifiers

	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
n_c	2	3	*4*	**1**	2	3
$\|D_c\|$	81	648	*5184*	**9**	72	576
Saturation rank	2(65.00|16.50|18.50)	4(50.38|15.88|33.75)	**1(70.13|12.13|17.75)**	*6(0.00|0.00|100.00)*	3(52.88|6.38|40.75)	5(31.37|9.63|59.00)
Accuracy rank	**1(73.19|15.38|11.44)**	2(54.81|14.50|30.69)	3(53.00|15.25|31.75)	*6(24.31|3.50|72.19)*	5(26.50|2.56|70.94)	4(40.50|4.19|55.31)
Complexity rank	*6(7.42|16.92|75.67)*	5(11.92|17.08|71.00)	4(37.75|9.42|52.83)	**1(96.08|2.58|1.33)**	3(52.58|9.83|37.58)	2(61.08|10.50|28.42)
Accuracy-complexity rank	4(40.30|16.15|43.55)	*6(33.36|15.79|50.84)*	3(45.38|12.33|42.29)	**1(60.20|3.04|36.76)**	5(39.54|6.20|54.26)	2(50.79|7.34|41.86)
Overall rank	2(48.53|16.26|35.20)	*6(39.03|15.82|45.15)*	**1(53.63|12.26|34.11)**	5(40.13|2.03|57.84)	4(43.99|6.26|49.76)	3(44.32|8.10|47.58)

Figure 9.6 illustrates n_c in terms of the red bars, and $|D_c|$ in terms of the black line. Note that both quantities are scaled to the same y-axis, however, the values for the red bars appear on the left hand side of the graph and the values for the black line appears on right side of the graph. Figure 9.6 was used in conjunction with Figures 9.4 and 9.5 to analyse the impact of the control parameters on the classifiers performance.

The results show that QPSO-N was the classifier with the lowest number of parameter configurations, but QPSO-N was also the classifier with the worst saturation and accuracy ranks. On the other hand, BP-WE was the classifier with the most parameter configurations, with 579 times more parameter configurations than QPSO-N. The large difference, however, lead to the highest saturation rank and overall rank, but also significantly worse accuracy and complexity scores. These results suggest that the classifiers with many or very few control parameters did not provide a good trade-off between performance, and the time taken to tune the control parameters.

QPSO-WD had the second least number of parameter configurations to test, with eight times more configurations than QPSO-N. QPSO-WD placed fourth overall and displayed average performance in the other rank categories. On the other hand, BP-

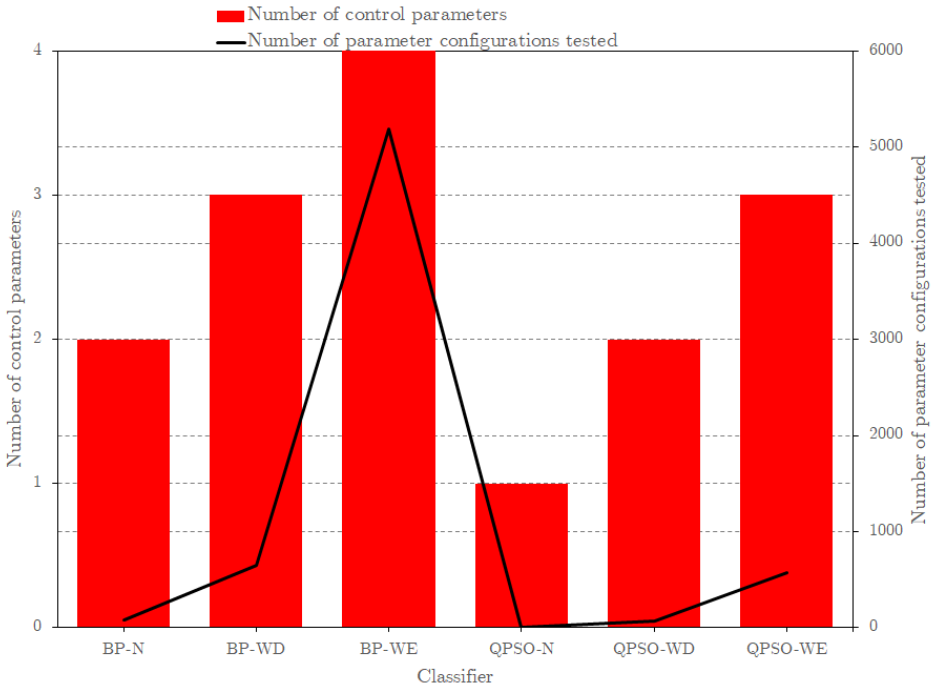


Figure 9.6: Control parameter statistics for classifiers

WD had the second most number of parameter configurations to test, with 72 times more configurations than QPSO-N, but also had no remarkable performance statistics in any of the performance categories. This provided further support that the classifiers with too many or too few control parameters did not provide a good trade-off between performance, and the time taken to tune the control parameters.

BP-N and QPSO-WE had the middle most number of parameter configurations, but also the most impressive performance. QPSO-WD had 64 times more parameter configurations than QPSO-N, and BP-N had only 9 times more parameter configurations than QPSO-N. Considering the overall ranks of BP-N and QPSO-WE, the BP-N had the best trade-off between performance and time taken to tune the control parameters, and the QPSO-WE had the second best trade-off. The results also indicate that BP-N was the most suitable classifier out of the BP classifiers tested for SDCPs, and QPSO-WE was the most suitable classifier out of the QPSO classifiers tested for SDCPs.

Lastly, in terms of the optimal values for the control parameters of the classifiers the following was observed:

The momentum term (α) and learning rate (η) values never exceeded values of 0.3, and were the majority of the time 0.1. This suggests that the ranges used for the these two control parameters could be reduced to $[0, 0.4]$. Furthermore, BP-N and BP-WD had the same α and η values. This provides an explanation for the similar performance trends of these two classifiers. BP-WE had the same α values as BP-N and BP-WD. However, the η values for BP-WE were different from BP-N and BP-WD for the electricity problems. Consequently, the performance trends of BP-WE differed significantly from the performance trends of BP-N and BP-WD on the electricity problems. This supported the finding that WE had an effect on the BP weights adjustment algorithm, whereas WD did not.

The above observations showed the BP weights adjustment algorithm required a very low learning rate and momentum for SDCPs in order to cope with the dynamic environments.

The radius (r) values of the regularised QPSO classifiers never exceed 0.25 and were mainly 0.1, which supports the recommendations by Harrison *et al.* [52]. On the other hand, QPSO-N generally had very high r values, e.g. 5. These high values are a result of the search trying to move back into feasible regions of the search space, through exploration, after complete saturation moved the search deep into infeasible regions of the search space. The differences between the r values of the QPSO classifiers supported the finding that WD and WE influenced the QPSO weights adjustment algorithm differently.

The regularisation lambdas (λ_r) of BP-WD and BP-WE favoured values under 0.01. Furthermore, the λ_r value for BP-WD was 0.0001 for all problems. Thus the range for the λ_r values of the regularised BP classifiers could be reduced for SDCPs.

On the other hand, the mode for λ_r for both regularised QPSO classifiers was 0.01. QPSO-WD, however, had a larger range of values for λ_r than QPSO-WE, i.e. 0.499 versus 0.09. Thus the value of λ_r could generally be chosen from a small range around 0.01. However, the value is problem and weights adjustment algorithm dependent.

The values for the weights relevancy threshold (w_0) were scattered over a wide large range for BP-WE. Values of w_0 for the QPSO-WE were also scattered over a wide range, but tended to favour larger values than those values used by BP-WE. QPSO-WE used the largest value out of the possible values for w_0, i.e. 1, for the electricity problems. Thus

the value ranges of w_0 are dependent on the underlying weights adjustment algorithms. A wide range for w_0 should therefore be used when optimising classifiers using WE on SDCP.

9.8 Weight distribution analysis

The discussions in sections 9.2 to 9.7 showed that the performance of the classifiers differed amongst each other and from expectations. However, the discussions could not reveal exactly why this was the case. Because the training algorithms trained the classifiers by changing the weights, the weight distributions gave a direct view of how each classifier handled the benchmark problems. Knowing how the classifiers handled the benchmark problems will assist in understanding why the performance differed. This section analyses the collective means of the two weights magnitude distribution performance measures $\bar{w}$ and σ_w, and the weights frequency distribution (Ξ_w) (refer to Section 8.4.8) in an effort to understand the variations in the performance of the classifiers. The key objectives of the weight distribution analysis are to determine:

1. Why the two weight adjustment algorithms, i.e. BP and QPSO, performed differently.
2. Why the regularisation aided the QPSO classifiers, but not BP classifiers.
3. How WD and WE compared in terms of the weight distribution, and what this meant in terms of their performance.
4. The effects of problem difficulty, environment, dimensionality, noise and complete saturation on the weight distribution of the classifiers, to further understand the performance results analysed thus far.

Table 9.15 presents the statistical and MWU-based ranking analysis of the weight magnitude distributions with regards to the classifiers for each of the problem domains. Table 9.16 presents the statistical and MWU-based ranking analysis of the weight magnitude distributions with regards to the classifier for each of the problem difficulties. Table 9.17 presents the statistical and MWU-based ranking analysis of the weight magnitudes

Table 9.15: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem domains for weight distribution performance measures

(a) Average weights magnitude ($\bar{w}$)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	0.2943±0.0392 4(36.25/3.75/60.00)	0.2751±0.0280 3(57.50/2.50/40.00)	*0.3046±0.0327* 5(22.50/1.25/76.25)	∞±∞ *6(0.00/0.00/100.00)*	**0.0668±0.0045** **1(100.00/0.00/0.00)**	0.0820±0.0060 2(80.00/0.00/20.00)
Sphere	*0.4812±0.0793* 5(20.00/13.75/66.25)	0.4510±0.0522 3(36.25/21.25/42.50)	0.4571±0.0632 4(36.25/20.00/43.75)	∞±∞ *6(0.00/0.00/100.00)*	**0.1980±0.0265** **1(91.25/6.25/2.50)**	0.2220±0.0317 2(82.50/6.25/11.25)
Thresholds	*0.5484±0.0773* 5(20.00/8.75/71.25)	0.5217±0.0560 4(31.25/10.00/58.75)	0.4261±0.0370 3(62.50/3.75/33.75)	∞±∞ *6(0.00/0.00/100.00)*	**0.2252±0.0151** **1(100.00/0.00/0.00)**	0.3783±0.0779 2(72.50/5.00/22.50)
SEA	*0.4032±0.0405* 4(46.25/18.75/35.00)	0.3911±0.0249 2(57.50/22.50/20.00)	0.3750±0.0211 **1(73.75/16.25/10.00)**	∞±∞ *6(0.00/0.00/100.00)*	**0.3436±0.0550** 3(71.25/5.00/23.75)	∞±∞ 5(20.00/0.00/80.00)
Electricity	*0.3072±0.0418* 4(20.00/0.00/80.00)	0.2242±0.0868 3(53.75/0.00/46.25)	**0.0921±0.0178** **1(97.50/0.00/2.50)**	∞±∞ *5(0.00/0.00/100.00)*	0.1881±0.0116 3(53.75/0.00/46.25)	0.1606±0.0053 2(75.00/0.00/25.00)

(b) Standard deviation of weights magnitude (σ_w)

Domain	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Hyperplane	*0.2754±0.0357* 5(21.25/2.50/76.25)	0.2539±0.0249 3(48.75/3.75/47.50)	0.2675±0.0394 4(45.00/3.75/51.25)	∞±∞ *6(0.00/0.00/100.00)*	**0.0538±0.0026** **1(100.00/0.00/0.00)**	0.0889±0.0366 2(80.00/0.00/20.00)
Sphere	0.3500±0.0737 3(52.50/13.75/33.75)	0.3257±0.0527 2(71.25/8.75/20.00)	0.3694±0.0894 4(40.00/10.00/50.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1563±0.0187** **1(100.00/0.00/0.00)**	*0.6119±0.2057* 5(20.00/0.00/80.00)
Thresholds	0.4861±0.1270 3(51.25/21.25/27.50)	0.4640±0.1022 2(58.75/20.00/21.25)	0.5357±0.1701 4(35.00/11.25/53.75)	∞±∞ *6(0.00/0.00/100.00)*	**0.1964±0.0055** **1(100.00/0.00/0.00)**	*0.7253±0.3217* 5(25.00/7.50/67.50)
SEA	0.3101±0.0469 3(60.00/12.50/27.50)	0.2942±0.0341 **1(80.00/8.75/11.25)**	*0.3356±0.0629* 4(43.75/7.50/48.75)	∞±∞ *6(0.00/0.00/100.00)*	**0.2570±0.0407** 2(78.75/6.25/15.00)	∞±∞ 5(20.00/0.00/80.00)
Electricity	*0.3099±0.0604* 5(21.25/0.00/78.75)	0.2426±0.0887 3(53.75/0.00/46.25)	0.2238±0.0453 4(50.00/1.25/48.75)	∞±∞ *6(0.00/0.00/100.00)*	0.1424±0.0084 2(76.25/1.25/22.50)	**0.1295±0.0042** **1(97.50/0.00/2.50)**

with regards to the classifier distributions for each of the problem environments. Table 9.18 presents the overall MWU-based ranking results of how the classifiers faired against each other over all the problems based on their weight magnitude distribution.

Figure 9.7 presents the aggregated weights frequency distributions (Ξ_w) for the classifiers (refer to Section 8.7.1). The scales of the y-axis for the graphs differ as follows. The BP classifiers all have the same y-axis scale, and the QPSO classifiers all have the same y-axis scale. Note that small artefacts occurred when the bin intervals changed from 0.1 to 1. However, this is to be expected because of the new bin interval is ten times greater than the previous bin interval.

Table 9.16: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem difficulty for weight distribution performance measures

(a) Average weights magnitude ($\bar{w}$)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	*0.4585±0.1507* 5(25.00/1.00/74.00)	0.3791±0.1761 4(60.00/3.00/37.00)	0.3525±0.1541 2(61.00/4.00/35.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1845±0.0761** **1(87.00/1.00/12.00)**	0.3552±0.2906 3(61.00/3.00/36.00)
Moderate-I	*0.4128±0.1139* 5(30.00/11.00/59.00)	0.3883±0.1271 4(44.00/14.00/42.00)	0.3459±0.1491 3(62.00/6.00/32.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.2126±0.1082** **1(77.00/5.00/18.00)**	∞±∞ 2(68.00/2.00/30.00)
Moderate-II	*0.3934±0.0864* 5(28.00/8.00/64.00)	0.3686±0.0938 4(42.00/11.00/47.00)	0.3145±0.1176 3(57.00/9.00/34.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1919±0.0763** **1(89.00/2.00/9.00)**	0.3169±0.2500 2(67.00/4.00/29.00)
Hard	*0.3627±0.0769* 5(31.00/16.00/53.00)	0.3546±0.0759 4(43.00/17.00/40.00)	0.3110±0.1256 3(54.00/14.00/32.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.2284±0.1074** **1(80.00/1.00/19.00)**	0.3417±0.2903 2(68.00/0.00/32.00)

(b) Standard deviation of weights magnitude (σ_w)

Difficulty	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Easy	0.4034±0.1449 3(45.00/3.00/52.00)	0.3350±0.1566 2(75.00/1.00/24.00)	0.4446±0.1896 5(37.00/2.00/61.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1468±0.0591** **1(94.00/0.00/6.00)**	*0.7023±0.5043* 4(46.00/0.00/54.00)
Moderate-I	*0.3636±0.1025* 4(40.00/9.00/51.00)	0.3390±0.1110 2(59.00/11.00/30.00)	0.3551±0.1394 5(40.00/7.00/53.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1696±0.0803** **1(91.00/3.00/6.00)**	∞±∞ 3(52.00/6.00/42.00)
Moderate-II	0.3263±0.0646 4(41.00/7.00/52.00)	0.3035±0.0600 2(65.00/4.00/31.00)	0.3187±0.0825 5(43.00/3.00/54.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1513±0.0587** **1(96.00/0.00/4.00)**	*0.4960±0.3620* 3(48.00/0.00/52.00)
Hard	0.2918±0.0508 4(39.00/21.00/40.00)	0.2868±0.0485 2(51.00/17.00/32.00)	0.2672±0.0625 3(51.00/15.00/34.00)	∞±∞ *6(0.00/0.00/100.00)*	**0.1771±0.0801** **1(83.00/3.00/14.00)**	*0.3922±0.3218* 5(48.00/0.00/52.00)

The weight distribution results in Tables 9.15 to 9.18, and Figure 9.7 show the following:

The weight distributions of the BP classifiers differed significantly in nature to that of the QPSO classifiers.

BP-N had a broad normal distribution with a negative mean. BP-WD did not alter this distribution drastically, but BP-WE did. BP-WE shifted most of the distribution to the negative side. The regularised BP classifiers in general had narrow weight distributions, that were skewed to the negative side. Furthermore, BP-WE saturated completely on the electricity A4 problem (refer to Section 9.4). The average weight magnitudes for the electricity domain, however, was very small.

The above observations show a large number of zero and negative weights for the BP

Table 9.17: Descriptive statistics and MWU-based ranking results of the classifiers with regards to the collective means of the problem environments for weight distribution performance measures

(a) Average weights magnitude ($\bar{w}$)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	*0.4091±0.1019* 5(30.56/14.44/55.00)	0.3978±0.0949 4(41.67/17.22/41.11)	0.3728±0.0593 3(46.67/12.78/40.56)	∞±∞ *6(0.00/0.00/100.00)*	**0.2145±0.0986** **1(91.67/1.67/6.67)**	0.3747±0.2701 2(65.00/2.78/32.22)
Abrupt	*0.4529±0.1672* 5(25.71/0.00/74.29)	0.3427±0.2007 3(65.71/0.00/34.29)	0.3229±0.1805 2(71.43/0.00/28.57)	∞±∞ *6(0.00/0.00/100.00)*	**0.1889±0.0709** **1(82.86/0.00/17.14)**	0.3938±0.3268 4(54.29/0.00/45.71)
Chaotic	*0.3960±0.1153* 5(27.03/5.41/67.57)	0.3538±0.1277 4(49.19/7.57/43.24)	0.2918±0.1687 2(67.57/5.41/27.03)	∞±∞ *6(0.00/0.00/100.00)*	**0.1974±0.0925** **1(75.14/3.24/21.62)**	∞±∞ 3(69.19/2.16/28.65)

(b) Standard deviation of weights magnitude (σ_w)

Environment	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
Progressive	0.3251±0.0923 3(46.11/17.78/36.11)	0.3161±0.0882 2(61.11/13.89/25.00)	0.3407±0.1172 4(43.33/10.56/46.11)	∞±∞ *6(0.00/0.00/100.00)*	**0.1713±0.0743** **1(93.33/1.11/5.56)**	*0.5553±0.3528* 5(33.33/2.22/64.44)
Abrupt	0.3978±0.1658 4(45.71/0.00/54.29)	0.3020±0.1716 2(77.14/0.00/22.86)	0.4440±0.2198 5(37.14/0.00/62.86)	∞±∞ *6(0.00/0.00/100.00)*	**0.1481±0.0544** **1(91.43/0.00/8.57)**	*0.7488±0.5751* 3(48.57/0.00/51.43)
Chaotic	*0.3572±0.1005* 5(35.68/4.32/60.00)	0.3188±0.1055 2(61.08/4.32/34.59)	0.3334±0.1440 4(43.24/4.32/52.43)	∞±∞ *6(0.00/0.00/100.00)*	**0.1538±0.0689** **1(88.65/2.16/9.19)**	∞±∞ 3(63.24/1.08/35.68)

classifiers. Too many zero and/or negative weights will result in net input signals that are zero or less. Thus, neurons with net input signals of zero or less will have activation values with zero gradients due to the ReLU activation function. The BP weight adjustment algorithm will *lose control over the weights*, i.e. not be able to adjust the weights, that result in such net input signals, because the algorithm is gradient-based. Once control is lost the BP algorithm will not be able to adjust the weight again. Because SDCPs are dynamic and unbounded in the real-world, weights need to be constantly adjusted. The BP classifiers would thereafter eventually fail for SDCPs. It can be argued that regularisation adds additional training information, i.e. model complexity, which will still provide a gradient in the case of weights being negative. However, the optimisation will only be focused on optimising model complexity and not accuracy. Furthermore, regularisation only drove weights faster to zero. Thus adding regularisation to the BP weights adjustment algorithm did not help. A possible solution for the problem of losing control over the weights could be to reinitialise the weights when the majority of weights

Table 9.18: MWU-based pairwise comparison of the classifiers for weight distribution performance measures (Wins/Ties/Losses percentages)

(a) Average weights magnitude ($\bar{w}$)

Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	0.00/28.75/71.25	16.25/13.75/70.00	100.00/0.00/0.00	6.25/2.50/91.25	20.00/0.00/80.00
BP-WD	71.25/28.75/0.00	—	26.25/23.75/50.00	100.00/0.00/0.00	13.75/2.50/83.75	25.00/1.25/73.75
BP-WE	70.00/13.75/16.25	50.00/23.75/26.25	—	100.00/0.00/0.00	30.00/0.00/70.00	42.50/3.75/53.75
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	0.00/0.00/100.00	0.00/0.00/100.00
QPSO-WD	91.25/2.50/6.25	83.75/2.50/13.75	70.00/0.00/30.00	100.00/0.00/0.00	—	71.25/6.25/22.50
QPSO-WE	80.00/0.00/20.00	73.75/1.25/25.00	53.75/3.75/42.50	100.00/0.00/0.00	22.50/6.25/71.25	—
Overall rank	5(28.50/9.00/62.50)	4(47.25/11.25/41.50)	3(58.50/8.25/33.25)	*6(0.00/0.00/100.00)*	**1(83.25/2.25/14.50)**	2(66.00/2.25/31.75)

(b) Standard deviation of weights magnitude (σ_w)

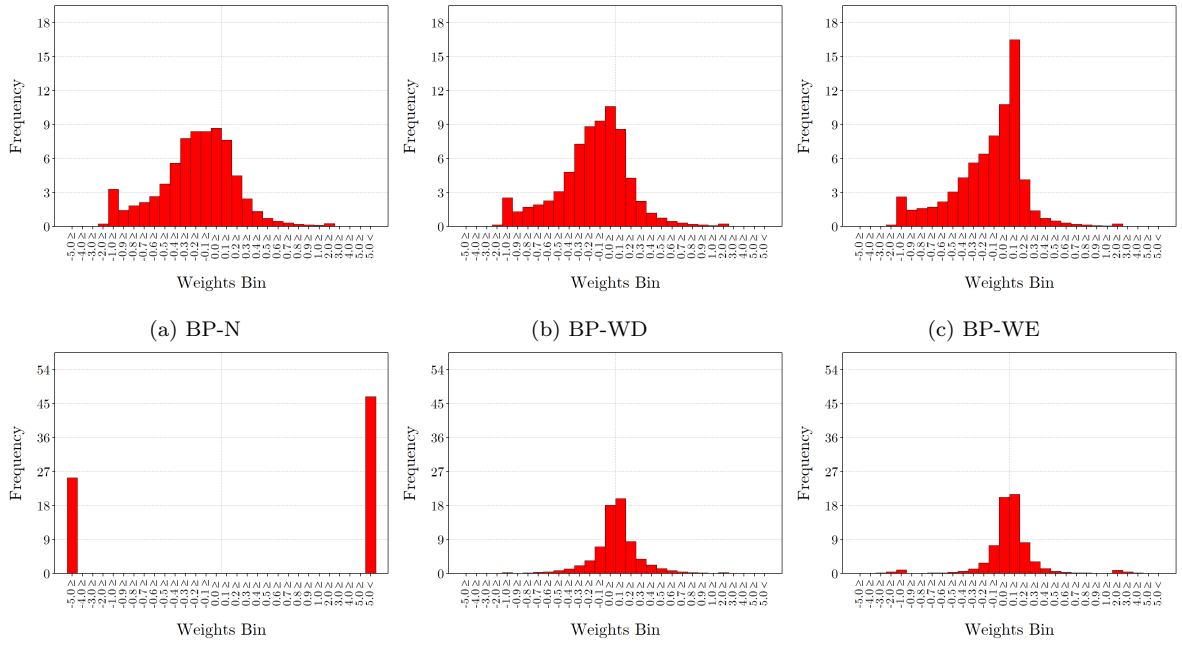
Classifier	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
BP-N	—	0.00/27.50/72.50	45.00/18.75/36.25	100.00/0.00/0.00	5.00/1.25/93.75	56.25/2.50/41.25
BP-WD	72.50/27.50/0.00	—	72.50/8.75/18.75	100.00/0.00/0.00	8.75/2.50/88.75	58.75/2.50/38.75
BP-WE	36.25/18.75/45.00	18.75/8.75/72.50	—	100.00/0.00/0.00	3.75/3.75/92.50	55.00/2.50/42.50
QPSO-N	0.00/0.00/100.00	0.00/0.00/100.00	0.00/0.00/100.00	—	0.00/0.00/100.00	0.00/0.00/100.00
QPSO-WD	93.75/1.25/5.00	88.75/2.50/8.75	92.50/3.75/3.75	100.00/0.00/0.00	—	80.00/0.00/20.00
QPSO-WE	41.25/2.50/56.25	38.75/2.50/58.75	42.50/2.50/55.00	100.00/0.00/0.00	20.00/0.00/80.00	—
Overall rank	3(41.25/10.00/48.75)	2(62.50/8.25/29.25)	5(42.75/6.75/50.50)	*6(0.00/0.00/100.00)*	**1(91.00/1.50/7.50)**	4(48.50/1.50/50.00)

are less than or equal to zero, or when accuracy starts to worsen.

Saturation in the BP classifiers was, therefore, caused by a loss of control over the weights, because of the combined effect of zero gradients and dynamic environments. The ReLU FFNN classifiers must, therefore, not use the BP weights adjustment algorithm when dealing with SDCPs.

QPSO-N, on the other hand, had all its weights distributed to the edges of negative and positive portions of the distributions. This is a definitive confirmation of complete saturation. The regularised QPSO classifiers both changed the weight distribution of QPSO-N to a narrow normal distribution around zero. The weight distribution of QPSO-WE was slightly more narrower than the weight distribution of QPSO-WD, but the weight value range of QPSO-WE was greater.

The above observations showed that the QPSO weights adjustment algorithm was immune to the zero gradients of the ReLU activation functions. QPSO-N, however, still



(a) BP-N (b) BP-WD (c) BP-WE

(d) QPSO-N (e) QPSO-WD (f) QPSO-WE

Figure 9.7: Weight frequency distribution graphs for the classifiers

suffered from saturation due to an uncontrolled explosion of weights. The difference between the QPSO-N and the regularised QPSO classifiers was that the regularised QPSO classifiers had more information about the search space to control, namely complexity performance. Hence, the *lack of search space information*, i.e. boundaries, was the cause of saturation for the QPSO classifiers.

Higher dimensional SDCPs lead to smaller weight magnitudes and distributions, than lower dimensional SDCPs. This was supported by all the classifiers, except QPSO-N. The higher the dimension of the problem, the more synapses there are compared to the number of outputs. Thus the less each hidden neuron has to contribute on average to the outputs in order to produce an output.

The average magnitude of the weights for QPSO-WE reached extreme levels in the noisy SEA problem domain, with just as extreme standard deviations. Thus noisy SDCPs interfered with the ability for QPSO-WE to control the weights. However, the interference had no significant effect on the overall performance of the QPSO-WE, because it was limited to Moderate-I chaotic SEA problems.

QPSO-WD had small weight magnitudes for the benchmark problems. This observation explains the stability that was seen in the complexity and saturation performance trends of QPSO-WD. Compared to BP-WD, QPSO-WD had significantly smaller weight magnitudes on average. Thus, the aggressiveness that WD showed towards penalising the weights was enhanced through the use of a dynamic weights adjustment algorithm.

The weight magnitude distributions for QPSO-WE and BP-N generally showed similar trends for the problem difficulties and problem environments. However, the distribution of weights for QPSO-WE conformed more to the normal distribution than BP-N. Thus, QPSO-WE showed that it had potential as a stream data classifier.

Problem difficulty had varying effects on the weight magnitudes distributions of the classifiers. The magnitudes of the weights for the classifiers were larger and more distributed for moderate-I problems than for moderate-II problems. In the case of the BP classifiers and QPSO-WE, the magnitudes of the weights were larger and more distributed for easy problems than for hard problems. There is a weak inverse relationship between the problem difficulty and the weights magnitude distribution for all the classifiers except QPSO-N and QPSO-WD. QPSO-WD weight magnitudes, however, tended to become larger and more distributed as the problem difficulty increased. Figure 9.8 illustrates the above by plotting the mean of the average weight magnitudes in Table 9.16 against the problem difficulties for each classifier. Note that Figure 9.8 excludes QPSO-N due to its infinitely large values.

The problem environments also had varying effects on the weight distributions of the classifiers. BP-N and QPSO-WE both had smaller weight magnitudes and distributions for progressive problems than for abrupt problems. The other classifiers showed the inverse thereof. This further supported the finding that QPSO-WE and BP-N shared some similarities in controlling the weights, especially in progressive and abrupt environments.

Weight distribution analysis was found to be critical in understanding the behaviours of classifiers in this study, because the analysis provided explanations for the unexpected behaviours in the performance of the classifiers.

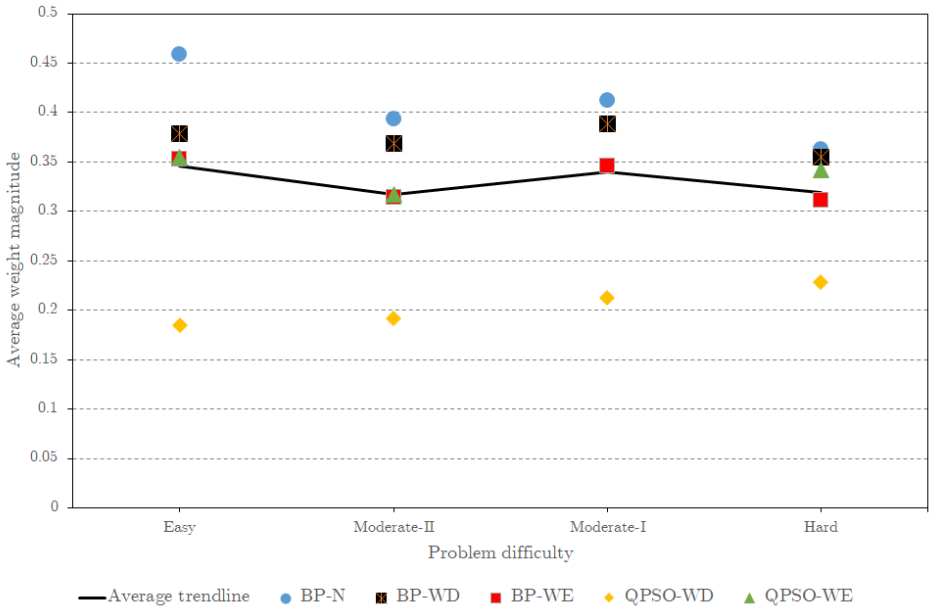


Figure 9.8: Average weight magnitude versus problem difficulty scatter plot

9.9 Swarm diversity analysis

This section analyses the swarm diversity performance measure $\mathcal{D}$ of the QPSO classifiers (refer to Section 8.4.9). The key objectives of the swarm diversity analysis are to determine:

1. The swarm diversity levels experienced by the various QPSO classifiers.
2. Whether or not the QPSO classifiers could adjust the swarm diversity as needed.
3. Whether or not the QPSO classifiers explored or exploited more.
4. The effects of problem difficulty, environment, dimensionality, noise and complete saturation on the exploration-exploitation trade-off of the QPSO classifiers.

Table 9.19 presents the statistical and MWU-based ranking analysis of the swarm diversity with regards to the QPSO classifiers for each of the problem domains. Table 9.20 presents the statistical and MWU-based ranking analysis of the swarm diversity with regards to the QPSO classifiers for each of the problem difficulties. Table 9.21 presents

the statistical and MWU-based ranking analysis of the swarm diversity with regards to the QPSO classifiers for each of the problem environments. Table 9.22 presents the overall MWU-based ranking results of how the QPSO classifiers faired against each other over all the problems based on their swarm diversity. The swarm diversity results in Tables 9.19 to 9.22 show the following:

Table 9.19: Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem domains for $\mathcal{D}$

Domain	Classifier		
	QPSO-N	*QPSO-WD*	*QPSO-WE*
Hyperplane	$\infty \pm \infty$	**0.5515±0.0448**	*0.6832±0.0669*
	3(0.00/0.00/100.00)	**1(100.00/0.00/0.00)**	2(50.00/0.00/50.00)
Sphere	$\infty \pm \infty$	*1.4283±0.3476*	**1.1051±0.6068**
	3(0.00/0.00/100.00)	2(56.25/6.25/37.50)	**1(87.50/6.25/6.25)**
Thresholds	$\infty \pm \infty$	**1.2556±0.1741**	*1.4449±0.4049*
	3(0.00/0.00/100.00)	**1(78.13/12.50/9.38)**	2(59.38/12.50/28.13)
SEA	$\infty \pm \infty$	***3.3617±1.1768***	$\infty \pm \infty$
	3(0.00/0.00/100.00)	2(50.00/0.00/50.00)	**1(100.00/0.00/0.00)**
Electricity	$\infty \pm \infty$	*1.6456±0.3540*	**1.4172±0.2540**
	3(0.00/0.00/100.00)	2(50.00/0.00/50.00)	**1(100.00/0.00/0.00)**

Table 9.20: Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem difficulty for $\mathcal{D}$

Difficulty	Classifier		
	QPSO-N	*QPSO-WD*	*QPSO-WE*
Easy	$\infty \pm \infty$	*1.2432±0.6265*	**0.7483±0.3104**
	3(0.00/0.00/100.00)	2(60.00/5.00/35.00)	**1(85.00/5.00/10.00)**
Moderate-I	$\infty \pm \infty$	***1.9026±1.3402***	$\infty \pm \infty$
	3(0.00/0.00/100.00)	2(62.50/5.00/32.50)	**1(82.50/5.00/12.50)**
Moderate-II	$\infty \pm \infty$	*1.3394±0.6276*	**0.9760±0.3721**
	3(0.00/0.00/100.00)	2(70.00/0.00/30.00)	**1(80.00/0.00/20.00)**
Hard	$\infty \pm \infty$	*2.1090±1.3496*	**2.0720±1.3742**
	3(0.00/0.00/100.00)	**1(75.00/5.00/20.00)**	2(70.00/5.00/25.00)

Swarm diversity for QPSO-N was extreme in all of the problem domains. Swarm diversity for QPSO-WE was extreme for the noisy SEA problems. The MWU ranking

Table 9.21: Descriptive statistics and MWU-based ranking results of the QPSO classifiers with regards to the collective means of the problem environments for $\mathcal{D}$

Environment	Classifier		
	QPSO-N	*QPSO-WD*	*QPSO-WE*
Progressive	$\infty \pm \infty$	*1.6429±1.1741*	**1.4955±1.2028**
	3(0.00/0.00/100.00)	**1(76.39/5.56/18.06)**	2(68.06/5.56/26.39)
Abrupt	$\infty \pm \infty$	*1.1877±0.4920*	**0.6416±0.3054**
	3(0.00/0.00/100.00)	2(57.14/0.00/42.86)	**1(92.86/0.00/7.14)**
Chaotic	$\infty \pm \infty$	***1.7412±1.0917***	$\infty \pm \infty$
	3(0.00/0.00/100.00)	2(59.46/2.70/37.84)	**1(87.84/2.70/9.46)**

indicated that swarm diversity of QPSO-WE was significantly less than the swarm diversity of the other classifiers 100% of the time for the SEA problems. This supports the findings that only a negligible number of runs for the SEA problems lead to the extreme swarm diversity. These observations showed that complete or high levels of saturation increased the weight magnitudes which in turn lead to high swarm diversity for the QPSO classifiers. Thus, complete saturation prevented the QPSO classifiers from exploiting a good solution.

