



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Training Feedforward Neural Networks with Bayesian Hyper-Heuristics

by

A.N. Schreuder

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

February, 2023

Publication:

*Schreuder, A.N. Training Feedforward Neural Networks with Bayesian Hyper-Heuristics. Master's Dissertation.
University of Pretoria, Department of Computer Science, Pretoria, South Africa. 2022*

Electronic, hyperlinked versions of this dissertation, including data and source code are available online at:

<https://github.com/arneschreuder/masters>

Training Feedforward Neural Networks with Bayesian Hyper-Heuristics

by

A.N. Schreuder

E-mail: an.schreuder@up.ac.za

Abstract

Many different heuristics have been developed and used to train *feedforward neural networks* (FFNNs). However, selection of the best heuristic to train FFNNs is a time consuming and non-trivial exercise. Careful, systematic selection is required to ensure that the best heuristic is used to train FFNNs. In the past, selection was done by trial and error. A modern approach is to automate the heuristic selection process. Often it is found that a single approach is not sufficient. Research has proposed the use of hybridisation of heuristics. One such approach is referred to as *hyper-heuristics* (HHs). HHs focus on dynamically finding the best heuristic or combinations of heuristics in heuristic-space by making use of heuristic performance information. One such implementation of a HH is a population-based approach that guides the search process by dynamically selecting heuristics from a heuristic-pool to be applied to different entities that represent candidate solutions to the problem-space, and work together to find good solutions. This dissertation introduces a novel population-based *Bayesian hyper-heuristic* (BHH). An empirical study is done by using the BHH to train FFNNs. An in-depth behaviour analysis is done and the performance of the BHH is compared to that of ten popular low-level heuristics each with different search behaviours. The chosen heuristic pool consists of classic gradient-based heuristics as well as meta-heuristics. The empirical process is executed on fourteen datasets consisting of classification and regression problems with varying characteristics. Results are analysed for statistical significance and the BHH is shown to be able to train FFNNs well and provide an automated method for finding the best heuristic to train the FFNNs at various stages of the training process.

Keywords: hyper-heuristics, meta-learning, feedforward neural networks, supervised learning, Bayesian statistics

Supervisor: Dr. A.S. Bosman
University of the Pretoria
Department of Computer Science

Supervisor: Prof. C.W. Cleghorn
University of the Witwatersrand
Department of Computer Science

Supervisor: Prof. A.P. Engelbrecht
Stellenbosch University
Department of Industrial Engineering, Computer Science Division

Degree: Master of Science, Computer Science (Artificial Intelligence)

“Men go abroad to wonder at the heights of mountains, at the huge waves of the sea, at the long courses of the rivers, at the vast compass of the ocean, at the circular motions of the stars and they pass by themselves without wondering.”

- St. Augustine

Acknowledgements

“I’ve made the most important discovery of my life. It’s only in the mysterious equation of love that any logical reasons can be found. I’m only here tonight because of you. You are the only reason I am . . . you are all my reasons.”

- Prof. J.F. Nash, Jr.

I would like to thank the following people, without whom this work would never have been possible:

- To my wife, Taylah. You are all my reasons.
- To my daughters, Beané and Aleah. Thank you for giving life meaning.
- To my parents, Adré and Engela. Thank you for always believing in me.
- To my sister, Jeannie and her husband, Jaco. Thank you for always motivating me.
- To my supervisors, Dr. A.S. Bosman, Prof. C.W. Cleghorn and Prof. A.P. Engelbrecht. Thank you for not giving up on me.
- To EPI-USE. Thank you for helping out where I needed it.
- To my creator, God. Thank You . . . for everything!

This project was made possible by the National Research Foundation (NRF) and the Centre for High Performance Computing (CHPC) Cluster at the Council for Scientific and Industrial Research (CSIR).

Contents

List of Figures	vii
List of Algorithms	xiii
List of Tables	xiv
1 Introduction	1
1.1 Summary of Research Domain	2
1.2 Problem Statement	4
1.3 Motivation	4
1.4 Objectives	6
1.5 Contributions	7
1.6 Dissertation Outline	8
2 Artificial Neural Networks	10
2.1 Biological Neuron	11
2.2 Artificial Neuron	12
2.2.1 Input	13
2.2.2 Weights	15
2.2.3 Net Input Signal	16
2.2.4 Biases	16
2.2.5 Activation Functions	17
2.2.6 Output	20
2.3 Artificial Neural Network	21
2.3.1 Applications	21
2.3.2 Architecture	22
2.3.3 Topology	23
2.3.4 Feedforward Neural Networks	23
2.4 Training	25
2.4.1 Supervised Learning	25
2.4.2 Error Functions	27
2.5 Summary	28

3	Heuristics	29
3.1	Optimisation	30
3.2	What is a heuristic?	31
3.3	Gradient-Based Heuristics	32
3.3.1	Backpropagation	32
3.3.2	Stochastic vs. Batch Training	35
3.3.3	Momentum	37
3.3.4	Nesterov Accelerated Gradients	38
3.3.5	Adaptive Gradients	39
3.3.6	Adaptive Learning Rate	40
3.3.7	Root Mean Squared Error Propagation	41
3.3.8	Adaptive Moments Estimation	41
3.4	Meta-Heuristics	42
3.4.1	Particle Swarm Optimisation	43
3.4.2	Differential Evolution	46
3.4.3	Genetic Algorithms	51
3.5	Summary	57
4	Hyper-Heuristics	58
4.1	Meta-Learning	60
4.2	What are Hyper-Heuristics?	60
4.3	Classification of Hyper-Heuristics	62
4.3.1	Source of Feedback	63
4.3.2	Heuristic Search Space	63
4.4	Summary	64
5	Probability	66
5.1	Overview of Probability	67
5.2	Conditional Probability and Independence	68
5.3	Two Laws of Probability for Multiple Events	69
5.4	Bayes' Theorem	69
5.5	Probability Distributions	71
5.5.1	Beta Probability Distribution	71
5.5.2	Dirichlet Probability Distribution	72
5.5.3	Bernoulli Probability Distribution	74
5.5.4	Binomial Probability Distribution	75
5.5.5	Categorical Probability Distribution	76
5.5.6	Multinomial Probability Distribution	77
5.6	Conjugate Priors	78
5.6.1	Binomial Likelihood	78

5.6.2	Categorical and Multinomial Likelihood	79
5.7	Bayesian Statistics	81
5.7.1	Frequentist vs. Bayesian Statistics	81
5.7.2	Bayesian Analysis	83
5.8	Summary	85
6	Bayesian Hyper-Heuristic	86
6.1	Overview	87
6.2	Architecture	88
6.3	Heuristic Pool	90
6.3.1	Heuristic Diversity	90
6.3.2	Heuristic Pool Size	91
6.3.3	Proxies	91
6.4	Entity Pool	93
6.4.1	Entity State	93
6.4.2	Population State	94
6.5	Performance Log	95
6.6	Credit Assignment Strategy	96
6.7	Selection Mechanism	98
6.7.1	Random Events	98
6.7.2	Independence	98
6.7.3	Bayes' Theorem	99
6.7.4	Predictive Model	99
6.7.5	Naïve Bayes	101
6.7.6	Numerical Stability	103
6.7.7	Mode Collapse	103
6.8	Optimisation Step	104
6.8.1	Concentration Parameters and Pseudo Counts	104
6.8.2	Maximum Likelihood Estimation	105
6.8.3	Maximum A Posteriori Estimation	108
6.9	Hyper-Parameters	110
6.9.1	Heuristic Pool	110
6.9.2	Population Size	111
6.9.3	Credit Assignment Strategy	111
6.9.4	Reselection Interval	112
6.9.5	Replay Window Size	112
6.9.6	Reanalysis Interval	112
6.9.7	Burn In	113
6.9.8	Discounted Rewards	113
6.9.9	Normalisation	113

6.9.10 Defaults	114
6.10 The BHH Algorithm	115
6.11 Summary	115
7 Methodology	117
7.1 Overview of Empirical Process	118
7.2 Datasets	119
7.2.1 Class Balancing	120
7.3 Models	120
7.4 Heuristics	121
7.5 BHH Baseline	121
7.6 Performance Measures	124
7.7 Stopping Conditions	125
7.8 Experiments	125
7.8.1 Behavioural Case Study	125
7.8.2 Standalone Heuristics	126
7.8.3 BHH Variants	126
7.9 Statistical Analysis	127
7.10 Implementation and Execution	128
7.11 Summary	128
8 Results	129
8.1 Overview	130
8.2 Behavioural Case Study	132
8.2.1 Performance Metrics	133
8.2.2 Concentration Parameters	134
8.2.3 Probability Distribution of Heuristic Selection Probabilities	136
8.2.4 Prior Heuristic Selection Probabilities	138
8.2.5 Posterior Heuristic Selection Probabilities	140
8.3 BHH vs. Low-Level Heuristics	142
8.4 Heuristic Pool	151
8.5 Population Size	155
8.6 Credit Assignment Strategy	160
8.7 Reselection Interval	164
8.8 Replay Window Size	169
8.9 Reanalysis Interval	173
8.10 Burn In	176
8.11 Normalisation	180
8.12 Discounted Rewards	184
8.13 Summary	188

9 Conclusion	191
9.1 Summary of Research Intent	192
9.1.1 Review of Problem Statement	192
9.1.2 Review of Research Motivation	192
9.1.3 Review of Research Objectives	192
9.2 Summary of Background Information	194
9.3 Summary of The Bayesian Hyper-Heuristic	194
9.4 Summary of Methodology	195
9.5 Summary of Results	197
9.5.1 Behavioural Case Study	198
9.5.2 BHH Baseline vs. Low-Level Heuristics	199
9.5.3 Heuristic pool	200
9.5.4 Population Size	200
9.5.5 Credit Assignment Strategy	200
9.5.6 Reselection Interval	201
9.5.7 Replay	201
9.5.8 Reanalysis Interval	201
9.5.9 Burn In	201
9.5.10 Normalisation	202
9.5.11 Discounted Rewards	202
9.6 Future Research Opportunities	202
9.7 Documentation and Data	205
9.8 Summary	205
Bibliography	206
A Acronyms	222
B Symbols	225
B.1 Chapter 2: Artificial Neural Networks	225
B.2 Chapter 3: Heuristics	228
B.3 Chapter 4: Hyper-Heuristics	233
B.4 Chapter 5: Probability	233
B.5 Chapter 6: Bayesian Hyper-Heuristic	238
B.6 Chapter 7: Methodology	241
B.7 Chapter 8: Results	241
C Datasets	243
D Statistical Analysis	244
D.1 BHH vs. Low-Level Heuristics	245

D.2	Heuristic Pool	248
D.3	Population Size	249
D.4	Credit Assignment Strategy	250
D.5	Reselection Interval	251
D.6	Replay Window Size	252
D.7	Reanalysis Internal	253
D.8	Burn In	254
D.9	Normalisation	255
D.10	Discounted Rewards	256
E	Derived Publications	257
	Index	258

List of Figures

2.1	The biological Neuron	11
2.2	The artificial neuron	12
2.3	The LReLU activation function	18
2.4	The sigmoid activation function	19
2.5	The hyperbolic tangent activation function	20
2.6	The results of softmax and argmax	22
2.7	A feedforward neural network	24
3.1	An illustration of <i>gradient descent</i> (GD) over various time steps showing the minimisation of the error with regards to weight value.	33
3.2	An illustration of <i>stochastic gradient descent</i> (SGD) fluctuations during training as taken from [125].	35
3.3	An illustration of <i>stochastic gradient descent</i> (SGD) with and without mo- mentum taken from [39].	37
3.4	An illustration of the weight update vector for <i>Nesterov accelerated gradients</i> (NAG) taken from [74].	38
3.5	An illustration of the uniform crossover operator as it applies to sexual recombination, resulting in two new offspring.	54
3.6	An illustration of the adapted uniform mutation operator as it applies to mutated offspring.	56
4.1	An illustration of the domain barrier that exists as a result of the separation between the low-level heuristics and the high-level heuristic as introduced by HHs.	61
4.2	A classification of HH approaches, according to two dimensions: (i) the source of feedback used during learning, and (ii) the nature of the heuristic search space.	62
5.1	A Venn-Diagram showing the proof of the additive law of probability for multiple events.	70
5.2	An illustration of the Beta probability distribution (left) [77] as well as the cumulative Beta probability distribution (right) [78] for various values of α and β	72

5.3	The <i>probability density functions</i> (PDFs) for the Dirichlet probability distribution over the 2-simplex. The concentration parameters, α , are varied. The values of the PDF are shown by the colour maps with contour lines at equal values as indicated in the colour bars [117].	74
5.4	An illustration of the coin-flip simulation for different sample sizes that show the convergence of the mean as per the <i>central limit theorem</i> (CLT).	75
5.5	The experimental outcomes for the mice-population experiments as were taken from [68]	82
5.6	An illustration of the prior and posterior probability distributions for the outcomes of the mice-population experiment, using a <i>Beta</i> prior, as was taken from [68].	83
6.1	An illustration of the architecture and high level components of the <i>Bayesian hyper-heuristic</i> (BHH).	89
6.2	The Beta probability distribution with varying α and β values.	114
7.1	Mapping of proxied heuristic state update operations as implemented by the BHH	123
8.1	The average train and test loss and accuracy plots over 30 epochs, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.	133
8.2	The average value of the concentration parameter α , are at indices 0, 6, 7, and 8 over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.	135
8.3	The average sampled heuristic selection probabilities, denoted θ , are at indices 0, 6, 7, and 8. The heuristic selection probabilities are sampled from the probability distribution, denoted $P(\theta \alpha)$, over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.	137
8.4	The average prior heuristic selection probabilities, $P(H \theta)$, are at indices 0, 6, 7, and 8. The prior heuristic selection probabilities are sampled from the probability distribution of heuristic selection probabilities, denoted by $P(\theta \alpha)$, over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.	139

8.5	The average calculated proportional posterior heuristic selection probabilities for heuristics (\mathbf{H}) at indices 0, 6, 7, and 8, given the application to entity e_0 and requiring a successful credit allocation, c_1 , from the <i>ibest</i> credit assignment strategy. The proportional posterior heuristic selection probabilities are calculated from the probabilistic model, denoted $P(\mathbf{H} \mathbf{E}, \mathbf{C}; \boldsymbol{\theta}, \phi, \psi)$, over 240 steps, and obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.	141
8.6	Descriptive plots for the average ranks of all low-level heuristics compared to three heuristic pool variants of the BHH baseline configuration, per dataset, across all independent runs and epochs.	145
8.7	Critical difference plots for the average ranks of all low-level heuristics compared to three heuristic pool variants of the baseline BHH, across all datasets, runs and epochs.	146
8.8	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the iris dataset over 30 epochs, illustrated in log scale.	147
8.9	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the car dataset over 30 epochs, illustrated in log scale.	148
8.10	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the bank dataset over 30 epochs, illustrated in log scale.	149
8.11	The train and test loss plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the fish toxicity dataset over 30 epochs, illustrated in log scale.	150
8.12	The train and test loss plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the parkinsons dataset over 30 epochs, illustrated in log scale.	150
8.13	Descriptive plots for the average ranks of the BHH with varying heuristic pools per dataset, across all independent runs and epochs.	153
8.14	Critical difference plots for the average ranks of the BHH with varying heuristic pools across all datasets, runs and epochs.	153
8.15	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the heuristic pool hyper-parameter on the abalone dataset over 30 epochs, illustrated in log scale.	154
8.16	The train and test loss plots for the experimental group comparing the performance of the BHH with different configurations of the heuristic pool hyper-parameter on the forest fires dataset over 30 epochs, illustrated in log scale.	155

8.17	Descriptive plots for the average ranks of BHH with varying population sizes per dataset, across all independent runs and epochs.	158
8.18	Critical difference plots for the average ranks of BHH with varying population sizes across all datasets, runs and epochs.	158
8.19	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the population size hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.	159
8.20	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the population size hyper-parameter on the student performance dataset over 30 epochs, illustrated in log scale.	159
8.21	Descriptive plots for the average ranks of the BHH with varying credit assignment strategies per dataset, across all independent runs and epochs. .	162
8.22	Critical difference plots for the average ranks of the BHH with varying credit assignment strategies across all datasets, runs and epochs.	162
8.23	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the credit assignment strategy hyper-parameter on the bank dataset over 30 epochs, illustrated in log scale.	163
8.24	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the credit assignment strategy hyper-parameter on the housing dataset over 30 epochs, illustrated in log scale.	163
8.25	Descriptive plots for the average ranks of the BHH with varying reselection interval values per dataset, across all independent runs and epochs.	167
8.26	Critical difference plots for the average ranks of the BHH with varying reselection interval values across all datasets, runs and epochs.	167
8.27	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the reselection interval hyper-parameter on the car dataset over 30 epochs, illustrated in log scale.	168
8.28	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the reselection interval hyper-parameter on the bike dataset over 30 epochs, illustrated in log scale.	168
8.29	Descriptive plots for the average ranks of the BHH with varying replay window sizes per dataset, across all independent runs and epochs.	171
8.30	Critical difference plots for the average ranks of the BHH with varying replay window sizes across all datasets, runs and epochs.	171

8.31	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the replay window size hyper-parameter on the diabetic dataset over 30 epochs, illustrated in log scale.	172
8.32	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the replay window size hyper-parameter on the fish toxicity dataset over 30 epochs, illustrated in log scale.	172
8.33	Descriptive plots for the average ranks of the BHH with varying reanalysis intervals per dataset, across all independent runs and epochs.	175
8.34	Critical difference plots for the average ranks of the BHH with varying reanalysis intervals across all datasets, runs and epochs.	175
8.35	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the reanalysis interval hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.	176
8.36	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the reanalysis interval hyper-parameter on the parkinsons dataset over 30 epochs, illustrated in log scale.	177
8.37	Descriptive plots for the average ranks of the BHH with varying burn in values per dataset, across all independent runs and epochs.	180
8.38	Critical difference plots for the average ranks of the BHH with varying burn in values across all datasets, runs and epochs.	180
8.39	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the burn in window size hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.	181
8.40	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the burn in window size hyper-parameter on the parkinsons dataset over 30 epochs, illustrated in log scale.	181
8.41	Descriptive plots for the average ranks of the BHH with normalisation toggled per dataset, across all independent runs and epochs.	184
8.42	Critical difference plots for the average ranks of the BHH with normalisation toggled across all datasets, runs and epochs.	184
8.43	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the normalisation hyper-parameter toggled on the abalone dataset over 30 epochs, illustrated in log scale.	185

8.44	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the normalisation hyperparameter toggled on the bike dataset over 30 epochs, illustrated in log scale.	185
8.45	Descriptive plots for the average ranks of the BHH with discounted rewards toggled per datasets, across all independent runs and epochs.	188
8.46	Critical difference plots for the average ranks of the BHH with discounted rewards toggled across all datasets, runs and epochs.	188
8.47	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the discounted rewards hyperparameter toggled on the wine quality dataset over 30 epochs, illustrated in log scale.	189
8.48	The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the discounted rewards hyperparameter toggled on the housing dataset over 30 epochs, illustrated in log scale.	189