Low dimensionality, noise and high temporal severity lead to the regularised QPSO classifiers exploring more, especially in the presence of noise. This was evident in the fact that benchmark problems exhibiting these characteristics, e.g. sphere, thresholds and SEA, had significantly higher swarm diversity than those benchmark problems that did not exhibit these characteristics. The results also showed that easier problems resulted in the regularised QPSO classifiers exploiting more, on average, than harder problems.

Figures 9.9 to 9.12 present the swarm diversity trend graphs. Note that Figures 9.9 to 9.12 include the plus and minus moving average standard deviation bands for the swarm diversity trend to provide insight into the stability of the swarm diversity trend.

Figure 9.9 presents the swarm diversity trend of the QPSO-N on the hyperplane A1 problems. The swarm diversity trends for QPSO-N took on a similar shape for all problems. Problems with higher temporal severity took longer to approach the extreme levels.

Figures 9.10 and 9.11 present the swarm diversity trends of the regularised QPSO

Table 9.22: MWU-based pairwise comparison of the classifiers for $\mathcal{D}$ (Wins/Ties/Losses percentages)

Classifier	Classifier		
	QPSO-N	*QPSO-WD*	*QPSO-WE*
QPSO-N	—	0.00/0.00/100.00	0.00/0.00/100.00
QPSO-WD	100.00/0.00/0.00	—	33.75/7.50/58.75
QPSO-WE	100.00/0.00/0.00	58.75/7.50/33.75	—
Overall rank	*3(0.00/0.00/100.00)*	2(66.88/3.75/29.38)	**1(79.38/3.75/16.88)**

classifiers for the A1 problems. Note that the swarm diversity trends for the A1 problems of the hyperplane, sphere, thresholds and electricity domains represent the typical shape of the swarm diversity trend experienced by the regularised QPSO classifiers in these domains. Decreasing spatial severity of the A1 problems lead to more volatile trends, while decreasing the temporal severity of the A1 problems resulted in smoother trends.

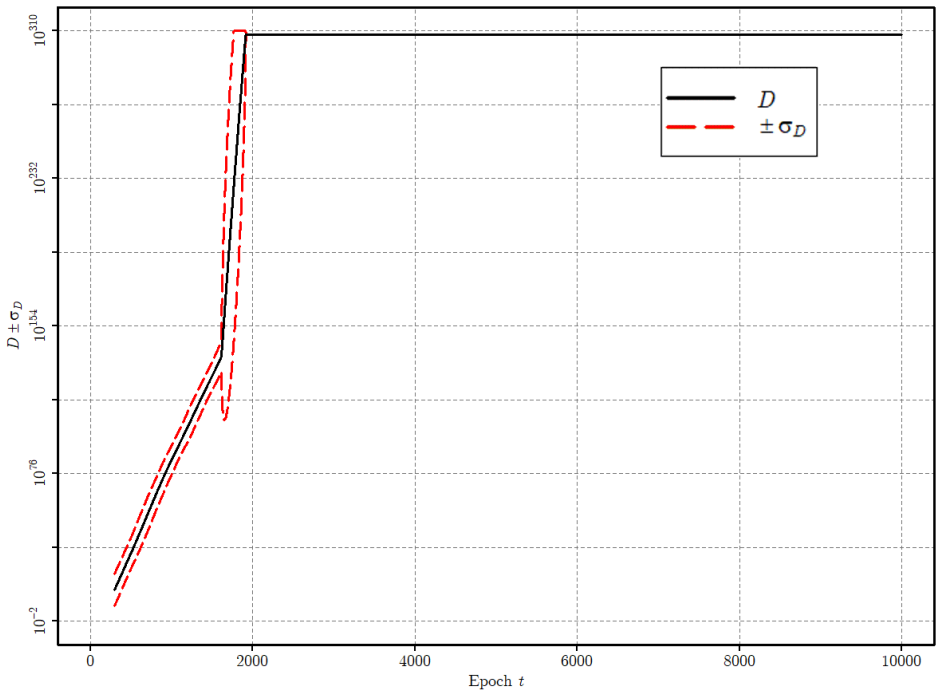


Figure 9.9: Swarm diversity trend of the QPSO-N classifier for the A1 hyperplane problem

Chaotic environments resulted in the highest levels of swarm diversity. Progressive

environments followed suit. The abrupt environments had significantly lower levels of swarm diversity than the progressive environments. This indicates that the regularised QPSO classifiers explored more while dealing with problems that had chaotic environments than those problems with other environments.

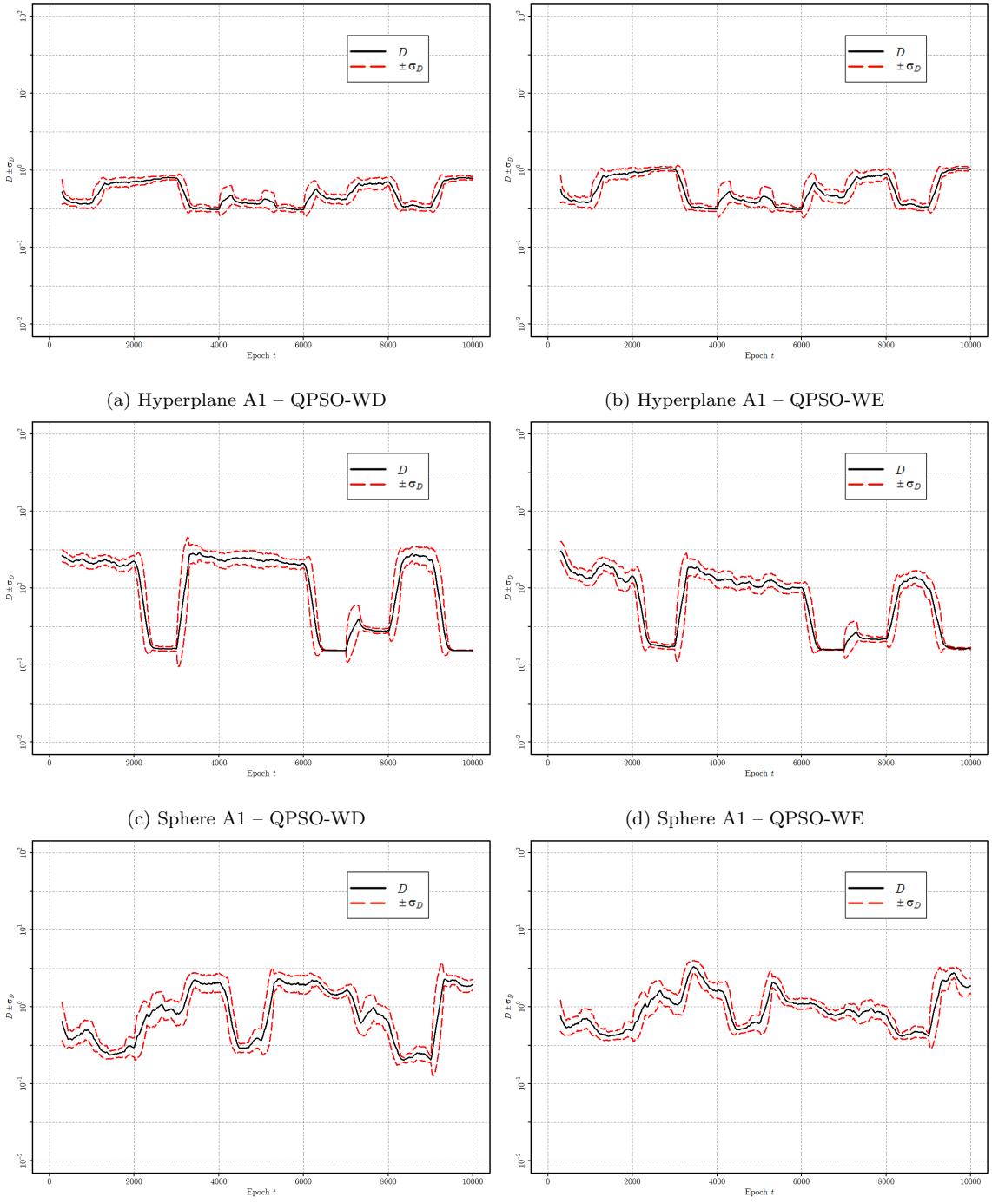
Decreasing the spatial severity of chaotic environments, to make them progressive, allowed for slightly more exploitative behaviour. Temporal severity, however, demonstrated the greatest effect on the swarm diversity of the regularised QPSO classifiers. That is, decreasing temporal severity dramatically decreases swarm diversity. Thus, allowing the classifier to exploit more.

The above observations show that regularised QPSO classifiers possess the ability to detect environment changes in various kinds of dynamic environments in the SDCPs, and to react accordingly.

The aggressive penalisation of the weights by QPSO-WD (refer to Section 9.8) prevented QPSO-WD from exploiting. This was evident from the fact that its swarm diversity was the majority of the time significantly higher than that of QPSO-WE.

QPSO-WE showed potential as a classifier for noisy SDCPs, because it was able to maintain swarm diversity in the noisy SDCPs. Note that there is a high probability that the electricity domain contains noise, because the SEA problems caused the swarm diversity trends of the QPSO-WD to have a similar shape to that of the electricity problems.

On a closing remark, the swarm diversity trends standard deviation bands were very narrow. Thus, swarm diversity trends were very consistent. The very volatile raw MSE trends observed in Section 9.5 did therefore not have much of an effect on the regularised QPSO classifiers ability to manage swarm diversity.



(a) Hyperplane A1 – QPSO-WD

(b) Hyperplane A1 – QPSO-WE

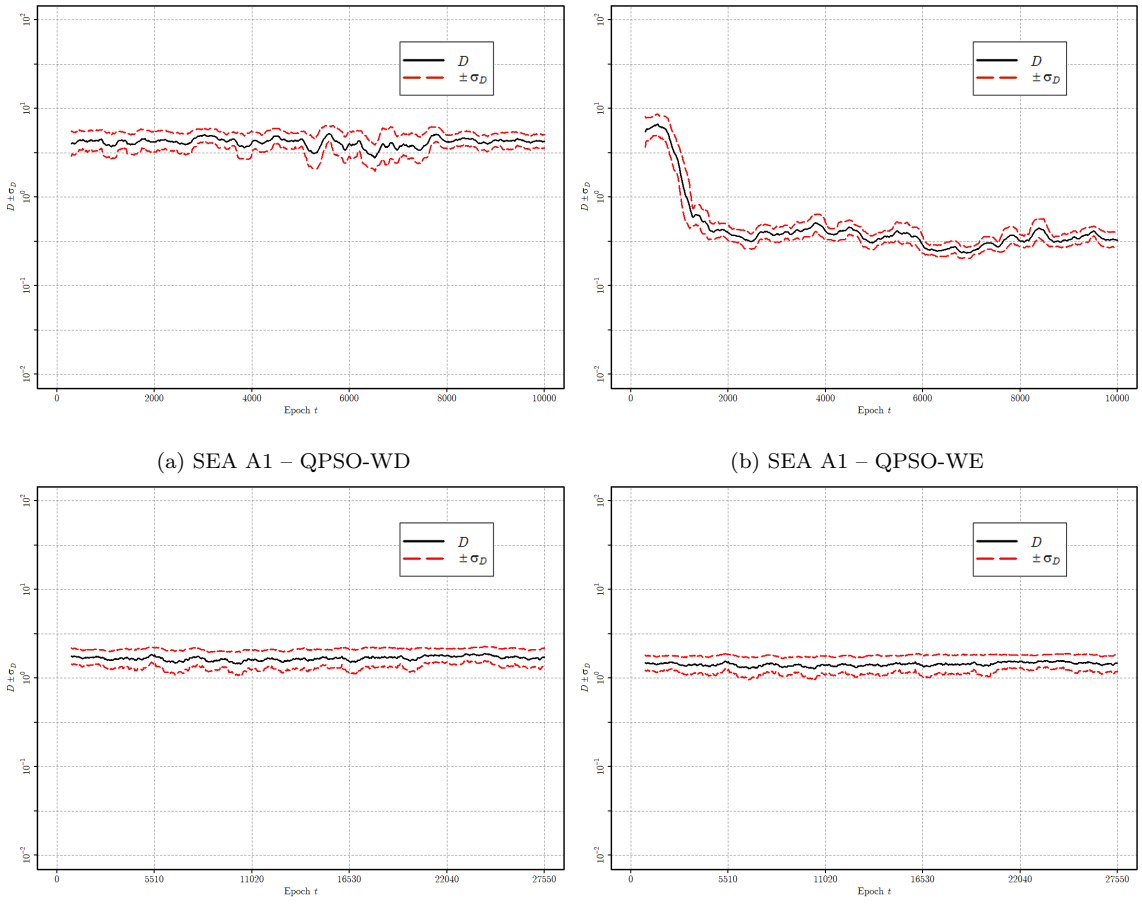
(c) Sphere A1 – QPSO-WD

(d) Sphere A1 – QPSO-WE

(e) Thresholds A1 – QPSO-WD

(f) Thresholds A1 – QPSO-WE

Figure 9.10: Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for the A1 hyperplane, A1 sphere and A1 thresholds problems



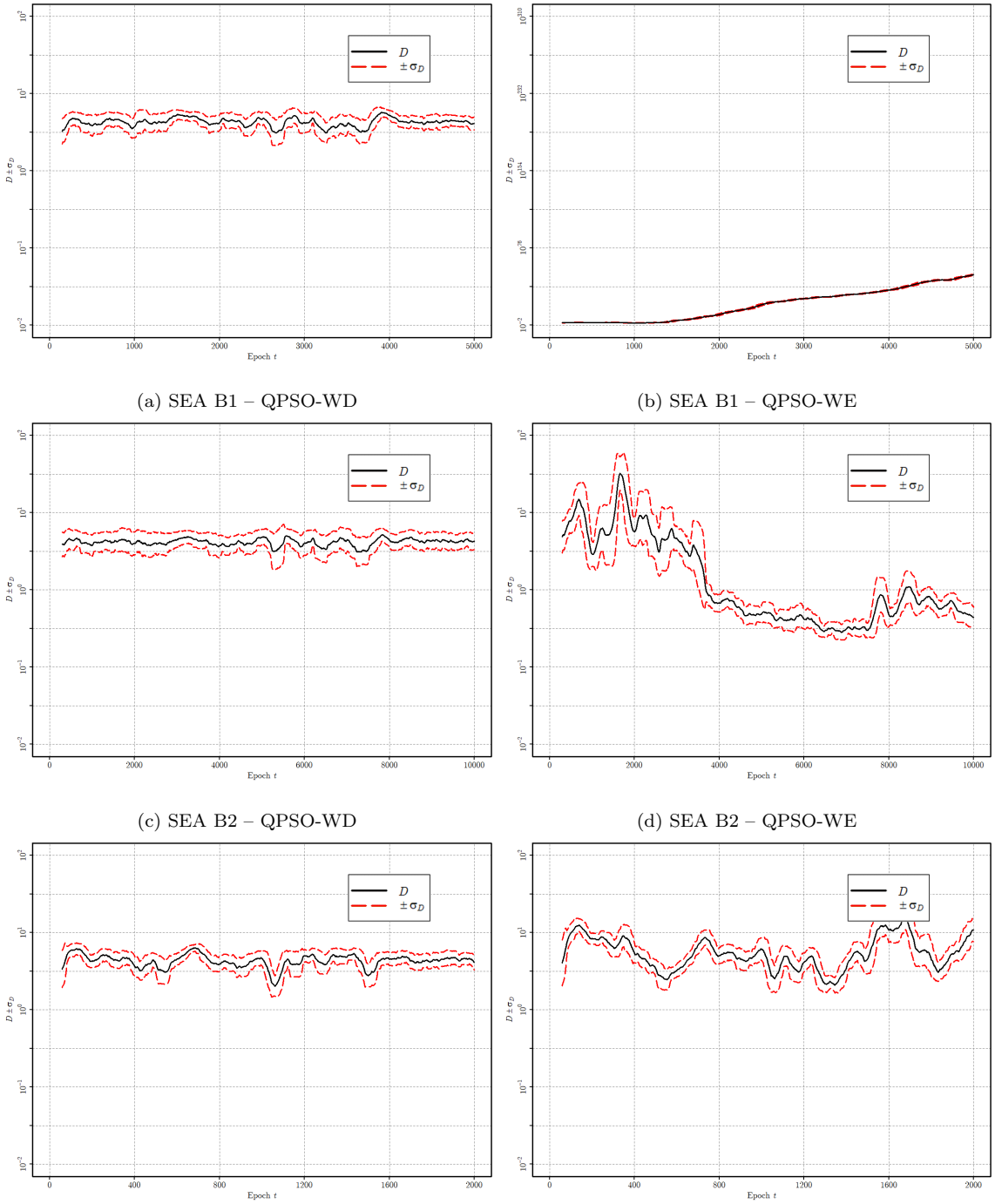
(a) SEA A1 – QPSO-WD

(b) SEA A1 – QPSO-WE

(c) Electricity A1 – QPSO-WD

(d) Electricity A1 – QPSO-WE

Figure 9.11: Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for the A1 SEA and A1 electricity problems



(a) SEA B1 – QPSO-WD

(b) SEA B1 – QPSO-WE

(c) SEA B2 – QPSO-WD

(d) SEA B2 – QPSO-WE

(e) SEA D2 – QPSO-WD

(f) SEA D2 – QPSO-WE

Figure 9.12: Swarm diversity trends of the QPSO-WD and QPSO-WE classifiers for selected SEA problems

9.10 Conclusion

This section concludes the empirical analysis. Section 9.10.1 presents the conclusions made with regards to the primary objective of the empirical analysis. Section 9.10.2 presents the conclusions made with regards to the secondary objectives of the empirical analysis.

9.10.1 Remarks on the primary objective

The primary objective of this analysis was to validate the nine hypotheses made in Section 8.1. The conclusions, with regards to the hypotheses, are as follows.

Hypothesis 1 stated that the proposed classifiers would outperform their counterparts, which were not regularised, on all of the SDCPs. The overall statistical ranks validated the hypothesis. BP-WE was considered the best BP classifier, and QPSO-WE was considered the best QPSO classifier. BP-WE, however, was 20% less accurate than BP-N, and ranked only marginally better than BP-N overall. BP-N had both the best overall accuracy rank and required significantly less parameter configuration to be tested, than BP-WE. Furthermore, BP-WD performed significantly worse than BP-N overall. Under consideration of all these aspects, hypothesis 1 was only valid for the QPSO classifiers.

Hypothesis 2 stated that QPSO-WD would outperform BP-WD on most of the SDCPs. The overall statistical ranks invalidated this hypothesis. QPSO-WD, however, did have a significantly better complexity performance rank than BP-WD. The saturation rank of the two classifiers also did not differ significantly. Furthermore, QPSO-WD had significantly less parameter configurations to test than BP-WD. Another problematic aspect with BP-WD was that the classifier suffered from a loss of control over its weights. Under consideration of all these aspects, hypothesis 2 was found to be partially validated.

Hypothesis 3 stated that QPSO-WE would outperform BP-WE on most of the SDCPs. The overall statistical ranks invalidated this hypothesis. QPSO-WE, however, did have a significantly better complexity performance rank. QPSO-WE also had significantly less parameter configurations to test than BP-WE. Like BP-WD, BP-WE

also suffered from the problematic loss of control over its weights. Under consideration of all these aspects, hypothesis 3 was found to be partially validated.

Hypothesis 4 stated that BP-WE and QPSO-WE would outperform their WD counterparts on all of the SDCPs. The overall statical ranks validated this hypothesis. Furthermore, the empirical analysis showed that the uniform weight penalisation approach by WD caused the under performance.

Hypothesis 5 stated that the proposed classifiers would have lower effective model complexity than their non-regularised counterparts on all of the SDCPs. The complexity ranks validated this hypothesis for the BP classifiers, but not for the QPSO classifiers. QPSO-N suffered from complete saturation. The complete saturation rendered the QPSO-N unusable and resulted in the abnormally high complexity performance. The complexity performance of QPSO-N is, therefore, considered as meaningless. Thus, the empirical analysis validated hypothesis 5 for the QPSO classifiers also.

Hypothesis 6 stated that the proposed classifiers would have lower levels of saturation than their non-regularised counterparts on all the SDCPs. The saturation ranks validated this hypothesis for the QPSO classifiers, but only partially validated the hypothesis for the BP classifiers. That is, only BP-WE had significantly lower saturation levels than BP-N. Furthermore, there were rising saturation trends amongst all of the BP classifiers due to the loss of control over their weights. This loss of control was accelerated through the use of regularisation. Thus, the empirical analysis invalidated hypothesis 6 for all the proposed BP classifiers.

Hypothesis 7 stated that the performance of the proposed classifiers would not scale well with an increase in noise. The empirical analysis validated this hypothesis for the BP classifiers. The regularised QPSO classifiers, however, performed very well in the noisy SEA SDCPs. The regularised QPSO classifiers, especially QPSO-WE, showed potential for noisy SDCPs.

Hypothesis 8 stated that BP-WD and BP-WE would not be able to handle the dynamic environments of the SDCPs as effectively as QPSO-WD and QPSO-WE would. Between the regularised BP and regularised QPSO classifiers, accuracy performance tended to favour the regularised BP classifiers, regardless of the dynamic environment of the SDCPs. The accuracy performance trends of the regularised BP classifiers, however,

showed signs of overfitting to previous environment instance, whereas the regularised QPSO classifiers did not. Furthermore, the regularised QPSO classifiers maintained stability in both saturation and complexity performance regardless of the environment of a SDCP. The empirical analysis, therefore, only had enough evidence to partially validate hypothesis 8. SDCPs with significantly more patterns need to be tested.

Hypothesis 9 stated that QPSO-WD and QPSO-WE would be able to maintain their swarm diversity when dealing with the SDCPs. The empirical analysis validated this hypothesis. Note that QPSO-WE did show some loss of swarm diversity control in a negligible amount of runs belonging to the SEA B1 problem.

All in all, five of the nine hypothesis were empirically validated, namely hypotheses 4, 5, 9. The remaining four hypotheses were partially validated, namely 1, 2, 3, 6, 7, 8.

9.10.2 Remarks on the secondary objectives

The secondary objectives of the empirical analysis were to gain a better understanding of how the classifiers dealt with SDCPs, and to determine what improvements could be made to the proposed classifiers. The conclusions with regards to the secondary objectives, are as follows.

Remarks on saturation

Saturation in the BP classifiers increased slowly over time. The trend was caused by a *loss of control over the weights*, due to the zero gradients of the ReLU activation functions in dynamic environments. The BP weights adjustment algorithm should, therefore, not be used to train ReLU FFNNs that deal with SDCPs. Regularisation only antagonises the loss of control over the weights, with BP-WE even experiencing complete saturation for the electricity A4 problem.

Saturation in the QPSO classifiers was severe when it occurred. The complete saturation was due to the weights exploding uncontrollably because of a *lack of search space information*, i.e. boundaries. Regularisation managed to successfully reduce the saturation experienced by the QPSO weights adjustment algorithm. QPSO-WE showed some ability with regards to distinguishing between necessary and unwanted saturation in most SDCPs. QPSO-WD, however, simply curbed all saturation regardless.

Complete saturation of the classifiers resulted in extreme levels of complexity reduction. The PCC results correlated to the number of output neurons. That is, the more output neurons a completely saturated classifier had, the more the accuracy performance degraded. The use of MSE measures in addition to the PCC measures assisted with detecting complete saturation.

Furthermore, complete saturation caused the accuracy performance trends based on the same error measure to group together, and the complexity reduction trends to stabilise at a high level, after some initial period of learning. Complete saturation, lastly, lead to meaningless exploration by the swarm of the QPSO classifiers.

Both φ_g and φ_v delivered almost exactly the same results. Thus, φ_g can be used by a streamed classifier during learning, if needed. Furthermore, the appropriate upper bound for the saturation measures, i.e. φ_v and φ_g, when using the proposed architecture was found to be one. Higher upper bounds produced irreverent findings about the saturation levels. The saturation measure, therefore, can be used with ReLU activation functions if the upper bound used is the same as the upper bound of the output neuron.

Lastly, there was no strong evidence to support the idea that the ReLU activation functions decreased the saturation levels of the hidden neurons in streamed data classifiers.

Remarks on accuracy

BP-N was the best at generalising and remembering patterns in SDCPs. However, the slow continuous rise of saturation warned of potential overfitting in the long run. Training accuracies tended to be very good for all classifiers, except QPSO-N.

The one-pass requirement resulted in highly volatile raw MSE_t and MSE_g trends of SDCPs. So much so that MSE_t and MSE_g would bounce between zero and one. The volatile raw MSE trends, however, did not stop the regularised QPSO classifiers from managing swarm diversity. Further research into methods for reducing the volatility, or creating classifiers that are capable of handling the volatility needs to be conducted.

QPSO-WD could not exploit solutions effectively, because of the aggressive penalisation of the weights by WD. Furthermore, the performance trends showed that the regularised QPSO classifiers were unable to adjust the rate at which information was

unlearnt as needed. If the regularised QPSO classifiers had some way of dynamically adjusting the unlearning rate, then they would gain a significant accuracy performance boost. A possible solution would be basing the aggressiveness of the penalisation of the weights on the accuracy performance, i.e. better accuracy performance means less aggressive penalisation and vice versa.

Lastly, the regularised QPSO classifiers generally had both low MSE and PCC errors.

Remarks on complexity

The regularised QPSO classifiers managed to get effective architectures close to the size of optimal architectures found by Rakitianskaia [94]. In some cases even better. Architecture selection should, thus, be employed by fully connected FFNN streamed data classifiers. Preferably with dynamic weights adjustment algorithms. Architecture selection algorithms for SDCPs must also function at a synapse-level instead of a neuron-level for the best performance.

The complexity performance results showed that the pruning algorithm proposed by Engelbrecht [31] was effective at determining the effective model for the FFNN classifiers. Ways of integrating the pruning algorithm with streamed algorithms should be researched as it can be an effective pruning algorithm for streamed data classifiers.

Furthermore, the ReLU activation functions stored less information than the sigmoid activation functions, because in general the classifiers did not manage to reduce the size of the effective architectures to that of the optimal architectures.

Lastly, the accuracy and complexity trends of the BP classifiers showed that the BP classifiers allocated information efficiently to the neurons during periods of high accuracy.

Lastly, the oversize synapse ratio, $n_{s_{or}}$, can be used to estimate computational complexity.

Remarks on noise

The results showed that regularised QPSO classifiers managed to unlearn noise, while the BP classifiers did not. Noise had an effect on the complexity performance of the classifiers. That is, the more a streamed data classifier captured noise, the less the classifier could reduce its model complexity. Furthermore, capturing noise lead to increased

levels of saturations, especially if there was prolonged exposure to noisy patterns.

Remarks on problem dimensionality

High dimensional SDCPs generally had a detrimental effect on the accuracy and complexity performance of the regularised QPSO classifiers, but not on the BP classifiers. Problem dimensionality did not have any significant effect on the saturation levels.

Remarks on problem difficulty

The problem difficulty classification scheme proposed in Section 8.3.6 was supported by the empirical findings. Moderate-I problems were found to be more difficult than moderate-II problems.

Lastly, increasing the window step size increased the intra-environment volatility in performance trends. On the other hand, increasing window frequency decreased the intra-environment volatility in performance trends and improved accuracy performance.

Remarks on dynamic environments

The BP classifiers saturated slower in SDCPs that had a high temporal severity and a low amount of information, i.e. a high window step size. On the other hand, high temporal severity prevented the regularised QPSO classifiers from learning, because their unlearning rates were too large. This further supported the need to do further research into dynamically adjusting the unlearning rates of the regularised QPSO classifiers.

The classifiers generally performed the best in abrupt environments, and second best in the chaotic environments. High levels of temporal severity, therefore, had a significantly more detrimental effect on the performance of the classifiers than the high levels of spatial severity. High spatial severity mostly countered the detrimental effect of high temporal severity.

Lastly, decreasing spatial severity of the A1 problems generally lead to more volatile swarm diversity trends, while decreasing the temporal severity of the A1 problems resulted in smoother swarm diversity trends. Thus, the regularised QPSO were able to manage their swarm diversity in all classes of dynamic environments.

Remarks on general behaviour

The BP and QPSO weights adjustment algorithm behaved significantly differently from each other for SDCPs. Behaviour amongst the BP classifiers did not differ significantly in the majority of the SDCPs. On the other hand, behaviour amongst the QPSO classifiers did differ significantly.

Remarks on overfitting

There were signs of overfitting amongst the classifiers in the accuracy performance trends. Overfitting could, however, not be validated by the overfitting indicators, because neither O_ρ nor O_{MSE_g} were appropriate overfitting indicators for SDCPs. Both measures failed, because of the volatile accuracy trends, dynamic environments, and for not taking the behaviour of MSE_t into account. Future research into more suitable measures is, therefore, required.

Remarks on control parameters

BP-N had the best trade-off between performance and the time taken to tune the control parameters. QPSO-WE provided the second best trade-off. This furthered the case for using QPSO-WE and BP-N for SDCP.

The value ranges of the momentum term (α) and learning rate (η) control parameters for the BP classifiers could both be reduced to $[0, 0.4]$. The finding that the BP classifiers kept their learning rate very low, showed that the BP classifiers required a very low learning rate and momentum for SDCPs in order to cope with the dynamic environments.

The values of the radius (r) control parameter for the regularised QPSO classifiers never exceed 0.25, and were mostly 0.1. The recommendation by Harrison *et al.* [52] to keep the radius of QPSO small was, therefore, supported by the empirical analysis.

The ranges and values of regularisation coefficient (λ_r) control parameter was found to be problem domain and weights adjustment algorithm dependent. Furthermore, values for the weights relevancy threshold (w_0) control parameter should be selected from a wide range, i.e. $(0, 1]$, when optimising the parameter for streamed data classifiers.

Remarks on benchmark suite

The benchmark suite must acquire more real-world problem domains as there is only one. Benchmark problems with significantly more patterns should be constructed from the problem domains, because the data streams did not allow the streamed classifiers to train long enough to confirm the effect of the performance trends that progressed slowly.

The benchmark suite must also acquire more artificial problems with extremely high temporal severity, because this was suggested to be a key characteristic of the real-world electricity domain. Benchmark SDCP that are quasi-static in relation to the other benchmark SDCPs must also be added, because none of the benchmark problems represented quasi-static environments.

9.11 Summary

This chapter presented the empirical analysis of the six classifiers that were investigated by this thesis. The analysis included the statistical analysis and performance trend analysis of the saturation, accuracy and complexity performance measures. These results were, further, used to rank the classifiers. Additional insight into the characteristics of the six classifiers were gained by analysing their overfitting behaviour, control parameters, weight distributions, and swarm diversity.

The empirical analysis validated several of the hypothesis made in Section 8.1. Furthermore, the empirical analysis provided important insight into the behavioural characteristics of the streamed data classifiers, as well as potential areas of improvement.

Next, Chapter 10 presents the conclusions made by this thesis.

Chapter 10

Conclusions

This chapter presents a summary of the work done in this thesis and the possible derivations thereof.

The remainder of the chapter is organised as follows. Section 10.1 summarises the conclusions made during the thesis with regards to the objectives set out in Section 1.2. Lastly, Section 10.2 provides possible topics for future research.

10.1 Summary of Conclusions

The main objective of this thesis was to investigate the application of regularised feed forward neural networks (FFNNs), trained by quantum particle swarm optimisation (QPSO), as classifiers for streamed data classification problems (SDCPs). This was accomplished as follows:

Background for the investigation was provided through detailed discussions on the topics of computational intelligence (CI), particle swarm optimisers (PSOs), artificial neural networks (ANNs) and streamed data problems (SDPs). The discussions focused on dynamic optimisation problems, QPSOs, FFNNs, regularisation, saturation, SDCP, and current literature on streamed data classifiers.

The ability to continuously adapt was found to be instrumental to the success of a streamed data classifier. Recent studies have successfully incorporated dynamic weights optimisation approaches, such as QPSO, into FFNNs for dynamic classification problems.

Studies thus far, however, have not considered architecture optimisation with dynamic weights optimisation for dynamic classification problems, let alone SDCP. This is despite the fact that efficient approaches, such as regularisation, exist.

This dissertation, therefore, proposed an online learning algorithm based on QPSO and regularisation to train FFNNs for SDCPs. weight decay (WD) and weight elimination (WE) were used as regularisers. Because regularisation literature with regards to SDCPs is limited, a back propagation (BP) variant of the learning algorithm was also proposed. The learning algorithms were each applied to a 3-layer FFNNs architecture, which used rectified linear unit (ReLU) activation functions and summation units. The four resulting classifiers were named QPSO-WD, QPSO-WE, BP-WD, and BP-WE.