List of Algorithms

1	The pseudo-code algorithm for the generic <i>gradient descent</i> (GD) heuristic.	34
2	The pseudo-code algorithm for the gbest PSO heuristic.	46
3	The pseudo-code algorithm for the binomial crossover technique for DE. . .	49
4	The pseudo-code algorithm for the exponential crossover technique for DE.	49
5	The pseudo-code for the general DE heuristic.	50
6	The pseudo-code for the generic EC heuristic.	52
7	The pseudo-code for the uniform crossover operator as used by <i>genetic algorithms</i> (GAs).	55
8	The pseudo-code for the uniform mutation operator as used by GAs.	55
9	The pseudo-code for the implementation of the <i>Bayesian hyper-heuristic</i> (BHH)	116

List of Tables

6.1	An example of a mapping of proxied state update operation maintained by the BHH.	92
6.2	An example of the performance log implemented by the BHH, showing the first five entities, their allocated heuristics and their resulting performance measurements for the first step of the training process.	96
6.3	Credit assignment strategy output table showing <i>ibest</i> credit assignment for the first five entities and their selected heuristics for step 1 of the training process.	97
7.1	Classification datasets	119
7.2	Regression datasets	119
7.3	Model configurations	120
7.4	Low-level heuristics and their hyper-parameter configurations.	122
7.5	The BHH baseline configuration as it is used in the empirical study.	124
7.6	BHH variants and their configuration	126
8.1	Empirical results showing test loss and statistics for different low-level heuristics compared to three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs, measured at the last epoch.	143
8.2	Empirical results showing normalised average rank and statistics for different low-level heuristics compared to three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs and epochs.	144
8.3	Empirical results showing average test loss and statistics for all heuristic pool configurations used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	151
8.4	Empirical results showing average rank and statistics for all heuristic pool configurations used by the BHH across multiple datasets, for all independent runs and epochs.	152

8.5	Empirical results showing average test loss and statistics for all population size configurations used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	156
8.6	Empirical results showing average rank and statistics for different population sizes used by the BHH across multiple datasets, for all independent runs and all epochs.	157
8.7	Empirical results showing average test loss and statistics for different credit assignment strategies used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	160
8.8	Empirical results showing average rank and statistics for different credit assignment strategies used by the BHH across multiple datasets, for all independent runs and epochs.	161
8.9	Empirical results showing average test loss and statistics for different reselection intervals used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	164
8.10	Empirical results showing average rank and statistics for different reselection intervals used by the BHH across multiple datasets, for all independent runs and epochs.	165
8.11	Empirical results showing average test loss and statistics for different replay window sizes used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	169
8.12	Empirical results showing average rank and statistics for different replay window sizes used by the BHH across multiple datasets, for all independent runs and epochs.	170
8.13	Empirical results showing average test loss and statistics for different reanalysis intervals used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	173
8.14	Empirical results showing average rank and statistics for different reanalysis intervals used by the BHH across multiple datasets, for all independent runs and epochs.	174
8.15	Empirical results showing average test loss and statistics for different burn in window sizes used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	177
8.16	Empirical results showing average rank and statistics for different burn in window sizes used by the BHH across multiple datasets, for all independent runs and epochs.	178
8.17	Empirical results showing average test loss and statistics for normalisation toggled by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	182

8.18	Empirical results showing average rank and statistics for normalisation toggled by the BHH across multiple datasets, for all independent runs and epochs.	183
8.19	Empirical results showing average test loss and statistics for discounted rewards toggled by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.	186
8.20	Empirical results showing average rank and statistics for discounted rewards toggled by the BHH across multiple datasets, for all independent runs and epochs.	187
D.1	ANOVA - Rank - BHH vs. Low-Level Heuristics	245
D.2	Kruskal-Wallis - BHH vs. Low-Level Heuristics	245
D.3	Post Hoc Comparisons - BHH vs. Low-Level Heuristics - Part A	246
D.4	Post Hoc Comparisons - BHH vs. Low-Level Heuristics - Part B	247
D.5	ANOVA - Rank - BHH Variant: Heuristic Pool	248
D.6	Post Hoc Comparisons - BHH Variant: Heuristic Pool	248
D.7	Kruskal-Wallis - BHH Variant: Heuristic Pool	248
D.8	ANOVA - Rank - BHH Variant: Population Size	249
D.9	Post Hoc Comparisons - BHH Variant: Population Size	249
D.10	Kruskal-Wallis - BHH Variant: Population Size	249
D.11	ANOVA - Rank - BHH Variant: Credit Assignment Strategy	250
D.12	Post Hoc Comparisons - BHH Variant: Credit Assignment Strategy	250
D.13	Kruskal-Wallis - BHH Variant: Credit Assignment Strategy	250
D.14	ANOVA - Rank - BHH Variant: Reselection Interval	251
D.15	Post Hoc Comparisons - BHH Variant: Reselection Interval	251
D.16	Kruskal-Wallis - BHH Variant: Reselection Interval	251
D.17	ANOVA - Rank - BHH Variant: Replay Window Size	252
D.18	Post Hoc Comparisons - BHH Variant: Replay Window Size	252
D.19	Kruskal-Wallis - BHH Variant: Replay Window Size	252
D.20	ANOVA - Rank - BHH Variant: Reanalysis Interval	253
D.21	Post Hoc Comparisons - BHH Variant: Reanalysis Interval	253
D.22	Kruskal-Wallis - BHH Variant: Reanalysis Interval	253
D.23	ANOVA - Rank - BHH Variant: Burn In	254
D.24	Post Hoc Comparisons - BHH Variant: Burn In	254
D.25	Kruskal-Wallis - BHH Variant: Burn In	254
D.26	ANOVA - Rank - BHH Variant: Normalisation	255
D.27	Post Hoc Comparisons - BHH Variant: Normalisation	255
D.28	Kruskal-Wallis - BHH Variant: Normalisation	255
D.29	ANOVA - Rank - BHH Variant: Discounted Rewards	256
D.30	Post Hoc Comparisons - BHH Variant: Discounted Rewards	256

D.31 Kruskal-Wallis - BHH Variant: Discounted Rewards	256
---	-----

Chapter 1

Introduction

“It is better to solve one problem five different ways, than to solve five problems one way.”

- George Polya

Machine learning (ML) is one of the most popular fields of research in *artificial intelligence* (AI) studies today. In recent years, ML research has seen some notable achievements in academia [58, 61, 99, 173], as well as the industry at large [103, 148, 149, 186]. ML research has grown tremendously over the past decade with successes like AlphaGo, which set new standards for AI capabilities by beating the world’s best Go player, Lee Sedol, 4-1 [149].

Over the past few years, modern hardware capabilities have improved to the point where workloads in the field of ML that were previously computationally infeasible, are now possible. One such sub-field of ML is *artificial neural networks* (ANNs). ANNs can generally be described as well-organised structures of mathematical computation and are inspired by the biological brain [45]. ANNs can be trained, which is the equivalent of “learning” from data. Learning gives rise to the ability to apply some form of decision making. Decision making as an ability provides a wide-range of applications from healthcare to finance and yields great interest in the field. With the improvement of hardware came an influx of ML researchers that focused their attention on training of ANNs.

Feedforward neural networks (FFNNs) are specific types of ANNs [133]. A popular field of focus for studying ANNs is the process by which these models are trained. The most common way of training FFNNs is *supervised learning*, which is a training technique that involves exposing the ANNs with input data and comparing the produced output data to that of predefined target data. Training of ANNs is seen as an optimisation problem. The ANN maintains a set of parameters, referred to

as “weights” and “biases”. A search algorithm known as a *heuristic* [124] is used to assign optimal values to the parameters of the ANN, such that a specified objective function is minimised. The research presented in this dissertation focuses on the development of a specific type of heuristic, called a *hyper-heuristic* (HH) to train FFNNs in a supervised learning approach.

This chapter provides the reader with a brief overview of the problem domain and outlines the research problem, objectives and purpose. The remainder of the chapter is presented as follows:

- **Section 1.1** provides the reader with brief summary of the research domain.
- **Section 1.2** outlines the research problem being addressed.
- **Section 1.3** provides a motivation for this research.
- **Section 1.4** presents the reader with the research objectives and outlines the goals of this dissertation.
- **Section 1.5** outlines the contributions of this research to the field.
- **Section 1.6** provides a full dissertation outline.

1.1 Summary of Research Domain

This section provides the reader with a brief summary of the research field and outlines key concepts that contribute to the topics discussed in this dissertation.

Although the landscape of what can be solved using ANNs today is extensive, there still exists no single model that can be generalised to solving multiple problem classes, across multiple problem domains. ANN training algorithms mostly yield problem specific solutions. This means that a particular approach that works well for one domain or problem class, often does not necessarily work for another. This problem is known as the *no free lunch theorem* (NFL) [178].

The performance and capabilities of ANNs is largely influenced by the learning process used. The learning process consists of multiple components. These include the type of underlying optimisation algorithm used, how the model parameters are initialised, the hyper-parameters used and the constraints of learning such as allowed search space and boundaries. Each of these elements influences how much a particular learning technique might focus on a particular solution (exploitation), in comparison to seeking out novel solutions (exploration) during training.

Techniques exist to dynamically balance the trade-off between exploration and exploitation during the search process. An example of such a technique is to dynamically adjust and learn the heuristic *hyper-parameters* as part of the learning process. This field of study is known as meta-learning [57]. Meta-learning of heuristic hyper-parameters as applied in the training of ANNs has shown to yield good generalisation results [80, 174].

The dynamic nature of the learning process leads towards the idea that ML models could be trained in a way such that the learning process and learning mechanism applied are not statically defined, but rather dynamic and under the control of some mechanism.

A recent suggestion related to the field of meta-learning is to dynamically select and/or adjust the heuristic used throughout the training process. This approach focuses on the hybridisation of learning paradigms. The main concept behind this paradigm is that the learning process is dynamic and problem specific. A particular learning technique might work well for one problem and not for another. At the same time, a particular learning technique might work well for a particular part of the search landscape, but not for another. By dynamically combining the best of different learning paradigms throughout the learning process, a trade-off can be made between exploration and exploitation as is required.

One such form of hybridisation of learning paradigms is referred to as *heterogeneous learning approaches*. Heterogeneous learning approaches make use of different *search behaviours* by selecting from a behaviour pool. Heterogeneous approaches have shown to balance the trade-off between exploration and exploitation [116].

A step further in the concept of hybridisation of learning paradigms is that of hybridisation of different *heuristics* as they are applied to some optimisation problem [20]. These methods are referred to as *hyper-heuristics* (HHs) and focus on finding the best heuristic in *heuristic space* to solve a specific problem. One such form of HH is a population-based approach that guides the search process by automatically selecting heuristics from a heuristic-pool to be applied to a collection of different candidate solutions in the solution-space. This collection of candidate solutions are referred to as a *population* of *entities*, where each *entity* is a single candidate solution to the problem being optimised. Population-based HHs implement a strategy where multiple heuristics work together to solve a problem. This technique requires a mechanism that selects a specific heuristic to be applied to a specific candidate solution.

Finding the best heuristic to use is non-trivial and some *selection strategy* must be used to select the best heuristic. HHs that implement such a selection strategy

are referred to as *selection HHs*. The term *selection* refers to the ability of the HH to select the best heuristic from a pool of low-level heuristics.

One specific type of selection HH is called a *multi-method population-based meta-heuristic* [171]. The term *multi-method* refers to the incorporation of different low-level heuristics, with different search behaviours, into the heuristic space. The term *population-based* refers to the utilisation of a population of entities that represent candidate solutions. Finally, the term *meta-heuristics* refers to the HH as a heuristic that does not have any domain knowledge and only makes use of information from the search process.

Grobler [64] mentioned that HHs have been shown to solve a number of problems including bin-packing, examination timetabling, production scheduling, the travelling salesman problem, vehicle routing problem and many more.

1.2 Problem Statement

Many different heuristics have been developed and used to train FFNNs [66, 111, 131]. Each of these heuristics has different search behaviours, characteristics, strengths and weaknesses. Finding the best heuristic to train the FFNN is required in order to yield optimal results. This process is often non-trivial and could be a time-consuming exercise. Consider that selection of the best heuristic as applied to optimisation problems, such as training FFNNs, is problem specific [3, 38, 126].

Careful, systematic selection is thus required to find and select the best heuristic to train FFNNs. In the past, researchers selected the best heuristic by trial and error. A set of heuristics and carefully selected hyper-parameters would be implemented, followed by an empirical test to evaluate the performance of each heuristic for a given problem domain [127]. In this way, researchers were able to determine which heuristics and which hyper-parameters performed well for different problems. However, this approach is problematic, because it is time-consuming and laborious.

1.3 Motivation

A process is required to alleviate the burden of having to exhaustively test each implementation of heuristic and hyper-parameters, for every problem being optimised.

In Section 1.2 above, it is mentioned that finding the best heuristic to train FFNNs is a timely and tedious process. A modern approach is to use HHs to automate the process of selecting the best heuristic when applied to some optimisation problem. The best heuristic might not be a single heuristic, but rather a hybridisation of

heuristics [127].

In the general context of optimisation, many different types of HHs have been implemented and applied to many different problems. Some notable examples include the simulated annealing-based HH by Dowsland et al. [37], the tabu-search HH of Burke et al. [19], the heterogeneous meta-hyper-heuristic by Grobler et al. [65] and work done by Van der Stockt et al. [171] on the analysis of selection hyper-heuristics for population-based *meta-heuristics* (MHs) in real-valued dynamic optimisation. However, research on the application of HHs in the context of FFNN training is scarce. Nel [115] provides the first research in this field, applying a HH to FFNN training.

Training of FFNNs is seen as a computational search problem. A *hyper-heuristic* (HH), as defined by Burke et al. [19], is “a search method or learning mechanism for selecting or generating heuristics to solve computational search problems”. HHs is a field of research aimed at the automated selection, combination, adaptation or generation of multiple lower-level heuristics to efficiently solve computational search problems.

Grobler et al. [65] mention that the focus of HHs is on automating the development of the *learning mechanism* used to find the best heuristic to obtain an appropriate solution to an optimisation problem. Grobler et al. explain that HHs employ a high-level heuristic that focuses on finding the best low-level heuristic in heuristic space that could include *single-method* and *multi-method* optimisation algorithms (heuristics). HHs relieve the burden of having to select the best heuristic to use by trial and error. Furthermore, HHs can generally be applied to multiple problems given that the set of low-level heuristics include heuristics that have the potential to solve the problem at hand [19]. Automation of the heuristic selection process and the general application to a wider range of problems are characteristics of HHs that can be beneficial when applied in the context of training FFNNs.

Note that training of FFNNs using HHs is not to be confused with the training of FFNNs using *ensemble networks* or *query by committees*. Pappa et al. [123] describe ensemble networks as a combination method in meta-learning whereby multiple ANNs are jointly used to solve a problem. Each member network in the ensemble is trained using a specified heuristic, and generally this heuristic is applied to the same member throughout the training process. The results of all of the member networks are then joined together using some consensus mechanism [184]. This mechanism could be weighted averages, majority voting or weighted voting. Ensemble networks do not search through the heuristic space to find the best training algorithm for each member, while HHs do.

This research takes a particular interest in developing a selection [HH](#) that makes use of probability theory and Bayesian statistical concepts to guide the heuristic selection process. This research develops the novel *Bayesian hyper-heuristic* ([BHH](#)), a new high-level heuristic that utilises a statistical approach, referred to as *Bayesian analysis*, which combines prior information with new evidence to the parameters of a selection probability distribution. This selection probability distribution is the mechanism by which the [HH](#) selects appropriate heuristics to train [FFNNs](#) during the training process. This process takes place dynamically and during the training process (online learning).

1.4 Objectives

The main objective of this research is focused on developing a novel *Bayesian hyper-heuristic* ([BHH](#)) selection mechanism in a [HH](#) framework that can be used to train [FFNNs](#). In order to reach this objective, the following sub-objectives are defined.

- Conduct a literature study on [ANNs](#) in order to provide the necessary background information of the [AN](#), [FFNNs](#) and the training process.
- Conduct a literature study on different types of heuristics that have been used to train [FFNNs](#). The literature study will provide the necessary background information to understand how different heuristics can be used in the set of heuristics to be selected to train the [FFNN](#).
- Conduct a literature study on meta-learning and [HHs](#) in order to provide the required background information necessary to propose and develop a new high-level heuristic and selection mechanism.
- Conduct a literature study on probability theory and Bayesian statistics such as Bayesian inference and Bayesian analysis to provide the necessary background information required to understand how Bayesian approaches can be used as a learning technique.
- Develop a novel [BHH](#) selection mechanism for a [HH](#) that makes use of Bayesian statistics to guide the [HH](#) search process while training [FFNNs](#) on different problems.
- Conduct an empirical study to show that the developed [BHH](#) can effectively be used to train [FFNNs](#).

- Conduct an empirical study to investigate the behavioural characteristics of the BHH as it is used to train FFNNs on an example problem.
- Conduct an empirical study and critically evaluate the performance of the developed BHH compared to individual heuristics in the heuristic space as they are used to train FFNNs on a number of different problems.
- Conduct an empirical study that investigates variations of the BHH and the effects of design decisions and hyper-parameters on the search process.
- Provide a statistical analysis of the results obtained from the empirical study.

1.5 Contributions

The results obtained from this research contribute to the field of study in the following ways.

- A novel heuristic selection operator is used that focuses on using Bayesian statistics to calculate the probability that a heuristic should be selected in order to efficiently train FFNNs. The resulting HH is referred to as the *Bayesian hyper-heuristic* (BHH).
- The results of the empirical study show with statistical significance and certainty that the BHH performs generally well on multiple problems. It is shown that, for each problem, the BHH performance is comparable to the best low-level heuristics included in the heuristic selection pool.
- The results of the empirical study show that the BHH is able to select the best heuristic to train FFNNs in general. This relieves researchers from the burden of having to do this selection process manually through trial and error.
- The results of the empirical study show that the BHH, given a diverse set of lower-level heuristics, will generally produce good results when applied to multiple problems at the same time.
- Finally, the results of the empirical study show that the BHH is capable of utilising *a priori*¹ knowledge in which a predefined selection bias is used for heuristics that are known to be well suited for certain problems.

¹Latin word, meaning “from what comes before”.

1.6 Dissertation Outline

The remainder of this dissertation is structured as follows:

- **Chapter 2** provides a literature study on [ANNs](#) and the various components that make up an *artificial neuron* ([AN](#)). The focus is on training of [FFNNs](#). It is shown how training of [FFNNs](#) is seen as an optimisation problem.
- **Chapter 3** provides details on various types of heuristics and [MHs](#) that have been used to train [FFNNs](#). A literature study is done to provide details on the search behaviours, application and implementation of each heuristic in the context of training [FFNNs](#).
- **Chapter 4** presents a literature study on the details of [HHs](#) and meta-learning in general. A discussion follows on the current landscape of [HH](#) research and a review of different selection approaches for [HHs](#) is conducted. It is shown how [HHs](#) are suitable for [FFNN](#) training.
- **Chapter 5** presents a literature study on probability theory. Probability distributions and conjugate priors are discussed in detail. The chapter concludes with a detailed discussion on Bayesian statistics, specifically focusing on Bayesian inference and analysis.
- **Chapter 6** presents the developed [BHH](#). It is shown how the [BHH](#) is implemented as a selection mechanism in the context of a [HH](#) framework. The [BHH](#) is shown to implement a Naïve Bayes classifier. The probabilistic model that is implemented is derived and discussed in detail. Finally, the chapter concludes with a detailed discussion on how Bayesian analysis is used to guide the heuristic selection process. The update step (training step) for prior probability concentration parameters is derived and discussed in detail. The [BHH](#) algorithmic implementation, variants and suggested application are presented as well.
- **Chapter 7** presents a detailed description of the empirical process and the setup of each experiment. It discusses the datasets used, the [FFNN](#) architecture and topology used, heuristics that are used, the configuration of hyper-parameters, initialisation techniques used and performance measures used. Finally, discussions follow on how the results are analysed. This includes a discussion on the method used to determine statistical significance.

- **Chapter 8** provides and discusses the results of the empirical study in detail. A baseline comparison is done by comparing the performance of the **BHH** to that of all the individual lower-level heuristics on all datasets. These datasets include classification and regression problems of various sizes and complexity. Detailed results are presented on the performance of the **BHH** as a selection mechanism for **HHs** and a brief comparison is made to other selection mechanisms. Finally, results are presented on the effects of the hyper-parameters of the **BHH** on the training process. Discussions follow on how the **BHH** is shown to automate the selection of the best heuristic during the training of **FFNNs**, as applied to a range of problems, alleviating the burden on researchers to apply traditional trial and error approaches.
- **Chapter 9** summarises the research done in this dissertation along with a brief overview of the findings made throughout the research process. A review of the research goals are given and suggestions for future research are made.

This dissertation is accompanied by a full index given at page 258 along with the following appendices:

- **Appendix A** provides a list of the important acronyms used or newly defined in the course of this work, as well as their associated definitions.
- **Appendix B** lists and defines the mathematical symbols used in this work, categorised according to the relevant chapter in which they appear.
- **Appendix C** provides details on the datasets used for the empirical analysis.
- **Appendix D** provides details on the outcomes of the statistical analyses.
- **Appendix E** lists the publications derived from this work.

To best view the illustrations, tables and figures presented throughout this dissertation, it is recommended that the dissertation be viewed in colour.

Chapter 2

Artificial Neural Networks

“Men ought to know that from the brain, and from the brain only, arise our pleasures, joy, laughter and jests, as well as our sorrows, pains, griefs, and tears.”

- Hippocrates

The human brain is one of the most complicated biological organs in nature. It gives us the ability to think and learn. The field of [ML](#) has taken a lot of inspiration from the biological brain which lead to the development of [ANNs](#) [139]. [ANNs](#) are used throughout this dissertation as the *model* to be trained using [HFs](#). The purpose of this chapter is to provide the necessary background information needed on [ANNs](#) and is structured as follows:

- **Section 2.1** gives background information on the [BN](#).
- **Section 2.2** introduces the [AN](#). Brief discussions follow on input, weights and biases, net input signal, activation functions, and output.
- **Section 2.3** introduces the [ANN](#). Brief discussions follow on [ANN](#) architecture, topology, and [FFNNs](#).
- **Section 2.4** provides details on the training process, supervised learning, training sets, stopping conditions, performance measures and error functions.
- **Section 2.5** provides a brief summary of the chapter.

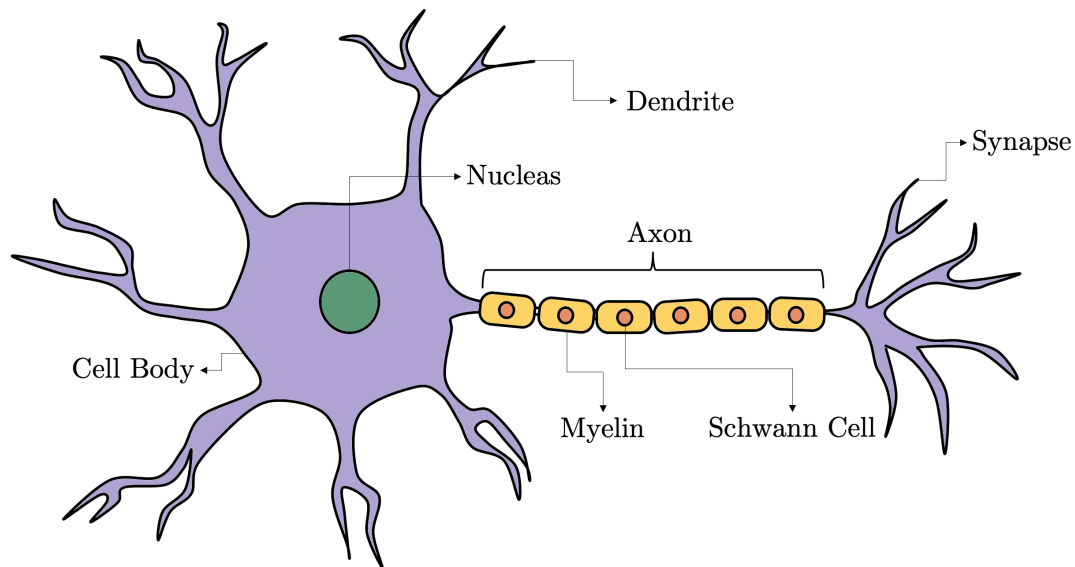


Figure 2.1: An illustration of the biological Neuron.

2.1 Biological Neuron

This section introduces the [BN](#) and provides the necessary background information that show how the [BN](#) has inspired the development of the [AN](#).

The biological neural systems are made up of microscopic nerve cells called neurons [85]. Figure 2.1 illustrates a single [BN](#). The main components of the [BN](#) include the cell body, *dendrites* and the *axon*. *Neural networks* ([NNs](#)) are formed through the connections between the axons and dendrites of various neurons. This is known as *synaptogenesis* [81]. Such a connection is referred to as a *synapse*. Communication takes place, through the synapse, by electro-chemical pulse and is often referred to as an *activation* or *action potential*. Communication signals propagate from the dendrites, through the cell body to the axon of a neuron, provoking a signal in the post-synaptic neuron [45]. The greater the connection between two neurons, the stronger the communication. Kennedy [91] defines stronger synapses as ones that contribute more depolarisation to the neural membrane upon activation than weaker ones. Stronger synapses have a higher probability of generating an action potential in the post-synaptic neuron. During activation, the pre-synaptic neuron release neurotransmitters that bind to the post-synaptic neuron. The frequent release of these molecules cause the synapse to grow. Connections that grow over time yield stronger signals (learning), while connections that are weak propagate low intensity signals and vanish over time (forgetting). The ability of synapses to strengthen and weaken over time is known as *synaptic plasticity* [81].

2.2 Artificial Neuron

This section introduces the [AN](#). Brief discussion follow on the various components that make up the [AN](#).

The [AN](#) implements a non-linear mapping from \mathbb{R}^I to \mathbb{R}^T , usually in the ranges $[0, 1]$ or $[-1, 1]$, depending on the activation function used [\[45\]](#) and is given as

$$f_{AN}: \mathbb{R}^I \rightarrow \mathbb{R}^T \quad (2.1)$$

where f_{AN} is the mapping function produced by the [AN](#), I is the total number of dimensions of the input in real-number space (\mathbb{R}), and T is the total number of dimensions of the target (desired output) in real-number space.

The [AN](#) implements various components and is illustrated in Figure 2.2.

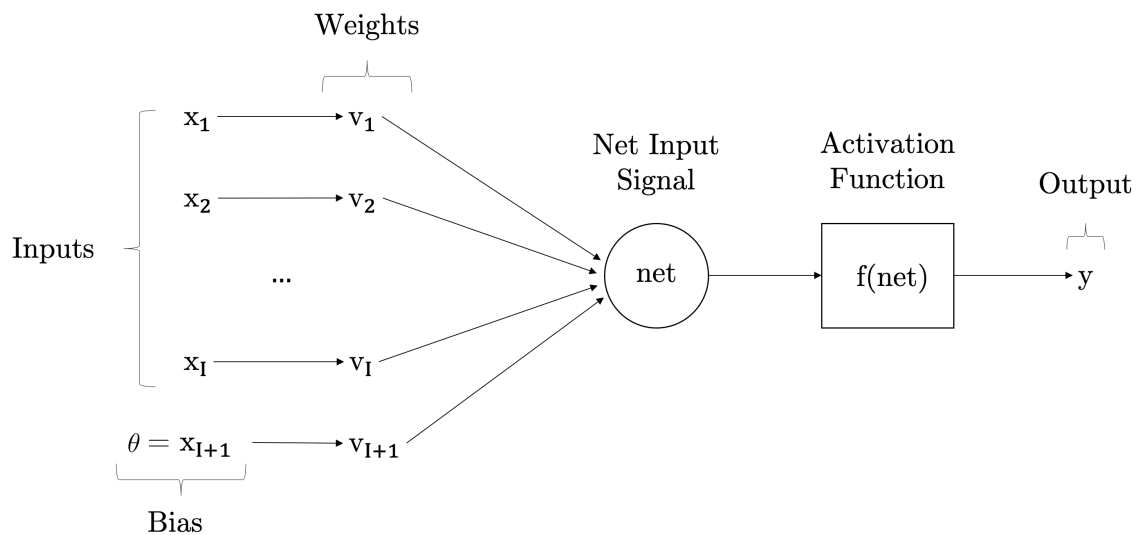


Figure 2.2: An illustration of the artificial neuron.

Each of the components of the [AN](#) is inspired by some element of the [BN](#). Brief descriptions of these are given as follows:

- **Input:** The input models the activations received from the pre-synaptic neuron or from some environment sensor. Input is represented by the input vector \mathbf{x} , in Figure 2.2.
- **Weights:** The weights model the synapses and connection strengths. Weights are represented by the weight vector \mathbf{v} , in Figure 2.2.
- **Net input signal:** The net input signal models the net resulting activations from all connected pre-synaptic neurons or environment sensors. The net input signal is represented by net , in Figure 2.2.

- **Biases:** The biases model a mechanism introduced to influence the strength of the activation (output signal) of the AN. The bias is represented by θ , in Figure 2.2.
- **Activation function:** The activation function models the action potential of the BN and is based on the net input signal. The activation function is represented by f , in Figure 2.2.
- **Output:** The output models the activation by the post-synaptic neuron. Output is represented by the output vector \mathbf{y} , in Figure 2.2.

The following sections provide a detailed discussion on each of the above mentioned components.

2.2.1 Input

Input signals are I -dimensional vectors of numerical values that are obtained either through some environment sensor or from other ANs. Input signals are often referred to as *features* or *independent variables* [54]. Throughout this dissertation, a single input vector is referred to as a *pattern*. Input data must first be pre-processed before it is presented to the AN. Input pre-processing techniques are presented in the following sections.

Input Pre-Processing

Input pre-processing improves the training process and contributes to the success of the practical application of the ANN [97]. The primary purpose of input pre-processing is to modify the input data so that it can better match predicted output data. There are many techniques to consider when pre-processing input data. These techniques include methods to encode data to certain formats, to scale to the correct ranges and to deal with incomplete, invalid or irrelevant data. For the purposes of this dissertation, *data encoding* and *normalisation* techniques are considered.

Encoding

For qualitative data, the class labels have to be converted from textual to numerical representations. In general, class labels are encoded as numerical vectors [17]. One such encoding technique is referred to as *one-hot encoding*. Harris and Harris [71] describe one-hot encoding as a group of vector elements where the classes are represented with a single high element (represented by 1) and all the others low (represented by 0). Each element y_k , where k is the k -th element of the encoded

vector \mathbf{y} , uniquely refers to a class. An activation, represented by a value of 1 at a given index thus represents a class associated with that index.

Normalisation

Numerical data must be normalised and scaled appropriately for the AN. In general, input data is normalised to a range that is appropriate for the activation function being used by the AN. Normalisation and scaling yields input data that is comparable to the output produced by the AN. For the purposes of this dissertation, the min-max scaler [1] and the standard score scaler [84] are considered.

Min-Max Scaler

The min-max scaler, also called *unity-based normalisation*, scales values to the range $[0, 1]$. The min-max scaler is used as a pre-processing technique for target values when the *sigmoid* activation function is used. The min-max scaler is given as

$$x'_{i,p} = \frac{x_{i,p} - x_{i_{min}}}{x_{i_{max}} - x_{i_{min}}} \quad (2.2)$$

where $x'_{i,p}$ is the normalised form of $x_{i,p}$, the i -th dimension of the input vector \mathbf{x}_p , $p \in \{1, 2, \dots, P\}$, where P is the total number of input patterns, $x_{i_{min}}$ and $x_{i_{max}}$ are respectfully the minimum and maximum values of the i -th dimension for all input vectors \mathbf{x}_p .

Standard Score Scaler

The standard score scaler, also known as the *z-score scaler*, scales values by subtracting the mean and scaling to the unit variance of each dimension i for all input patterns \mathbf{x}_p , $p \in \{1, 2, \dots, P\}$. The standard score scaler is used as a pre-processing technique for target values when the *hyperbolic tangent* activation function is used. The standard score scaler is given as

$$x'_{i,p} = \frac{x_{i,p} - \mu_i}{\sigma_i} \quad (2.3)$$

where μ_i and σ_i^2 are respectfully the mean and unit variance of the i -th dimension of all input vectors \mathbf{x}_p .

2.2.2 Weights

The connection strength that synapses in the [BN](#) have is modelled in the [AN](#) as weight vectors of numerical values associated with each dimension of the input. Weights can either dampen or strengthen the input by some negative or positive numerical value. Changes in the weight associated with a feature changes the influence that that particular feature has on the predicted output. Finding the correct value for each weight, such that the [AN](#) yields optimal output, is an optimisation problem [\[161\]](#). Research has shown that weight initialisation plays an important role in the efficient training of [ANNs](#) [\[162\]](#).

Weight Initialisation

Weight initialisation is the process by which candidate solutions (represented by the weights of the [AN](#)) to the problem are “placed” in the search space. Weight initialisation influences the speed of convergence, the probability of convergence and the generalisation capabilities of [ANNs](#) [\[49\]](#). Finding the optimal initialisation values for weights is non-trivial and can be seen as another optimisation problem [\[46, 152, 180\]](#).

Weight initialisation is dependent on the activation function used. If weights are initialised as values that are too small, the vanishing gradients problem can occur [\[70\]](#). If weights are initialised as values that are too big, output of the activation function would not be in the active range. This leads to unit saturation, and exploding gradients can occur [\[70\]](#).

Many different weight initialisation techniques have been developed [\[46\]](#). For the purposes of this dissertation, focus is put on random uniform sampling, *Glorot uniform* (*Xavier*) sampling and *Glorot normal* sampling [\[58\]](#). Brief discussions on each of these weight initialisation techniques are presented as follows.

Random Uniform Sampling

Random uniform sampling initialises weights uniformly in the range $[\omega_{min}, \omega_{max}]$, written as $\omega_i \sim U(\omega_{min}, \omega_{max})$, where the ω_{min} and ω_{max} are respectfully the lower and upper bounds of the uniform distribution. Suggested parameter values are $(-1, 1)$ or $(-0.5, 0.5)$ [\[119\]](#).

Glorot Uniform Sampling

Glorot uniform sampling is a specialisation of random uniform sampling, whereby $\omega_{max} = \sqrt{\frac{6}{f_{anin} + f_{anout}}}$ and f_{anin} is the number of input neurons to the weight vector

and *fanout* is the number of output neurons from the weight vector.

Glorot Normal Sampling

Glorot normal sampling initialises weights by sampling from a truncated normal distribution centred on a mean of 0 and with $\sigma = \sqrt{\frac{2}{I+K}}$, where σ is the standard deviation of the distribution. I and K are respectfully the number of input and output units in the weight vector. Glorot normal sampling has been shown to decrease training time [58].

2.2.3 Net Input Signal

ANs accumulate the net resulting input signal from all input dimensions into a value called the *net input signal*, expressed as *net*. This signal is passed to the activation function, which provokes an artificial action potential in the AN.

A common way by which the net input signal is calculated is by means of *summation units* (SUs) [45], which compute the net input signal as the weighted sum of all input signals and is given as

$$net = \sum_{i=1}^I x_i v_i \quad (2.4)$$

where x_i is the i -th dimension of the input vector \mathbf{x} , and v_i is the i -th dimension of the weight vector \mathbf{v} , associated with the given input dimension.

2.2.4 Biases

A bias/threshold term θ is introduced to help translate the output of the activation function [10]. The value of θ can be learned during the training process along with the weights of the ANN. In order to simplify equations, the input and weight vectors are augmented such that the input and hidden layers have an additional neuron/unit, called the *bias unit* [45]. The net input signal can then be rewritten to consider the bias unit, leading to an augmented net input signal.

Augmented Net Input Signal

The augmented net input signal, that includes the bias unit, has the form $net' = net - \theta$, with $\theta = x_{i+1}v_{i+1}$. A constant value $x_{i+1} = -1$ can be used, meaning that the weight associated with the bias unit, v_{i+1} , is optimised along with the other weights during the optimisation process. The net input signal, as given in Equation (2.4), then changes as

$$\begin{aligned}
net' &= net - \theta \\
&= \sum_{i=1}^I x_i v_i - \theta \\
&= \sum_{i=1}^I x_i v_i + x_{I+1} v_{I+1} \\
&= \sum_{i=1}^{I+1} x_i v_i
\end{aligned} \tag{2.5}$$

2.2.5 Activation Functions

An activation function is used to model the action potential of the AN [75, 185]. The activation function takes as a parameter, the augmented net input signal. When the output produced by the activation function surpasses some threshold value τ , we consider that neuron to have “fired” an output signal. Activation functions thus model *phase shift*. In the context of classification problems, activation functions form decision boundaries between classes. In the context of regression problems, activation functions try to approximate some function that maps the input data to some target data.

In general, activation functions produce a non-linear mapping of \mathbb{R}^I to the range $[0, 1]$ or $[-1, 1]$ as shown in Equations (2.6) and (2.7) below.

$$f_{AN} : \mathbb{R}^I \rightarrow [0, 1] \tag{2.6}$$

$$f_{AN} : \mathbb{R}^I \rightarrow [-1, 1] \tag{2.7}$$

Many different activation functions have been developed [88]. For the purposes of this dissertation, focus is put on the *rectified linear unit* (ReLU) [87, 113], the *leaky rectified linear unit* (LReLU) [105], the sigmoid [100] and the hyperbolic tangent [104] activation functions.