The investigation empirically evaluated the proposed classifiers by pitting them against each other, and their non-regularised counterparts, namely QPSO-N and BP-N, on 80 benchmark problems. The benchmark problems were derived from five problem domains using the sliding window algorithm.

To provide a better understanding of the benchmark problems, this thesis proposed several novel methods for quantitatively evaluating SDCPs. The methods included an algorithm to extract the environment instances from an extract of a SDCP; a set of quantitative measures that can be used to determine the spatial and temporal severity in SDCPs; and a classification scheme to quantify, and describe how difficult it is for a classifier to accurately learn a SDCPs. These methods were successfully applied to the 80 benchmark problems, and provided significant insight into the findings of the empirical investigation.

The results of the benchmarking process were analysed using a statistically-sound approach employing descriptive statistics, Mann-Whitney U (MWU)-based ranking, and performance trend analysis. The following findings were made by the empirical analysis:

The BP-N classifier learned SDCPs quickly and accurately. BP-N was the most accurate classifier of all the classifiers. BP-N, however, had a tendency to experience increasing saturation as the data stream progressed. The same increasing saturation trend occurred in BP-WD and BP-WE, with BP-WE completely saturating in some cases.

Saturation in the BP classifiers was caused by a loss of control over the weights,

i.e. the learning algorithm was unable to adjust the weights. This loss of control was a result of the combined effect that the dynamic environments and the zero gradients of the ReLU activation function had on the BP algorithm. Furthermore, regularisation only accelerated the loss of control.

QPSO-WE provided well-rounded performance, and overall showed the most potential as a streamed data classifier. The main issue with the QPSO-WE was its accuracy performance. On the other hand, the aggressive weight penalisation of WD resulted in very stable complexity and saturation performance, but degraded accuracy performance significantly. The only time QPSO-WD was found worth considering was in noisy SDCPs.

QPSO-N completely saturated all the time. Unlike the BP classifiers, saturation in the QPSO classifiers was caused by a lack of search space information, i.e. boundaries imposed by additional constraints on the classifier.

With regards to complexity performance, the regularised QPSO classifiers managed to get effective architectures close to the size of the optimal architectures found by Rakitianskaia [94]. In some cases even better. The regularised BP classifiers failed to get architectures close to the optimal architectures.

Architecture selection should, therefore, be employed by fully connected FFNN streamed data classifiers, preferably with dynamic weights adjustment algorithms. Furthermore, architecture selection algorithms for SDCPs should function at a synapse-level instead of a neuron-level.

Lastly, the performance trends for accuracy measures experienced high levels of volatility. This volatility was found to be one of the main causes for the unforeseen behaviours of the classifiers. The one-pass requirement was identified as the main culprit, because it does not allow the classifier to see the entire environment instance per epoch.

The above findings by this thesis showed that regularised FFNN classifiers, using a dynamic weight adjustment algorithm, such QPSO, had potential. However, several improvements need to be made to make the QPSO classifiers suitable for SDCPs.

10.2 Future Work

The following future work is derived from the findings made during the course of this thesis:

Future research into alternatives to the various components making up the proposed classifiers should be done. Potential avenues include the investigation into the usage of other dynamic PSOs weight adjustment algorithms, using product units, instead of summation units, using sigmoid and other bounded activation functions, and using other regularisers.

In terms of alternative algorithmic settings, the performance of small swarm sizes for PSO classifiers should be investigated. If smaller swarms result in similar accuracy, then the computational complexity of the classifiers can be reduced. Another potential area that can be investigated is to use different error functions for the classifiers. Because SDCPs are classification problems, a possible starting point is to use the cross entropy loss function instead of the mean square error (MSE) to train the classifiers. Another algorithmic setting that should be investigated is the use of different weight initialisation ranges, such as *He normal initialisation* proposed by He *et al.* [55].

The empirical analysis showed that the current set of benchmark SDCPs needs more benchmark problems. Thus the benchmark suite should be built on to create a more comprehensive benchmark suite for future research. This includes adding more domains, especially real-world domains, and more examples of quasi-static, abrupt, and chaotic environments. Additionally, benchmark problems with longer data streams should also be added. Another avenue that can be considered is to add additional functionality to the sliding windows algorithm, such as random shuffling of patterns that are repeated in an environment instance and noise generation.

The empirical analysis revealed six behaviours/characteristics pertaining to the classifiers, that if mitigated or exploited, could improve performance. Firstly, the regularised QPSO classifiers had a tendency to get both low MSE and percentage correct classifications (PCC) errors. A potential starting point would be to investigate the effect of reducing the range of the output neurons to $[0.1, 0.9]$. That is, if the activation value of the output neuron is more than 0.9, then the value is changed to 1.

Secondly, the classifiers experienced volatile accuracy performance trends due to the

one-pass requirement. If the volatility could be reduced, then performance could be improved. A potential starting point is to investigate the use of fading factors to calculate the training error as suggested by Gama *et al.* [45].

Thirdly, noise had a negative effect on the classifiers. Research into creating a low computational complexity noise-filtering mechanism for SDCP classifiers should be done. A starting point would be to first do a more detailed investigation into the effect of noise on the proposed classifiers, and the current approaches to noise filtering.

Fourth, the "loss of control over weights" in the BP classifiers caused unwanted saturation. The ReLU activation function was the main cause of this. Alternative activation functions for the BP classifiers should investigated, such as the sigmoid or the leaky ReLU activation functions.

The fifth behaviour was the unwanted saturation for the QPSO classifiers that was caused by a "lack of search space information". Regularisation clearly reduced this issue, but not completely. Additional forms of controlling saturation in the classifiers should be investigated, such as velocity clamping and search space boundaries.

The sixth and final behaviour that should be considered is that the regularised QPSO classifiers could not manage their unlearning rate dynamically, because of the static values of the regularisation coefficient, λ_r, and the relevance threshold, w_0. A possible solution would be basing these control parameters on accuracy in such a way that better accuracy performance would result in less aggressive penalisation and vice versa. Another possible solution would be to dynamically adjust η for the BP classifiers and λ_r for all the classifiers.

The pruning algorithm by Engelbrecht [31] proved effective in determining the effective model complexity. Future research into adapting it to replace regularisation should be done. If done successfully, this would allow effective computational complexity to be realised during learning. A starting point would be to see if basing the effective reduction in complexity, Ω_r, on the generalisation set, D_g, instead of the validation set, D_v, returns similar results to what was observed during the empirical analysis.

Finding alternative overfitting measures for the streamed data classifiers is another topic for future research, because the current measures proved inadequate. If alternatives are not found, then applying algorithms to SDCP that are dependent on detecting

overfitting would not be possible.

Investigate the use of regularisation with PSO in a non-ANN context, for the purpose of potentially replacing velocity clamping.

Lastly, the streamed benchmark problem approach to tuning control parameters proved effective, but not ideal. A self-adaptive version of the proposed classifiers should, therefore, be investigated in the future. A possible starting point is looking at the self-adaptive QPSO proposed by Pamparà and Engelbrecht [90].

Bibliography

[1] C.C. Aggarwal. *Data Streams: Models and Algorithms*. Springer, first edition, 2007.

[2] E. Alpaydn. *Introduction to Machine Learning*. The MIT Press, second edition, 2010.

[3] P.J. Angeline. Tracking extrema in dynamic environments. In *Proceedings of the 6th International Conference on Evolutionary Programming*, pages 335–345. Springer, April 1997.

[4] M. Azzeh. Software effort estimation based on optimized model tree. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, September 2011.

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

[6] T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *Applications of Evolutionary Computing. EvoWorkshops. Lecture Notes in Computer Science*, volume 3005, pages 489–500. Springer, 2004.

[7] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computing*, 10(4):459–472, August 2006.

[8] T. Blackwell, J. Branke, and X. Li. Particle swarms for dynamic optimization problems. In *Swarm Intelligence*, pages 193–217. Springer, 2008.

[9] T.M. Blackwell and P.J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pages 19–26. Morgan Kaufmann Publishers Incorporated, July 2002.

[10] A. Bosman, A. Engelbrecht, and M. Helbig. Fitness landscape analysis of weight-elimination neural networks. *Neural Processing Letters*, 48(1):353–373, August 2018.

[11] J. Branke, E. Salihoğlu, and Ş. Uyar. Towards an analysis of dynamic environments. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1433–1440. ACM, 2005.

[12] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[13] F. Chu, Y. Wang, and C. Zaniolo. An adaptive learning approach for noisy data streams. In *Proceedings of the International Conference on Data Mining*, pages 351–354. IEEE, November 2004.

[14] C. W. Cleghorn. Particle swarm optimization: Understanding order-2 stability guarantees. In *Proceedings of the International Conference on the Applications of Evolutionary Computation*, pages 535–549. Springer, 2019.

[15] C. W. Cleghorn and B. Stapelberg. Particle swarm optimization: Stability analysis using n-informers under arbitrary coefficient distributions, 2020.

[16] C.W. Cleghorn and A. Engelbrecht. Particle swarm optimizer: The impact of unstable particles on performance. In *Proceedings of the Symposium Series on Computational Intelligence*, pages 1–7. IEEE, December 2016.

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

[18] C.W. Cleghorn and A.P. Engelbrecht. Particle swarm convergence: An empirical investigation. In *Proceedings of the Congress on Evolutionary Computation*, pages 2524–2530. IEEE, July 2014.

[19] C.W. Cleghorn and A.P. Engelbrecht. Particle swarm stability: a theoretical extension using the non-stagnate distribution assumption. *Swarm Intelligence*, 12(1):1–22, March 2018.

[20] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1951–1957. IEEE, July 1999.

[21] Y. Cui, C. Surpur, S. Ahmad, and J. Hawkins. A comparative study of htm and other neural network models for online sequence learning with streaming data. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1530–1538. IEEE, July 2016.

[22] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[23] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, October 2012.

[24] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the SIGKDD International Conference on Knowledge discovery and data mining*, volume 6, pages 71–80. ACM, August 2000.

[25] J.G.O.L. Duhain and A.P. Engelbrecht. Towards a more complete classification system for dynamically changing environments. In *Proceedings of the Congress on Evolutionary Computation*, pages 1–8. IEEE, June 2012.

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

[27] K.B. Dyer, R. Capo, and R. Polikar. Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):12–26, January 2014.

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

[29] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 84–88. IEEE, July 2000.

[30] R.C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the Congress on Evolutionary Computation*, pages 94–100, May 2001.

[31] A. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, 12(6):1386–1399, November 2001.

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

[33] A.P. Engelbrecht. *Computational Intelligence: An introduction*. John Wiley and Sons Ltd., second edition, 2007.

[34] A.P. Engelbrecht. *Heterogeneous Particle Swarm Optimization*, pages 191–202. Springer, September 2010.

[35] A.P. Engelbrecht. Particle swarm optimization: Global best or local best? In *Proceedings of the BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 124–135. IEEE, September 2013.

[36] A.P. Engelbrecht. Particle swarm optimization: Iteration strategies revisited. In *Proceedings of the 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 119–123. IEEE, September 2013.

[37] A.P. Engelbrecht. Roaming behaviour of unconstrained particles. In *Proceedings of the BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 104–111. IEEE, September 2013.

[38] A.P. Engelbrecht. Asynchronous particle swarm optimization with discrete crossover. In *Proceedings of the Symposium on Swarm Intelligence*, pages 1–8. IEEE, December 2014.

[39] W. Ertel. *Introduction to Artificial Intelligence*. Springer, first edition, 2011.

[40] S.E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann Publishers Incorporated, 1989.

[41] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006.

[42] M. Fernndez-Redondo and C. Hernndez-Espinosa. Weight initialization methods for multilayer feedforward. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 119–124. D-Facto public, April 2001.

[43] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I.H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, July 1998.

[44] J. Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, April 2012.

[45] J. Gama, R. Sebastião, and P.P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, January 2009.

[46] E. Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–510, December 1989.

[47] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[48] D. Gies and Y. Rahmat-Samii. Vector evaluated particle swarm optimization (vepso): optimization of a radiometer array antenna. In *Proceedings of the Antennas and Propagation Society Symposium*, volume 3, pages 2297–2300. IEEE, June 2004.

[49] S.U. Guan and S. Li. Incremental learning with respect to new incoming input attributes. *Neural Processing Letters*, 14(3):241–260, December 2001.

[50] A. Gupta and S.M. Lam. Weight decay backpropagation for noisy data. *Neural Networks*, 11(6):1127–1138, 1998.

[51] S.L. Harris and D.M. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, arm edition, 2016.

[52] K. Harrison, B.M. Ombuki-Berman, and A.P. Engelbrecht. The effect of probability distributions on the performance of quantum particle swarm optimization for solving dynamic optimization problems. In *Proceedings of the Symposium Series on Computational Intelligence*, pages 242–250. IEEE, December 2015.

[53] K.R. Harrison, A.P. Engelbrecht, and B.M. Ombuki-Berman. The sad state of self-adaptive particle swarm optimizers. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 431–439, July 2016.

[54] E. Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3–4):157–325, 2015.

[55] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the International Conference on Computer Vision*, pages 1026–1034. IEEE, 2015.

[56] M. Helbig and A.P. Engelbrecht. Analysing the performance of dynamic multi-objective optimisation algorithms. In *Proceedings of the Congress on Evolutionary Computation*, pages 1531–1539. IEEE, June 2013.

[57] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, December 1991.

[58] R. Huang and S. Tong. Evolving product unit neural networks with particle swarm optimization. In *Proceedings of the International Conference on Image and Graphics*, pages 624–628. IEEE, September 2009.

[59] E.J. Hughes. Evolutionary many-objective optimisation: many once or one many? In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 222–227. IEEE, September 2005.

[60] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh SIGKDD International Conference on Knowledge discovery and data mining*, pages 97–106. ACM, August 2001.

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

[62] A. Jadhav and L. Deshpande. An efficient approach to detect concept drifts in data streams. In *Proceedings of the 7th International Advance Computing Conference*, pages 28–32. IEEE, January 2017.

[63] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *The Journal of Soft Computing*, 9(1):3–12, January 2005.

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

[65] C. Kim, S.H. Cha, Y.J. An, and N. Wilson. On roc curve analysis of artificial neural network classifiers. In *Proceedings of the 30th International Florida Artificial Intelligence Research Society Conference*, pages 318–321. AAAI Publications, May 2017.

[66] S.B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, April 2013.

[67] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, and M. Wozniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

[68] A. Krogh and J.A. Hertz. A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 950–957. Morgan Kaufmann Publishers Incorporated, December 1991.

[69] B. Krose and P. van der Smagt. *An introduction to Neural networks*. University of Amsterdam, eighth edition, November 1996.

[70] R. V. Kulkarni, S. H. Patil, and R. Subhashini. An overview of learning in data streams with label scarcity. In *Proceedings of the International Conference on Inventive Computation Technologies*, volume 2, pages 1–6, August 2016.

[71] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, May 2015.

[72] Y. LeCun, L., G.B. Orr, and K.R. Müller. *Efficient BackProp*, volume 7700, pages 9–50. Springer, 1998.

[73] D.H. Lee and D.S. Kang. The application of the artificial neural network ensemble model for simulating streamflow. *Procedia Engineering*, 154:1217–1224, 2016.

[74] L.R. Leerink, C.L. Giles, B.G. Horne, and M.A. Jabri. Learning with product units. *Advances in Neural Information Processing Systems*, 7:537–544, 1995.

[75] J. Leskovec, A. Rajaraman, and J.D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, second edition, November 2014.

[76] N.Y. Liang, G.B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.

[77] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, April 1988.

[78] N. Littlestone and M.K. Warmut. The weighted majority algorithm. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 256–261. IEEE, October 1989.

[79] D. Liu, T.S. Chang, and Y. Zhang. A constructive algorithm for feedforward neural networks with incremental training. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(12):1876–1879, December 2002.

[80] V. Losing, B. Hammer, and H. Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.

[81] A.L. Maas, A.Y. Hannun, and A.Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[82] K.M. Malan and A.P. Engelbrecht. Algorithm comparisons and the significance of population size. In *Proceedings of the Congress on Evolutionary Computation*, pages 914–920. IEEE, June 2008.

[83] H.B. Mann and D.R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60, January 1947.

[84] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

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

[86] H. Mhaskar, Q. Liao, and T. Poggio. Learning functions: When is deep better than shallow. *CBMM Memo*, (045), 2016.

[87] R.W. Morrison. Performance measurement in dynamic environments. In J. Branke, editor, *Proceedings of GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, August 2003.

[88] B. Ngom, A. Boly, and R. Chiky. "forgetting functions" in the context of data streams for the benefit of decision-making. In *Proceedings of the International Workshop on Computational Intelligence for Multimedia Understanding*, pages 1–5, October 2016.

[89] O. Olorunda and A.P. Engelbrecht. Measuring exploration/exploitation in particle swarm using swarm diversity. In *Proceedings of the Congress on Evolutionary Computation*, pages 1128–1134. IEEE, May 2008.

[90] G. Pamparà and A.P. Engelbrecht. Self-adaptive quantum particle swarm optimization for dynamic environments. Unpublished article (currently under review), 2019.

[91] K. Potdar, T.S. Pardawala, and C.D. Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, 175:7–9, 10 2017.

[92] S. Pramod and O.P. Vyas. Data stream mining: A review on windowing approach. *Global Journal of Computer Science and Technology Software and Data Engineering*, 12(11), 2012.

[93] M. Pratama, P. P. Angelov, J. Lu, E. Lughofer, M. Seera, and C. P. Lim. A randomized neural network for data streams. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3423–3430. IEEE, May 2017.

[94] A. Rakitianskaia. Using particle swarm optimisation to train feedforward neural networks in dynamic environments. Master's thesis, University of Pretoria, December 2011.

[95] A. Rakitianskaia and A. Engelbrecht. Weight regularisation in particle swarm optimisation neural network training. In *Proceedings of the Symposium on Swarm Intelligence*, pages 1–8. IEEE, December 2014.

[96] A. Rakitianskaia and A. Engelbrecht. Saturation in pso neural network training: Good or evil? In *Proceedings of the Congress on Evolutionary Computation*, pages 125–132. IEEE, May 2015.

[97] A. Rakitianskaia and A. Engelbrecht. Measuring saturation in neural networks. In *Proceedings of the Symposium Series on Computational Intelligence*, pages 1423–1430. IEEE, December 2015.

[98] A. Rakitianskaia and A.P. Engelbrecht. Training neural networks with pso in dynamic environments. In *Proceedings of the Congress on Evolutionary Computation*, pages 667–673. IEEE, May 2009.

[99] A.S. Rakitianskaia and A.P. Engelbrecht. Training high-dimensional neural networks with cooperative particle swarm optimiser. In *Proceedings of the Congress on Evolutionary Computation*, pages 4011–4018. IEEE, July 2014.

[100] A. Robël. The dynamic pattern selection algorithm: Effective training and controlled generalization of backpropagation neural networks. Technical report, Institut fur Angewandte Informatik, Technische Universitat, Berlin, 1994.

[101] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, November 1958.

[102] A. Sancho-Asensio, A. Orriols-Puig, and E. Golobardes. Robust on-line neural learning classifier system for data stream classification tasks. *Soft Computing*, 18(8):1441–1461, August 2014.

[103] A.F.C. Santos, Í.P. Teles, O.M.P. Siqueira, and A.A. de Oliveira. *Big Data: A Systematic Review*, pages 501–506. Springer, 2017.

[104] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the International Conference on Evolutionary Computation*, pages 69–73. IEEE, May 1998.

[105] Y. Singh and A.S. Chauhan. Neural networks in data mining. *Journal of Theoretical and Applied Information Technology*, 5(6):37–42, 2009.

[106] B.R. Smith. *Neural Network Enhancement of Closed-Loop Controllers for Ill-Modeled Systems with Unknown Nonlinearities*. PhD thesis, Virginia Polytechnic Institute and State University, Virginia, December 1997.

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

[108] S. Sonoda and N. Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017.

[109] Z. Telec, B. Trawiński, T. Lasota, and G. Trawiński. Evaluation of neural network ensemble approach to predict from a data stream. In *Proceedings of the International Conference on Computational Collective Intelligence*, pages 472–482. Springer, September 2014.

[110] C.M. Teng. A comparison of noise handling techniques. In *Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference*, pages 269–273. AAAI, January 2001.

[111] C.K. Tham. On-line learning using hierarchical mixtures of experts. In *Proceedings of the Fourth International Conference on Artificial Neural Networks*, pages 347–351. IET, June 1995.

[112] A. Tsymbal. The problem of concept drift: Definitions and related work. Technical report, Trinity College, Dublin, 2004.

[113] J.M. Twomey and A.E. Smith. Performance measures, consistency, and power for artificial neural network models. *Mathematical and Computer Modelling*, 21(1-2):243–258, January 1995.

[114] F. van den Bergh and A.P. Engelbrecht. Training product unit networks using cooperative particle swarm optimisers. In *Proceedings of the International Joint Conference on Neural Networks*, pages 126–131. IEEE, July 2001.

[115] A.B. van Wyk and A.P. Engelbrecht. Analysis of activation functions for particle swarm optimised feedforward neural networks. In *Proceedings of the Congress on Evolutionary Computation*, pages 423–430. IEEE, July 2016.

[116] H. Wang, W. Fan, P.S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the SIGKDD International Conference on Knowledge discovery and data mining*, volume 9, pages 226–235. ACM, August 2003.

[117] F. Webster. *Theories of the Information Society*. Routledge, third edition, 2006.

[118] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Generalization by weight-elimination with application to forecasting. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, volume 3, pages 875–882. Morgan Kaufmann Publishers Incorporated, 1990.

[119] P.J. Werbos. *Beyond Regression: New tools for prediction and analysis in the Behavioural Sciences*. PhD thesis, Harvard University, Boston, 1974.

[120] L.F.A. Wessels and E. Barnard. Avoiding false local minima by proper initialization of connections. *IEEE Transactions on Neural Networks*, 3(6):899–905, 1992.

[121] B.M. Wilamowski. Neural network architectures and learning. In *Proceedings of the International Conference on Information Technology*, pages TU1–TU12. IEEE, 2003.

[122] D.R. Wilson and T.R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[123] K. Woods and K.W. Bowyer. Generating roc curves for artificial neural networks. *IEEE Transactions on Medical Imaging*, 16(3):329–337, June 1997.

[124] R. Xu and D.C. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):14–23, May 2005.

[125] F. Yoav and E.S. Robert. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14:771–780, October 1999.

[126] Z. Zainuddin and O. Pauline. Function approximation using artificial neural networks. *International Journal of Systems Applications, Engineering and Development*, 1(4):173–178, December 2007.

[127] C. Zhang, H. Shao, and Y. Li. Particle swarm optimisation for evolving artificial neural network. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2487–2490. IEEE, October 2000.

Appendix A

Performance trend graphs

This appendix provides the performance trend graphs for the saturation, accuracy and complexity performance trend analysis done in section 9.4. Figure A.1 provides the legend for the performance trend graphs in figures A.2 to A.41. Note that all these graphs are discussed in section 9.4.

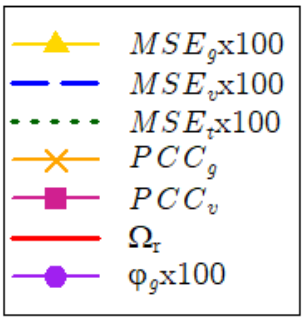
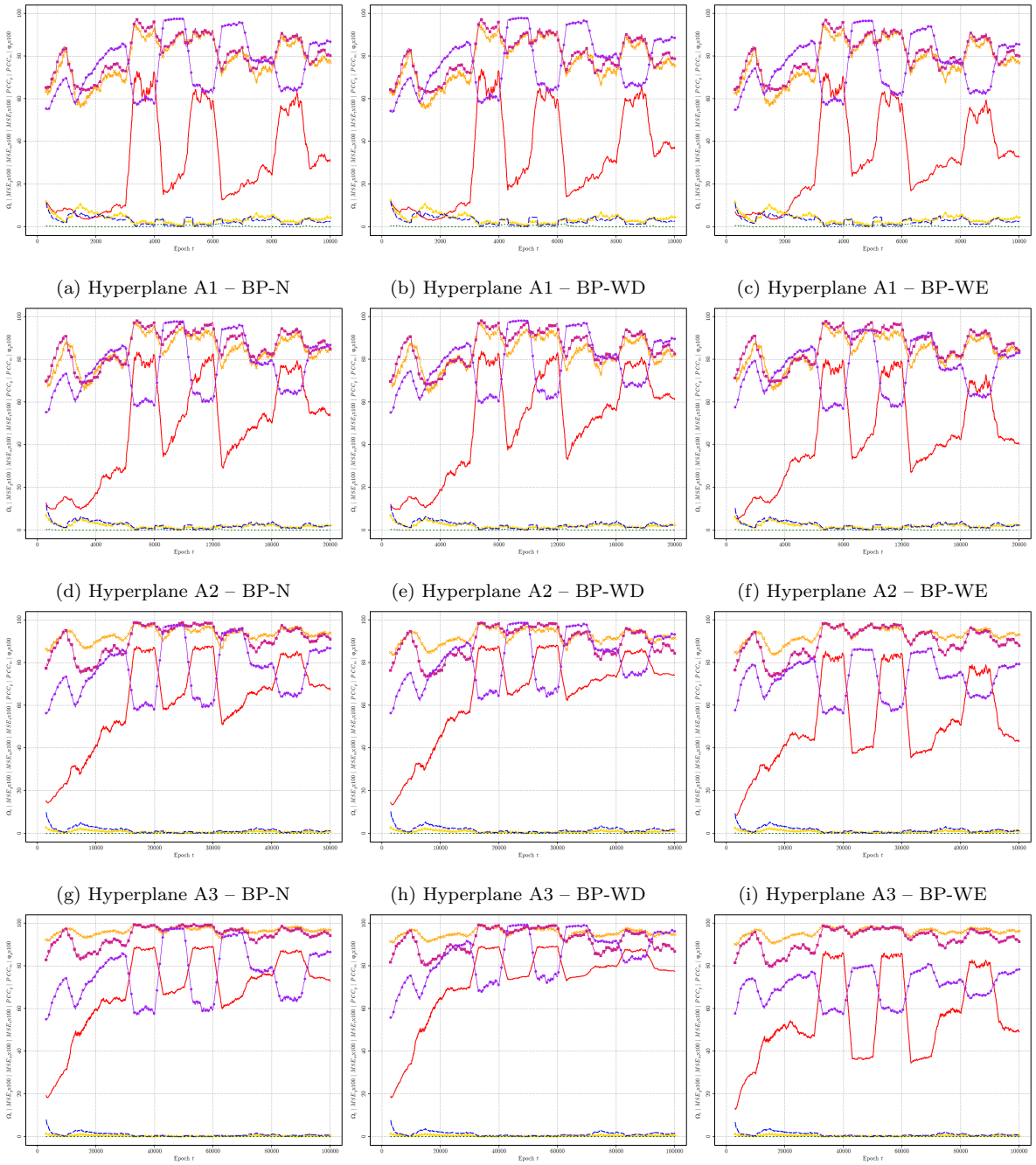


Figure A.1: Legend for the performance trend analysis graphs



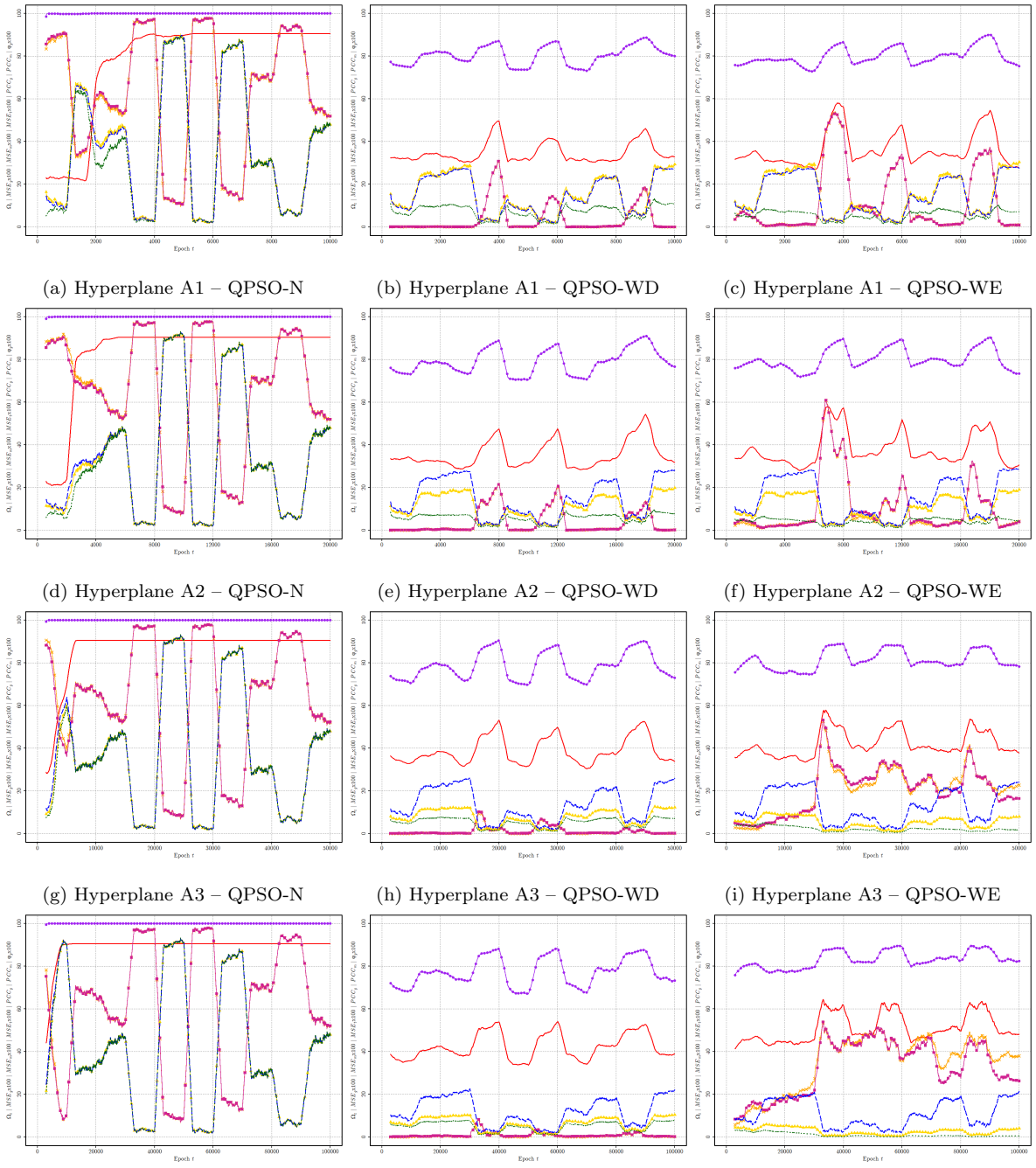
(a) Hyperplane A1 – BP-N (b) Hyperplane A1 – BP-WD (c) Hyperplane A1 – BP-WE

(d) Hyperplane A2 – BP-N (e) Hyperplane A2 – BP-WD (f) Hyperplane A2 – BP-WE

(g) Hyperplane A3 – BP-N (h) Hyperplane A3 – BP-WD (i) Hyperplane A3 – BP-WE

(j) Hyperplane A4 – BP-N (k) Hyperplane A4 – BP-WD (l) Hyperplane A4 – BP-WE

Figure A.2: Saturation, accuracy, and complexity performance trends of the BP classifiers for the A1 – A4 Hyperplane problems



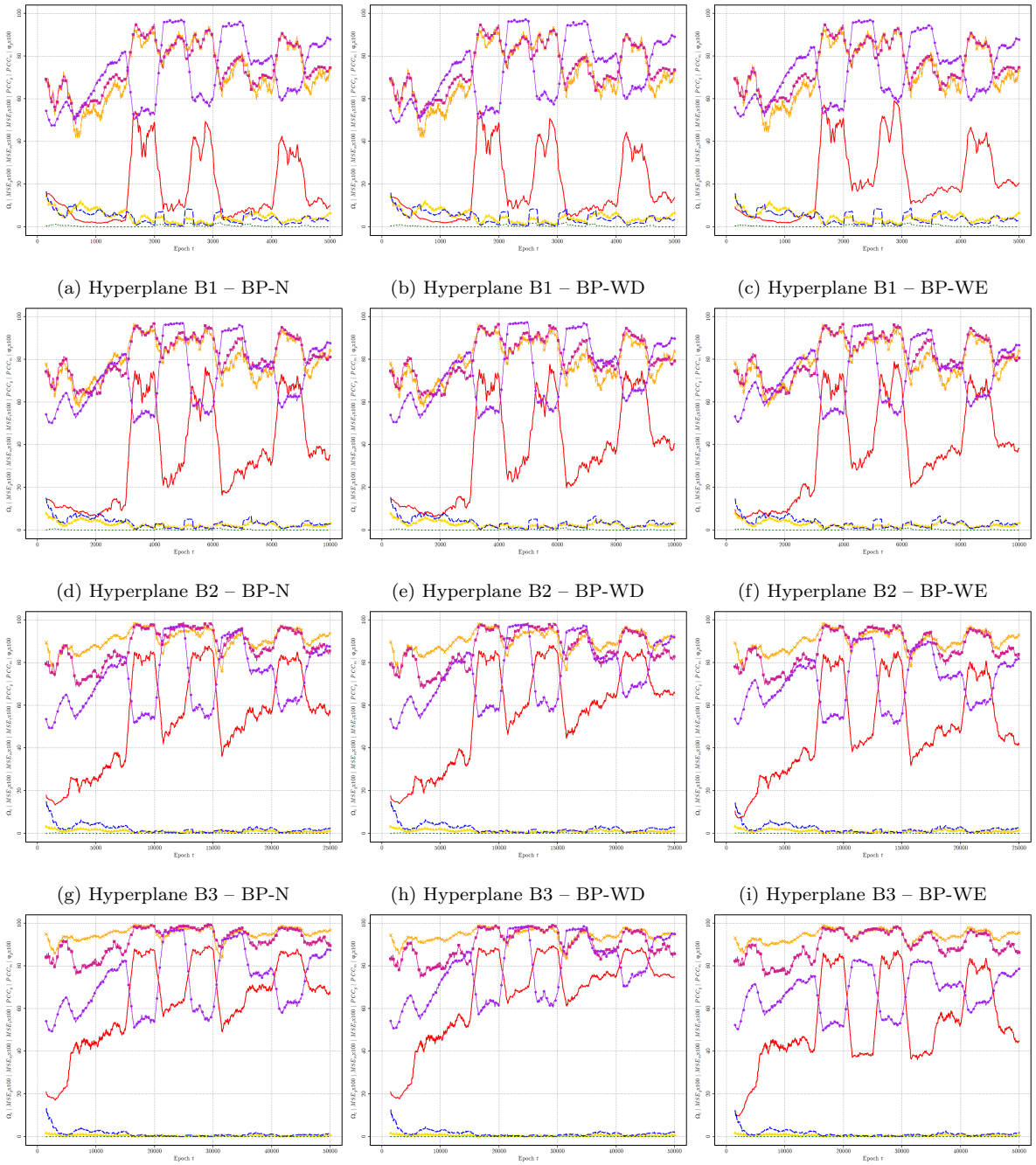
(a) Hyperplane A1 – QPSO-N (b) Hyperplane A1 – QPSO-WD (c) Hyperplane A1 – QPSO-WE