Rectified Linear Unit

The *rectified linear unit* (ReLU) activation function is an activation function defined in the positive part of its argument and is given as

$$f(x) = x^+ = \max(0, x) \tag{2.8}$$

An advantage of ReLU is that it is not susceptible to the vanishing gradients

problem [179] which occurs when gradients in the first layers of a multi-layer ANN approach zero, and have no effect on the training process.

Leaky Rectified Linear Unit

The *leaky rectified linear unit* (LReLU) activation function is a variant of the ReLU activation function that avoids zero gradients in the negative part of its argument by introducing a scaling parameter, $\alpha > 0$ [179]. Similar to ReLU, LReLU is not susceptible to the vanishing gradients problem. However, LReLU does not suffer from the dying ReLU problem [165], which occurs when the neurons that use ReLU activation functions become *inactive* and only output 0 due to a negative net input signal. The LReLU activation function is given as

$$f_{AN}(net - \theta) = \begin{cases} net - \theta & \text{if } net \geq \theta \\ \alpha(net - \theta) & \text{otherwise} \end{cases} \quad (2.9)$$

By introducing a parameter $\alpha > 0$, negative net input signals will still yield non-zero activations, resulting in non-zero gradients and avoiding gradient saturation. Non-zero gradients are required by some heuristics such as SGD in order to be able to effectively train ANNs [70]. An illustration of LReLU with various values for α and $\theta = 0$ is given in Figure 2.3.

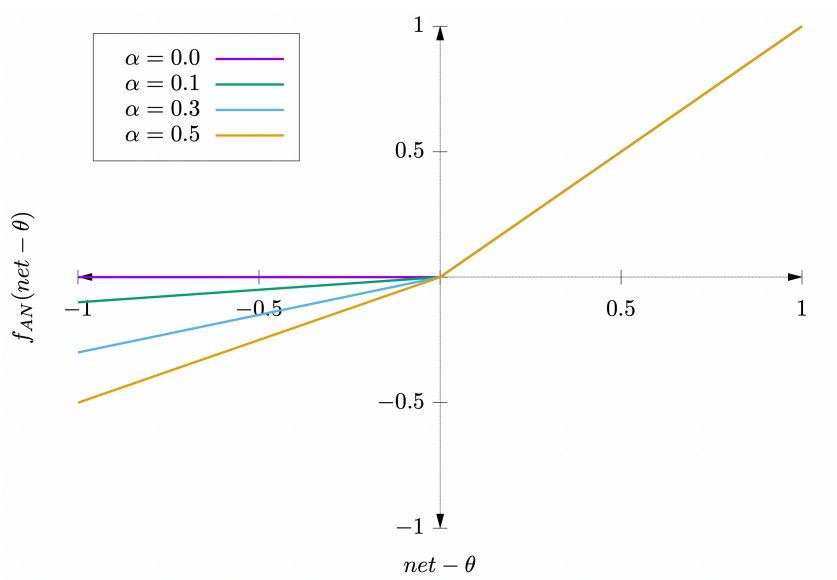


Figure 2.3: An illustration of the LReLU activation function with various values for α and $\theta = 0$.

Sigmoid

The sigmoid activation is the continuous differentiable approximation of the step function, which was used in the original perceptron model developed by Rosenblatt [138], and yields output in the range $(0, 1)$. The sigmoid activation function is given as

$$f_{AN}(net - \theta) = \frac{1}{1 + e^{-\lambda(net-\theta)}} \quad (2.10)$$

where λ is a control parameter that controls the steepness of the sigmoid activation function and is usually set to $\lambda = 1$. An illustration of the sigmoid activation function with various values for λ and $\theta = 0$ is given in Figure 2.4.

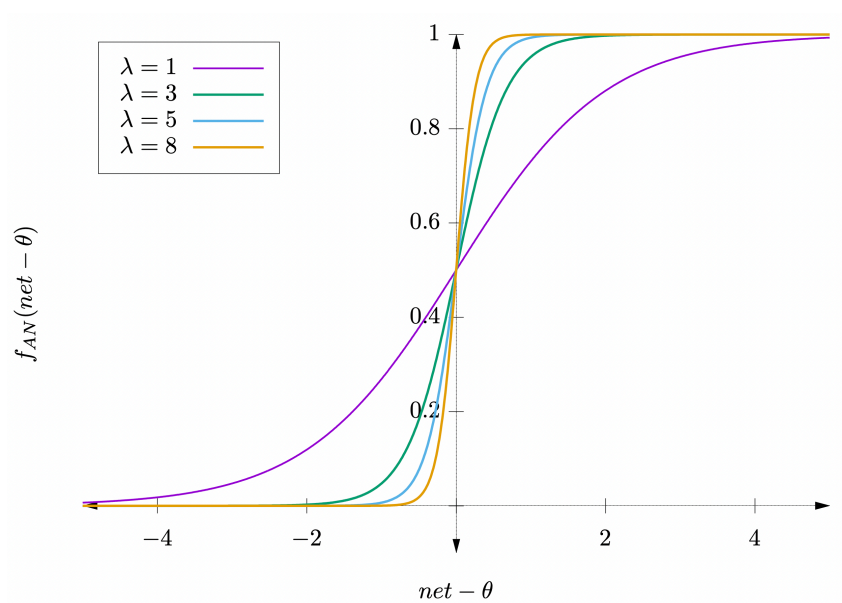


Figure 2.4: An illustration of the sigmoid activation function with various values for λ and $\theta = 0$.

Hyperbolic Tangent

The hyperbolic tangent activation function has a similar shape to that of the sigmoid activation function, but yields output in the range $(-1, 1)$. The hyperbolic tangent activation function is given as

$$f_{AN}(net - \theta) = \frac{e^{\lambda(net-\theta)} - e^{-\lambda(net-\theta)}}{e^{\lambda(net-\theta)} + e^{-\lambda(net-\theta)}} \quad (2.11)$$

An illustration of the hyperbolic tangent activation function with various values for λ and $\theta = 0$ is given in Figure 2.5.

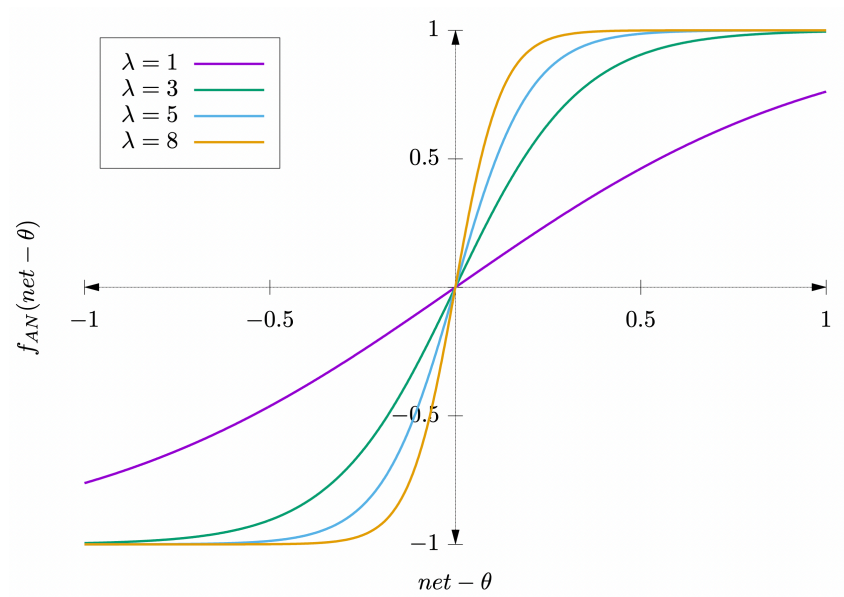


Figure 2.5: An illustration of the hyperbolic tangent activation function with various values for λ and $\theta = 0$.

2.2.6 Output

Output signals are numerical values that represent the output of the AN's activation function. Output signals are often referred to as *predictions*. Output values need to be post-processed to ensure the practical application of the AN.

Output Post-Processing

Output post-processing converts output values to ranges that better match that of the target values. The post-processing techniques that are applicable depend on the type of problem (regression or classification), pre-processing techniques used, as well as the activation function used. For the purposes of this dissertation, data decoding, normalisation and re-scaling techniques are considered.

Decoding

Data decoding refers to the process of undoing the encoding process. For min-max scaling, data is converted back to the range (x_{min}, x_{max}) . In the context of an AN, binary logistic regression problems map the output to the positive or negative class, usually separating the class decision boundary using a threshold value τ . In the context of multiple ANs' output, the output is a vector of numerical values. The one-hot encoded output is then decoded by mapping the index of the output vector that yields the highest activation (argmax) to its associated class.

Softmax

The softmax function, also known as the *softargmax* [60, p. 184] or *normalised exponential function* [12] is a generalisation of the logistic function that converts a K -dimensional output vector \mathbf{y} into a K -dimensional output vector \mathbf{y}' where each element y'_k is in the range $(0, 1)$ and all elements sum up to 1 as is shown in Equation (2.12) below.

$$\mathbf{y}' : \mathbb{R}^K \rightarrow \left\{ \mathbf{y}' \in \mathbb{R}^K \mid y'_k \in (0, 1), \sum_{k=1}^K y'_k = 1 \right\} \quad (2.12)$$

The softmax function is then given as

$$y'_k = \frac{e^{y_k}}{\sum_{k=1}^K e^{y_k}} \quad (2.13)$$

Argmax

The argmax function is similar to the softmax function, with the difference that the element $y_k, k \in \{1, 2, \dots, K\}$ with the highest output value is set to 1 and the rest are set to 0. All elements still sum to 1, but the activation is only observed at index k where the activation is 1. The argmax function is given as

$$y'_k = \begin{cases} 1 & \text{if } y_k = \max(y_1, y_2, \dots, y_K) \\ 0 & \text{otherwise} \end{cases} \quad (2.14)$$

Figure 2.6 illustrates the comparison of transformations of the output vector \mathbf{y} , where the sigmoid, the softmax and the argmax activation functions are used.

2.3 Artificial Neural Network

Multiple ANs can be organised and used together forming a “network” of ANs referred to as an *artificial neural network* (ANN). This section presents ANNs and discussions follow on their applications, architecture, and topologies. Specific reference is made to *feedforward neural network* (FFNN), a specific type of AN used in this dissertation.

2.3.1 Applications

ANNs are inspired by the biological brain. Our brains model biological neural networks with neuron counts in the hundreds of billions and synapse counts in the

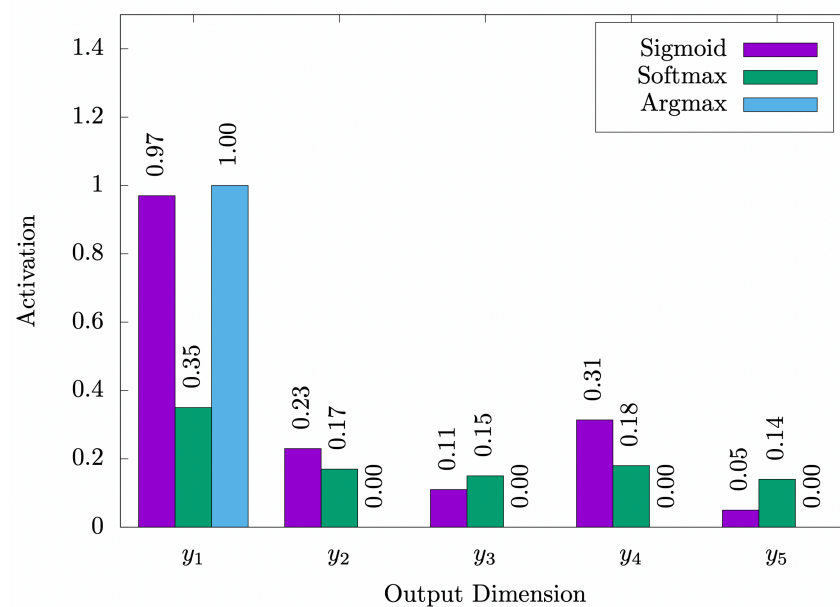


Figure 2.6: An illustration of softmax and argmax applied after the sigmoid activation function.

trillions. To emulate the computational capability of the human brain is not yet possible on modern day hardware. Sandberg et al. [142] approximates a computational requirement of 256 000 terabytes/s to emulate the entire brain. Despite our shortage in hardware capabilities, ANNs have been successfully applied to a range of problem classes. Engelbrecht [45] summarises some common problems that are solved using ANNs and can be listed as follows:

- **Classification:** Predicting the class of an input vector [92].
- **Pattern Matching:** Producing closely associated patterns based on an input vector [21, 96].
- **Pattern Completion:** Completing the missing parts of an input vector [32].
- **Optimisation:** Producing optimal values of parameters in a optimisation problems [153].
- **Data Mining:** Feature discovery in large datasets [150].

The composition of ANs in an ANN is expressed as the *architecture* and *topology* of the AN. The exact composition to use depends on the problem being solved.

2.3.2 Architecture

The architecture of the ANN refers to the way in which ANs are organised. ANNs can be organised in layers where a single layer can contain multiple ANs. Generally,

each layer makes use of the same activation function. Output from one layer is propagated as input to the next layer. This dissertation focuses on the simplest architecture, containing three particular layers, including the input, hidden and output layers. ANNs with this type of architecture are usually referred to as *shallow NNs*. A description of each layer is given as follows.

Input Layer

The input layer contains the input data to the ANN. Since the input layer simply provides the input data, some literature do not consider the input layer as an actual part of the ANN [45].

Hidden Layer

The hidden layer contains a collection of “hidden” ANs, also referred to as *hidden units* or *nodes*. Hidden units are used if the target data is not linearly separable [45]. It has been shown that ANNs that incorporate monotonically increasing differentiable activation functions can approximate any continuous function with just one hidden layer, given that the hidden layer has enough hidden units [79].

Output Layer

The output layer contains the final activations or the predictions of the ANN. These outputs can be used to measure the performance of the ANN.

2.3.3 Topology

The topology of the ANN refers to the way that layers of ANs are connected to each other. There are many different topologies [108]. For the purposes of this dissertation focus is put on a *fully connected* topology where each AN in one layer is connected to all ANs in the next, without any cycles [183].

2.3.4 Feedforward Neural Networks

FFNNs were the first and simplest type of ANNs developed [144] and implement input, hidden and output layers by arranging them in sequential order. Furthermore, FFNNs implement fully connected topologies. In FFNNs, information moves forward, in one direction, from the input nodes, through the hidden nodes and finally to the output nodes. Depending on the optimisation algorithm used, error correction

information can be propagated backwards through the network. An illustration of a **FFNN** is given in Figure 2.7 below.

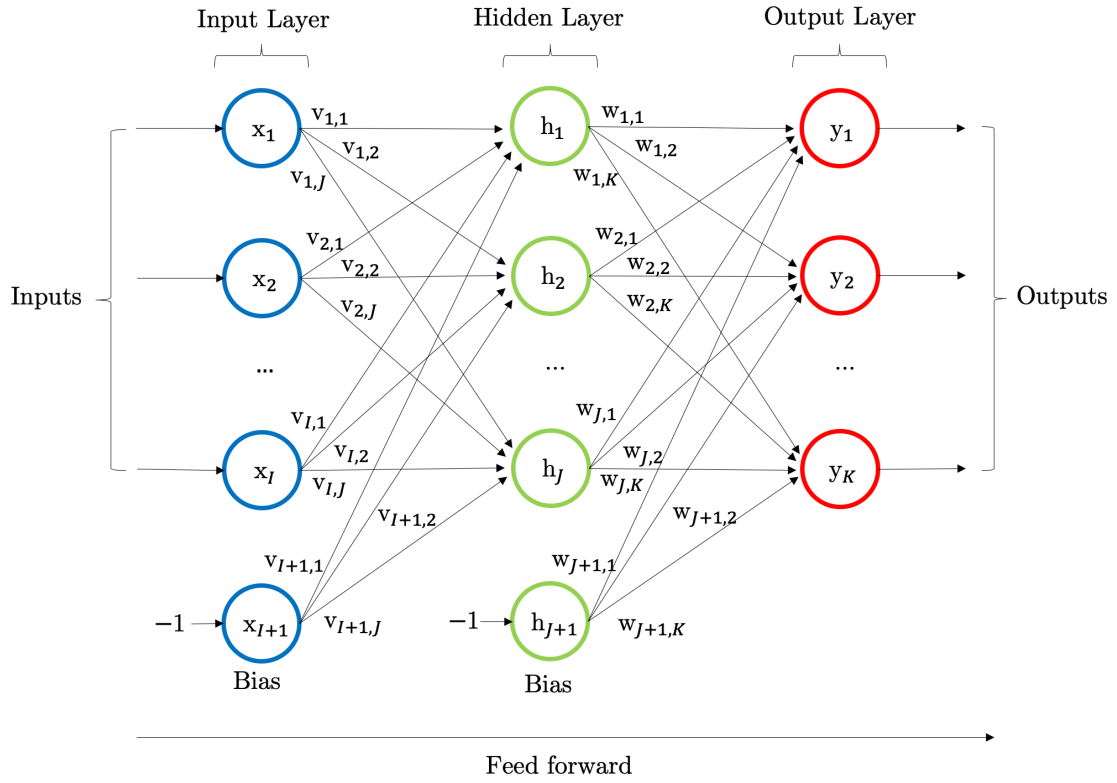


Figure 2.7: An illustration of a feedforward neural network implementing input, hidden and output layers using a fully connected topology.

In Figure 2.7, x_i refers to the i -th dimension in the input vector \mathbf{x} , h_j refers to the j -th dimension in the hidden layer, y_k refers to the k -th dimension in the output vector \mathbf{y} , $v_{i,j}$ refers to the weight associated with input node x_i and the hidden node h_j , and $w_{j,k}$ refers to the weight associated with hidden node h_j and the output node y_k .

Assuming the use of **SUs** and bias weights, the output for the **FFNN** at index k , denoted y_k , is calculated as

$$\begin{aligned}
 y_k &= f(\text{net}_{h,y}) \\
 &= f\left(\sum_{j=0}^{J+1} h_j w_{j,k}\right) \\
 &= f\left(\sum_{j=0}^{J+1} f(\text{net}_{i,h}) w_{j,k}\right) \\
 &= f\left(\sum_{j=0}^{J+1} f\left(\sum_{i=0}^{I+1} x_i v_{i,j}\right) w_{j,k}\right)
 \end{aligned} \tag{2.15}$$

The remainder of this dissertation makes use of [FFNNs](#) and is sometimes referred to as *the model*.

2.4 Training

Details on the training of [FFNNs](#) are presented in this section along with detailed discussions on supervised learning and loss functions.