(d) Hyperplane A2 – QPSO-N (e) Hyperplane A2 – QPSO-WD (f) Hyperplane A2 – QPSO-WE

(g) Hyperplane A3 – QPSO-N (h) Hyperplane A3 – QPSO-WD (i) Hyperplane A3 – QPSO-WE

(j) Hyperplane A4 – QPSO-N (k) Hyperplane A4 – QPSO-WD (l) Hyperplane A4 – QPSO-WE

Figure A.3: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the A1 – A4 Hyperplane problems



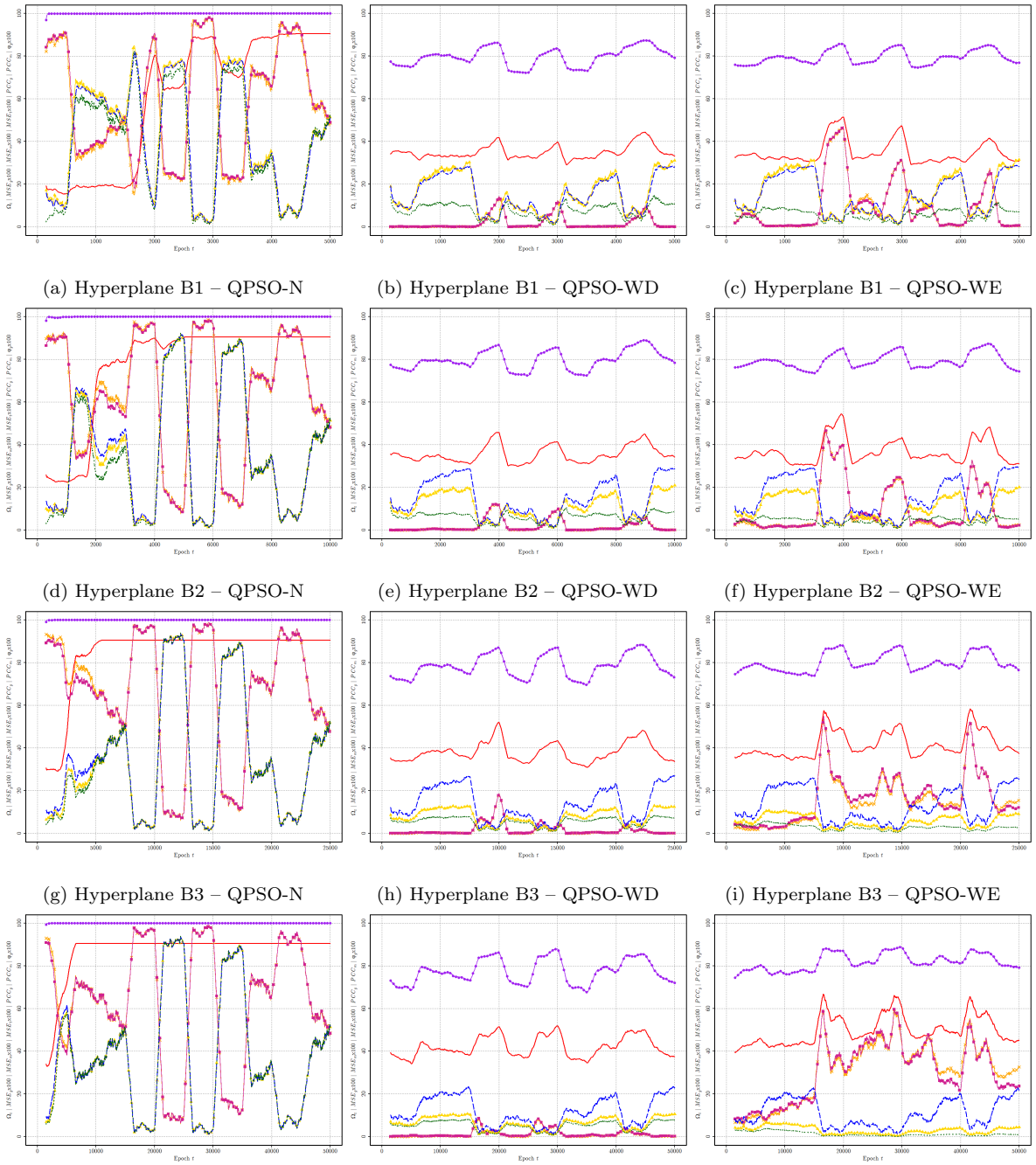
(a) Hyperplane B1 – BP-N (b) Hyperplane B1 – BP-WD (c) Hyperplane B1 – BP-WE

(d) Hyperplane B2 – BP-N (e) Hyperplane B2 – BP-WD (f) Hyperplane B2 – BP-WE

(g) Hyperplane B3 – BP-N (h) Hyperplane B3 – BP-WD (i) Hyperplane B3 – BP-WE

(j) Hyperplane B4 – BP-N (k) Hyperplane B4 – BP-WD (l) Hyperplane B4 – BP-WE

Figure A.4: Saturation, accuracy, and complexity performance trends of the BP classifiers for the B1 – B4 Hyperplane problems



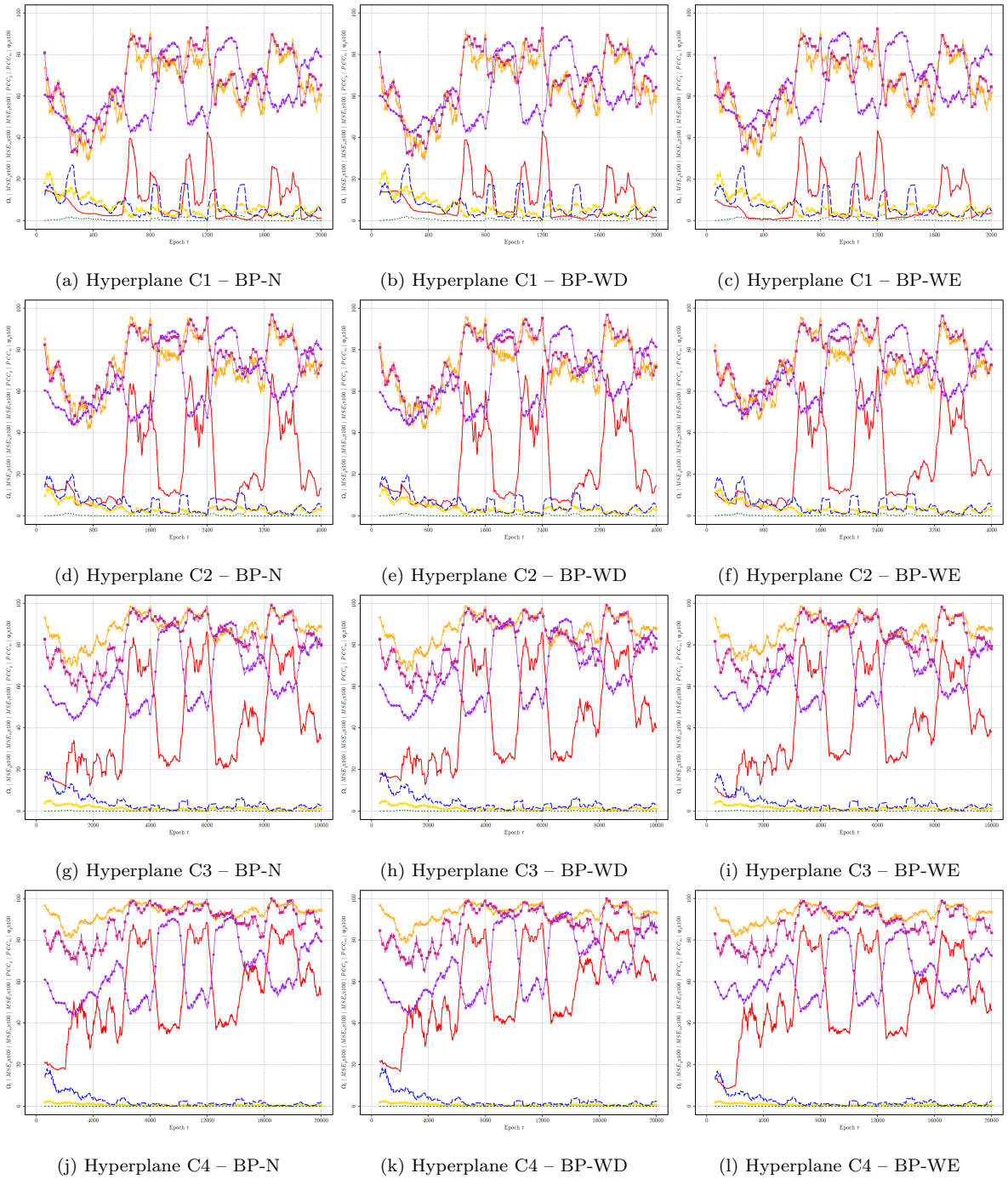
(a) Hyperplane B1 – QPSO-N (b) Hyperplane B1 – QPSO-WD (c) Hyperplane B1 – QPSO-WE

(d) Hyperplane B2 – QPSO-N (e) Hyperplane B2 – QPSO-WD (f) Hyperplane B2 – QPSO-WE

(g) Hyperplane B3 – QPSO-N (h) Hyperplane B3 – QPSO-WD (i) Hyperplane B3 – QPSO-WE

(j) Hyperplane B4 – QPSO-N (k) Hyperplane B4 – QPSO-WD (l) Hyperplane B4 – QPSO-WE

Figure A.5: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the B1 – B4 Hyperplane problems



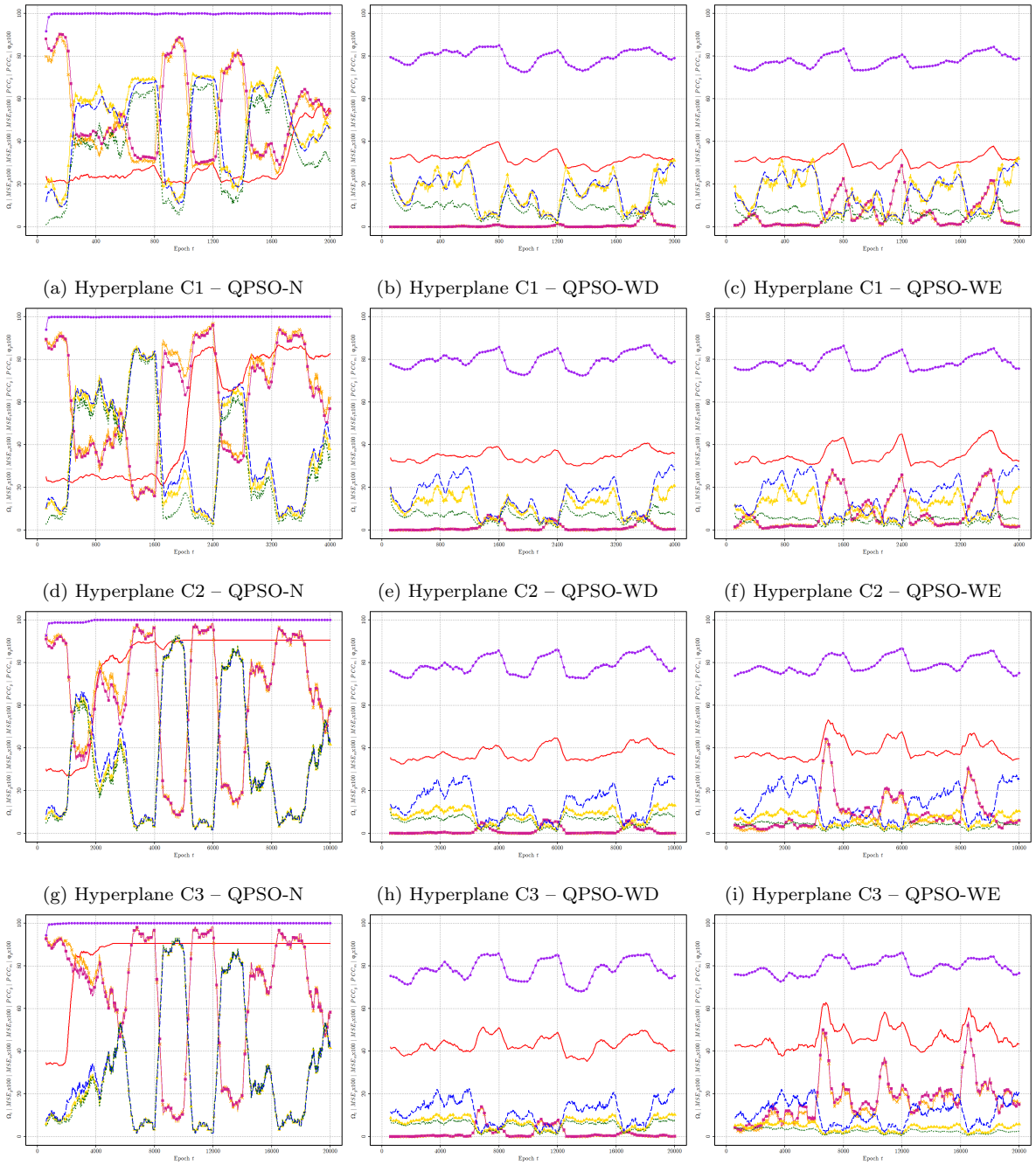
(a) Hyperplane C1 – BP-N (b) Hyperplane C1 – BP-WD (c) Hyperplane C1 – BP-WE

(d) Hyperplane C2 – BP-N (e) Hyperplane C2 – BP-WD (f) Hyperplane C2 – BP-WE

(g) Hyperplane C3 – BP-N (h) Hyperplane C3 – BP-WD (i) Hyperplane C3 – BP-WE

(j) Hyperplane C4 – BP-N (k) Hyperplane C4 – BP-WD (l) Hyperplane C4 – BP-WE

Figure A.6: Saturation, accuracy, and complexity performance trends of the BP classifiers for the C1 – C4 Hyperplane problems



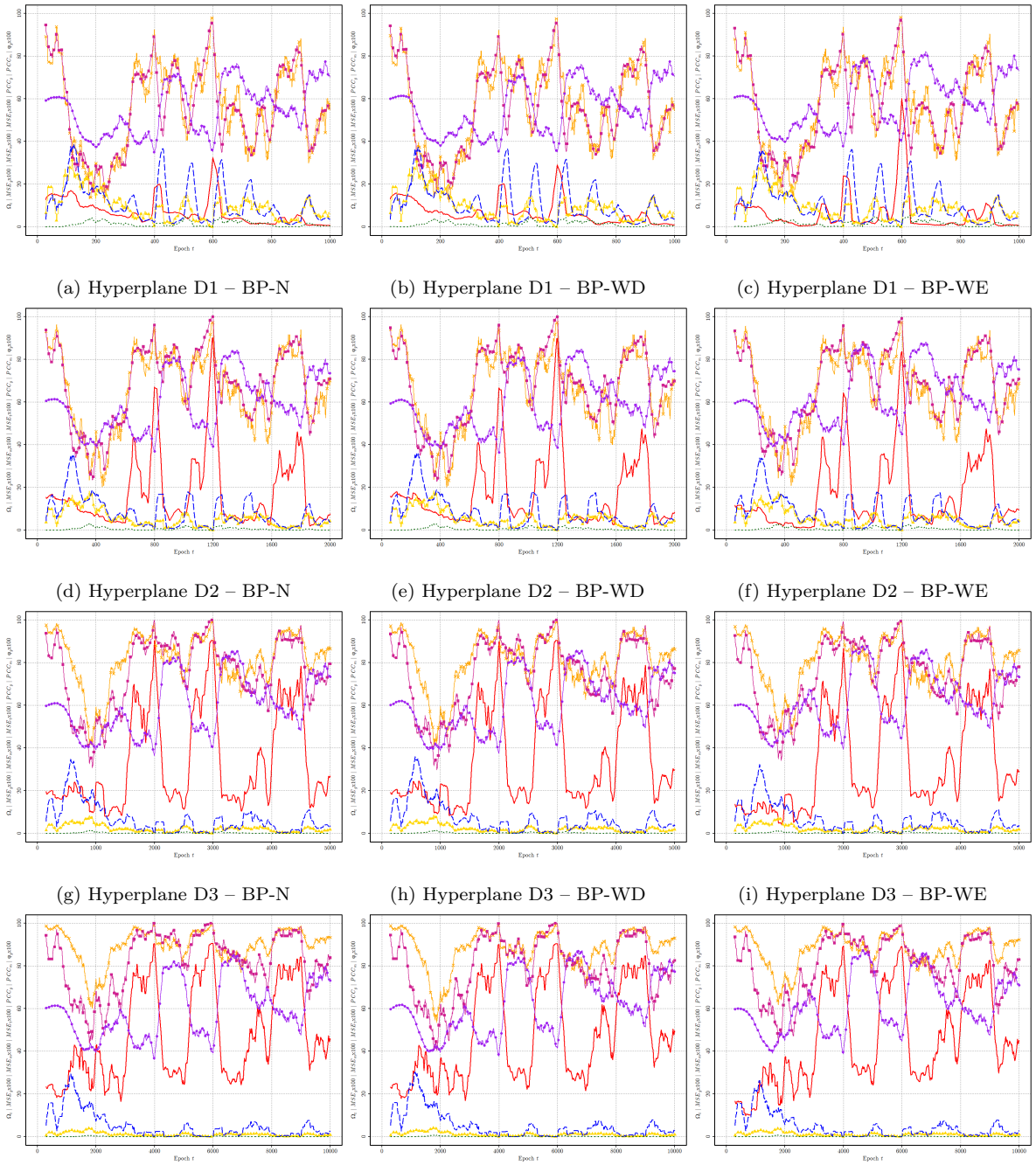
(a) Hyperplane C1 – QPSO-N (b) Hyperplane C1 – QPSO-WD (c) Hyperplane C1 – QPSO-WE

(d) Hyperplane C2 – QPSO-N (e) Hyperplane C2 – QPSO-WD (f) Hyperplane C2 – QPSO-WE

(g) Hyperplane C3 – QPSO-N (h) Hyperplane C3 – QPSO-WD (i) Hyperplane C3 – QPSO-WE

(j) Hyperplane C4 – QPSO-N (k) Hyperplane C4 – QPSO-WD (l) Hyperplane C4 – QPSO-WE

Figure A.7: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the C1 – C4 Hyperplane problems



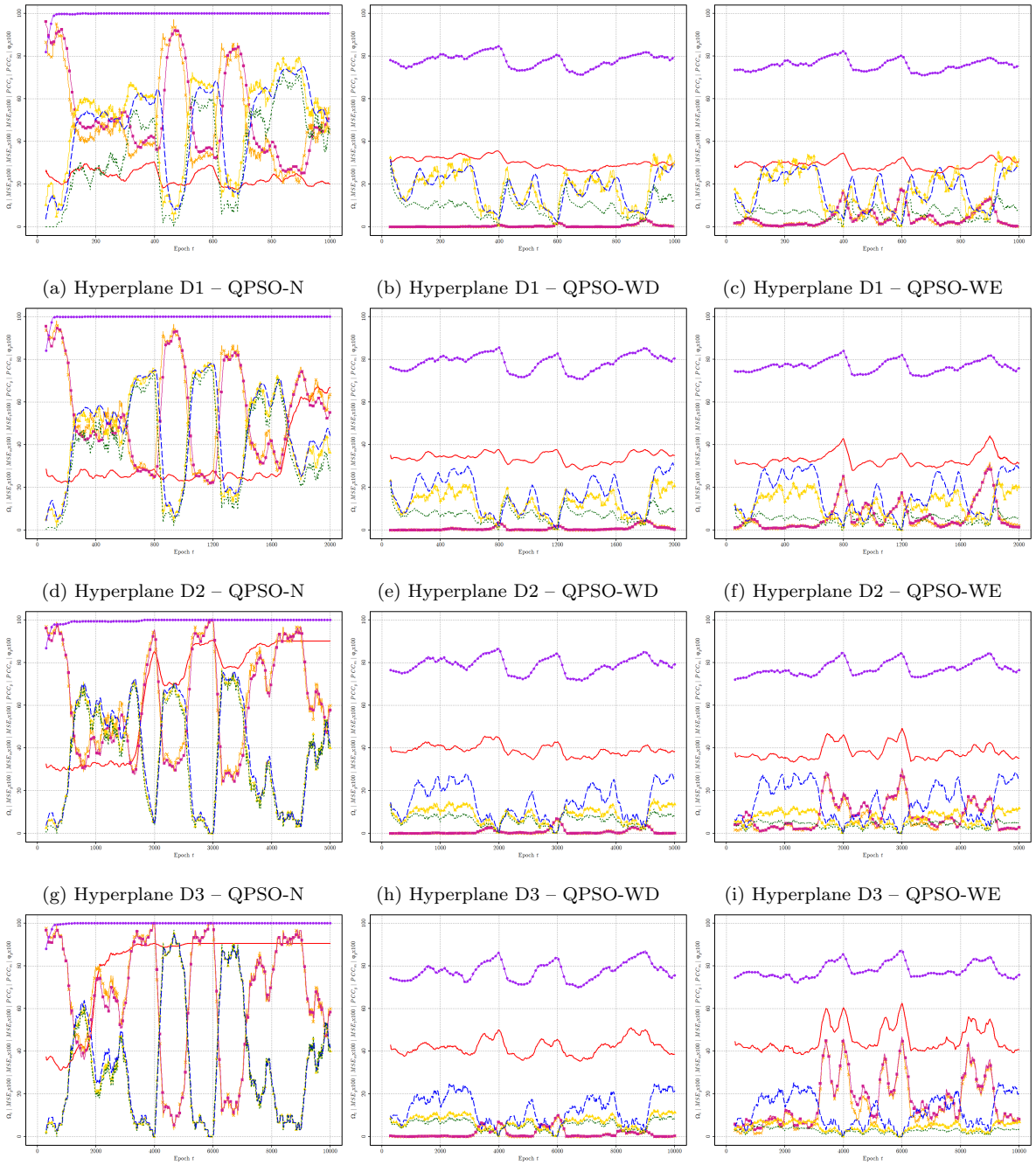
(a) Hyperplane D1 – BP-N (b) Hyperplane D1 – BP-WD (c) Hyperplane D1 – BP-WE

(d) Hyperplane D2 – BP-N (e) Hyperplane D2 – BP-WD (f) Hyperplane D2 – BP-WE

(g) Hyperplane D3 – BP-N (h) Hyperplane D3 – BP-WD (i) Hyperplane D3 – BP-WE

(j) Hyperplane D4 – BP-N (k) Hyperplane D4 – BP-WD (l) Hyperplane D4 – BP-WE

Figure A.8: Saturation, accuracy, and complexity performance trends of the BP classifiers for the D1 – D4 Hyperplane problems



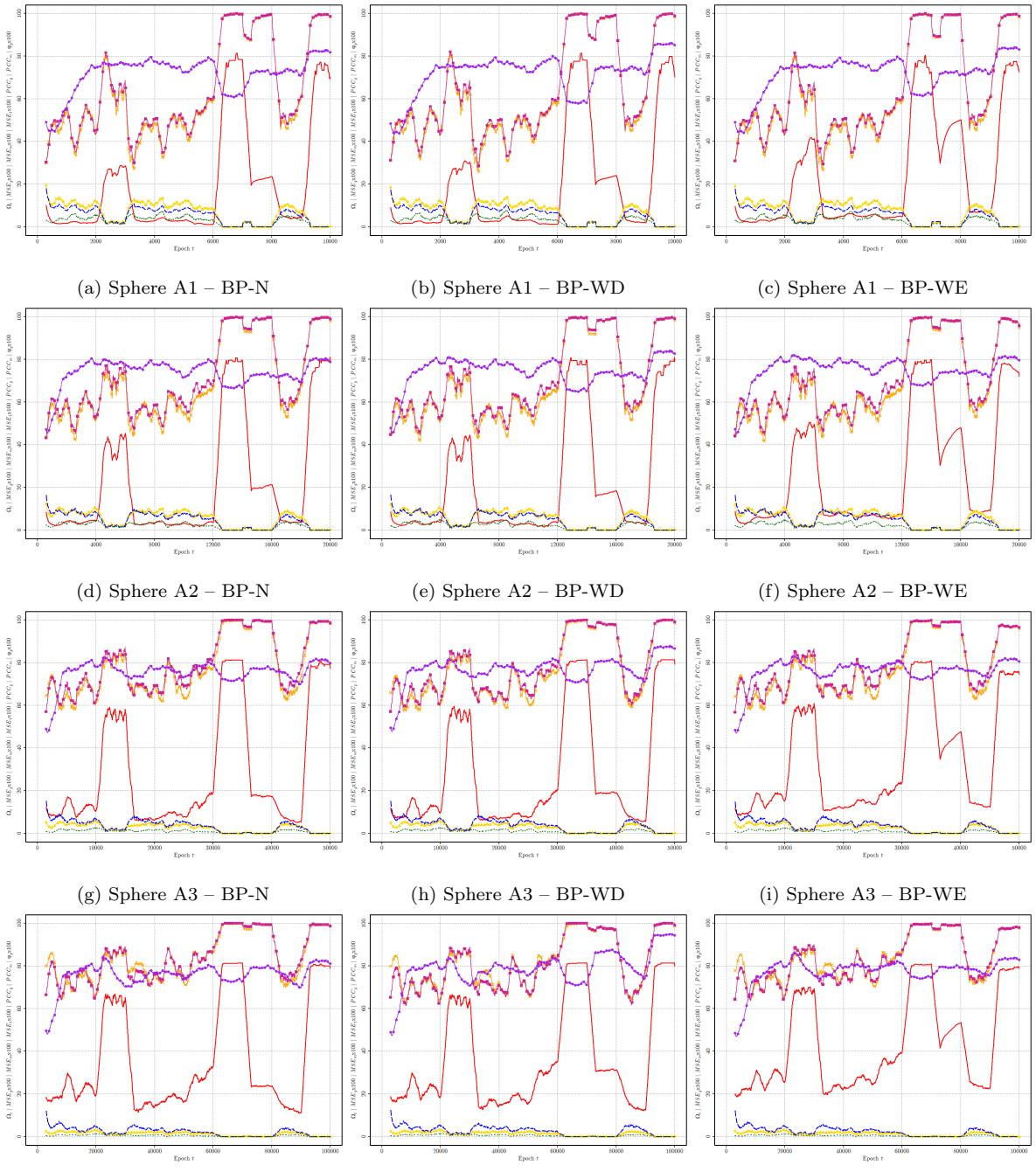
(a) Hyperplane D1 – QPSO-N (b) Hyperplane D1 – QPSO-WD (c) Hyperplane D1 – QPSO-WE

(d) Hyperplane D2 – QPSO-N (e) Hyperplane D2 – QPSO-WD (f) Hyperplane D2 – QPSO-WE

(g) Hyperplane D3 – QPSO-N (h) Hyperplane D3 – QPSO-WD (i) Hyperplane D3 – QPSO-WE

(j) Hyperplane D4 – QPSO-N (k) Hyperplane D4 – QPSO-WD (l) Hyperplane D4 – QPSO-WE

Figure A.9: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the D1 – D4 Hyperplane problems



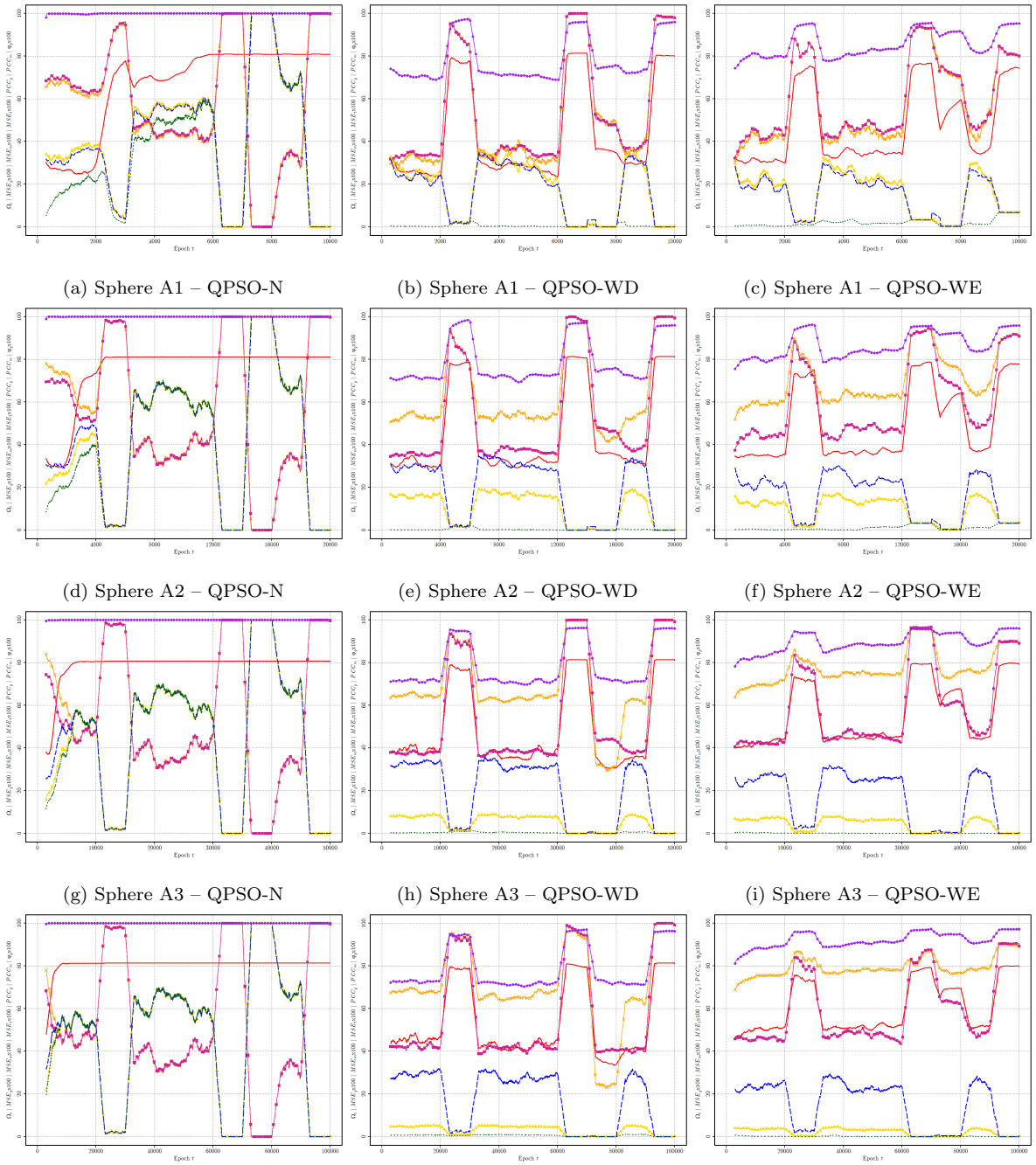
(a) Sphere A1 – BP-N (b) Sphere A1 – BP-WD (c) Sphere A1 – BP-WE

(d) Sphere A2 – BP-N (e) Sphere A2 – BP-WD (f) Sphere A2 – BP-WE

(g) Sphere A3 – BP-N (h) Sphere A3 – BP-WD (i) Sphere A3 – BP-WE

(j) Sphere A4 – BP-N (k) Sphere A4 – BP-WD (l) Sphere A4 – BP-WE

Figure A.10: Saturation, accuracy, and complexity performance trends of the BP classifiers for the A1 – A4 Sphere problems