Training is the process whereby the weights of the [FFNN](#) are systematically changed with the aim of improving the *performance* of the [FFNN](#). During the training process, the [FFNN](#) is exposed to data while trying to produce some target outcome.

The degree to which the produced outcome differs from the target outcome is referred to as *loss*. Since training of [FFNNs](#) is an optimisation problem, the goal of the training process is to minimise the loss of the [FFNNs](#) as it related to input and target data.

Finding the optimal weights that produce the best performance on a given task is an optimisation problem. The optimisation algorithm used to find the optimal weights is referred to as a *heuristic*. Heuristics search for possible solutions in the solution-space and make use of information from the search space to guide to process.

2.4.1 Supervised Learning

Supervised learning is the process of training where the data that is presented to the [FFNN](#) during training, includes the desired solution [56]. The [FFNN](#) learns the mapping function from the input to the target output [16]. The desired solutions are referred to as *labels*. Supervised learning can be used for both classification and regression problems.

The training data that is used during supervised learning, is split proportionally into a *training* and *validation* set. Data in the training set is used to train the [FFNN](#) [86] and update the weights, while the validation dataset is used for hyperparameter tuning, where the parameters of the training process are altered, but not the weights of the [FFNN](#).

Exposing the [FFNN](#) to all training data once is referred to as an epoch. The [FFNN](#) is trained until some stopping condition is reached. Once the stopping condition has been reached, the *performance* of the [FFNN](#) is evaluated. The performance is directly related to the loss of the [FFNN](#) as it relates to a *test dataset*. The test dataset is a collection of data that is not used during training and is used to determine the

generalisation capabilities of the [FFNN](#) as it relates to unseen data. Overfitting and underfitting describe two possible outcomes of the training process.

Overfitting

Overfitting describes a scenario where the trained [FFNN](#) performs well on training data, but does not generalise well to never before seen data from the test set [56, 160]. Géron [56] describes overfitting as the case where [FFNN](#) is too complex relative to the noisiness of the training data.

Underfitting

Underfitting describes a scenario where the [FFNN](#) is not able to effectively learn the underlying structure of the training data [56, 160]. Géron [56] describe underfitting as the case where the [FFNN](#) is too simple relative to the underlying structure of the training data.

There are two types of supervised learning algorithms based on when weights are updated [45]. These include stochastic training and batch training.

Stochastic Training

Stochastic training, also known as *online learning*, is a supervised learning variation whereby weights are adjusted after each training pattern presented. Stochastic training benefits from shuffling data in the training dataset before presenting it to the [FFNN](#), in order to avoid overfitting or memorising the order in which patterns are presented [45]. It has been shown that shuffling the training data can speed up convergence [8].

Batch Training

Batch training, also known as *offline learning*, is a supervised training variation whereby weight changes are accumulated and used to adjust the weights only once, after all the training patterns have been presented.

Mini-Batch Training

Research suggests a trade-off between stochastic and batch training by making use of mini-batches [8]. Mini-batch training is similar to batch training, however weights are updated after β patterns have been presented, where β is the mini-batch size.

Performance metrics are used during training on the training set and evaluation on the test set. There are many different performance measurements that can be used, however this dissertation focuses on performance measures related to loss. Loss is calculated using an error function. The following section presents the reader with more detail on the error functions that can be used during the training process.

2.4.2 Error Functions

This dissertation focuses on a number of error functions, including *sum squared error* (SSE), *mean squared error* (MSE), *root mean squared error* (RMSE), *mean absolute error* (MAE), *binary cross entropy* (BinXE), *categorical cross entropy* (CatEX) and *sparse categorical cross entropy* (SparseCatXE).

Sum Squared Error

The SSE is given as

$$\epsilon = \sum_{p=1}^P \sum_{k=1}^K (\hat{y}_{k,p} - y_{k,p})^2 \quad (2.16)$$

where $\hat{y}_{k,p}$ is k -th dimension of the target output of pattern p , $y_{k,p}$ is k -th dimension of the predicted output vector \mathbf{y}_p for pattern p , P is the number of patterns in the mini-batch, and K is the number of dimensions in the output vector \mathbf{y} .

Mean Squared Error

The MSE is given as

$$\epsilon = \frac{\sum_{p=1}^P \sum_{k=1}^K (\hat{y}_{k,p} - y_{k,p})^2}{PK} \quad (2.17)$$

Root Mean Squared Error

The RMSE is given as

$$\epsilon = \sqrt{\frac{\sum_{p=1}^P \sum_{k=1}^K (\hat{y}_{k,p} - y_{k,p})^2}{PK}} \quad (2.18)$$

Mean Absolute Error

The MAE is given as

$$\epsilon = \frac{\sum_{p=1}^P \sum_{k=1}^K |\hat{y}_{k,p} - y_{k,p}|}{PK} \quad (2.19)$$

Binary Cross-Entropy

[BinXE](#) is used in classification problems, where there are only two classes in the target output data. [BinXE](#) is given as

$$\epsilon = - \frac{\sum_{p=1}^P \sum_{k=1}^K (\hat{y}_{k,p} \log(y_{k,p}) + (1 - \hat{y}_{k,p}) \log(1 - y_{k,p}))}{PK} \quad (2.20)$$

Categorical Cross-Entropy

[CatEX](#) is used in classification problems where the target output or *label* is a one-hot encoded vector. [CatEX](#) is given as

$$\epsilon = - \frac{\sum_{p=1}^P \sum_{k=1}^K \sum_{c=1}^C (\mathbb{1}_{\hat{y}_{k,p} \in C_c} \log(y_{k,p} [\mathbb{1}_{y_{k,p} \in C_c}]))}{PK} \quad (2.21)$$

where $\mathbb{1}$ is the indicator function that the k -th index observation belongs to the c -th class. C is the total number of unique class labels. If $C = 2$, then [BinXE](#) can be used instead.

Sparse Categorical Cross-Entropy

[SparseCatXE](#) error function is similar to [CatEX](#) with the only difference being that the target output or label is a one-hot embedding of a class represented as an integer, $c \in \{1, 2, \dots, C\}$.

2.5 Summary

This chapter presented background information on the [BN](#). The [AN](#) was introduced and discussions followed on the various components that make up the [AN](#). Details on input, weights and biases, net input signal, activation functions, and output were provided. The [ANN](#) was introduced. [ANN](#) design was described in terms of architecture and topologies. Special emphasis was put on [FFNNs](#). Background information on the training of [FFNNs](#) was presented. A variant of training, called supervised learning was presented and led to discussions on datasets, performance measures and training outcomes. The chapter concluded with a number of error functions that can be used to calculate the loss in the context of supervised learning.

Chapter 3

Heuristics

“It is not the strongest of the species that survives. It is also not the most intelligent that survives. It is the one that is the most adaptable to change.”

- Charles Darwin

Many different techniques have been used to train [FFNNs](#) [94]. Finding the best technique to use to train a [FFNN](#) has been shown to be problem dependent in many cases [93]. Every technique has its characteristics, constraints, advantages and disadvantages. At the time of writing, the majority of work that is published around the training of [FFNNs](#), involves the use of gradient-based techniques [115]. Gradient-based techniques are not without flaws and can, for example, yield slow convergence or get trapped in local optima [110]. Other techniques have also been used to successfully train [FFNNs](#), including [MHs](#) such as *particle swarm optimisation* ([PSO](#)) [131, 172], *differential evolution* ([DE](#)) [47] and *genetic algorithms* ([GAs](#)) [67].

Chapter 2 briefly introduced the reader to the concept of heuristics and *meta-heuristics* ([MHs](#))s. This chapter presents more detailed background information on various different heuristics that have been used to train [FFNNs](#). Broadly speaking, this chapter focuses on two different groups of heuristics, including classical gradient-based approaches and population-based [MHs](#). Each technique is presented and discussed in detail. Pseudo-code algorithms are provided for each technique and discussions follow on advantages, disadvantages, capabilities and limitations. The remainder of this chapter is structured as follows:

- **Section 3.1** provides a brief review of optimisation. It is shown that training of [FFNNs](#) is an optimisation problem.

- **Section 3.2** provides background information on the origins and definition of the term *heuristic*. It is shown that heuristics are a class of algorithms that are used to solve optimisation problems.
- **Section 3.3** presents seven low-level, gradient-based heuristics, including *stochastic gradient descent* (SGD), *momentum* (Momentum), *Nesterov accelerated gradients* (NAG), *adaptive gradients* (Adagrad), *root mean squared error propagation* (RMSProp), *adaptive learning rate* (Adadelata) and *adaptive moment estimation* (Adam).
- **Section 3.4** presents three different population-based MHs, including *particle swarm optimisation* (PSO), *differential evolution* (DE) and *genetic algorithms* (GAs).
- **Section 3.5** provides a brief summary of the chapter.

3.1 Optimisation

Optimisation is the task of finding a solution to a given problem that is better than alternative solutions. Better stated by Oldewage [120], optimisation is the task of finding values for a set of variables such that some measure of optimality is satisfied given a set of constraints. Engelbrecht [45] breaks optimisations problems down into three components:

- An **objective function**: Represents the quantity to be optimised and is used as the “measure of optimality”. Optimisation can be defined in terms of the minimisation or maximisation of the objective function f .
- A **set of unknowns or independent variables**: Affects the outcome of the objective function f and is denoted as x . $f(x)$ is thus the quantification of the objective function over the unknowns, represented by x . Note that x could be a scalar value, a vector or a matrix and notation is left out for simplicity.
- A **set of constraints**: Restrict and limit the values that can be assigned to the unknowns, represented by x . Optimisation problems that must adhere to a set of constraints are referred to as *constraint satisfaction problems* (CSPs).

Optimisation problems come in a wide variety, and can be defined in terms of the number of variables used (uni- vs. multivariate), the number of objective functions used (single- vs. multi-objective), the degree of linearity (linear vs. quadratic/polynomial), the number of optima (uni- vs. multi-modal), the nature of the environment

(static vs. dynamic), the types of variables used (separable vs. inseparable, discrete vs. continuous) and the set of constraints that the solution must adhere to (constrained vs unconstrained).

Optima can be defined as *local* or *global* optima. Local optima is the best optimisation of $f(x)$ in a neighbourhood of solutions, while the global optima is the best optimisation of $f(x)$ over all solutions in the solution space.

As stated in Chapter 2, the training of an FFNNs is a particular type of optimisation problem, where the goal is to find the configuration of *weights*, such that the FFNN yields output that minimises some loss function. The mechanism by which the optimal weights for a FFNN is sought out, is executed by an optimisation algorithm known as a heuristic.

3.2 What is a heuristic?

The term *heuristic* comes from the Latin word *heuristicus* which means “to find out or discover”. Romanycia et al. [137] provide a complete study on the history and origins of the term *heuristic*. From their research, a proposal is made to define heuristics in the context of *artificial intelligence* (AI), as any device, be it a program, rule, piece of knowledge, which is added to a problem-solving system, in expectation that, on average, the performance will improve.

In the context of this dissertation, a heuristic refers to an algorithmic search technique that serves as a guide to a search process where good solutions to a optimisation problem is being sought out. Different heuristics make use of different information during the search process [93]. During training of FFNNs, heuristics such as gradient-based heuristics make use of the derivatives obtained by evaluating the FFNN. It can thus be said that gradient-based heuristics make use of information directly from the *search space*. On the contrary, heuristics such as MHs make use of meta-information obtained as a result of evaluating the FFNN [13]. The meta-information that MHs make use of could include ranked-performance of a population of candidate solutions, referred to as *entities*. Meta-heuristics are useful when there is imperfect information about the search space [11], and are generally less problem-specific than other classes of heuristics [13]. This dissertation takes a particular interest in gradient-based heuristics and MHs.

3.3 Gradient-Based Heuristics

Gradient-based heuristics are optimisation techniques that make use of derivatives obtained from evaluating the model being optimised. Specifically, in the context of a minimisation problem, these techniques are called *gradient descent* (GD) heuristics as they *minimise* some loss function. GD is generally attributed to Cauchy [101], who first suggested it in 1847. In 1907 Hadamard [69] independently proposed a similar method.

Although gradient-based heuristics were not the first heuristics used to train FFNNs [45], they are certainly the most widely used. Gradient-based heuristics have become increasingly popular partly due to their simplicity and low computational overhead compared to other heuristics such as MHs and other second-order derivative methods such as Newtons' method. There are many variants of gradient-based heuristics, however, they all fundamentally apply the same generic *gradient descent* (GD) framework called *backpropagation* (BP).

3.3.1 Backpropagation

Chapter 2 introduced supervised learning and presented a number of loss functions. In the context of supervised learning, loss functions produce a scalar value ϵ , that represents the error between the output of the FFNN and the desired output. When using GD to train FFNNs, the loss function is used to adjust the weights of the FFNN in order to minimise the error [45]. Engelbrecht [45] states that training of FFNNs using GD, is done by calculating the gradient of ϵ in *weight-space*, and then moving the weight vector along the negative gradient. An illustration of GD is given in Figure 3.1.

In the context of training *shallow* FFNNs using supervised learning, the error signal is propagated backwards in the network from the output layer, through the hidden layer to the input layer, updating the weights at the output and hidden layers. The algorithm that propagates the error signal backwards is known as *backpropagation* (BP). BP was popularised by Werbos [177]. Nel [115] states that BP provides a procedure for updating the network weights, layer by layer, by evaluating the derivatives of the error function \mathcal{E} , with respect to the weights at each layer. Engelbrecht [45] describes the BP process in two steps:

- **Feedforward Pass:** During this phase the output values of the FFNN is calculated for each training pattern.
- **Backward Propagation:** During this phase the error signal is propagated

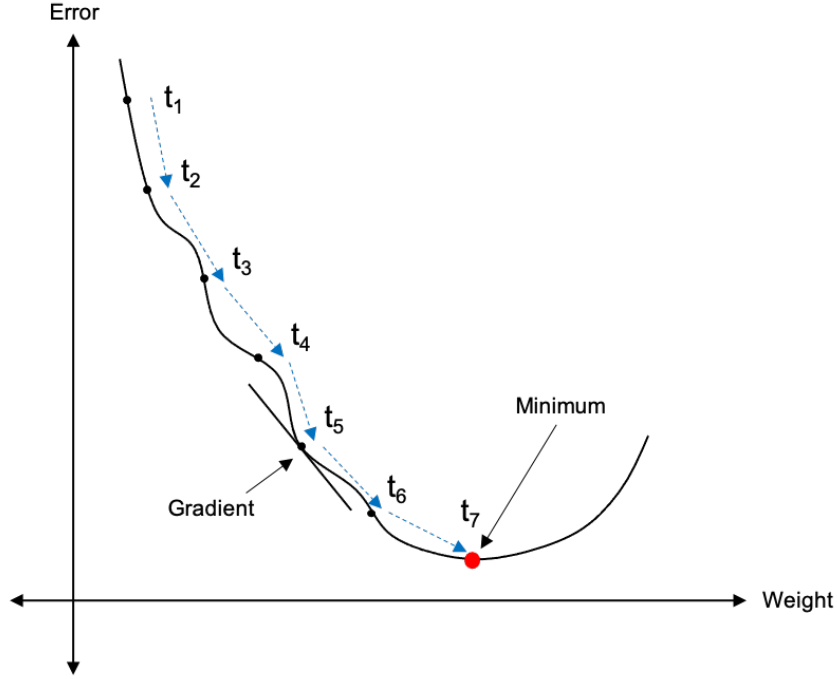


Figure 3.1: An illustration of *gradient descent* (GD) over various time steps showing the minimisation of the error with regards to weight value.

backwards from the output layer, through the hidden layer, to the input layer of the FFNN. Weights at the output and hidden layers are then adjusted as functions of the backpropagated error signal.

The update step for GD by means of BP can be formulated as is shown in Equations (3.1) to (3.4). The general weight update step is given as

$$w_i(t) = w_i(t - 1) + \Delta w_i(t) \quad (3.1)$$

where w_i is the weight value at index i , t is the time step, $\Delta w_i(t)$ is the weight update vector at index i and time step t . The delta weight term $\Delta w_i(t)$, is given as

$$\Delta w_i(t) = -\eta \frac{\partial \epsilon}{\partial w_i} \quad (3.2)$$

where η is the learning rate and $\frac{\partial \epsilon}{\partial w_i}$ is the *gradient* of the error, as a result of a loss function \mathcal{E} , relative to the weight at index i . The learning rate controls the step-size that is taken in the direction of the negative gradient at each time step. Assuming the use of SSE as the loss function, the partial derivative of ϵ , relative to the weight vector \mathbf{w} , at index i , is given as

$$\frac{\partial \epsilon}{\partial w_i} = -2(t_{i,p} - o_{i,p}) \frac{\partial f}{\partial net_p} z_{i,p} \quad (3.3)$$

where $t_{i,p}$ refers to the i -th element of the target value for pattern p , $o_{i,p}$ refers to the i -th element of the output value for pattern p , f refers to the activation function, net_p refers to the net input signal for pattern p , and $z_{i,p}$ refers to the input value at index i for pattern p .

Finally, assuming the use of the sigmoid activation function, the partial derivative of the activation function, relative to the net input signal is given as

$$\frac{\partial f}{\partial net_p} = o_p(1 - o_p) \quad (3.4)$$

Again, assuming the use of [SSE](#) as the loss function, the pseudo-code implementation for the generic [GD](#) algorithm is taken from Engelbrecht [45] and is presented in Algorithm 1 below.

Algorithm 1 The pseudo-code algorithm for the generic *gradient descent* ([GD](#)) heuristic.

Initialise the [FFNN](#) weights and biases, the learning rate η and the time step/epoch $T = 0$;

while stopping conditions are not met **do**

 Let $\epsilon_T = 0$;

for each training pattern p **do**

 Calculate the output of the hidden and output layers (feedforward);

 Compute the error signals of the hidden and output layers;

 Adjust the weights of the output and hidden layers (backpropagate);

$\epsilon_T += [\epsilon_{T_p} = \sum_{k=1}^K (t_{k,p} - o_{k,p})^2]$;

end for

$T += 1$;

end while

A simplified notation is proposed from which the gradient-based heuristics can be compared. Consider a simplification of the update-step for [SGD](#) as presented in Equations (3.1) and (3.2) where $\frac{\partial \epsilon}{\partial w_i}$ is simply represented as the gradient term \mathbf{g} , resulting in a simplified weight update step and is given as

$$\mathbf{w} = \mathbf{w} - \eta \mathbf{g} \quad (3.5)$$

where \mathbf{w} refers to the weight vector, η refers to the learning rate as before and \mathbf{g}

refers to the gradient of the error function relative to weight vector. Note that some subscripts related to layers, indices, time steps and training patterns are omitted for convenience.

3.3.2 Stochastic vs. Batch Training

This section shines light on the algorithmic implementation of BP with specific context to stochastic and batch training. The implementation of GD using stochastic training is referred to as *stochastic gradient descent* (SGD).