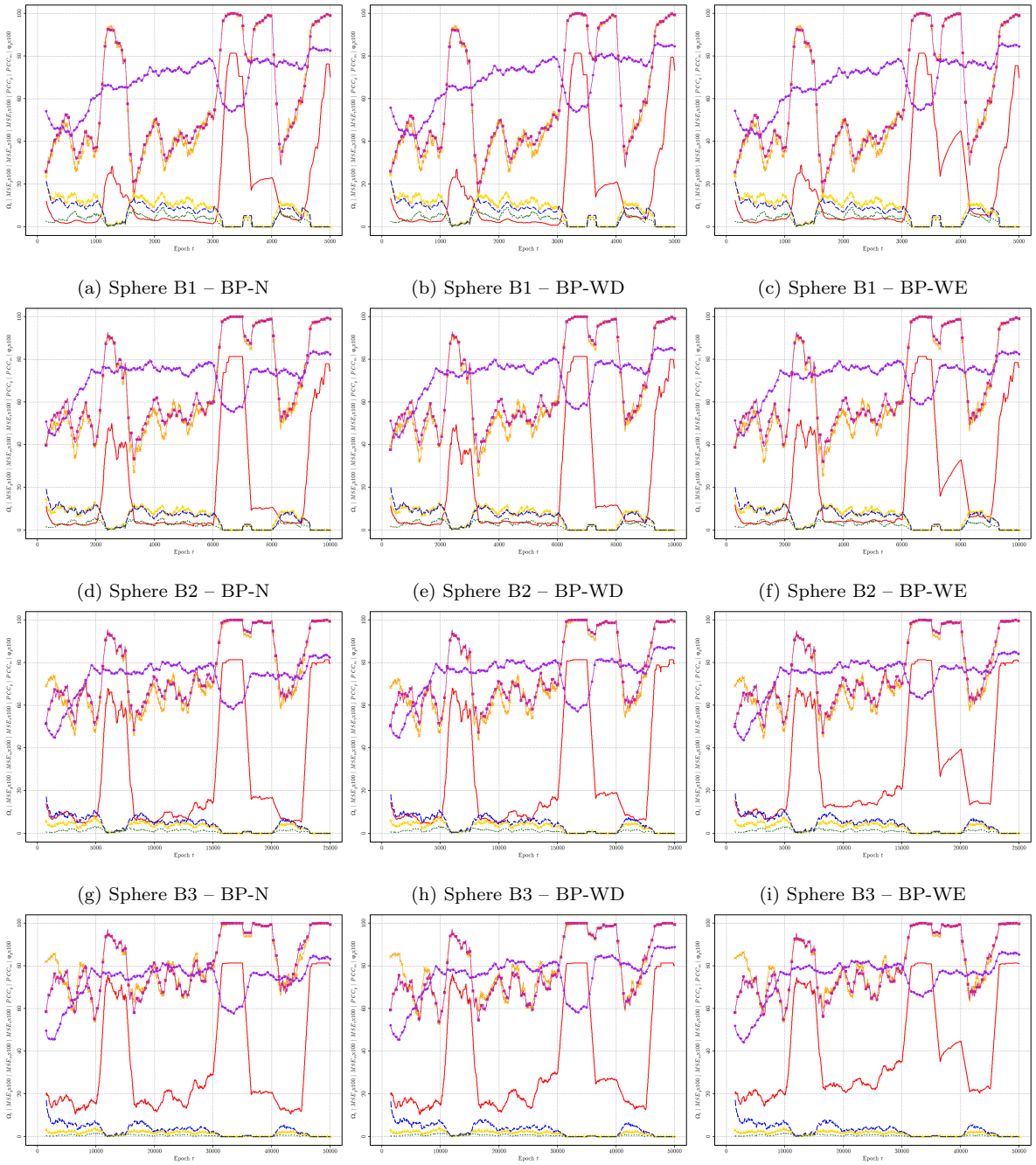
(a) Sphere A1 – QPSO-N (b) Sphere A1 – QPSO-WD (c) Sphere A1 – QPSO-WE

(d) Sphere A2 – QPSO-N (e) Sphere A2 – QPSO-WD (f) Sphere A2 – QPSO-WE

(g) Sphere A3 – QPSO-N (h) Sphere A3 – QPSO-WD (i) Sphere A3 – QPSO-WE

(j) Sphere A4 – QPSO-N (k) Sphere A4 – QPSO-WD (l) Sphere A4 – QPSO-WE

Figure A.11: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the A1 – A4 Sphere problems



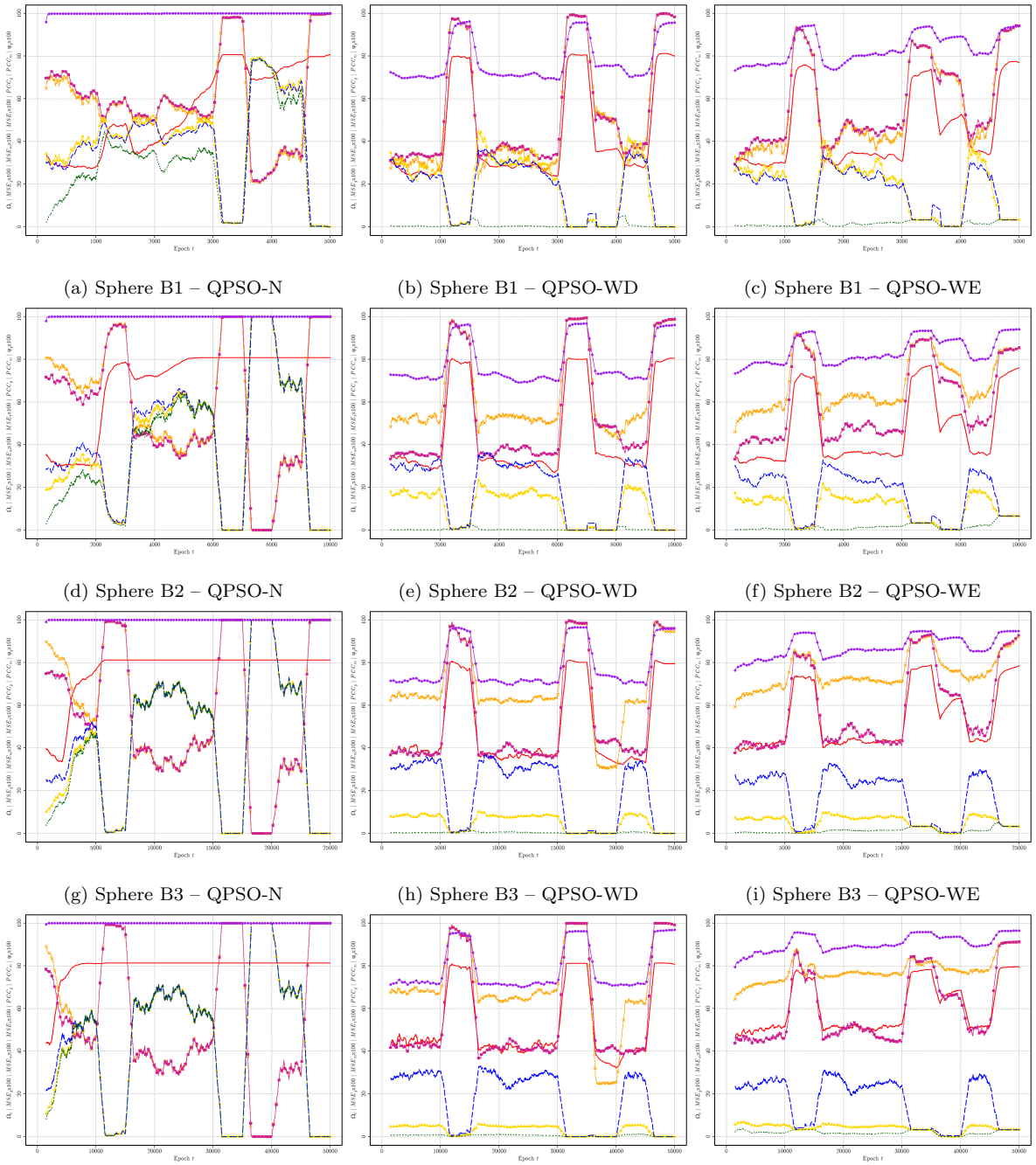
(a) Sphere B1 – BP-N (b) Sphere B1 – BP-WD (c) Sphere B1 – BP-WE

(d) Sphere B2 – BP-N (e) Sphere B2 – BP-WD (f) Sphere B2 – BP-WE

(g) Sphere B3 – BP-N (h) Sphere B3 – BP-WD (i) Sphere B3 – BP-WE

(j) Sphere B4 – BP-N (k) Sphere B4 – BP-WD (l) Sphere B4 – BP-WE

Figure A.12: Saturation, accuracy, and complexity performance trends of the BP classifiers for the B1 – B4 Sphere problems



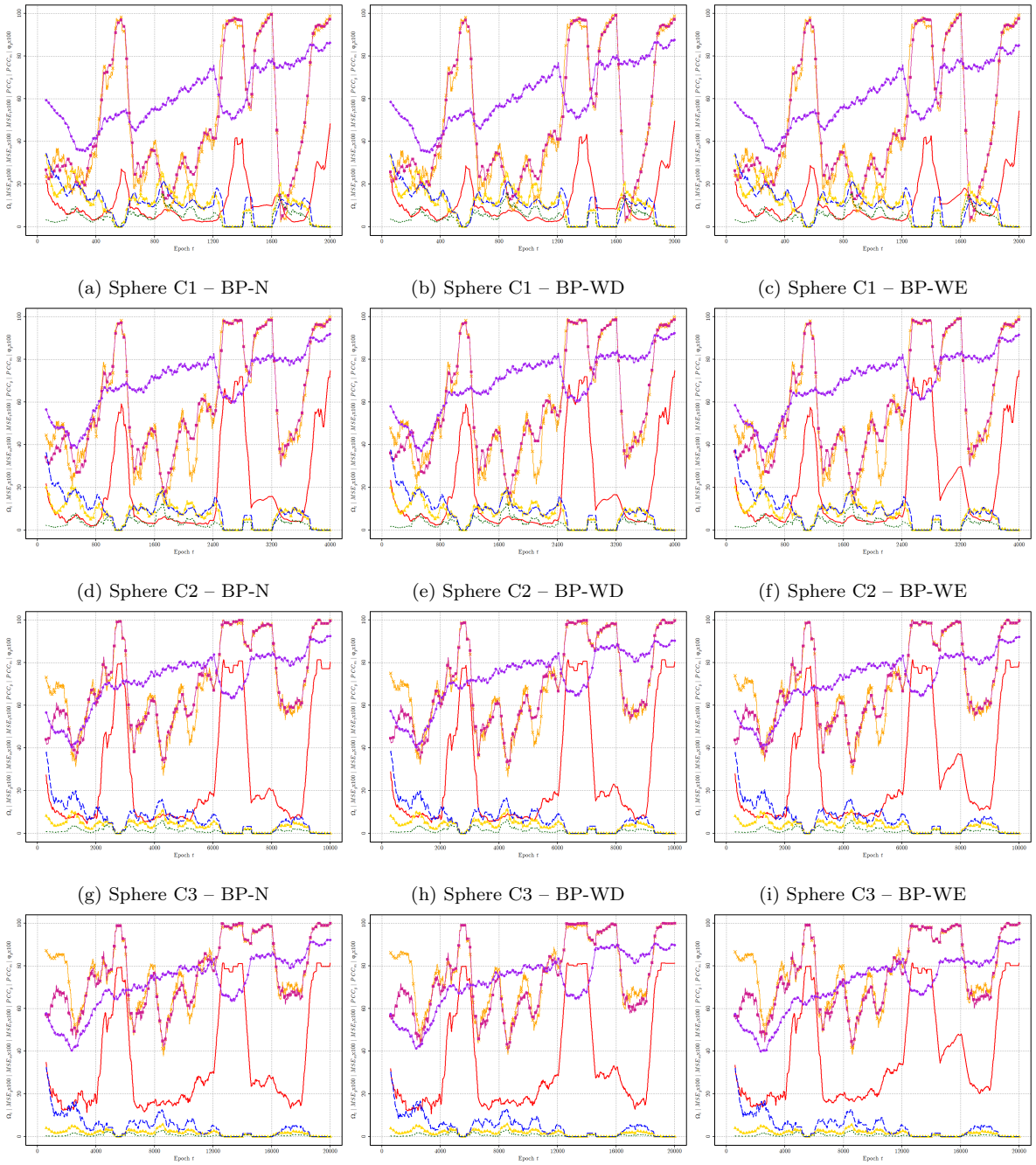
(a) Sphere B1 – QPSO-N (b) Sphere B1 – QPSO-WD (c) Sphere B1 – QPSO-WE

(d) Sphere B2 – QPSO-N (e) Sphere B2 – QPSO-WD (f) Sphere B2 – QPSO-WE

(g) Sphere B3 – QPSO-N (h) Sphere B3 – QPSO-WD (i) Sphere B3 – QPSO-WE

(j) Sphere B4 – QPSO-N (k) Sphere B4 – QPSO-WD (l) Sphere B4 – QPSO-WE

Figure A.13: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the B1 – B4 Sphere problems



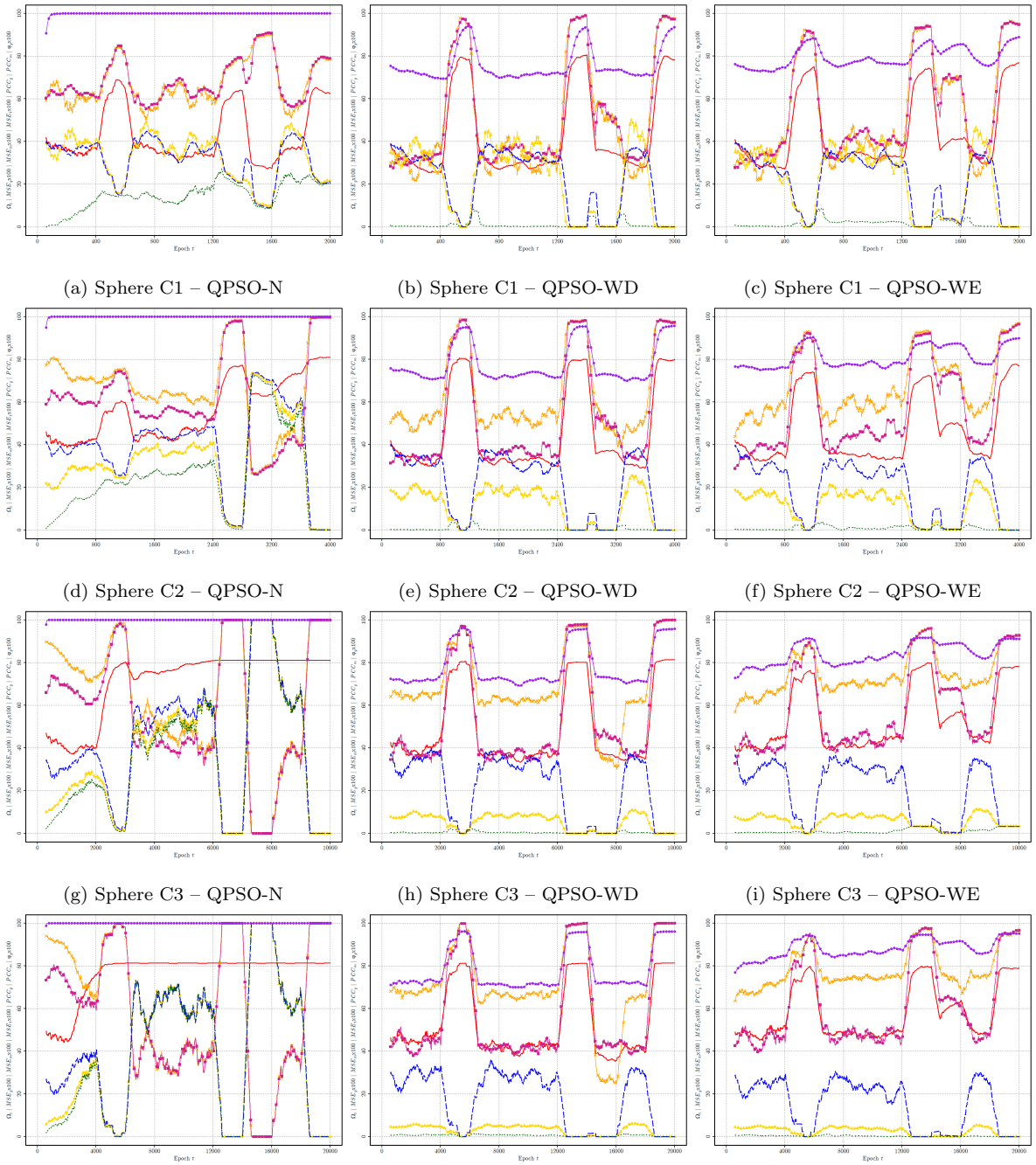
(a) Sphere C1 – BP-N (b) Sphere C1 – BP-WD (c) Sphere C1 – BP-WE

(d) Sphere C2 – BP-N (e) Sphere C2 – BP-WD (f) Sphere C2 – BP-WE

(g) Sphere C3 – BP-N (h) Sphere C3 – BP-WD (i) Sphere C3 – BP-WE

(j) Sphere C4 – BP-N (k) Sphere C4 – BP-WD (l) Sphere C4 – BP-WE

Figure A.14: Saturation, accuracy, and complexity performance trends of the BP classifiers for the C1 – C4 Sphere problems



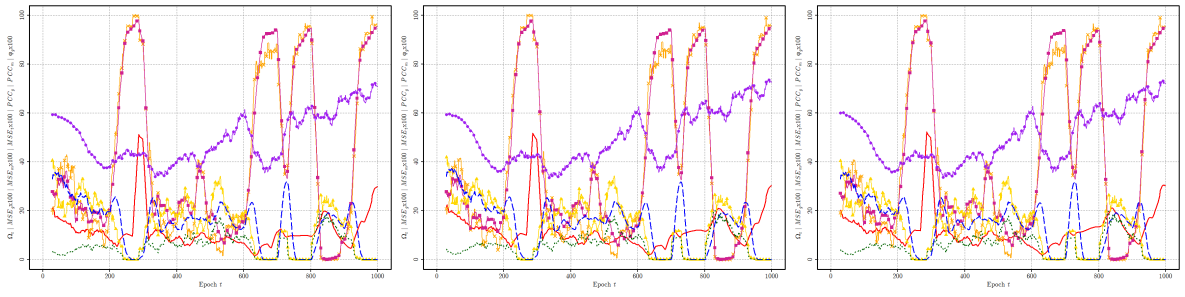
(a) Sphere C1 – QPSO-N (b) Sphere C1 – QPSO-WD (c) Sphere C1 – QPSO-WE

(d) Sphere C2 – QPSO-N (e) Sphere C2 – QPSO-WD (f) Sphere C2 – QPSO-WE

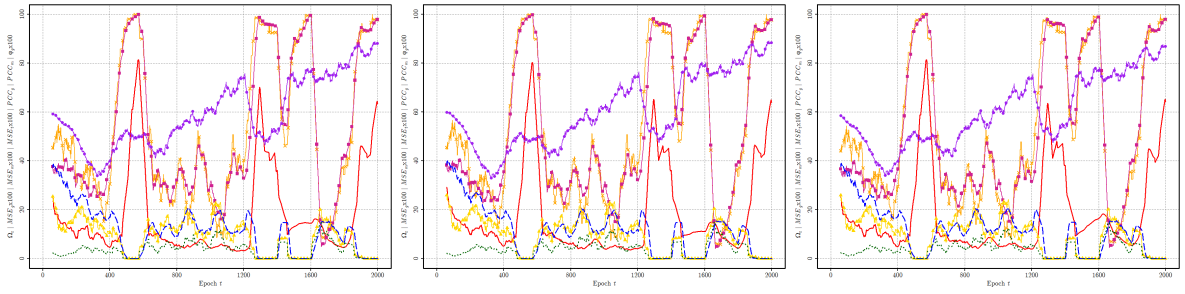
(g) Sphere C3 – QPSO-N (h) Sphere C3 – QPSO-WD (i) Sphere C3 – QPSO-WE

(j) Sphere C4 – QPSO-N (k) Sphere C4 – QPSO-WD (l) Sphere C4 – QPSO-WE

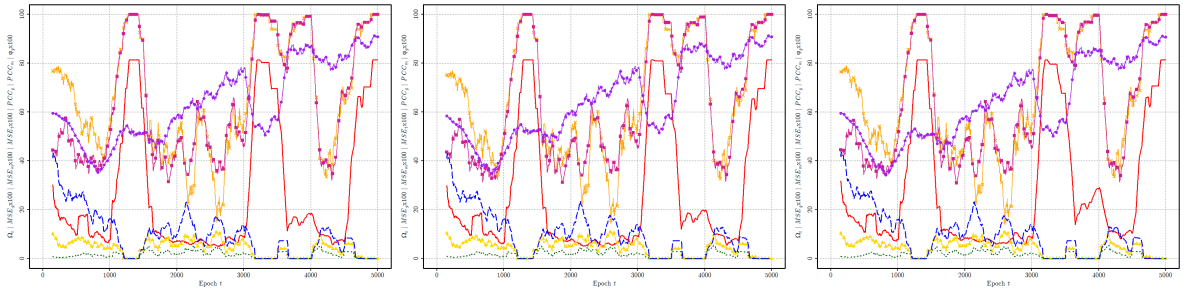
Figure A.15: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the C1 – C4 Sphere problems



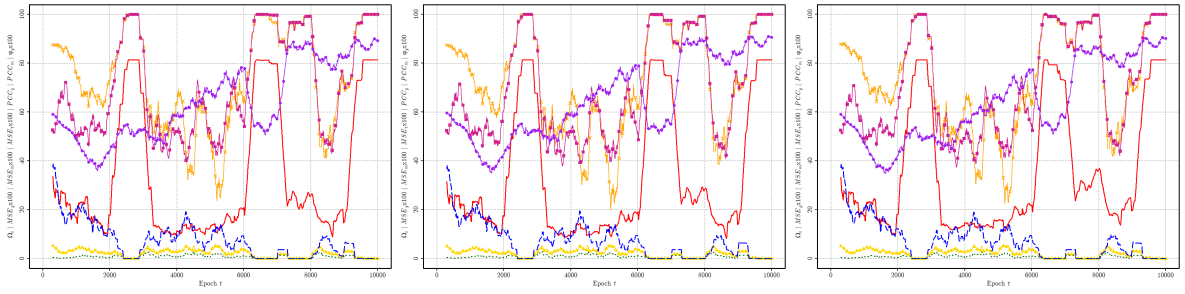
(a) Sphere D1 – BP-N (b) Sphere D1 – BP-WD (c) Sphere D1 – BP-WE



(d) Sphere D2 – BP-N (e) Sphere D2 – BP-WD (f) Sphere D2 – BP-WE

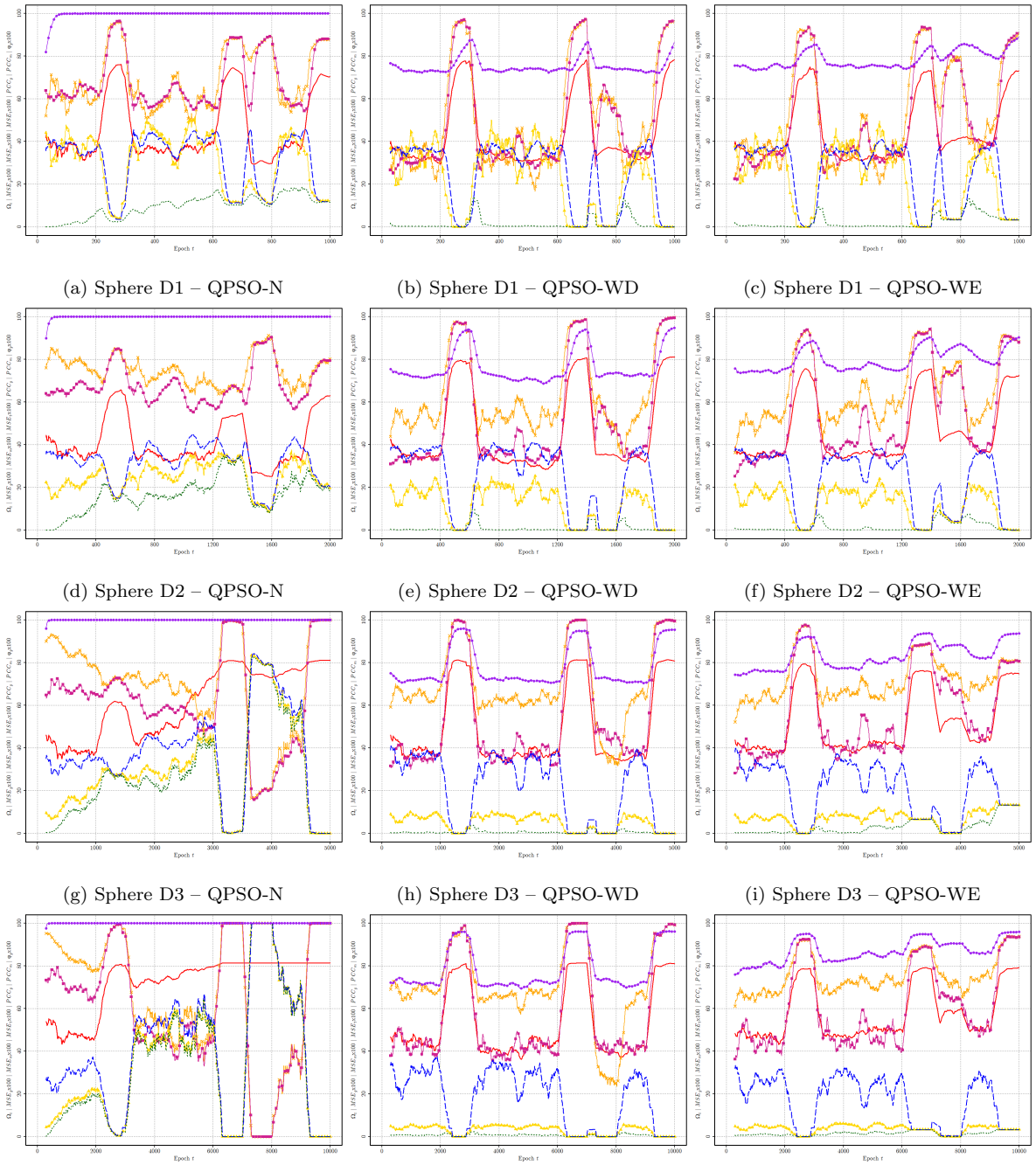


(g) Sphere D3 – BP-N (h) Sphere D3 – BP-WD (i) Sphere D3 – BP-WE



(j) Sphere D4 – BP-N (k) Sphere D4 – BP-WD (l) Sphere D4 – BP-WE

Figure A.16: Saturation, accuracy, and complexity performance trends of the BP classifiers for the D1 – D4 Sphere problems



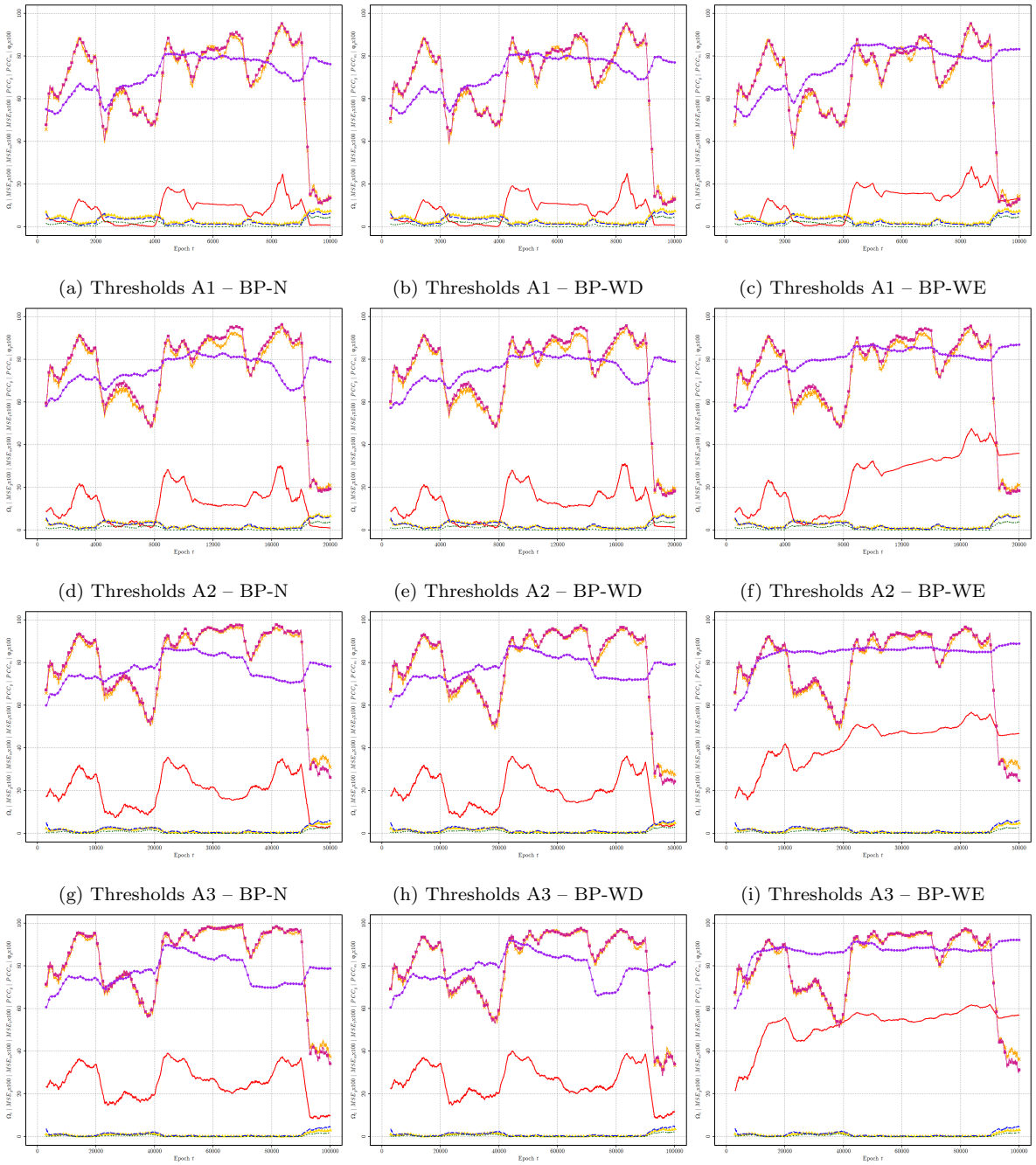
(a) Sphere D1 – QPSO-N (b) Sphere D1 – QPSO-WD (c) Sphere D1 – QPSO-WE

(d) Sphere D2 – QPSO-N (e) Sphere D2 – QPSO-WD (f) Sphere D2 – QPSO-WE

(g) Sphere D3 – QPSO-N (h) Sphere D3 – QPSO-WD (i) Sphere D3 – QPSO-WE

(j) Sphere D4 – QPSO-N (k) Sphere D4 – QPSO-WD (l) Sphere D4 – QPSO-WE

Figure A.17: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the D1 – D4 Sphere problems



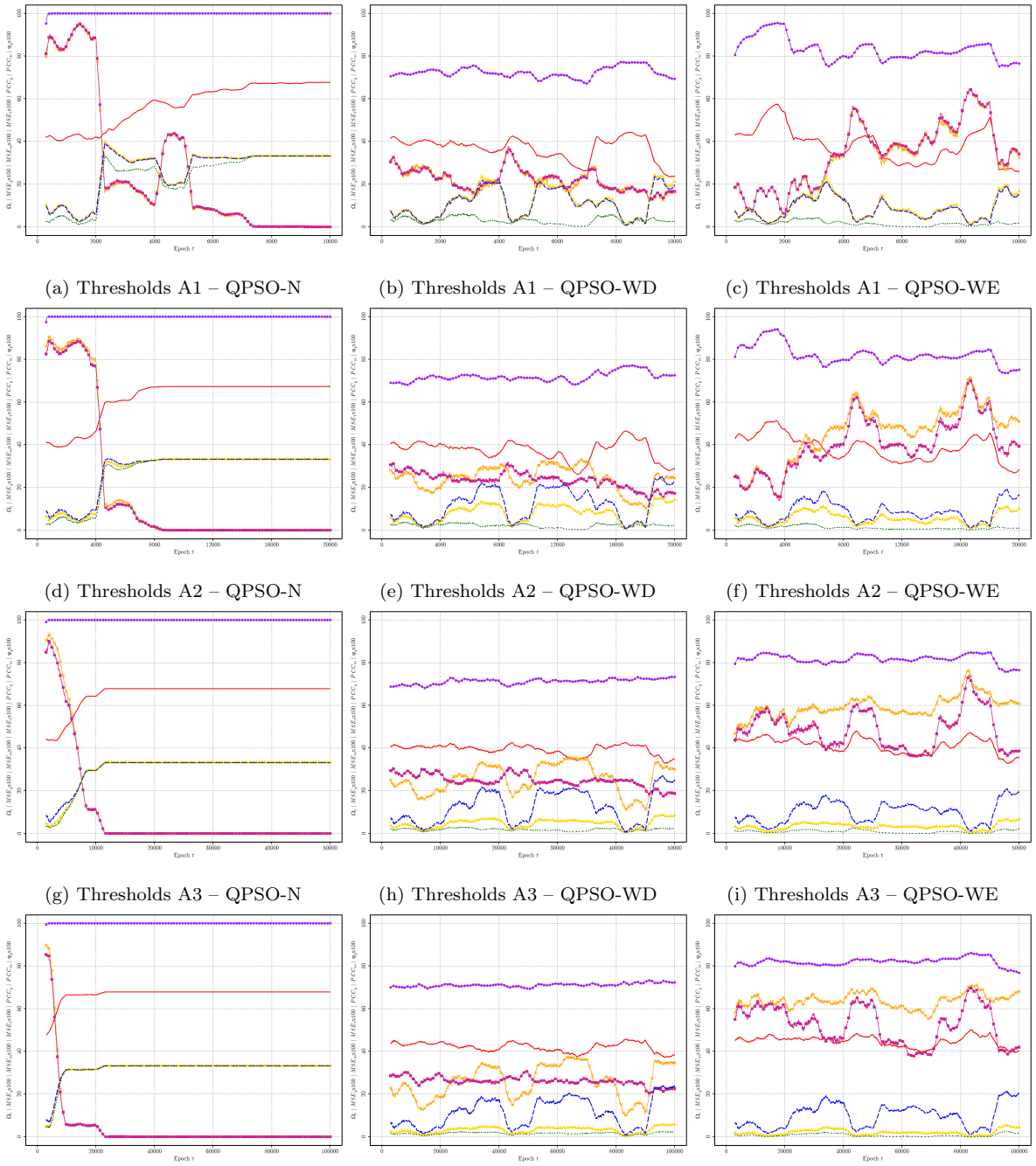
(a) Thresholds A1 – BP-N (b) Thresholds A1 – BP-WD (c) Thresholds A1 – BP-WE

(d) Thresholds A2 – BP-N (e) Thresholds A2 – BP-WD (f) Thresholds A2 – BP-WE

(g) Thresholds A3 – BP-N (h) Thresholds A3 – BP-WD (i) Thresholds A3 – BP-WE

(j) Thresholds A4 – BP-N (k) Thresholds A4 – BP-WD (l) Thresholds A4 – BP-WE

Figure A.18: Saturation, accuracy, and complexity performance trends of the BP classifiers for the A1 – A4 Thresholds problems