SGD was one of the first widely used heuristics to train FFNN, however, it is not without flaws. With stochastic training, only one training pattern is presented at each iteration/epoch. As such, weight updates are done with high variance and noise [140]. An illustration of the fluctuations caused by SGD during training is given in Figure 3.2.

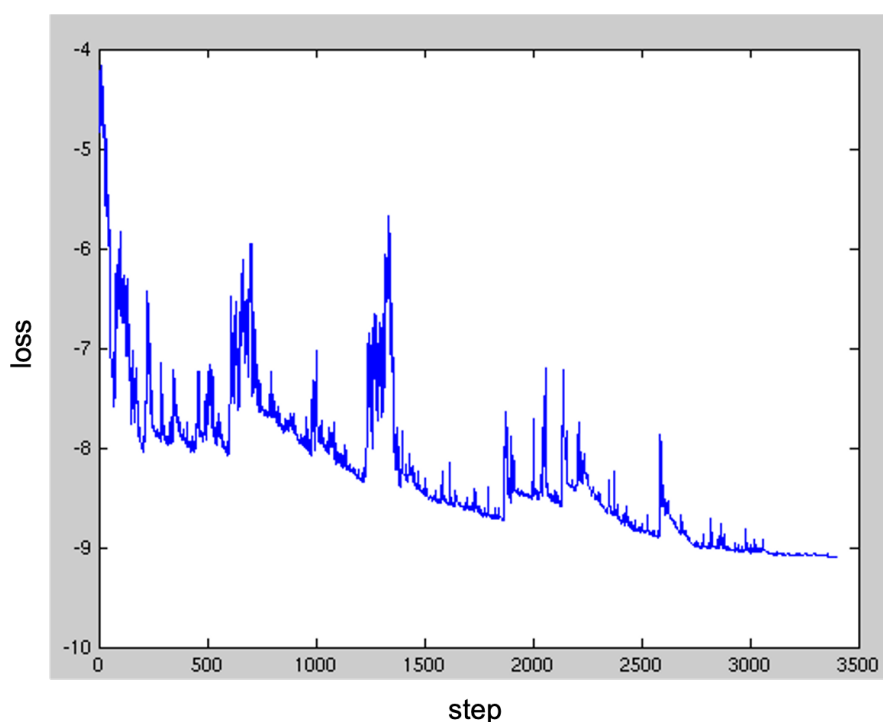


Figure 3.2: An illustration of *stochastic gradient descent* (SGD) fluctuations during training as taken from [125].

With batch training, all the training patterns are presented at once and the FFNN converges to the global minimum of the loss function w.r.t \mathbf{w} (assuming the loss function is convex). However, this is computationally expensive and impractical to implement in many situations. While SGD is able to jump out of local minima into new, potentially better local minima [140], SGD complicates convergence to

the minima as [SGD](#) can potentially keep overshooting better minima. By slowly decreasing the learning rate η during training, the same convergence behaviour is achieved as with batch training.

A compromise is to make use of mini-batches of training patterns, where a small number of training patterns are presented to the [FFNN](#) at once. This is referred to as *mini-batch* training. Input patterns from mini-batches are approximations of the total population of training patterns. According to Ruder [140] this has two main advantages:

- Mini-batch training reduces the high variance of weight updates as observed for [SGD](#), which leads to better convergence.
- Mini-batch training allows for the implementation of [GD](#) using highly optimised matrix operations, common to the state-of-the-art [ML](#) libraries used today.

In the context of this dissertation, the implementation of [SGD](#) refers to the mini-batch training implementation of [GD](#). Although mini-batch training does provide a compromise between stochastic and batch training, there are still a number of challenges faced by mini-batch training. These include:

- The appropriate value to use for the learning rate η , is difficult to determine and is often problem-specific. A learning rate that is too small causes premature exploitation, leading to slow and bad convergence. On the contrary, a learning rate that is too high may lead to bad learning outcomes as the heuristic keeps overshooting good minima.
- Learning rate schedules [135] can be introduced to dynamically change the learning rate throughout the training process, however, these schedules and their parameters have to be defined a priori and are often problem specific [28].
- The learning rate that has been introduced so far is applied to all elements of the weight update vector. If the training data is sparse and the features have varying frequencies, an equal update to all weight elements is inefficient. Larger weight updates are required for less frequently occurring features. Inefficient updates of all elements in the weight vectors contribute to the credit assignment problem [141], common to [GD](#) variants.
- It is difficult to avoid getting trapped in local minima, especially for highly non-convex loss functions used for training [FFNNs](#). Dauphin et al. [31] mentions the role of saddle points in getting trapped in local minima. Saddle points

are points at which one dimension slopes upwards, while another dimension slopes downwards. Ruder [140] mentions that these saddle points are usually surrounded by plateaus of the same error, leading to gradients that are close to zero in all dimensions.

Alternative variants have been proposed that lead to better control over the convergence characteristics caused by GD. The first of these GD variants include Momentum.

3.3.3 Momentum

Research shows that SGD has difficulty navigating ravines [158]. Ravines are areas where the surface curves much more steeply in one dimension than in another. These ravines are common around local minima. As such, SGD is shown to oscillate across the slopes of the ravine while only making minor progress towards the local minima.

Momentum [129] is a variant of SGD that helps accelerate SGD in the relevant direction, dampening oscillations. Ruder [140] mentions that this is done by adding a fraction α , of the weight update vector of the previous time step to the current weight update vector. An illustration of SGD with and without momentum is given in Figure 3.3.

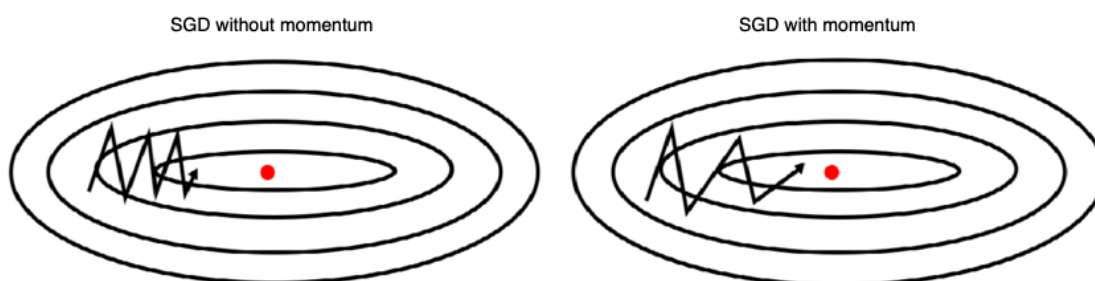


Figure 3.3: An illustration of *stochastic gradient descent* (SGD) with and without momentum taken from [39].

The accumulation of momentum is given as

$$\mathbf{v} = \alpha \mathbf{v} - \eta \mathbf{g} \quad (3.6)$$

while the update step is then amended as

$$\mathbf{w} = \mathbf{w} + \mathbf{v} \quad (3.7)$$

By redefining the **SGD** update steps as shown above in Equations (3.6) and (3.7), the **Momentum** heuristic allows for the increase of momentum for dimensions whose gradients point in the same direction, while simultaneously reducing momentum for dimensions whose gradients change direction, leading to faster convergence and less oscillation. The momentum term α is usually set to 0.9 [45, 140].

3.3.4 Nesterov Accelerated Gradients

Nesterov accelerated gradients (**NAG**) is a variant of the **Momentum** heuristic developed by Sutskever et al. [157] and is inspired by Nesterov's [118] work on optimising convex functions. **NAG** provides an improvement to the momentum accumulation term by providing a look-ahead term that better refines the weight update step.

In the **NAG** heuristic, the gradient is not calculated w.r.t the current weights, but rather w.r.t the approximate future positions of the weights [140, 157]. An illustration of the weight update vector using **NAG** is taken from Geoffrey Hinton's lecture on mini-batch **GD** [74] and is presented in Figure 3.4 below.

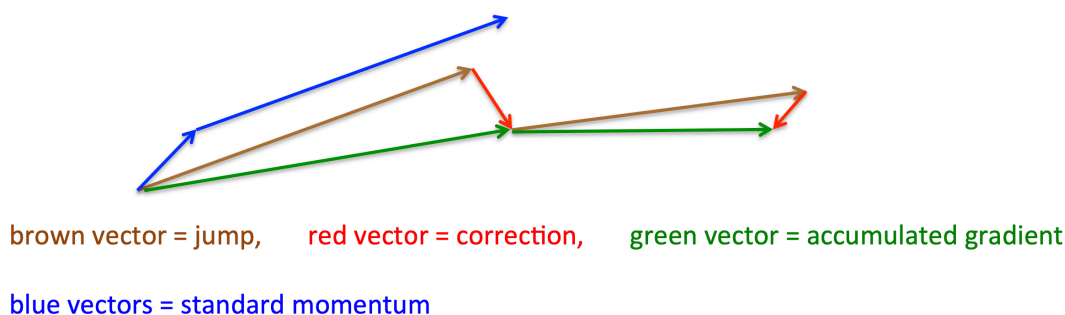


Figure 3.4: An illustration of the weight update vector for *Nesterov accelerated gradients* (**NAG**) taken from [74].

The difference in the update step for **Momentum** and the update step for **NAG** as developed by Sutskever et al. [157] is described by Ruder [140] as follows. **Momentum** first calculates the current gradient, represented by the small blue vector in Figure 3.4 and then takes a large step in the direction of the updated accumulated gradient, presented by the big blue vector. In contrast, **NAG** first takes a big step in the direction of the previous accumulated gradient presented by the brown vector. At this point the gradient is measured and **NAG** makes a correction represented by the red vector. The complete update step is represented by the green vector. By anticipating approximate future positions of the weights, the weight update step as defined by **NAG** controls the optimisation process from going too fast and results in increased responsiveness [9]. Along with the same velocity update step for **Momentum**, as

presented in Equation (3.6), the **NAG** weight update step is given as

$$\mathbf{w} = \mathbf{w} + \alpha \mathbf{v} - \eta \mathbf{g} \quad (3.8)$$

3.3.5 Adaptive Gradients

Adaptive gradients (**Adagrad**) is an adaptation of **SGD** that implements a learning rate for every element in the weight vector and is developed by Duchi et al. [41]. Ruder [140] mentions that **Adagrad** adapts the learning rate to the elements of the weight vector, by performing small updates (i.e. low learning rates) for weights associated with frequently occurring features and larger updates (i.e. high learning rates) for weights associated with infrequent features. For this reason, **Adagrad** is well suited for dealing with situations where training data is sparse.

In the **GD** variants presented thus far, the weight updates have been applied by making use of the same learning rate η , for all elements of the weight update vector. **Adagrad** uses a different learning rate for every weight element w_i , at every time step t . The weight update step for **Adagrad** is given as

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \quad (3.9)$$

where w_i is the i -th element of the weight vector, $G_{t,ii} \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element i, i , is the sum of the squared gradients w.r.t. w_i , up to time step t and ϵ is a smoothing term that avoids division by zero and is set to an insignificant value such as 1×10^{-8} . Since \mathbf{G}_t contains the sum of the squares of the gradients along its diagonal, the weight update step can be vectorised and updated using the matrix-vector product, represented by \odot , between $G_{t,i}$ and $g_{t,i}$. The simplified update step as implemented by **Adagrad** is given as

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \odot g_{t,i} \quad (3.10)$$

Adagrad's main benefit is that it does not require manual tuning of the learning rate η . However, a problem with **Adagrad** is in the accumulation of squared gradients in the denominator, represented by \mathbf{G}_t . Since every term that is added is positive, \mathbf{G}_t keeps growing, leading to a situation where the learning rate shrinks to the point that the heuristic is no longer able to learn.

3.3.6 Adaptive Learning Rate

Zeiler [182] presents [Adadelata](#), an improvement of [Adagrad](#) that eliminates the accumulation of squared gradients in the denominator. Ruder [140] mentions that instead of accumulating all past squared gradients as in the case of [Adagrad](#), [Adadelata](#) restricts the window of accumulation to a window with a fixed size W . However, storing W previous squared gradients is very inefficient. Instead, the accumulation of gradients over W steps is defined recursively as a decaying average of all past squared gradients. The moving average of squared gradients at time step t , denoted by $E[\mathbf{g}^2]_t$ depends on a fraction α of the previous average of squared gradients and the current gradient, similar to the update step for [Momentum](#). Also similar to [Momentum](#), α is set to 0.9. The update step for $E[\mathbf{g}^2]_t$ is given as

$$E[\mathbf{g}^2]_t = \alpha E[\mathbf{g}^2]_{t-1} + (1 - \alpha) \mathbf{g}_t^2 \quad (3.11)$$

To illustrate the difference between [Adagrad](#) and [Adadelata](#), the [SGD](#) weight update step for [Adagrad](#), as presented in (3.10) is rewritten such that the diagonal matrix \mathbf{G}_t , is replaced with the moving average of squared gradients and is given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{E[\mathbf{g}^2]_t + \epsilon}} \mathbf{g}_t \quad (3.12)$$

Since the denominator is the *root mean squared* ([RMS](#)) error criterion of the gradient, the criteria short-hand notation is used and is given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{RMS[\mathbf{g}]_t} \mathbf{g}_t \quad (3.13)$$

Zeiler [182] noted that the units of the update step as presented in Equation (3.13) do not match. The weight update vector should have the same hypothetical units as the weight vector. To fix the aforementioned problem, another exponentially decaying average is defined in terms of squared weight updates and is given as

$$E[\Delta \mathbf{w}^2]_t = \alpha E[\Delta \mathbf{w}^2]_{t-1} + (1 - \alpha) \Delta \mathbf{w}_t^2 \quad (3.14)$$

The *root mean squared* ([RMS](#)) error of weight updates is given as

$$RMS[\Delta \mathbf{w}]_t = \sqrt{E[\Delta \mathbf{w}^2]_t + \epsilon} \quad (3.15)$$

Note that $RMS[\Delta \mathbf{w}]_t$ is unknown at time step t . $RMS[\Delta \mathbf{w}]_t$ is approximated

with the *root mean squared* (RMS) of weight updates until the previous time step. The learning rate in Equation (3.13) is then replaced. The weight update step for [Adadelta](#) is then concluded and is given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{RMS[\Delta \mathbf{w}]_{t-1}}{RMS[\mathbf{g}]_t} \mathbf{g}_t \quad (3.16)$$

The main advantage of [Adadelta](#) is in removing the learning rate, which eliminates the need for hyper-parameter tuning of the learning rate.

3.3.7 Root Mean Squared Error Propagation

Root mean squared error propagation (RMSProp), presented by Hinton et al. [74] is similar to [Adadelta](#) and was developed independently around the same time. RMSProp is the same as the first weight update vector for [Adadelta](#), presented in Equation (3.12). A disadvantage of RMSProp is that it still includes the learning rate term η that needs to be tuned. For RMSProp, Hinton et al. [74] suggests α be set to 0.9 and η be set to 0.001.

3.3.8 Adaptive Moments Estimation

Adaptive moment estimation (Adam) is another variant of SGD that includes adaptive learning rates and is presented by Kingma et al. [94]. In addition to storing an exponentially decaying average of past squared gradients like [Adadelta](#) and RMSProp, Adam also stores an exponentially decaying average of past gradients, similar to Momentum [140]. Heusel et al. [73] uses the analogy that, if Momentum can be seen as a ball running down a slope, then Adam behaves like a heavy ball with friction, which prefers flat minima in the error surface. The decaying averages for past gradients and past squared gradients is given in Equations (3.17) and (3.18) respectively.

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1 - \beta_1) \mathbf{g}_t \quad (3.17)$$

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1 - \beta_2) \mathbf{g}_t^2 \quad (3.18)$$

In Equations (3.17) and (3.18) above, β_1 and β_2 are decay rates, similar to α for Momentum. Kingma et al. [94] suggest default values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$.

Ruder [140] mentions that \mathbf{m}_t and \mathbf{v}_t presented above are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. Kingma et al. [94] mentions that because \mathbf{m}_t and \mathbf{v}_t are initialised to be vectors of 0's, they are biased towards 0. The aforementioned bias is especially prominent during the initial time steps and/or when the decay rates β_1 and β_2 are small (β_1 and β_2 are close to 1). The bias-corrected first and second moment estimates are presented in Equations (3.19) and (3.20) respectively.

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t} \quad (3.19)$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t} \quad (3.20)$$

The Adam weight update rule is then given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t \quad (3.21)$$

3.4 Meta-Heuristics

Gradient-based heuristics are sensitive to the problem that they are applied to, with hyper-parameter selection often dominating the research focus [7, 50]. Blum et al. [13] mention that since the 1980's, a new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are referred to as *meta-heuristics*.

This section aims to introduce the concept of MHs, with focus on population-based meta-heuristics. Three well known MHs, that have been shown to train FFNN, are presented. These MHs include *particle swarm optimisation* (PSO), *differential evolution* (DE) and *genetic algorithms* (GAs). Carvalho et al. [22] compared various PSO variants for training FFNNs. Espinal et al. [47] compared DE and PSO when applied to FFNN training and Gupta et al. [67] compared BP to a GA for training FFNNs.

The term *meta-heuristic* (MH) was first introduced by Glover [59] in 1986 and is derived from the composition of two Greek words. Heuristic derives from the verb *heuriskein* which means “to find”. The prefix, *meta*, means “beyond, in an upper level”. MHs were often called *modern heuristics* [134]. Blum et al. [13] mention that there is a debate as to what the formal definition of MHs is and suggests the

definition of **MHs** to be “high level strategies for exploring search spaces by using different methods”.

The biggest difference between **MHs** and gradient-bases heuristics is that **MHs** make use of meta-information obtained as a result of evaluating the **FFNN** during training and is not limited to information about the search space [13].

3.4.1 Particle Swarm Optimisation

Particle swarm optimisation (**PSO**) is a stochastic population-based search algorithm based on the social behaviour of birds in a flock [90]. By definition, the **PSO** heuristic is nature-inspired. **PSOs** were first presented by Kennedy et al.[90]. Kennedy and Eberhart first applied **PSOs** to train of **FFNNs** [44, 89]. The application of **PSOs** in the context of training **FFNN** have been widely studied [131, 172]. This section aims to provide the details of **PSO** implementation.

This dissertation uses the term *entity* for candidate solutions and a *population* for a collection of entities, in the general context of population-based **MHs**. Engelbrecht [45] mentions that in **PSOs** individual candidate solutions are referred to as *particles* and the population is referred to as a *swarm*. These particles are “flown” through a hyper-dimensional search space. Changes in particle position is due to social-psychological tendencies of individuals to emulate the success of other individuals. The changes in the particle position are then influenced by the experience or knowledge of the particle’s neighbours. The social behaviour of particles is modelled such that they stochastically return to previously successful regions in the search space.

Van Wyk [172] mentions that the swarm is usually arranged in a predefined structure, called a *neighbourhood topology* that governs the communication between particles. Two specific configurations of neighbourhood topologies that exists are referred to as *local best* (*lbest*) **PSO** and *global best* (*gbest*) **PSO**. There are two main differences between the two approaches in terms of their convergence characteristics [42]. These include:

- Due to the larger particle interconnectivity of *gbest* **PSO**, the heuristic converges faster than with *lbest* **PSO**. Engelbrecht [45] mentions that faster convergence comes at a cost of less diversity.
- As a consequence of larger diversity, the *lbest* **PSO** is less susceptible to getting trapped in local minima.

Shi et al. [146] proposed a modification of the original **PSO** as was presented by Kennedy et al. [90]. Their implementation focuses on the *gbest* **PSO** with *inertia*

weights. This dissertation focuses on the Shi et al. [146] implementation of *global best PSO* and particles in this specific implementation has a number of properties associated with them [172]. These include:

- **Position:** Refers to the candidate solution that is represented by the particle and defines the particle position within the optimisation problem's hyper-dimensional solution space. Let the current position vector, for particle i , at time step t be denoted by $\mathbf{x}_i(t)$. Let I denote the population size and J denote the search space dimensionality.
- **Velocity:** Represents a step size for the particle in the search space. The velocity at vector, for particle i , at time step t is denoted $\mathbf{v}_i(t)$.
- **Solution Quality:** Refers to the evaluation of the particle's position with respect to the objective function. Let $f(\mathbf{x}_i(t))$ denote the quality of the solution represented by the particle's position.
- **Personal Best Position:** Refers to a cognitive memory construct, where each particle keeps track of their personal best position found during optimisation thus far. The personal best position is denoted $\mathbf{y}_i(t)$.
- **Global Best Position:** Refers to a social memory construct, where each particle has a reference to the best solution found in the particle's neighbourhood thus far. In the case of *gbest PSO*, the global best position is the best position of the entire swarm. The global best position is denoted $\hat{\mathbf{y}}_i(t)$.

During initialisation, particles are randomly placed within the search space by sampling from a uniform distribution such that $\mathbf{x}_i \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$ and the velocity is set to 0. At time step 0, the particle's initial position is set to be the particle's personal best solution such that $\mathbf{y}_i(0) = \mathbf{x}_i(0)$. The particle's update step is then broken into two parts, including a velocity update step presented in Equation (3.22) followed by a position update step as presented in Equation (3.23).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \quad (3.22)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (3.23)$$

In Equations (3.22) and (3.23) above, i refers to the i -th particle in the swarm and j refers to the j -th dimension of the particle's position. The velocity update step consists of:

- **Previous Velocity:** Denoted by the term $v_{ij}(t)$. This term represents the particle's momentum at index j and is used to formulate an update step for the particle in the search space. Van Wyk [172] mentions that it forces the particle to maintain a consistent direction, preventing drastic changes in terms of update steps. This term is then scaled by the *inertia* weight control parameter, denoted w . Inertia weight was introduced by Shi et al. [146] as a mechanism to control the exploration and exploitation abilities of the swarm.
- **Cognitive Component:** Denoted by the term $c_1 r_{1_j}(t)[y_{ij} - x_{ij}(t)]$. This component represents the particle's personal experience at index j . It introduces an attractor to the particle's personal best position so far. The cognitive component is stochastically scaled with random numbers $\mathbf{r}_1 \sim U(0, 1)^J$ and the cognitive acceleration coefficient c_1 is used to control the influence of the cognitive attractor.
- **Social Component:** Denoted by the term $c_2 r_{2_j}(t)[\hat{y}_{ij} - x_{ij}(t)]$. This component represents the particle's social experience at index j . It introduces an attractor to the swarm's best position so far. The social component is also stochastically scaled with random numbers $\mathbf{r}_2 \sim U(0, 1)^J$, while also introducing the social acceleration coefficient c_2 that is used to control the influence of the social attractor.

Van Wyk [172] mentions that when PSOs were first developed, it was possible for particle velocities to become inappropriately large during optimisation, leading to situations where particles fly out of the feasible search space. This is known as *swarm explosion* and occurs when there are frequent changes in the global best position. In order to address this issue, the concept of *velocity clamping* was introduced [42]. The idea behind velocity clamping is to restrict particle velocities to some \mathbf{V}_{max} threshold, modelling a form of terminal velocity. Velocity clamping is applied after the velocity update step and is given in Equation (3.24) below.

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } -V_{max,j} < v'_{ij}(t+1) < V_{max,j} \\ -V_{max,j} & \text{if } v'_{ij}(t+1) \leq -V_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t+1) \geq V_{max,j} \end{cases} \quad (3.24)$$

\mathbf{V}_{max} is another hyper-parameter that must be defined a priori. Appropriate values for \mathbf{V}_{max} may prevent swarm explosion, but also has an effect on the exploration and exploitation of the heuristic. If \mathbf{V}_{max} is small, particle update steps are small,

resulting in exploitation [42]. If V_{max} is big, it allows for larger update steps, promoting more exploration.

It should be noted that the choice of control parameters play a vital role in the behaviour and characteristics of the PSO. Van den Berg and Engelbrecht [168, 169] have done extensive work on the effects of different values for control parameters. For the purposes of this dissertation, the c_1 and c_2 control parameter are set to 1.496180 and the inertia weight w is set to 0.0729844 as these correspond to values used in [43] and have been shown to be appropriate for a number of problems.

An example of the pseudo-code implementation of the *gbest* PSO is taken from [45] and is given in Algorithm 2.

Algorithm 2 The pseudo-code algorithm for the *gbest* PSO heuristic.

Create swarm of N entities, each with J dimensions;

while stopping condition are not met **do**

for each particle $i = 1, \dots, N$ **do**

 // Set the personal best position;

if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**

$\mathbf{y}_i = \mathbf{x}_i$;

end if

 // Set the global best position;

if $f(\mathbf{x}_i) < f(\hat{\mathbf{y}})$ **then**

$\hat{\mathbf{y}} = \mathbf{x}_i$;

end if

end for

for each particle $i = 1, \dots, N$ **do**

for each dimension $j = 1, \dots, J$ **do**

 // Perform velocity update step;

$v_{ij} = wv_{ij} + c_1r_{1j}[y_{ij} - x_{ij}] + c_2r_{2j}[\hat{y}_{ij} - x_{ij}]$

 // Perform position update step;

$x_{ij} = x_{ij} + v_{ij}$

end for

end for

end while

3.4.2 Differential Evolution

This section aims to introduce the next population-based MH, called *differential evolution* (DE). Similar to PSO, DE is a stochastic population-based search strategy

developed by Storm and Price [128] in 1995. DE shares a lot of similarities with other evolutionary MH paradigms such as PSOs and GAs. However, DE differs significantly in how distance and direction information from the current population is used to guide the search process [45]. Originally, DE was focused on multi-dimensional real-valued optimisation problems, but unlike gradient-based heuristics, it does not require any gradient information. DE does not require the underlying optimisation problem to be differentiable and can thus be applied to problems that are discrete, noisy and dynamic [136].

Lots of research has been done on using DE to train FFNNs. Some notable work include [82, 110, 151]. In these works, the authors often highlight the low computational complexity and simplicity of implementation for DE.

Similar to other *evolutionary algorithms* (EAs), variation from one generation to the next is achieved through the application of crossover and mutation operators. Engelbrecht [45] mentions that for other EAs, if both crossover and mutation operators are used, crossover is applied first, after which the generated offspring is mutated. Furthermore, other EAs sample mutation step sizes from some probability distribution. DE differs from the aforementioned EAs in two ways. Firstly, mutation is applied first to generate a *trial vector*, which is then used within the crossover operator to produce one offspring. Secondly, mutation step sizes are not sampled from prior known probability distributions.

In DE, mutation step sizes are influenced by the differences in positions of different entities in the current population. The positions of entities in the population provide valuable information about the fitness landscape, a concept DE aims to exploit in order to find optimal solutions. There are three main components to the DE heuristic. These include mutation, crossover and selection operators [128]. Each of these is presented in detail in the following sections.

Mutation

The purpose of the mutation operator is to produce a trial vector for each entity in the current population by mutating a *target* vector with a *weighted differential* [45]. The trial vector is used in the crossover operator to produce offspring. The mutation process then follows as such. For each parent $\mathbf{x}_i(t)$ generate a trial vector $\mathbf{u}_i(t)$ as follows:

- Select a target vector, $\mathbf{x}_{i_1}(t)$ from the population that is not the same as the parent i.e. $i \neq i_1$. Various different selection strategies can be used to select the target vector.

- Randomly select two other individuals $\mathbf{x}_{i_2}(t)$ and $\mathbf{x}_{i_3}(t)$. Importantly, all of these entities must be unique such that $i \neq i_1 \neq i_2 \neq i_3$ and $i_2, i_3 \sim U(1, N)$ where N is the size of the swarm/population.
- Selected individual entities are then used to calculate the trial vector by perturbing the target vector as presented in Equation (3.25) below.

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \quad (3.25)$$

In Equation (3.25) $\beta \in (0, \infty)$ is the scale factor and controls the amplification of the differential variation [45].

Crossover

In the context of EAs, reproduction and recombination is done through the crossover operation. The same applies to DE. The DE crossover operator implements a discrete recombination of the trial vector, $\mathbf{u}_i(t)$, as was generated in Equation (3.25) above, and the parent vector $\mathbf{x}_i(t)$ to produce new offspring $\mathbf{x}'_i(t)$. The crossover operator is given in Equation (3.26) below.

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in \mathcal{J} \\ x_{ij}(t) & \text{otherwise} \end{cases} \quad (3.26)$$

In Equation (3.26), $x_{ij}(t)$ refers to the j -th element of the vector $\mathbf{x}_i(t)$ and \mathcal{J} refers to a set of crossover points or indices at which perturbation is done. Different techniques for determining the set \mathcal{J} , has been proposed [154, 155]. These include:

- **Binomial crossover:** A crossover mask is generated by randomly selecting indices from the set of possible crossover points $\{1, 2, \dots, J\}$ where J is the problem dimension. Binomial crossover is presented in Algorithm 3. In Algorithm 3, p_r is the crossover probability. The higher the value of p_r , the more points will be included in the set \mathcal{J} . A *Bernoulli* distribution can be used to generate the binomial crossover mask. Note that due to the probabilistic nature of this process, it is possible that no crossover points are selected. To counteract this situation, a randomly selected crossover point j^* is included in the set \mathcal{J} such that $\mathcal{J} \neq \emptyset$ where \emptyset is the empty set.
- **Exponential crossover:** Engelbrecht [45] states that with exponential crossover, a sequence of adjacent crossover points are selected from some randomly selected crossover index. This means that the set of possible crossover points \mathcal{J} is

a circular array in indices. Exponential crossover does not require the selection of an additional crossover point j^* as this technique includes at the very least one index, which is the starting index that is randomly selected. From the starting index, the next index is selected until $U(0, 1) \geq p_r$ or $|\mathcal{J}| = N$, and p_r is the same crossover probability as mentioned above for binomial crossover. The implementation of exponential crossover is given in Algorithm 4.

Algorithm 3 The pseudo-code algorithm for the binomial crossover technique for DE.

```

 $j^* \sim U(1, N);$ 
 $\mathcal{J} \leftarrow \mathcal{J} \cup \{j^*\};$ 
for each  $j \in \{1, \dots, N\}$  do
    if  $U(0, 1) < p_r$  and  $j \neq j^*$  then
         $\mathcal{J} \leftarrow \mathcal{J} \cup \{j\};$ 
    end if
end for

```

Algorithm 4 The pseudo-code algorithm for the exponential crossover technique for DE.

```

 $\mathcal{J} \leftarrow \{\};$ 
 $j \sim U(0, N - 1);$ 
repeat
     $\mathcal{J} \leftarrow \mathcal{J} \cup \{j + 1\};$ 
     $j = (j + 1) \bmod N$ 
until  $U(0, 1) \geq p_r$  or  $|\mathcal{J}| = N;$ 

```

Selection

Selection refers to the technique that is used to determine which entities are included in the mutation operator to produce a trial vector [45]. Various selection operators have been suggested [154, 155]. With reference to the mutation operator, most DE implementations make use of random selection or the best entity is used as the target vector $\mathbf{x}_{i_1}(t)$. To construct the population for the next generation, deterministic selection is used. As such, a parent is replaced if the offspring produces a better solution than the parent such that $f(\mathbf{x}'_i(t)) \leq f(\mathbf{x}_i(t))$. Engelbrecht [45] states that deterministic selection for the next generation ensures the average fitness of the population does not deteriorate.

General Differential Evolution Algorithm

Algorithm 5 is taken from Engelbrecht [45] and presents the general DE algorithm. The population is initialised by randomly placing entities in the search space such that the positions of the entities are confined to some search boundary. As such, $x_{ij}(t) \sim U(x_{min,j}, x_{max,j})$, where $x_{min,j}$ and $x_{max,j}$ define the search boundaries.

Algorithm 5 The pseudo-code for the general DE heuristic.

```

Set the generation counter,  $t = 0$ ;
Initialise the control parameters,  $\beta$  and  $p_r$ 
while stopping condition not met do
    for each entity  $\mathbf{x}_i(t) \in \mathcal{C}(t)$  do
        Evaluate the fitness,  $f(\mathbf{x}_i(t))$ ;
        Create the trial vector,  $\mathbf{u}_i(t)$  by applying the mutation operator;
        Create an offspring,  $\mathbf{x}'_i(t)$  by applying the crossover operator;
        if  $f(\mathbf{x}'_i(t))$  is better than  $f(\mathbf{x}_i(t))$  then
            Add  $\mathbf{x}'_i(t)$  to  $\mathcal{C}(t + 1)$ ;
        else
            Add  $\mathbf{x}_i(t)$  to  $\mathcal{C}(t + 1)$ ;
        end if
    end for
end while
Return the individual with the best fitness as the solution;

```

As with other heuristics, DE also contains a set of control parameters. These include:

- **Population size:** The population size has a direct influence on the exploration ability of the DE heuristic [45]. The larger the population size, the more differential vectors are available and thus, more directions can be explored.
- **Scaling Factor:** The scaling factor, $\beta \in (0, \infty)$ controls the amplification of the differential variations $(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t))$. A lower scaling factor leads to smaller step sizes and as a result, convergence will take longer. Larger values facilitate exploration, but could cause the algorithm to overshoot. Similar to other heuristics, adaptive mechanisms can be used to dynamically alter the scaling factor throughout the optimisation process.
- **Recombination Probability:** The probability of recombination, p_r has a direct influence on the diversity of the DE heuristic [45]. This parameter

controls the number of elements that are included during crossover. The higher the probability of recombination, the more variation is introduced in the new population. Similar to the scaling factor, dynamic techniques can be used to adjust this the recombination probability during optimisation.

DE/ $x/y/z$ Notation

Many variants of DE have been created and researched [107]. A general notation for DE heuristic variants have been developed by Storn [154, 155]. The notation follows the form $DE/x/y/z$ where x , y and z refer to the components that are used by the particular DE. A breakdown of this notation is provided as follows:

- x : The selection mechanism for the target vector.
- y : The number of difference vectors to include.
- z : The type of crossover operator used.

For this dissertation, *random* and *best entity* selection, a single difference vector and binomial and exponential crossover are considered. This results in the DE notations as follows:

- DE/rand/1/bin
- DE/best/1/bin
- DE/rand/1/exp
- DE/best/1/exp

3.4.3 Genetic Algorithms

EC refers to a collection of nature-inspired optimisation algorithms that lend their foundation to biological evolution. Engelbrecht [45] mentions that EC refers to computer-based problem solving systems that use computational models of evolutionary processes such as natural selection, survival of the fittest and reproduction. Charles Darwin's theory of *natural selection* [29] became the foundation of biological evolution [30]. Engelbrecht [45] summarises the Darwinian theory of evolution as follows. In a world with limited resources and stable populations, each individual competes with others for survival. Those individuals with the “best” characteristics (traits) are more likely to survive and to reproduce and those characteristics will be passed on to their offspring. These desirable characteristics are inherited by

the following generations and (over time) become dominant among the population. Evolution via natural selection of a randomly chosen population of entities can be seen as a search through the space of possible chromosome values. This makes the EC search process a stochastic search for an optimal solution to the given problem.

So far, two population-based MHs have been introduced. These include PSO and DE. The DE heuristic that was presented in Section 3.4.2 is one type of EC algorithm. This section introduces another population-based, nature-inspired optimisation EC algorithm, referred to as *genetic algorithms* (GAs). The details of the implementation of GAs is given in this section. GAs have been widely used to train FFNNs [109, 111, 147].

GAs were first proposed by Fraser [55] and later by Bremermann et al. [15] and Reed et al. [132]. However, Holland [76] is widely regarded as the father of GAs. Similar to DEs, GAs are also nature-inspired population-based MHs and model genetic evolution of entities in a hypothetical population. As with other EAs, GAs implement a number of operators that drive the optimisation process. Primarily, GAs implement selection, modelling survival of the fittest and crossover, modelling reproduction. The remainder of this section presents the aforementioned operators in more detail.

For sake of clarity, a generic EC algorithm is presented first, as a frame of reference. The generic EC algorithm is referred to as the canonical GA (CGA) and was proposed by Holland [76]. The generic EC algorithm is taken from [45] and is presented in Algorithm 6 below.

Algorithm 6 The pseudo-code for the generic EC heuristic.

```

Let  $t = 0$  be the generation counter;
Create and initialise a  $J$ -dimensional population  $\mathcal{C}(0)$ , to consist of  $N$  individuals;
while stopping condition not met do
    Evaluate the fitness,  $f(\mathbf{x}_i(t))$  of each individual  $\mathbf{x}_i(t)$ ;
    Perform reproduction to create offspring;
    Select the new population,  $\mathcal{C}(t + 1)$ ;
    Advance to the new generation, i.e.  $t = t + 1$ 
end while

```

From Algorithm 6 above, it can be seen that a number of components influence the search process. These include:

- **Encoding:** Refers to the representation of a candidate solution to some optimisation problem as the of some entity. Every element in the chromosomes

is referred to as a *gene* or *allele*.

- **Fitness Function:** Refers to the objective function that measures the fitness of an entity. Fitness refers to the survivability of an entity and measures the strength of a candidate solution represented by the entity's chromosomes.
- **Initialisation:** Refers to the initialisation strategy used to generate the initial population. Often entities' chromosomes are uniformly sampled in the feasible search space for the underlying optimisation problem.
- **Selection:** Refers to the techniques that are used to select entities for reproduction and generation of the new population as well as the selection of genes for mutation. Selection is implemented through selection operators.
- **Reproduction:** Refers to the generation of the next population and is implemented through crossover operators.

The initial implementations of EC heuristics such as GAs did not contain a mutation operator as it was only introduced later [45]. The following sections provide the crossover, mutation and selection operators for GAs.