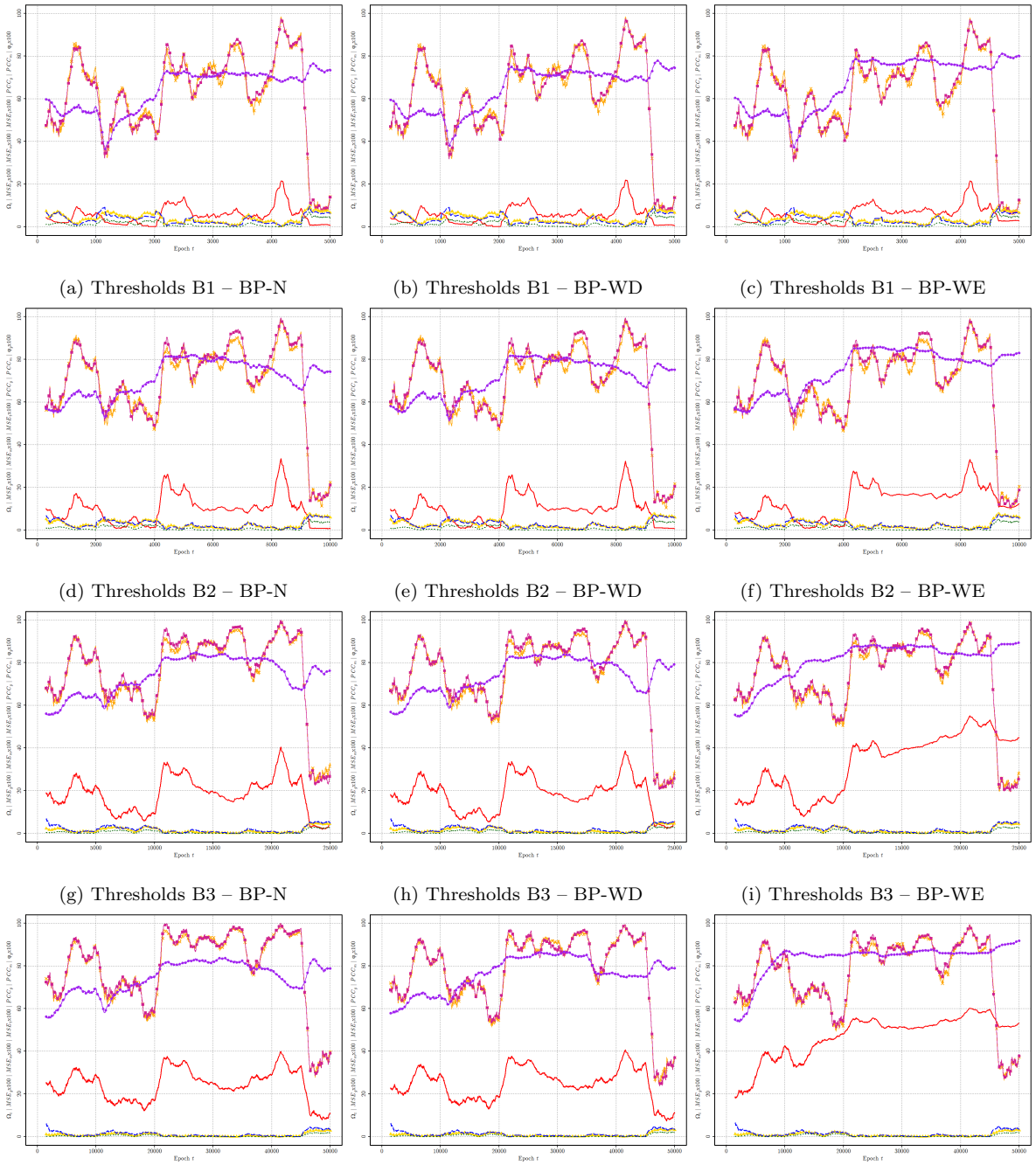
(a) Thresholds A1 – QPSO-N (b) Thresholds A1 – QPSO-WD (c) Thresholds A1 – QPSO-WE

(d) Thresholds A2 – QPSO-N (e) Thresholds A2 – QPSO-WD (f) Thresholds A2 – QPSO-WE

(g) Thresholds A3 – QPSO-N (h) Thresholds A3 – QPSO-WD (i) Thresholds A3 – QPSO-WE

(j) Thresholds A4 – QPSO-N (k) Thresholds A4 – QPSO-WD (l) Thresholds A4 – QPSO-WE

Figure A.19: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the A1 – A4 Thresholds problems



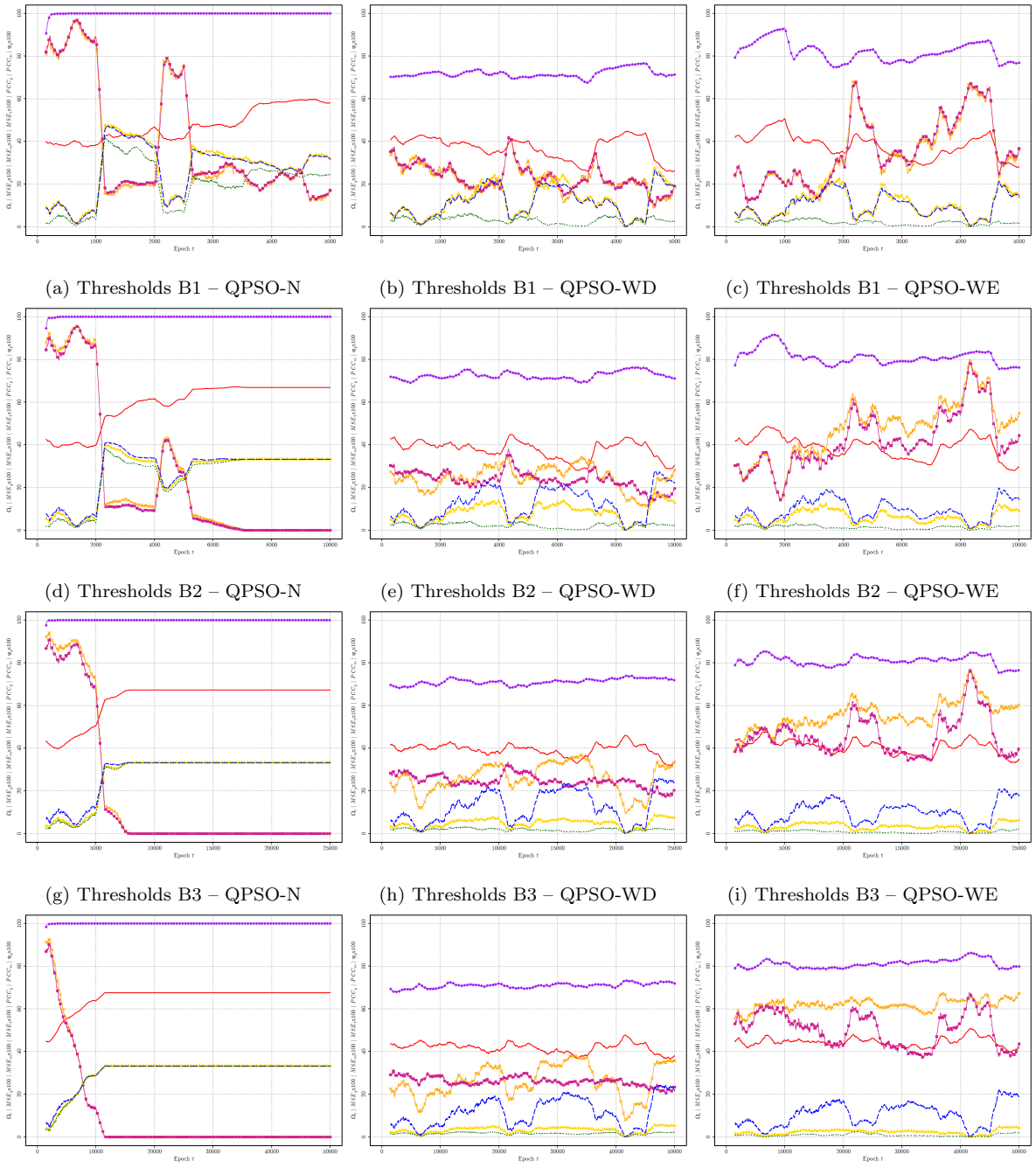
(a) Thresholds B1 – BP-N (b) Thresholds B1 – BP-WD (c) Thresholds B1 – BP-WE

(d) Thresholds B2 – BP-N (e) Thresholds B2 – BP-WD (f) Thresholds B2 – BP-WE

(g) Thresholds B3 – BP-N (h) Thresholds B3 – BP-WD (i) Thresholds B3 – BP-WE

(j) Thresholds B4 – BP-N (k) Thresholds B4 – BP-WD (l) Thresholds B4 – BP-WE

Figure A.20: Saturation, accuracy, and complexity performance trends of the BP classifiers for the B1 – B4 Thresholds problems



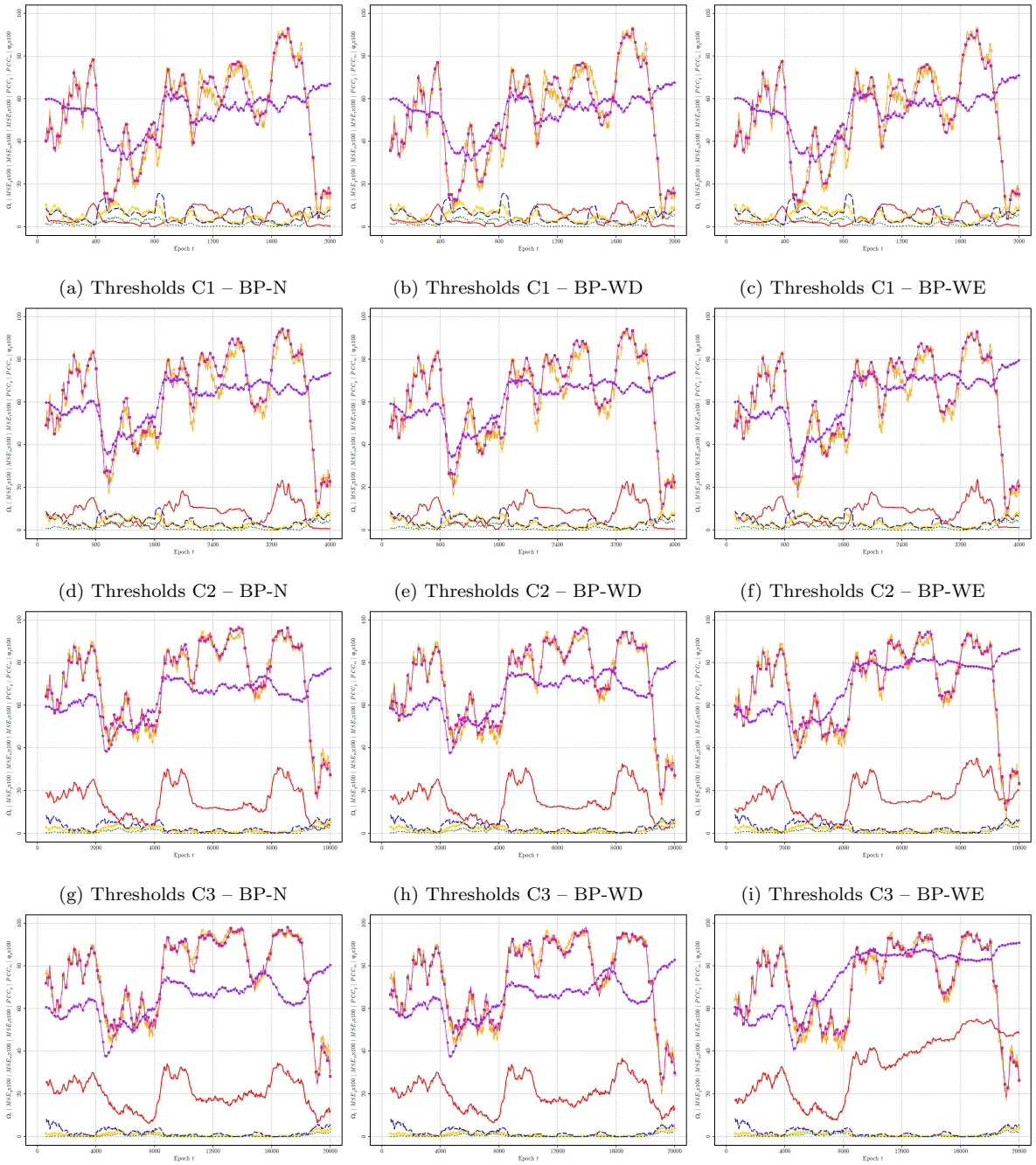
(a) Thresholds B1 – QPSO-N (b) Thresholds B1 – QPSO-WD (c) Thresholds B1 – QPSO-WE

(d) Thresholds B2 – QPSO-N (e) Thresholds B2 – QPSO-WD (f) Thresholds B2 – QPSO-WE

(g) Thresholds B3 – QPSO-N (h) Thresholds B3 – QPSO-WD (i) Thresholds B3 – QPSO-WE

(j) Thresholds B4 – QPSO-N (k) Thresholds B4 – QPSO-WD (l) Thresholds B4 – QPSO-WE

Figure A.21: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the B1 – B4 Thresholds problems



(a) Thresholds C1 – BP-N (b) Thresholds C1 – BP-WD (c) Thresholds C1 – BP-WE

(d) Thresholds C2 – BP-N (e) Thresholds C2 – BP-WD (f) Thresholds C2 – BP-WE

(g) Thresholds C3 – BP-N (h) Thresholds C3 – BP-WD (i) Thresholds C3 – BP-WE

(j) Thresholds C4 – BP-N (k) Thresholds C4 – BP-WD (l) Thresholds C4 – BP-WE

Figure A.22: Saturation, accuracy, and complexity performance trends of the BP classifiers for the C1 – C4 Thresholds problems

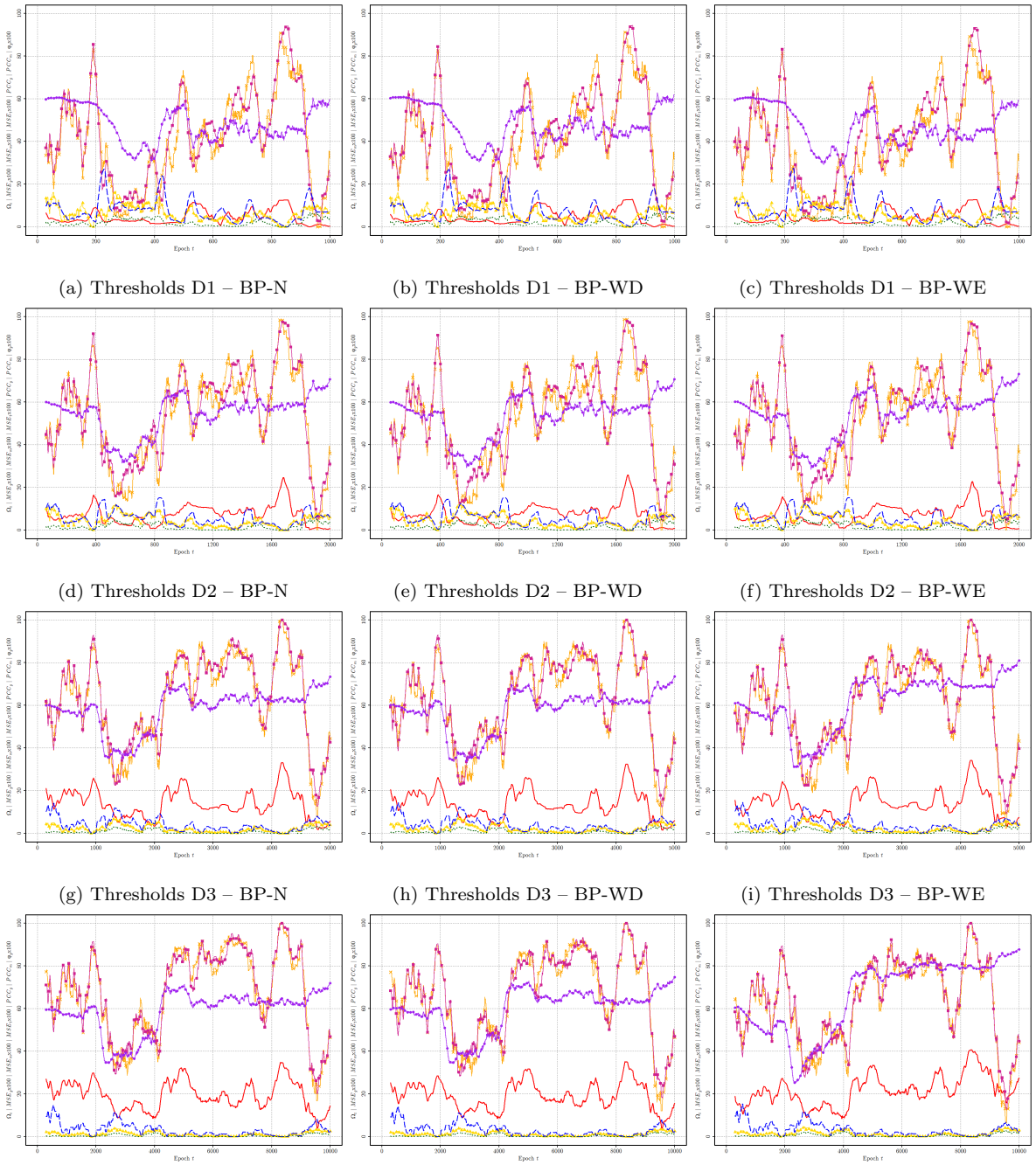
(a) Thresholds C1 – QPSO-N (b) Thresholds C1 – QPSO-WD (c) Thresholds C1 – QPSO-WE

(d) Thresholds C2 – QPSO-N (e) Thresholds C2 – QPSO-WD (f) Thresholds C2 – QPSO-WE

(g) Thresholds C3 – QPSO-N (h) Thresholds C3 – QPSO-WD (i) Thresholds C3 – QPSO-WE

(j) Thresholds C4 – QPSO-N (k) Thresholds C4 – QPSO-WD (l) Thresholds C4 – QPSO-WE

Figure A.23: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the C1 – C4 Thresholds problems



(a) Thresholds D1 – BP-N (b) Thresholds D1 – BP-WD (c) Thresholds D1 – BP-WE

(d) Thresholds D2 – BP-N (e) Thresholds D2 – BP-WD (f) Thresholds D2 – BP-WE

(g) Thresholds D3 – BP-N (h) Thresholds D3 – BP-WD (i) Thresholds D3 – BP-WE

(j) Thresholds D4 – BP-N (k) Thresholds D4 – BP-WD (l) Thresholds D4 – BP-WE

Figure A.24: Saturation, accuracy, and complexity performance trends of the BP classifiers for the D1 – D4 Thresholds problems

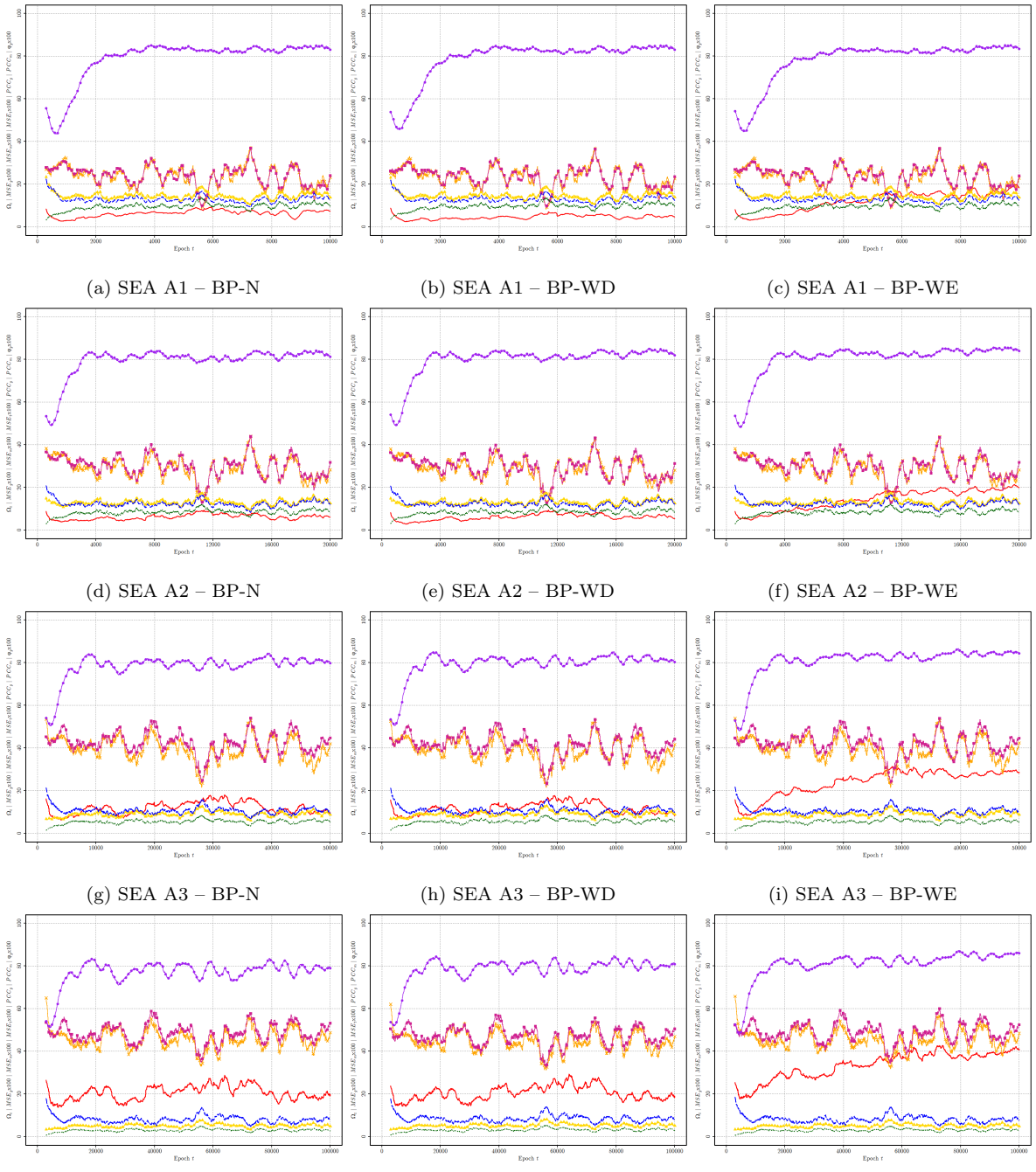
(a) Thresholds D1 – QPSO-N (b) Thresholds D1 – QPSO-WD (c) Thresholds D1 – QPSO-WE

(d) Thresholds D2 – QPSO-N (e) Thresholds D2 – QPSO-WD (f) Thresholds D2 – QPSO-WE

(g) Thresholds D3 – QPSO-N (h) Thresholds D3 – QPSO-WD (i) Thresholds D3 – QPSO-WE

(j) Thresholds D4 – QPSO-N (k) Thresholds D4 – QPSO-WD (l) Thresholds D4 – QPSO-WE

Figure A.25: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the D1 – D4 Thresholds problems



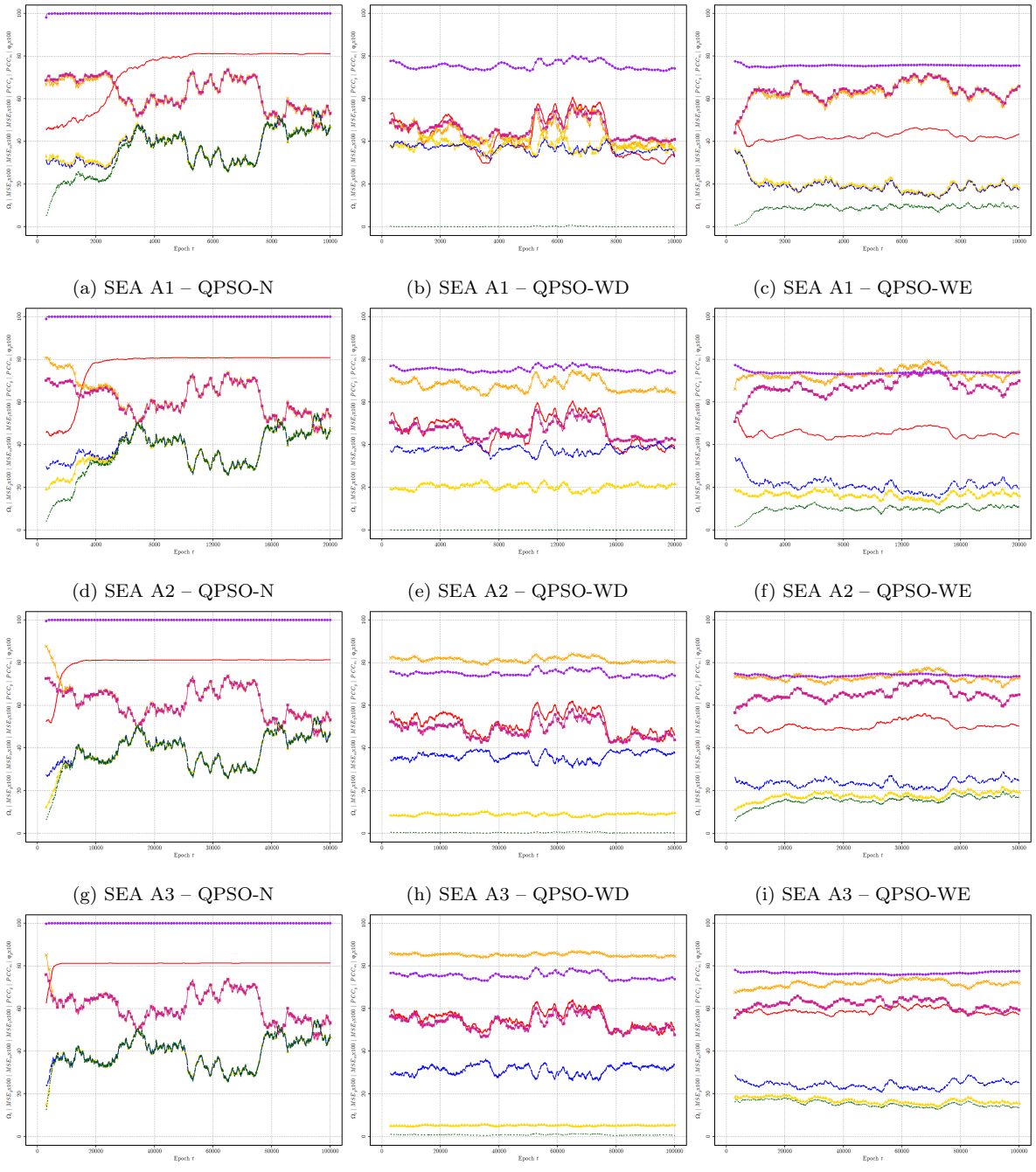
(a) SEA A1 – BP-N (b) SEA A1 – BP-WD (c) SEA A1 – BP-WE

(d) SEA A2 – BP-N (e) SEA A2 – BP-WD (f) SEA A2 – BP-WE

(g) SEA A3 – BP-N (h) SEA A3 – BP-WD (i) SEA A3 – BP-WE

(j) SEA A4 – BP-N (k) SEA A4 – BP-WD (l) SEA A4 – BP-WE

Figure A.26: Saturation, accuracy, and complexity performance trends of the BP classifiers for the A1 – A4 SEA problems



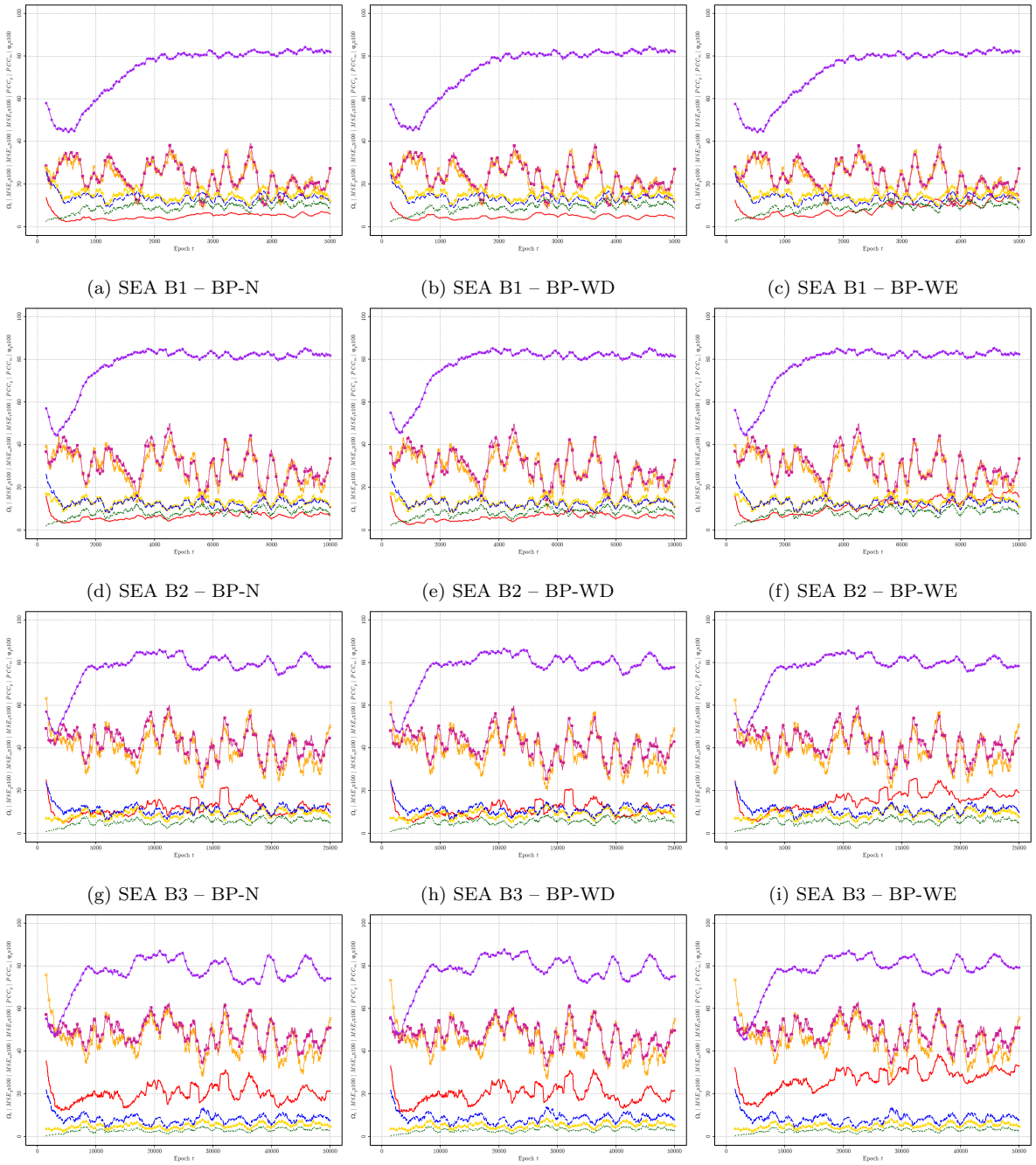
(a) SEA A1 – QPSO-N (b) SEA A1 – QPSO-WD (c) SEA A1 – QPSO-WE

(d) SEA A2 – QPSO-N (e) SEA A2 – QPSO-WD (f) SEA A2 – QPSO-WE

(g) SEA A3 – QPSO-N (h) SEA A3 – QPSO-WD (i) SEA A3 – QPSO-WE

(j) SEA A4 – QPSO-N (k) SEA A4 – QPSO-WD (l) SEA A4 – QPSO-WE

Figure A.27: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the A1 – A4 SEA problems



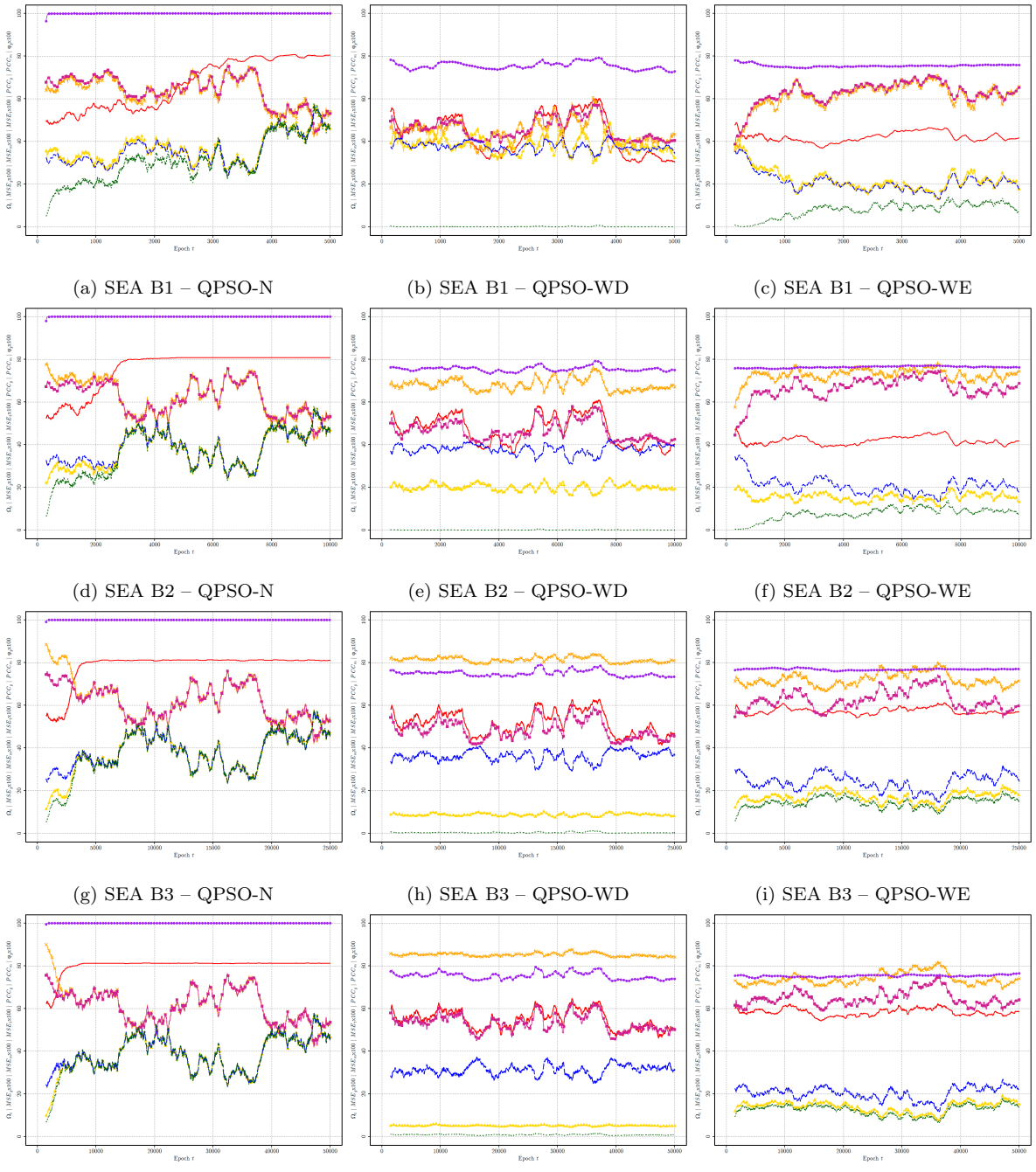
(a) SEA B1 – BP-N (b) SEA B1 – BP-WD (c) SEA B1 – BP-WE

(d) SEA B2 – BP-N (e) SEA B2 – BP-WD (f) SEA B2 – BP-WE

(g) SEA B3 – BP-N (h) SEA B3 – BP-WD (i) SEA B3 – BP-WE

(j) SEA B4 – BP-N (k) SEA B4 – BP-WD (l) SEA B4 – BP-WE

Figure A.28: Saturation, accuracy, and complexity performance trends of the BP classifiers for the B1 – B4 SEA problems



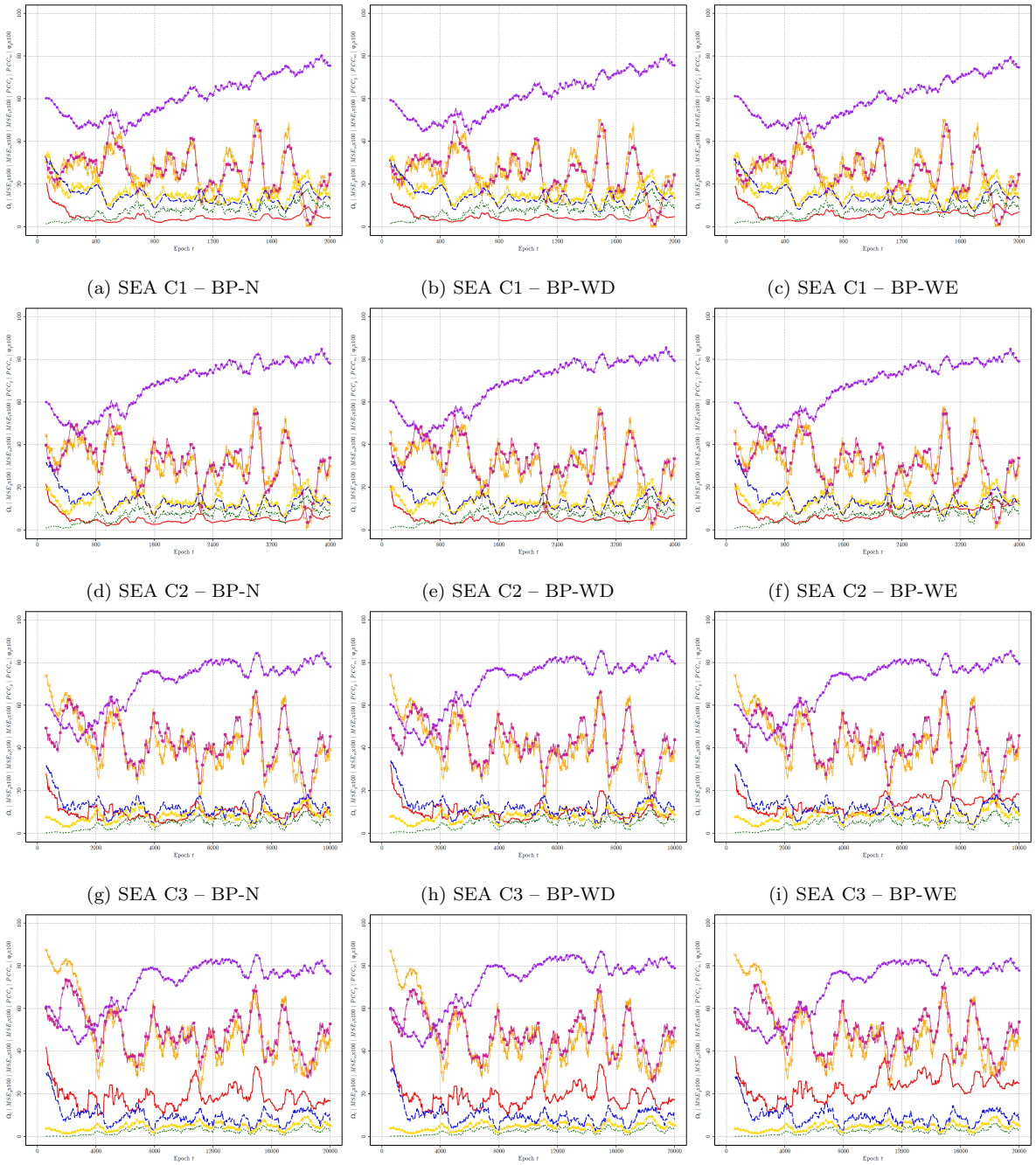
(a) SEA B1 – QPSO-N (b) SEA B1 – QPSO-WD (c) SEA B1 – QPSO-WE

(d) SEA B2 – QPSO-N (e) SEA B2 – QPSO-WD (f) SEA B2 – QPSO-WE

(g) SEA B3 – QPSO-N (h) SEA B3 – QPSO-WD (i) SEA B3 – QPSO-WE

(j) SEA B4 – QPSO-N (k) SEA B4 – QPSO-WD (l) SEA B4 – QPSO-WE

Figure A.29: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the B1 – B4 SEA problems



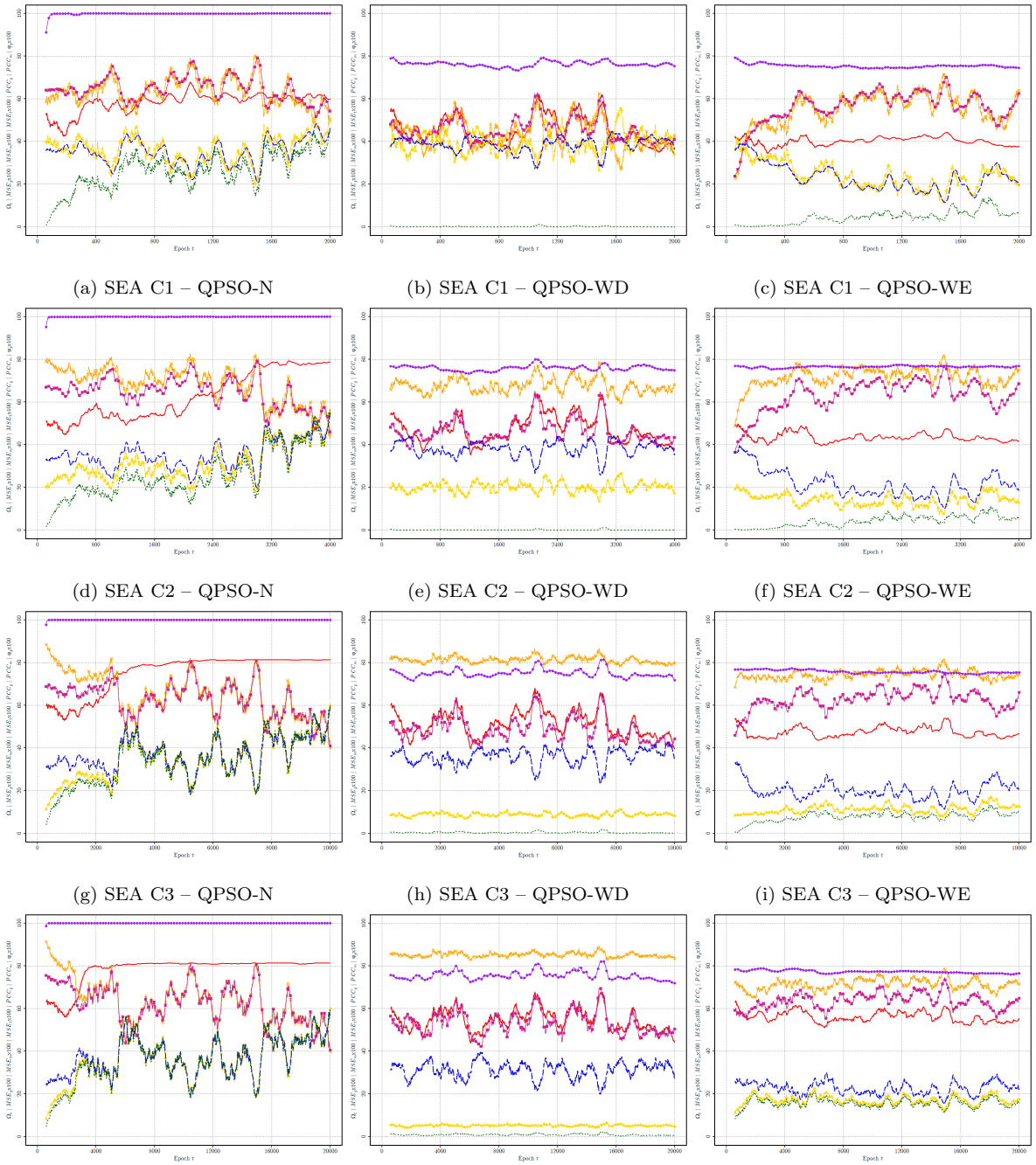
(a) SEA C1 – BP-N (b) SEA C1 – BP-WD (c) SEA C1 – BP-WE

(d) SEA C2 – BP-N (e) SEA C2 – BP-WD (f) SEA C2 – BP-WE

(g) SEA C3 – BP-N (h) SEA C3 – BP-WD (i) SEA C3 – BP-WE

(j) SEA C4 – BP-N (k) SEA C4 – BP-WD (l) SEA C4 – BP-WE

Figure A.30: Saturation, accuracy, and complexity performance trends of the BP classifiers for the C1 – C4 SEA problems



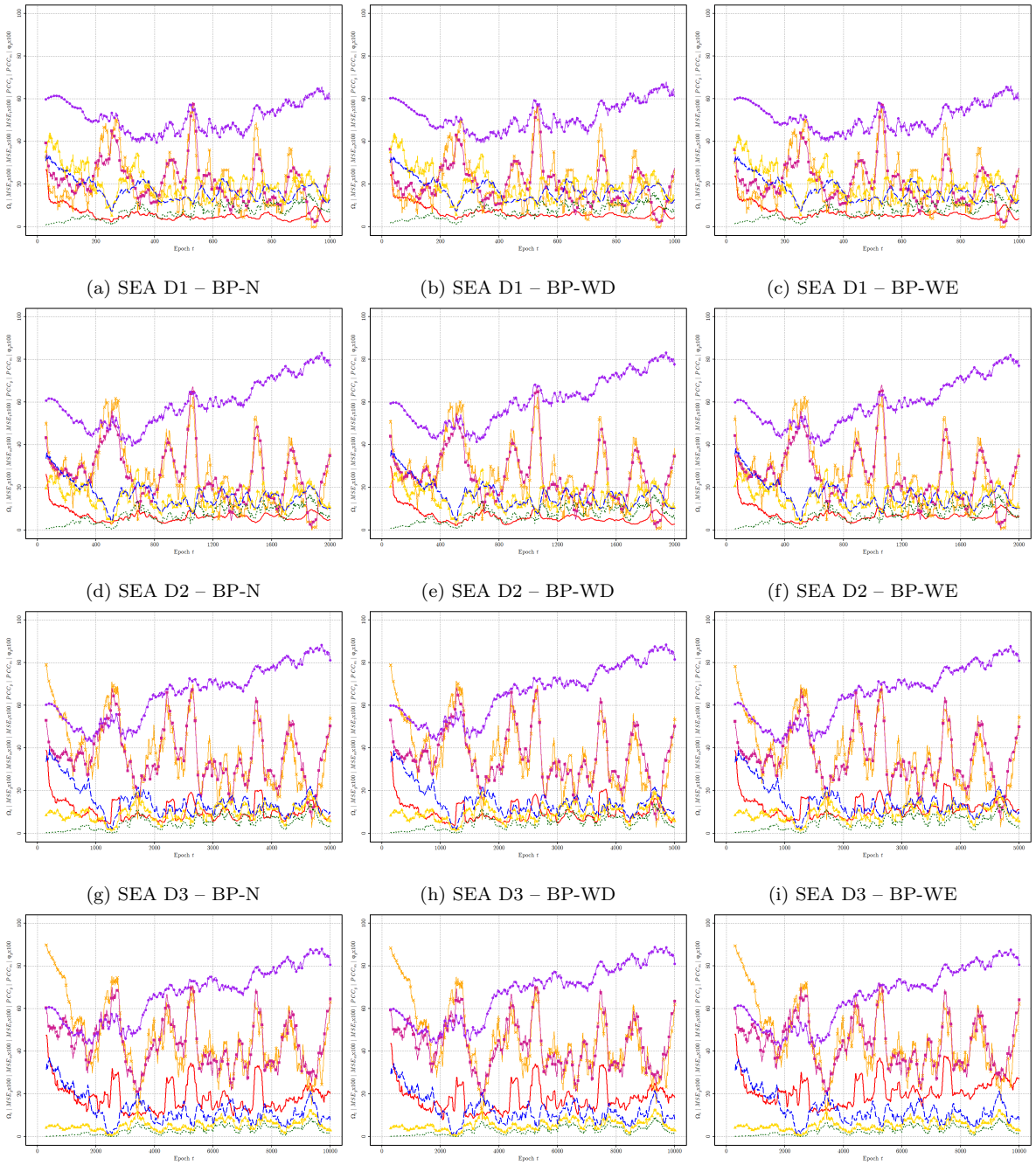
(a) SEA C1 – QPSO-N (b) SEA C1 – QPSO-WD (c) SEA C1 – QPSO-WE

(d) SEA C2 – QPSO-N (e) SEA C2 – QPSO-WD (f) SEA C2 – QPSO-WE

(g) SEA C3 – QPSO-N (h) SEA C3 – QPSO-WD (i) SEA C3 – QPSO-WE

(j) SEA C4 – QPSO-N (k) SEA C4 – QPSO-WD (l) SEA C4 – QPSO-WE

Figure A.31: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the C1 – C4 SEA problems



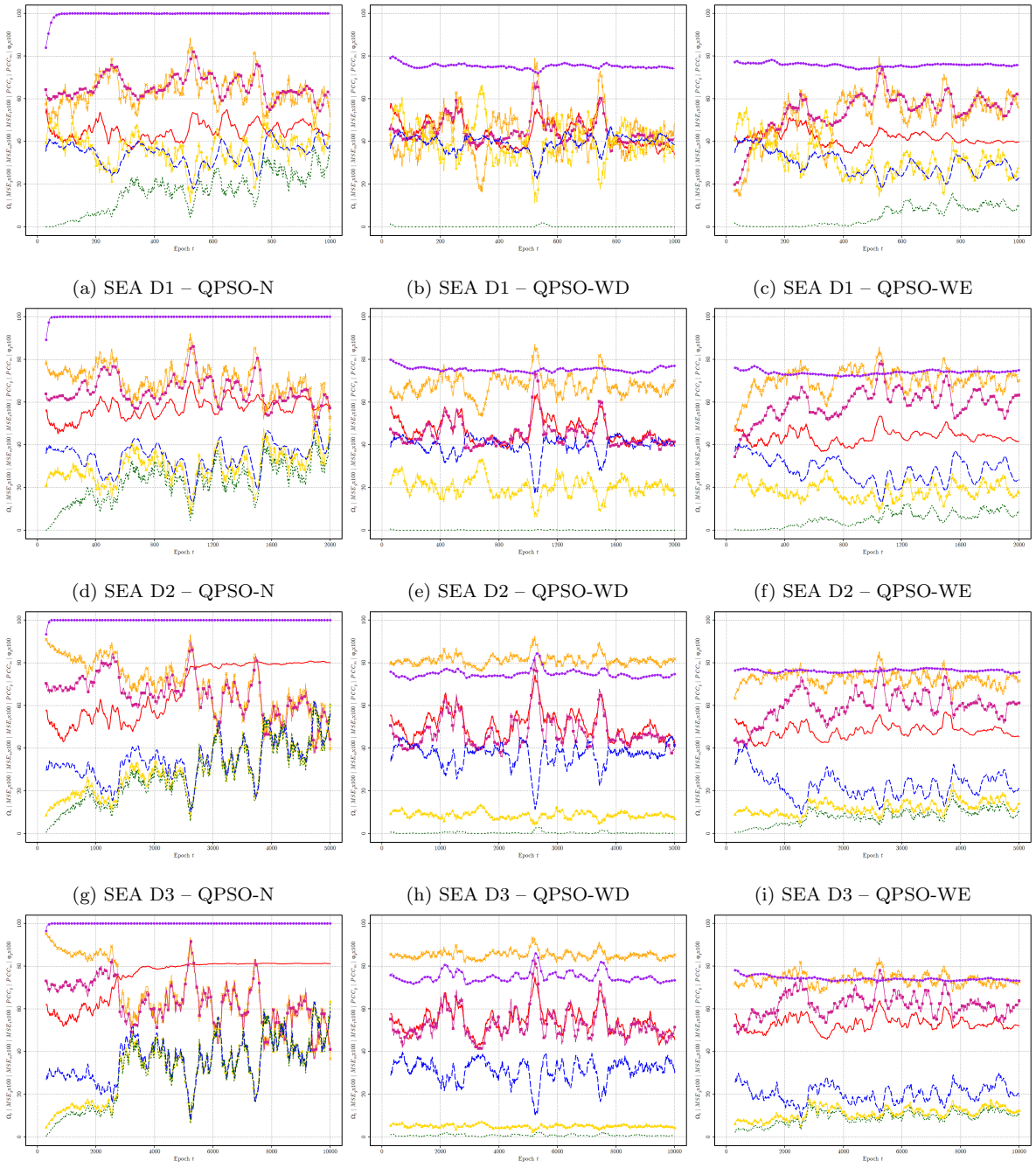
(a) SEA D1 – BP-N (b) SEA D1 – BP-WD (c) SEA D1 – BP-WE

(d) SEA D2 – BP-N (e) SEA D2 – BP-WD (f) SEA D2 – BP-WE

(g) SEA D3 – BP-N (h) SEA D3 – BP-WD (i) SEA D3 – BP-WE

(j) SEA D4 – BP-N (k) SEA D4 – BP-WD (l) SEA D4 – BP-WE

Figure A.32: Saturation, accuracy, and complexity performance trends of the BP classifiers for the D1 – D4 SEA problems



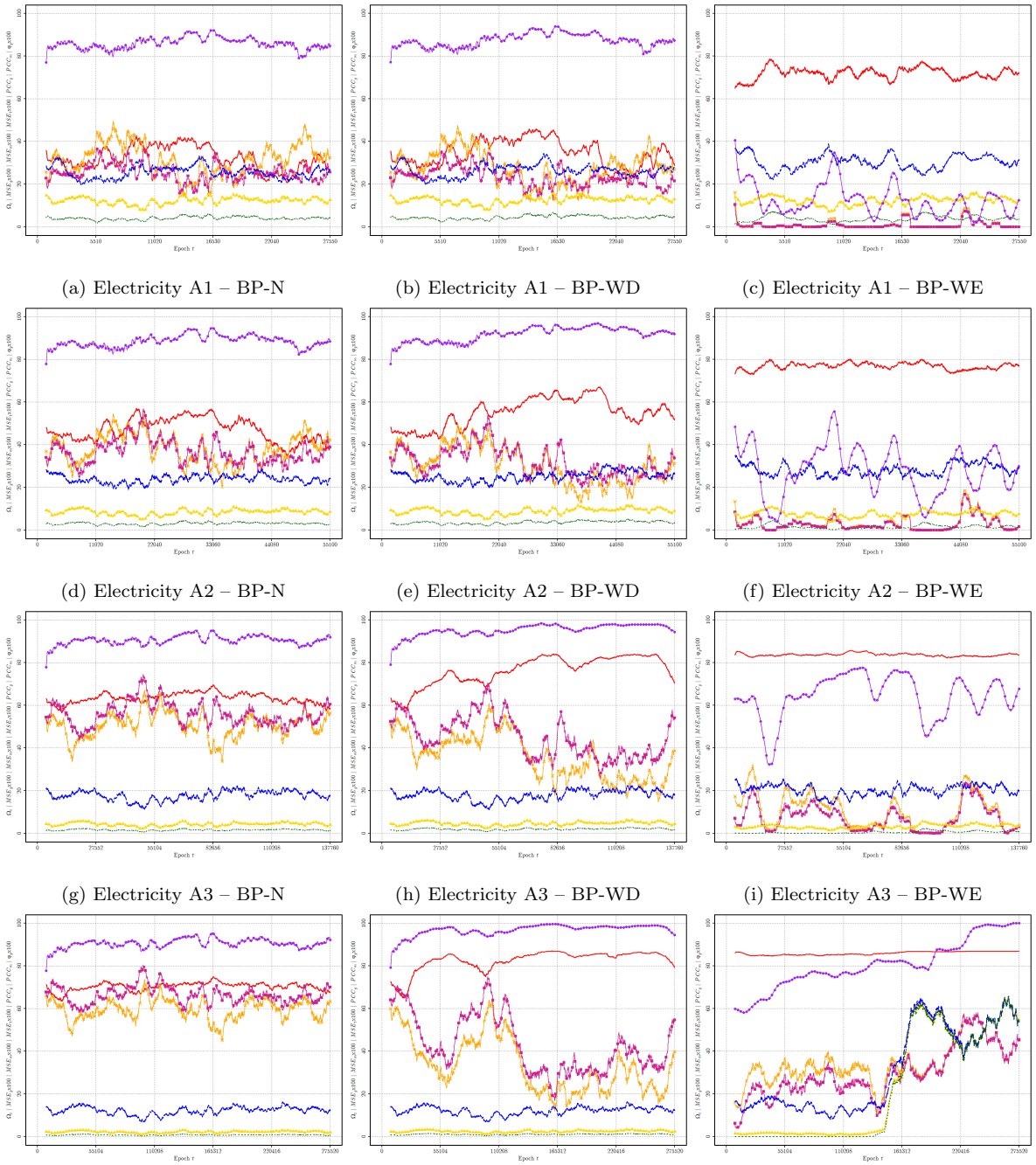
(a) SEA D1 – QPSO-N (b) SEA D1 – QPSO-WD (c) SEA D1 – QPSO-WE

(d) SEA D2 – QPSO-N (e) SEA D2 – QPSO-WD (f) SEA D2 – QPSO-WE

(g) SEA D3 – QPSO-N (h) SEA D3 – QPSO-WD (i) SEA D3 – QPSO-WE

(j) SEA D4 – QPSO-N (k) SEA D4 – QPSO-WD (l) SEA D4 – QPSO-WE

Figure A.33: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the D1 – D4 SEA problems



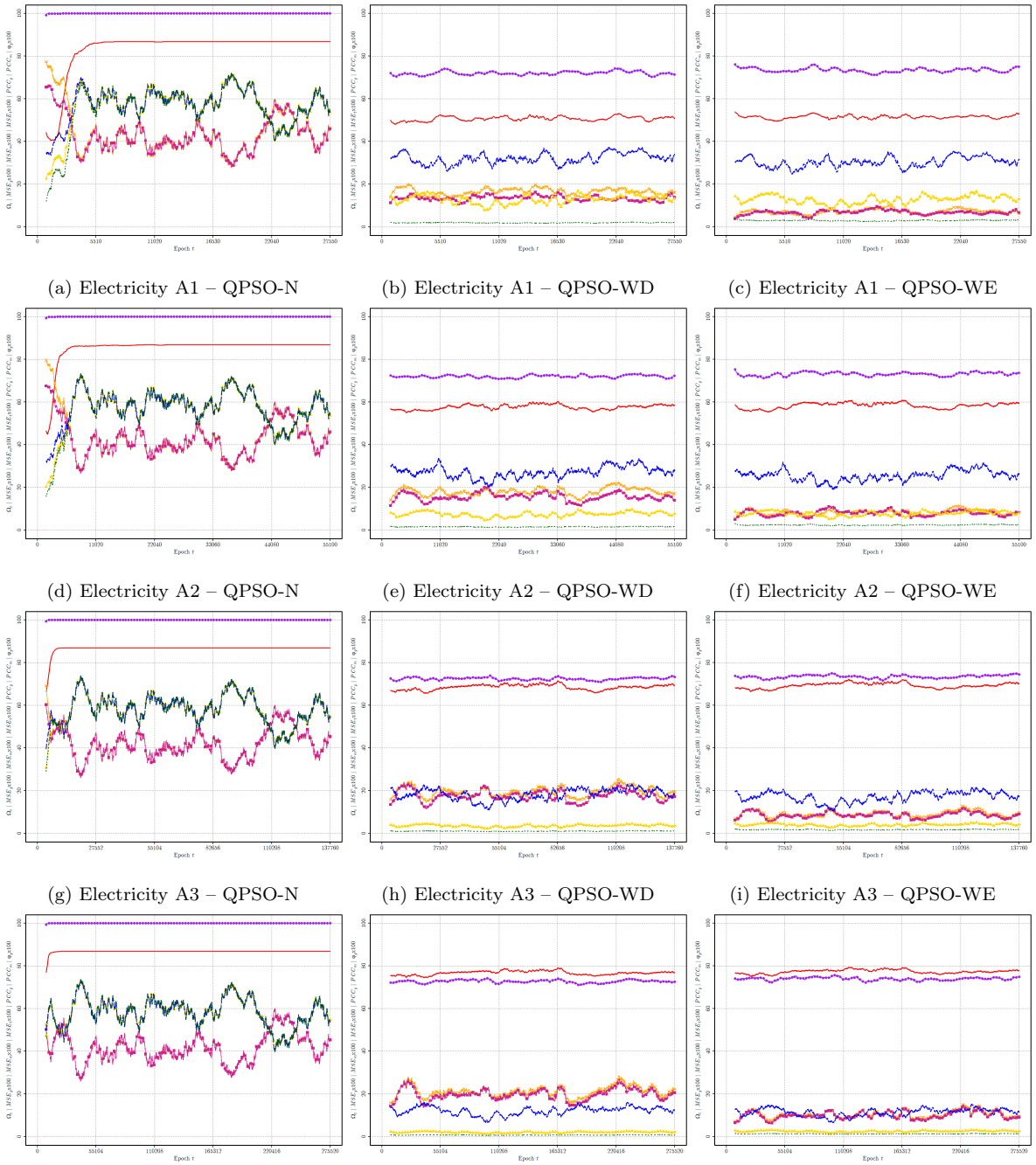
(a) Electricity A1 – BP-N (b) Electricity A1 – BP-WD (c) Electricity A1 – BP-WE

(d) Electricity A2 – BP-N (e) Electricity A2 – BP-WD (f) Electricity A2 – BP-WE

(g) Electricity A3 – BP-N (h) Electricity A3 – BP-WD (i) Electricity A3 – BP-WE

(j) Electricity A4 – BP-N (k) Electricity A4 – BP-WD (l) Electricity A4 – BP-WE

Figure A.34: Saturation, accuracy, and complexity performance trends of the BP classifiers for the A1 – A4 Electricity problems



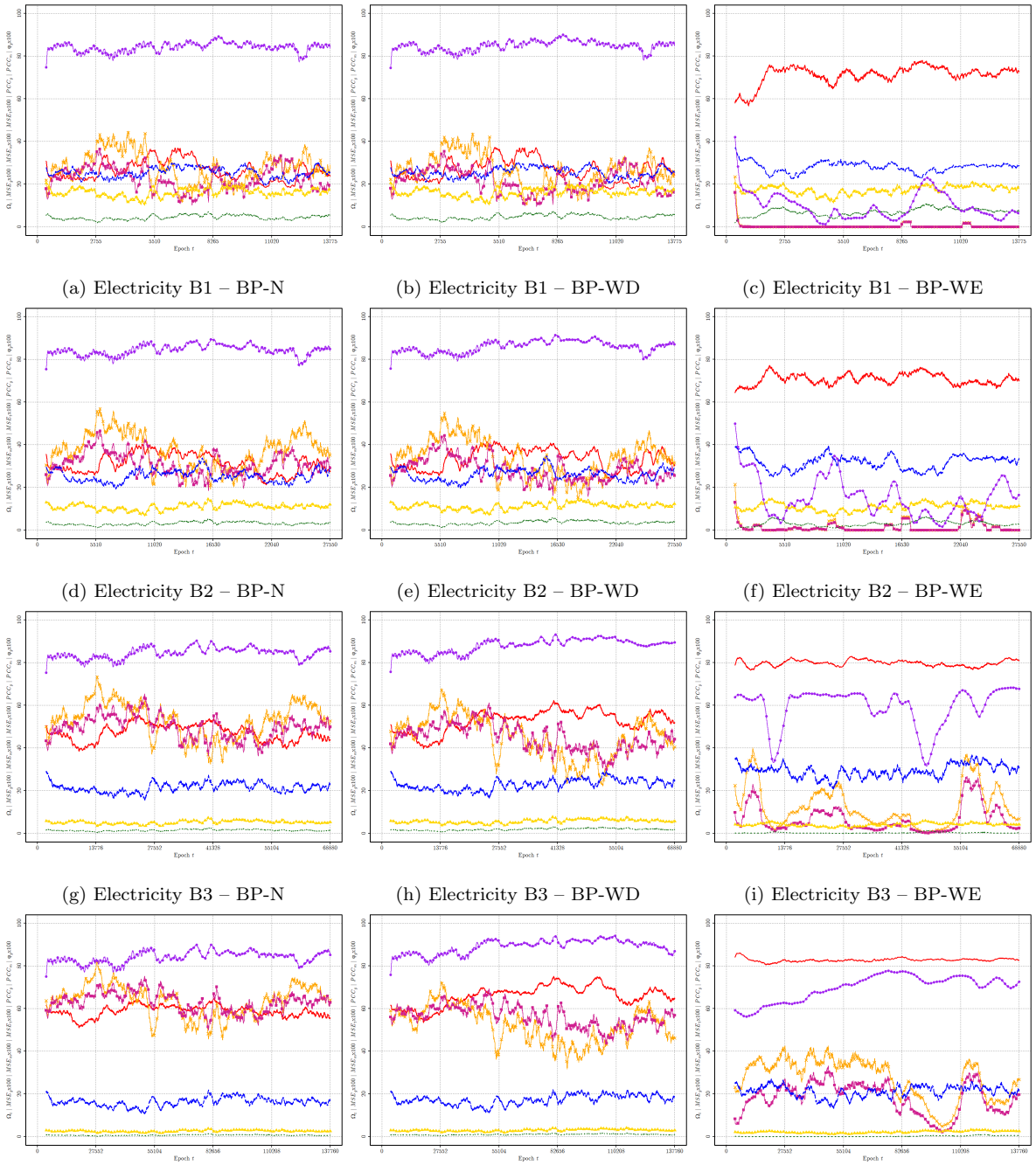
(a) Electricity A1 – QPSO-N (b) Electricity A1 – QPSO-WD (c) Electricity A1 – QPSO-WE

(d) Electricity A2 – QPSO-N (e) Electricity A2 – QPSO-WD (f) Electricity A2 – QPSO-WE

(g) Electricity A3 – QPSO-N (h) Electricity A3 – QPSO-WD (i) Electricity A3 – QPSO-WE

(j) Electricity A4 – QPSO-N (k) Electricity A4 – QPSO-WD (l) Electricity A4 – QPSO-WE

Figure A.35: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the A1 – A4 Electricity problems



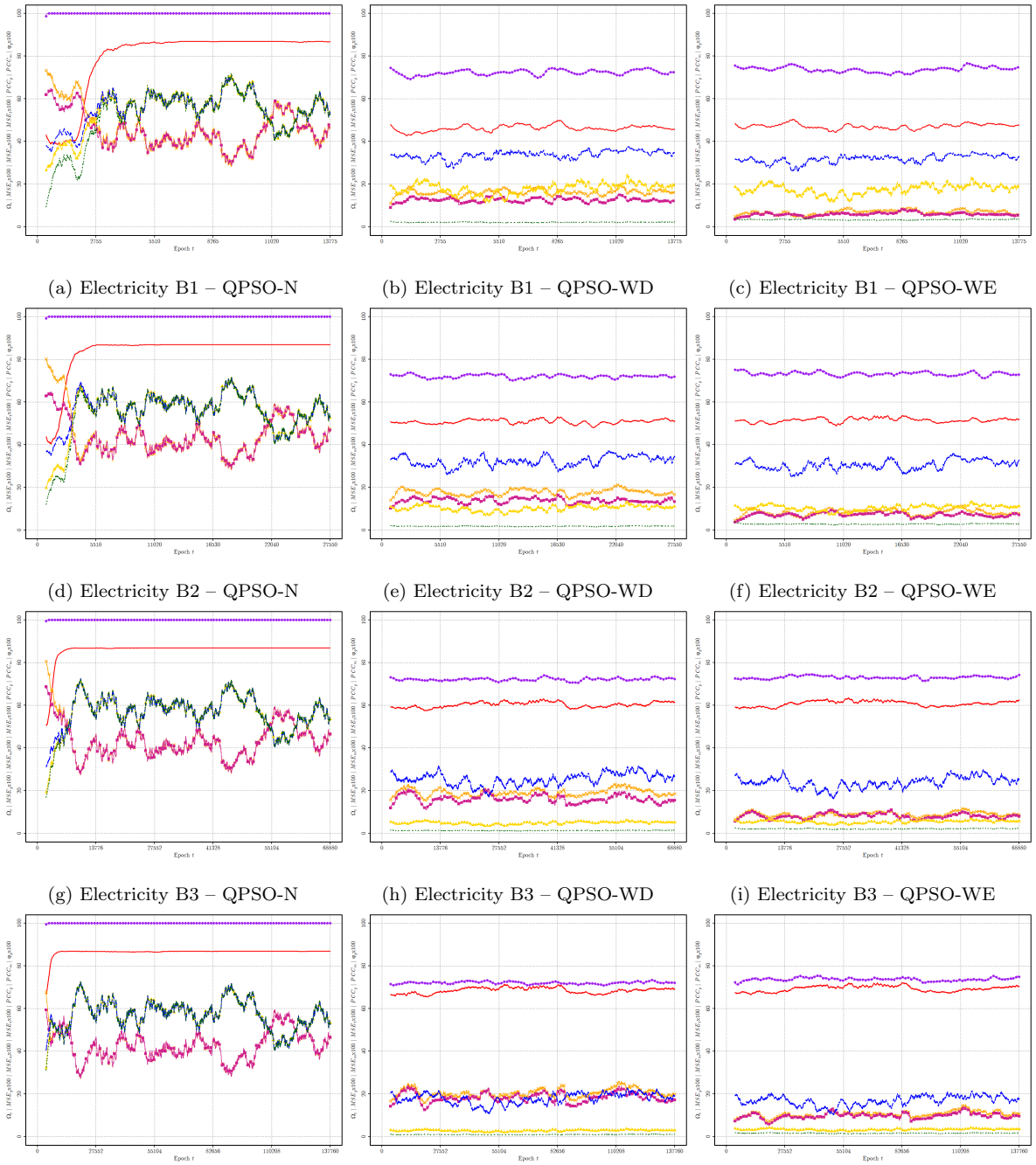
(a) Electricity B1 – BP-N (b) Electricity B1 – BP-WD (c) Electricity B1 – BP-WE

(d) Electricity B2 – BP-N (e) Electricity B2 – BP-WD (f) Electricity B2 – BP-WE

(g) Electricity B3 – BP-N (h) Electricity B3 – BP-WD (i) Electricity B3 – BP-WE

(j) Electricity B4 – BP-N (k) Electricity B4 – BP-WD (l) Electricity B4 – BP-WE

Figure A.36: Saturation, accuracy, and complexity performance trends of the BP classifiers for the B1 – B4 Electricity problems



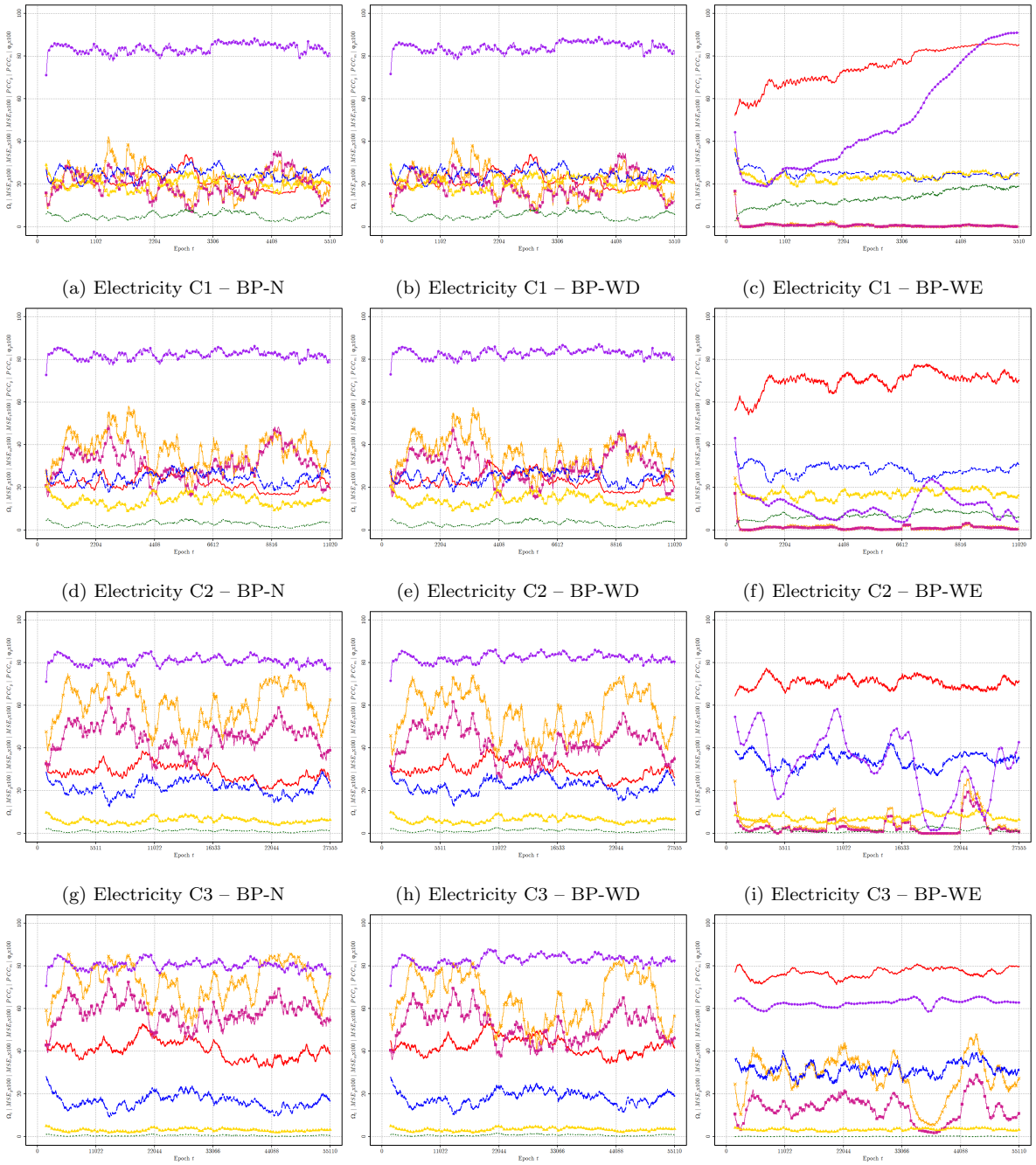
(a) Electricity B1 – QPSO-N (b) Electricity B1 – QPSO-WD (c) Electricity B1 – QPSO-WE

(d) Electricity B2 – QPSO-N (e) Electricity B2 – QPSO-WD (f) Electricity B2 – QPSO-WE

(g) Electricity B3 – QPSO-N (h) Electricity B3 – QPSO-WD (i) Electricity B3 – QPSO-WE

(j) Electricity B4 – QPSO-N (k) Electricity B4 – QPSO-WD (l) Electricity B4 – QPSO-WE

Figure A.37: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the B1 – B4 Electricity problems



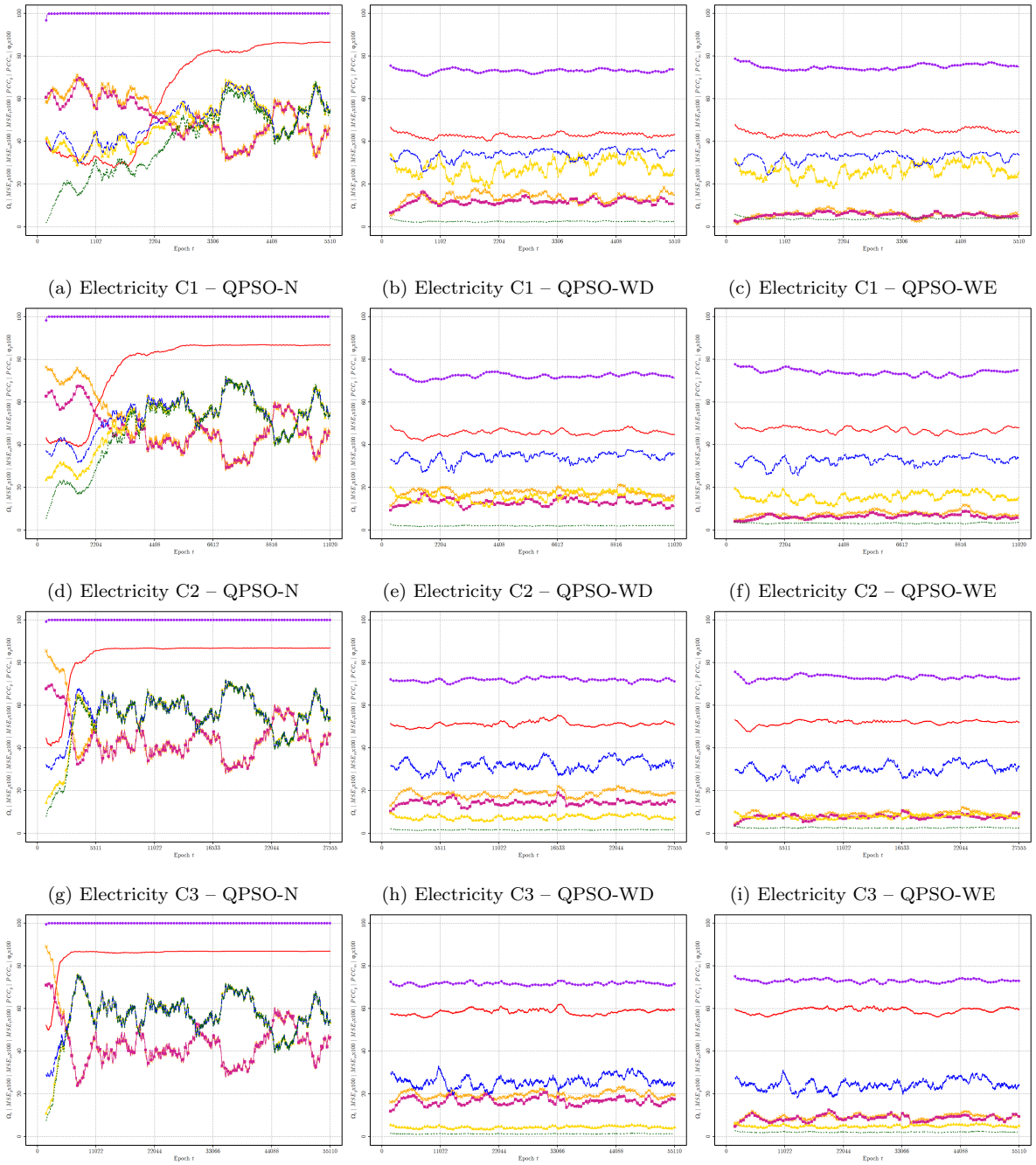
(a) Electricity C1 – BP-N (b) Electricity C1 – BP-WD (c) Electricity C1 – BP-WE

(d) Electricity C2 – BP-N (e) Electricity C2 – BP-WD (f) Electricity C2 – BP-WE

(g) Electricity C3 – BP-N (h) Electricity C3 – BP-WD (i) Electricity C3 – BP-WE

(j) Electricity C4 – BP-N (k) Electricity C4 – BP-WD (l) Electricity C4 – BP-WE

Figure A.38: Saturation, accuracy, and complexity performance trends of the BP classifiers for the C1 – C4 Electricity problems



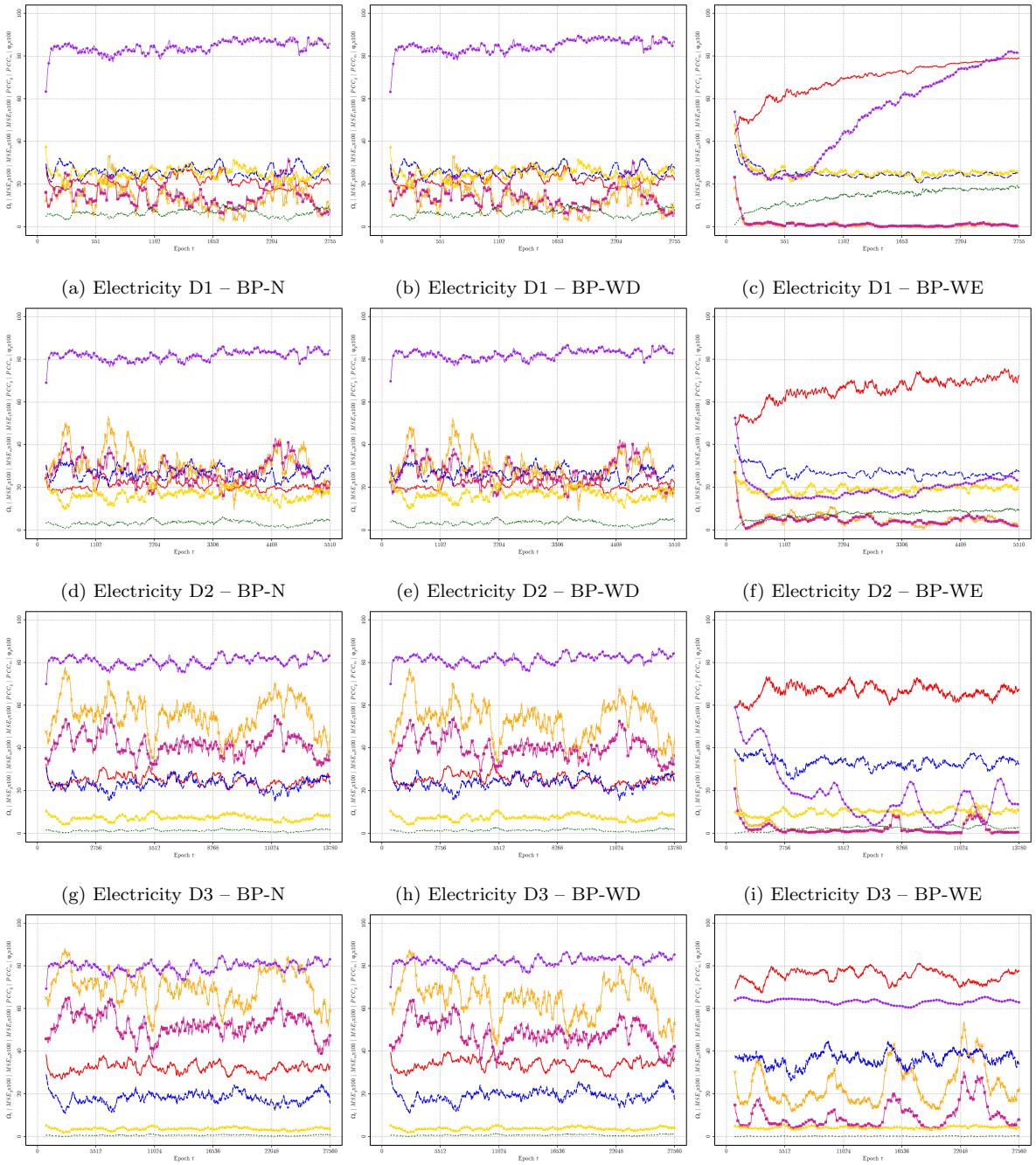
(a) Electricity C1 – QPSO-N (b) Electricity C1 – QPSO-WD (c) Electricity C1 – QPSO-WE

(d) Electricity C2 – QPSO-N (e) Electricity C2 – QPSO-WD (f) Electricity C2 – QPSO-WE

(g) Electricity C3 – QPSO-N (h) Electricity C3 – QPSO-WD (i) Electricity C3 – QPSO-WE

(j) Electricity C4 – QPSO-N (k) Electricity C4 – QPSO-WD (l) Electricity C4 – QPSO-WE

Figure A.39: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the C1 – C4 Electricity problems



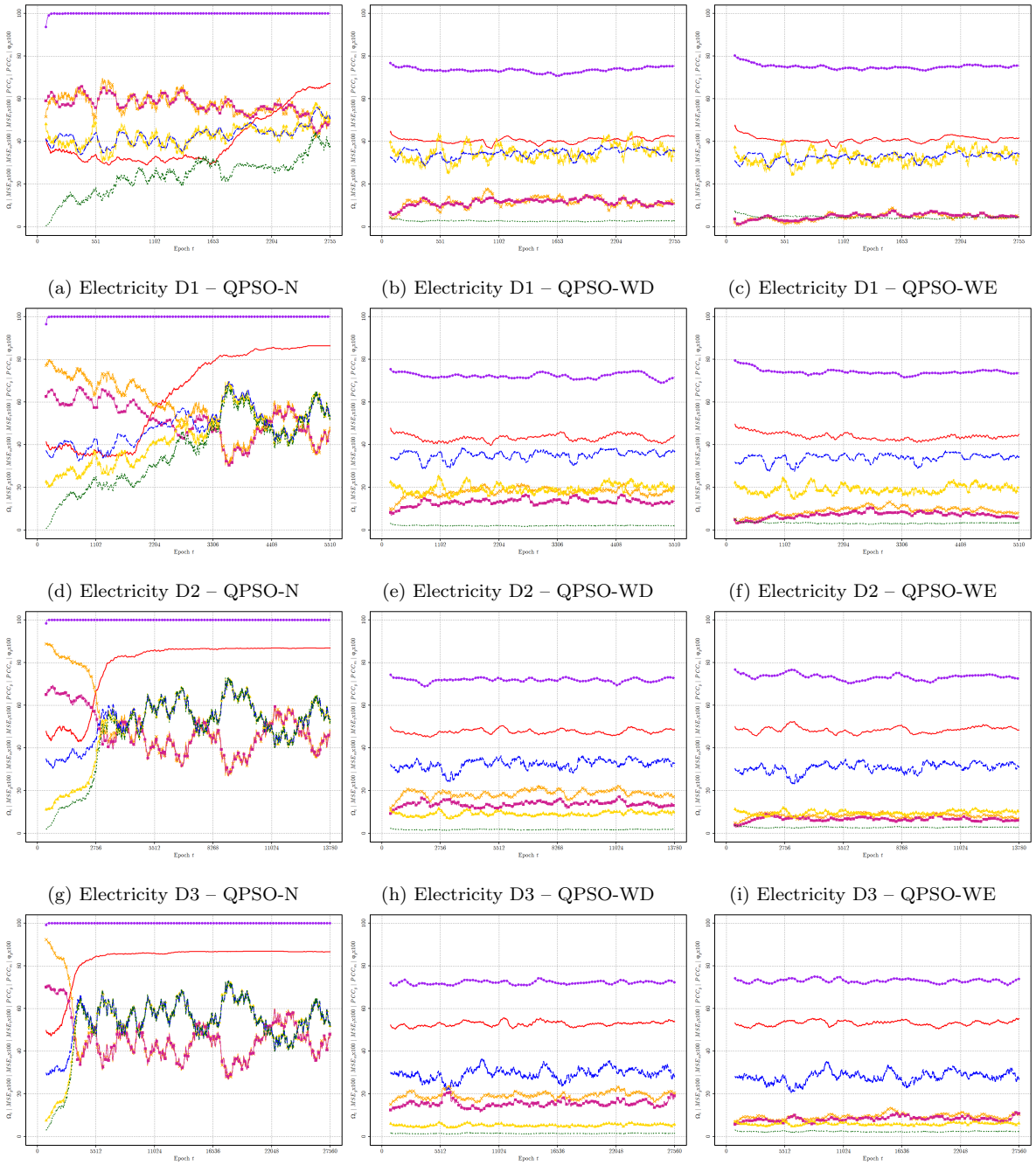
(a) Electricity D1 – BP-N (b) Electricity D1 – BP-WD (c) Electricity D1 – BP-WE

(d) Electricity D2 – BP-N (e) Electricity D2 – BP-WD (f) Electricity D2 – BP-WE

(g) Electricity D3 – BP-N (h) Electricity D3 – BP-WD (i) Electricity D3 – BP-WE

(j) Electricity D4 – BP-N (k) Electricity D4 – BP-WD (l) Electricity D4 – BP-WE

Figure A.40: Saturation, accuracy, and complexity performance trends of the BP classifiers for the D1 – D4 Electricity problems



(a) Electricity D1 – QPSO-N (b) Electricity D1 – QPSO-WD (c) Electricity D1 – QPSO-WE

(d) Electricity D2 – QPSO-N (e) Electricity D2 – QPSO-WD (f) Electricity D2 – QPSO-WE

(g) Electricity D3 – QPSO-N (h) Electricity D3 – QPSO-WD (i) Electricity D3 – QPSO-WE

(j) Electricity D4 – QPSO-N (k) Electricity D4 – QPSO-WD (l) Electricity D4 – QPSO-WE

Figure A.41: Saturation, accuracy, and complexity performance trends of the QPSO classifiers for the D1 – D4 Electricity problems

Appendix B

Acronyms

A list of all the acronyms and their definitions are provided in this appendix.

AI	artificial intelligence
ANN	artificial neural network
BP	back propagation
CI	computational intelligence
CNN	convolutional neural network
CPSO	charged particle swarm optimisation
CVFDT	concept-adapting very fast decision tree
DNN	deep neural network
ELM	extreme learning machine
FFNN	feed forward neural network
GA	genetic algorithm
gBest	global best
GD	gradient descent
HPSO	heterogeneous particle swarm optimisation
HTM	hierarchical temporal memory
lBest	local best
MLP	multi-layer perceptron
MSE	mean square error
MWU	Mann-Whitney U

NN	neural network
OS-ELM	online sequential extreme learning machine
PCC	percentage correct classifications
PGC	percentage good classifications
PSO	particle swarm optimisation
QPSO	quantum particle swarm optimisation
ReLU	rectified linear unit
RNN	recurrent neural network
ROC	reciever operating characteristic
RT2McRVFLN	recurrent type-2 random vector functional link network
RdNN	random neural network
SDP	streamed data problem
SDCP	streamed data classification problem
SSE	sum square error
VFDT	very fast decision tree
WD	weight decay
WE	weight elimination

Appendix C

Symbols

This appendix provides an explanation for each of the symbols used throughout this thesis. The symbols are arranged by the chapter in which they were defined.

C.1 Chapter 2

I	The number of inputs for an approximation problem.
K	The number of outputs for an approximation problem.
D	An arbitrary dataset of patterns.
D_p	The p-th pattern in dataset D
$\|D\|$	The cardinality of the dataset D.
$\vec{z}$	The inputs vector for a pattern.
$\vec{t}$	The targets vector for a pattern.

C.2 Chapter 3

n_p	The number of particles in a PSO swarm.
N	An arbitrary number.
$f(\vec{x})$	A objective function that describe optimisation problem.
$\vec{x}$	A PSO particle's position vector.

$\vec{v}$	A PSO particle's velocity vector.
$\vec{y}$	A PSO particle's personal best position vector.
$\vec{\hat{y}}$	A PSO particle's neighbourhood best position vector.
x_{ij}	The j-th element of the position vector of particle i in the swarm of a PSO.
v_{ij}	The j-th element of the velocity vector of particle i in the swarm of a PSO.
y_{ij}	The j-th element of the personal best position vector of particle i in the swarm of a PSO.
$\hat{y}_{ij}$	The j-th element of the neighbourhood best position vector of particle i in the swarm of a PSO.
ω	The momentum term of a PSO's particle.
c_1	The cognitive component's positive acceleration co-efficient.
c_2	The social component's positive acceleration co-efficient.
$\vec{r}_1$	The cognitive component's random uniformly sampled vector.
$\vec{r}_2$	The social component's random uniformly sampled vector.
t	The current iteration for an optimisation algorithm.
$\mathcal{D}$	The diversity of a PSO's swarm described by Equation (3.3).
$\bar{x}_j$	The j-th element of the position vector of the PSO swarm's spatial centre.
$v_{\vec{max}}$	The maximum velocity vector used by velocity clamping.
χ	The constriction coefficient determined by Equation (3.4).
κ	A constant in the Equation (3.4).
ϕ	The sum of c_1 and c_2.
d	The sampling distribution parameter of a QPSO.
r	The sampling radius parameter of a QPSO.

C.3 Chapter 4

ϵ	An arbitrary error value.
n_n	The total number of neurons in a ANN, including bias neurons.
n_s	The total number of synapses in a ANN, including synapses from bias neurons.
n_l	The total number of layers in a ANN.
n_i	The number of input neurons, excluding bias neurons, in a FFNN.
n_h	The number of hidden neurons, excluding bias neurons, in the hidden layer of a 3-layer FFNN.
n_k	The number of output neurons in a FFNN.
z_i	The value of the input neuron i.
h_h	The activation value of the hidden neuron h.
o_k	The activation value of the output neuron k.
w_{kh}	The weight of the synapse from hidden neuron h to output neuron k in a 3-layer FFNN.
w_{hi}	The weight of the synapse from input neuron i to hidden neuron h.
net	The net input value of a neuron.
net_{in}	The net input signal for the incoming synapses.
n_{in}	The number of incoming synapses for a neuron.
$w_{in,j}$	The weight of the j-th incoming synapse of a neuron.
$v_{in,j}$	The value of the source neuron of the j-th incoming synapse of a neuron.
net_{Σ}	The net input value of a summation unit neuron.
$f_{Sig}(net)$	The sigmoid activation function.
λ	The steepness factor of an activation function.
θ	The shifting factor of an activation function.
$f_{ReL}(net)$	The rectified linear activation function defined by Equation (4.4).
$max(...)$	The maximum value function for a set of values.
$f_L(net)$	The linear activation function.

E_g	The generalisation error of an ANN.
E_v	The validation error of an ANN.
E_t	The training error of an ANN.
D_g	The data set used to calculate E_g, know as the generalisation set.
D_v	The data set used to calculate E_v, know as the validation set.
D_t	The data set used to calculate E_t, know as the training set.
p_g	The D_g's splitting percentage.
p_v	The D_v's splitting percentage.
p_t	The D_t's splitting percentage.
$\|D\|$	The number of data points in dataset D.
$t_{p,k}$	The k-th target of the target vector $\vec{t}$ of pattern p.
$o_{p,k}$	The k-th output neuron's value when processing pattern p with the ANN.
y_p	The "goodness" value for pattern p when using percentage good classifications (PGC).
$\vec{o}_p$	The outputs vector of the ANN when processing pattern p.
n_{s_e}	The number of effective synapses in an ANN.
n_{n_e}	The number of effective neurons in an ANN.
w_j	The j-th weight in an ANN.
b_w	The width of a bin, i.e. interval, when binning.
$\bar{E}_v$	The moving average of E_v.
σ_{E_v}	The standard deviation of $\bar{E}_v$.
m_p	The period of a moving average.
$\rho(t)$	The generalisation factor at epoch t.
$\varphi_\rho(t)$	The generalisation factor constraint at epoch t
$min(...)$	The minimum value function for a set of values.
$\bar{\rho}$	The current moving average of ρ.
σ_ρ	The standard deviation of $\bar{\rho}$.
$fanin$	The N_{is} of the destination neuron of a synapse.

n_e	The number of epochs for which a learning algorithm learns an environment instance.
n_e^*	The maximum n_e per environment instance.
E_r	The regularisation term.
E'_t	The regularised error function.
λ_r	The regularisation coefficient.
w_0	The relevancy threshold constant, used by WE.
α	The momentum term in BP.
η	The learning rate term in BP.
w_j	The j-th weight in a ANN.
$\Delta w_j(t)$	The change, weight w_j experiences at epoch t.
φ_{b_w}	The binning saturation measure, defined by Equation (4.20).
B	The number of bins.
f_b	The frequency of bin b.
$\bar{g}'_b$	The average activation value, scaled to the range $[-1, 1]$, for the activation value bin b.

C.4 Chapter 6

$\vec{b}_p$	The binned input vector of $\vec{z}_p$.
I_{b_w}	The set of environment instances extracted by algorithm 4 using b_w.
Θ_{b_w}	The spatial severity of a SDCP, defined by Equation (6.1).
$I_{b_w,e}$	The set of binned patterns in environment instance e in I_{b_w}.
$I_{b_w,e,p}$	The p-th pattern in $I_{b_w,e}$.
$\Delta_\Theta(I_{b_w,e,p})$	The spatial change for $EI_{B,dp,e,p}$.
Δ^*_Θ	The maximum euclidean distance by which a target vector can change.
τ_{b_w}	The temporal severity of a SDCP, defined by Equation (6.2).
Θ'_{b_w}	The normalised spatial severity of a SDCP, defined by Equation (6.3).
S	A arbitrary set of sequentially-ordered data stream datasets.

S_d	The d-th sequentially-ordered data stream dataset in S.
τ'_{b_w}	The normalised temporal severity of a SDCP, defined by Equation (6.4).
Λ_{b_w}	The dynamic environment classification of a SDCP, defined by Equation (6.5).
ζ_{Q,b_w}	The degree to which a SDCP's environment is dominated by quasi-static characteristics.
ζ_{A,b_w}	The degree to which a SDCP's environment is dominated by abrupt characteristics.
ζ_{P,b_w}	The degree to which a SDCP's environment is dominated by progressive characteristics.
ζ_{C,b_w}	The degree to which a SDCP's environment is dominated by chaotic characteristics.
ζ_{b_w}	The degree to which a SDCP's environment is dominated by Λ_{b_w}, defined by Equation (6.10).

C.5 Chapter 7

$\Delta w_{kh}(t)$	The change, weight w_{kh} experiences at epoch t.
f'_{ReL}	The partial derivative of $f_{ReL}(net_\Sigma)$ with respect to net_Σ.
δ_{o_k}	The error signal of output neuron k, and is defined by Equation (7.3).
J	The number of hidden neurons, excluding bias neurons in the hidden layer.
$\Delta w_{hi}(t)$	The change, weight w_{hi} experiences at epoch t.
δ_{h_h}	The error signal of hidden neuron h, and is defined by Equation (7.5).
$\Delta w_{ki}(t)$	The change, weight w_{ki} experiences at epoch t during BP.
$net_{h_h,p}$	The net value of hidden neuron h, when the FFNN process pattern p in dataset D.

C.6 Chapter 8

a_i	A co-efficient of the i-th hyperplane in the moving hyperplane problem domain.
c	A arbitrary user-selected constant.
y	A arbitrary value.
θ	A problem's class threshold.
c_i	The i-th element of the hypersphere's centre.
r^2	The squared radius of a hypersphere.
r^2_θ	The squared radius of the threshold hypersphere.
f_1	The first threshold line of sliding threshold problem domain.
f_2	The second threshold line of sliding threshold problem domain.
θ_1	The threshold of f_1.
θ_2	The threshold of f_2.
$f_3(\vec{z})$	The dimension extracting function used by the sliding thresholds problem domain.
v_s	The linearly scaled value calculated by Equation (8.11).
v_u	The original value that needs to be linearly scaled.
$v_{u,min}$	The lower bound of range that v_u exists in.
$v_{u,max}$	The upper bound of range that v_u exists in.
$v_{s,min}$	The lower bound of range that v_s exists in.
$v_{s,max}$	The upper bound of range that v_s exists in.
n_w	The number of windows generated by the sliding windows technique. n_w is defined by Equation (8.12).
w_m	The the size of the windows created by the sliding windows technique.
w_f	The window repetition frequency used by the sliding windows technique.
w_s	The step size used by the sliding windows technique.
$\|w_j\|$	The magnitude of the j-th weight in an ANN.
$\bar{w}$	The average weight magnitude of an ANN.

σ_w	The standard deviation in weight magnitude of an ANN.
Ξ_w	The frequency distribution of the weights in an ANN.
$S^{(p)}$	The output sensitivity matrix for pattern p.
$S_{k,j}^{(p)}$	The output sensitivity of the j-th synapse for the j-th output, using the pattern p.
$N_j^{(p)}$	The output sensitivity norm for the j-th synapse, using pattern p.
$\bar{N}_j$	The average sensitivity norm for the j-th synapse.
σ_j^2	The sample variance of the $N_j^{(p)}$.
Υ_j	The sensitivity test statistic for the j-th synapse.
σ_0^2	The maximum amount of variance that a synapse may display before being considered relevant during Variance nullity testing.
H_0	The null hypothesis for Variance nullity testing.
χ^2	The Chi-squared distribution.
$\chi^2_{df,c}$	is χ^2 using df degrees of freedom and c confidence interval.
α_v	is the significance level for Variance nullity testing.
Υ_c	The critical value for Variance nullity testing.
T	The total number of epochs an FFNN learned for during learning.
$\epsilon(t)$	The error ϵ's value, for the best model found at the end or start of epoch t.
$\bar{C}_\epsilon$	The collective mean of error ϵ, calculated using Equation (8.13).
MSE_g	MSE based on D_g.
MSE_v	MSE based on D_v.
MSE_t	MSE based on D_t.
PCC_g	PCC based on D_g.
PCC_v	PCC based on D_v.
$\varphi_{0.1,1}$	$\varphi_{0.1}$ based on the activation value range $[0, 1]$.
$\varphi_{0.1,5}$	$\varphi_{0.1}$ based on the activation value range $[0, 5]$.
$\varphi_{0.1,10}$	$\varphi_{0.1}$ based on the activation value range $[0, 10]$.

φ_g	The saturation measure based on the minimum of $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$, using D_g.
φ_v	The saturation measure based on the minimum of $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$, using D_v.
O_{MSE_v}	The boolean result of the overfitting constraint Equation (4.9).
O_ρ	The boolean result of the overfitting constraint Equation (4.10).
n_a	The number of activated neurons in a FFNN.
n_b	The total number of bias neurons in a FFNN.
Ω_{FFNN}	The lower bound computational complexity of processing a pattern with a 3-layer FFNN.
Ω_{FFNN_e}	: Lower-bound complexity of processing a pattern with the effective model of a FFNN, using $\sigma_0^2 = 0.0001$ and $\alpha_p = 0.01$.
Ω_r	The effective reduction in lower-bound computational complexity as a percentage, defined by Equation (8.20).
$n_{h_{or}}$	The effective hidden neuron oversize ratio which was determined by dividing b_{h_e} by the number of hidden neurons (including bias neurons) in the optimal architecture for the particular problem domain.
$n_{s_{or}}$	The effective synapse oversize ratio which was determined by dividing n_{s_e} by the number of synapses in the optimal architecture for the particular problem domain.
n_c	is the number of tunable control parameters of a classifier.
$\|D_c\|$	is the number of control parameter configurations tested for a classifier.
$\mathcal{V}_i$	is the set of potential values for the i-th control parameter of a classifier.
$\rho_{x,y}$	The Pearson correlation coefficient of the two arbitrary series x and y.
x_i	The i-th value in an arbitrary series x.
y_i	The i-th value in an arbitrary series y.