Crossover

As with DE, crossover operators model the reproduction of entities in the population. Broadly speaking, the crossover operators can be divided into three main categories [45] and are based on arity of the operator i.e. the number of parents used for reproduction.

- **Asexual:** Offspring are generated from one parent.
- **Sexual:** Offspring are generated from two parents and can produce one or two offspring.
- **Multi-recombination:** Offspring are generated from more than two parents and can produce one or more offspring.

Engelbrecht [45] mentions that crossover operators can be further classified based on their encoding/representation scheme. These include binary-specific operators used for binary representations and operators focused on floating-point representations. Since the focus is put on training FFNNs, from this point on, floating-point representations are assumed.

During crossover, parents are selected using a selection operator. As with DEs, recombination is applied probabilistically and thus, selection of a parent does not guarantee reproduction. Each parent has a probability p_c of producing offspring. Usually a high crossover probability is used [45]. In addition to recombination, GAs implement a replacement policy where fit offspring can replace weaker parents in the population.

Although floating-point representations of chromosomes are assumed, binary crossover operators can also be used, since they produce a mask that defines how parents are recombined. Specifically, in the context of this dissertation, focus is put on *uniform* crossover. Uniform crossover refers to a crossover operator where an J -dimensional crossover mask is generated randomly [159]. Uniform crossover is illustrated in Figure 3.5 below and the algorithm for uniform crossover is given in Algorithm 7. In Algorithm 7, p_x is the bit-swapping probability.

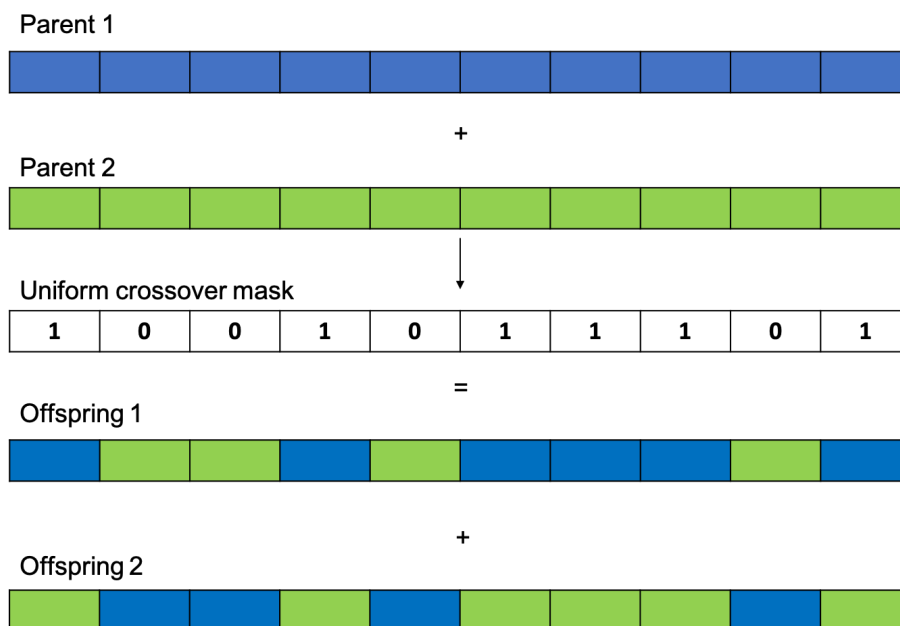


Figure 3.5: An illustration of the uniform crossover operator as it applies to sexual recombination, resulting in two new offspring.

Mutation

The mutation operator is applied in order to introduce new genetic material into an existing entity [45]. In doing so, diversity is added into the genetic characteristics of the population. Mutation is applied at a certain mutation probability p_m , to each gene of the offspring $\mathbf{x}_i(t)$ to produce mutated offspring $\mathbf{x}'_i(t)$. Engelbrecht [45] mentions that the mutation probability, also referred to as the mutation rate, is

Algorithm 7 The pseudo-code for the uniform crossover operator as used by [GAs](#).

```

Initialise the mask,  $m_j(t) = 0, \forall j = 1, \dots, J$ ;
for  $j = 1$  to  $J$  do
    if  $U(0, 1) \leq p_x$  then
         $m_j(t) = 1$ ;
    end if
end for

```

usually small such that $p_m \in [0, 1]$ to ensure that good solutions are not distorted too much.

Similar to the crossover operator, mutation operators can be classified according to the representation scheme used. In the context of training [FFNNs](#), binary crossover operators such as the *uniform* mutation operator can be used to generate a mutation mask that specifies which genes are mutated. For the purposes of this dissertation, the application of the mutation operator on $x_{ij}(t)$ results in a small update step for that gene such that $x'_{ij}(t) = x_{ij}(t) + v_{ij}(t)$. In this case, $v_{ij}(t)$ is sampled using Glorot uniform sampling within the bounds $(-limit, limit)$, as was presented in Chapter 2. An adaptation of the uniform mutation operator is provided in Algorithm 8 below.

Algorithm 8 The pseudo-code for the uniform mutation operator as used by [GAs](#).

```

for  $i = 1$  to  $N$  do
    for  $j = 1$  to  $J$  do
        if  $U(0, 1) \leq p_m$  then
            Sample update step  $v_{ij}(t) \sim U(-limit, limit)$ 
             $x'_{ij}(t) = x_{ij}(t) + v_{ij}(t)$ ;
        end if
    end for
end for

```

An illustration of the adapted uniform mutation operator is presented in Figure 3.6 below.

Selection

Selection is a widely used concept in all [EAs](#) and models survival of the fittest in the evolutionary context. The main idea behind the selection operator is to emphasise better solutions [45]. Selection is applied in two of the main steps of [EAs](#). These

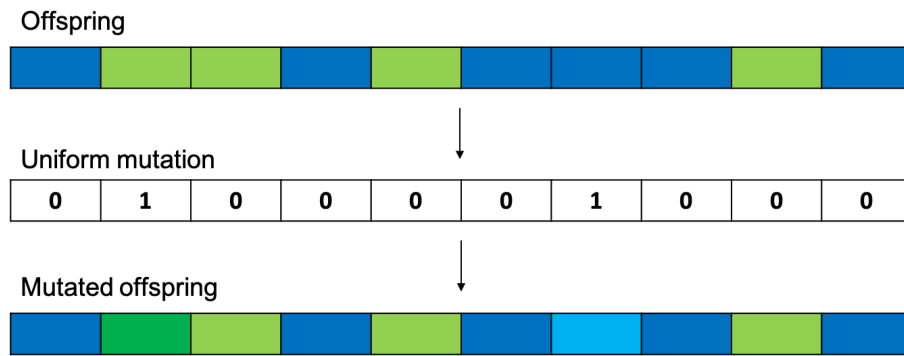


Figure 3.6: An illustration of the adapted uniform mutation operator as it applies to mutated offspring.

include:

- **Selection of the new population:** A new population of candidate solutions are selected at the end of each generation to serve as the population for the next generation. The new population can be selected from both the parents and offspring. The selection operator is thus responsible for ensuring that good entities survive to the next generation.
- **Reproduction:** Offspring is created through the crossover and/or the mutation operators. In terms of crossover, good solutions should have a high probability of reproducing to ensure that offspring contain genetic material of the best entities. In terms of mutation, selection mechanisms should focus on weaker entities. By mutating weak entities, the hope is to introduce better traits, increasing their chance to survive.

Engelbrecht [45] mentions that selection operators are characterised by their *selective pressure*. Selective pressure is defined as the speed at which the best entity's solution will occupy the entire population by repeated application of the selection operator alone [5]. A selection operator with a high selective pressure rapidly decreases the diversity in the population, possibly leading to premature convergence. A high selective pressure limits the exploration abilities of the population. Selection operators should maintain a balance between exploration and exploitation.

Various selection mechanisms have been proposed. For the purposes of this dissertation, focus is put on the following selection mechanisms and concepts.

- **Tournament Selection:** Tournament selection selects a group of N_t entities randomly from the population such that $N_t < N$, where N is the population size. The performance of the selected N_t entities are then compared and the

best entities from this group are selected and returned by the operator. It should be mentioned that for sexual crossover of two parents, tournament selection is applied twice, first for the first parent and then again for the second parent. Engelbrecht [45] mentions that tournament selection prevents the best entities from dominating, provided that N_t is not too large. This results in a lower selective pressure. If N_t is too small, the chances that bad entities will be selected increase.

- **Rank-based Selection:** Rank-based selection uses the rank-ordered fitness values to determine the probability of selection and not the absolute fitness value. Engelbrecht [45] mentions that the advantage of this approach is that the best entities will not dominate the selection process. Non-deterministic linear sampling selects an entity $\mathbf{x}_i(t)$ such that $i \sim U(0, U(0, N - 1))$. Importantly, in the context of a minimisation problem, entities are first sorted in decreasing order of fitness value, assuming that the best heuristic is then contained at index 0, while the worst entity is contained at index $N - 1$.
- **Elitism:** Elitism refers to the process of ensuring that the best entities from the current population survive to the next generation. The best entities are simply passed on to the next generation without mutation. The more entities that survive to the next generation, the lower the diversity of the new population.

3.5 Summary

This chapter provided detailed background information on heuristics. Different heuristics were presented that have been shown to be able to train **FFNNs**. Two main groups of heuristics were presented and discussed in detail. These include gradient-based heuristics and *meta-heuristics* (**MHs**). A number of different variants for each group have been presented and was discussed in detail.

Chapter 4

Hyper-Heuristics

“The capacity to learn is a gift; the ability to learn is a skill; the willingness to learn is a choice.”

- Brian Herbert

Chapter 3 introduced the concept of a heuristic as an optimisation algorithm that is used in a search process that seeks out optimal solutions to an optimisation problem. Finding the best heuristic to use for a given optimisation problem is non-trivial. Furthermore, finding optimal solutions is often dependent on using the appropriate configurations. In the context of training FFNNs, these configurations could include heuristic hyper-parameters and model architecture. It is often the case that the application of a specific configuration only applies to a specific problem and does not necessarily translate or generalise to other problems [93].

The optimal configuration of heuristics to use, along with hyper-parameters and model architecture, lies in some hyper-dimensional *configuration space* yielding yet again, another optimisation problem on its own. Traditionally, if multiple configurations are to be considered, an empirical process of trial and error is executed. Trial and error approaches are time consuming and laborious.

One particular approach to the configuration optimisation problem mentioned above, is to try and automate the process to find the best configuration. One such automated process involves considering many different configurations in an iterative fashion, sweeping over different configurations parameters within the configuration space. Techniques such as *Design of Experiments* (DoE), proposed by Fisher et al. [53], follow an iterative approach, where a configuration space is defined in terms of its extremities, while retrieving candidate configurations by interpolating over the configuration space. Iterative approaches implement a form of offline learning where the optimisation of configuration only happens after all configurations have been considered and evaluated.

More modern techniques automate the search for optimal configuration as part of the underlying search process itself, resulting in a form of dynamic configuration during the search process. An example of such a technique is to dynamically adjust the heuristic *hyper-parameters* as part of the optimisation process. This field of study is known as meta-learning [57]. The term *learning* refers to the optimisation of the configuration and hyper-parameters.

A recent suggestion related to the field of meta-learning is to dynamically select and/or adjust the heuristic characteristics and behaviours used throughout the search process and not just the heuristic hyper-parameters. This approach focuses on the hybridisation of heuristic paradigms. By dynamically combining the best of different paradigms throughout the learning process, a trade-off can be made between focussing on a particular solution (exploitation), in comparison to seeking out novel solutions (exploration) during the search process. The dynamic trade-off between exploration and exploitation could also be seen as a dynamic trade-off between the strengths and weaknesses of different heuristics during the search process.

One such form of hybridisation of heuristic paradigms is referred to as *heterogeneous approaches*. Heterogeneous approaches make use of different *search behaviours* by selecting from a behaviour pool. Heterogeneous approaches have shown to balance the trade-off between exploration and exploitation [116].

Another form of hybridisation of heuristic paradigms is that of hybridisation of different *heuristics* that are applied to some optimisation problem [20]. These methods are referred to as *hyper-heuristics* (HHs) and focus on finding the best heuristic in *heuristic space* to solve a specific problem. This dissertation takes a particular interest in developing a novel selection HH that can be used to train FFNNs. This chapter provides the reader with the necessary background information of meta-learning and HHs. The remainder of the chapter is structured as follows:

- **Section 4.1** provides the reader with a brief definition and discussion on meta-learning.
- **Section 4.2** presents the concept of a HH and a formal definition is provided. It is shown how HHs implement a form of meta-learning.
- **Section 4.3** provides a detailed discussion on the classification and types of HHs. Distinction is made between online and offline learning mechanisms. Discussions follow on selection and generation HHs, and construction and perturbation mechanisms.
- **Section 4.4** provides a brief summary of the chapter.

4.1 Meta-Learning

Meta-learning is inspired by a branch of meta-cognition that is concerned with learning about *learning processes*. Meta-learning studies how learning systems can increase efficiency through experience [174] and is broadly defined as the learning process concerned with the concept of *learning to learn* [164]. The goal of meta-learning is to provide a learning/optimisation process that is flexible to the problem domain or task under consideration.

Vilalta et al. [174] mention that meta-learning differs from base-learning in the scope of the level of adaptation. Meta-learning is concerned with learning how to choose the right configuration and biases dynamically. On the contrary, for *base-learning*, biases and configurations are predefined and fixed a priori. Thrun et al. [164] mention that meta-learning aims at discovering ways to dynamically search for the best learning strategy as the number of tasks increase, implying some form of generalisation of the learning process to other problems.

In the context of training FFNNs, meta-learning could be applied to dynamically adjust the hyper-parameters of a heuristic during the training process. K -fold cross-validation is a common technique that applies dynamic adjustment of hyper-parameters during the training process [2]. K -fold cross-validation is based on the idea that the optimal hyper-parameters for a given problem can be found by evaluating the performance of the heuristic on a subset of the training data.

Dynamic adjustment of hyper-parameters might not be enough. Different search behaviours or techniques might be needed to balance the trade-off between exploration and exploitation. The following section provides a brief introduction to the concept of a *hyper-heuristic* (HH), a modern approach that builds on the foundations of meta-learning.

4.2 What are Hyper-Heuristics?

The term *hyper-heuristic* (HH) was first used in 1997 by Burke et al. [19] and was used to describe a protocol that combines several AI methods in the context of automated theorem proving. The term was independently used in 2000 by Cowling et al. [27] to describe “heuristics to choose heuristics”.

Burke et al. [19] define HHs as search methods or learning mechanism for selecting or generating heuristics to solve computational search problems. Burke et al. [18] mentions that a HH is a high-level approach that, given a particular problem instance and a number of low-level heuristics, can select and apply an appropriate low-level

heuristic at each decision point.

Grobler [64] states that HHs promote the design of more generally applicable search methodologies and tend to perform relatively well on a large set of different problems, in contrast to specialised algorithms, which typically focus on outperforming the state-of-the-art for a single application.

HHs implement a form of meta-learning that is concerned with the selection of the best heuristic from a pool of heuristics to solve a given problem. In the context of population-based HHs, an entity pool exists that represent a pool of candidate solutions to the given problem. Each entity in the entity pool is assigned its own low-level heuristic from the heuristic pool.

The selection of the best heuristic to apply to a candidate solution, is based on the performance of the heuristic relative to that particular candidate solution at a particular point in the search process. It can be said that HHs are concerned with finding the best heuristic in *heuristic space*, while the underlying low-level heuristics find solutions in the feasible *search/solution space*. HHs, introduce a domain barrier that separates the information that the high-level heuristic and low-level heuristics use during the search process and is illustrated in Figure 4.1 below.

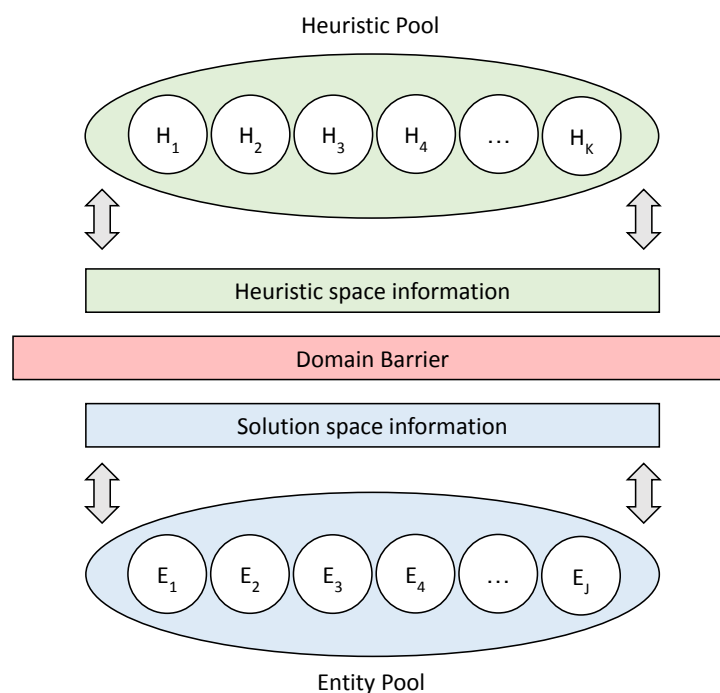


Figure 4.1: An illustration of the domain barrier that exists as a result of the separation between the low-level heuristics and the high-level heuristic as introduced by HHs.

Figure 4.1 shows the separation of information that is accessible to the entity pool and heuristic pool. At a higher level abstraction, the high level heuristic implemented by HHs, do not make use of domain specific information such as an entity's position

or gradient from the search space.

To summarise, Grobler [64] highlights two fundamental ideas behind HHs. Firstly, the recognition that the process of selecting or designing efficient hybrid and/or cooperative heuristics can be regarded as a computational search problem in itself. Secondly, there is significant potential to improve search methodologies by the incorporation of learning mechanisms that can adaptively guide the search. These two fundamental ideas have inspired different types of hyper-heuristics [19].

In the general context of optimisation, many different types of HHs have been implemented and applied to many different problems. Some notable examples include the simulated annealing-based HH by Dowsland et al. [37], the tabu-search HH of Burke et al. [19], the heterogeneous meta-hyper-heuristic by Grobler et al. [65] and work done by Van der Stockt et al. [171] on the analysis of selection hyper-heuristics for population-based MHs in real-valued dynamic optimisation. Research on the application of HHs in the context of FFNN training is still scarce. Nel [115] provides the first research in this field, applying a HH to FFNN training. The following section provides a framework for HH classification.

4.3 Classification of Hyper-Heuristics

Burke et al. [19] proposed a modern classification scheme used to classify HHs. According to the proposed classification scheme, HHs are classified in two dimensions. These include the *source of feedback* used during learning and the nature of the *heuristic search space*. Figure 4.2 presents the classification scheme as proposed by Burke et al. [19].

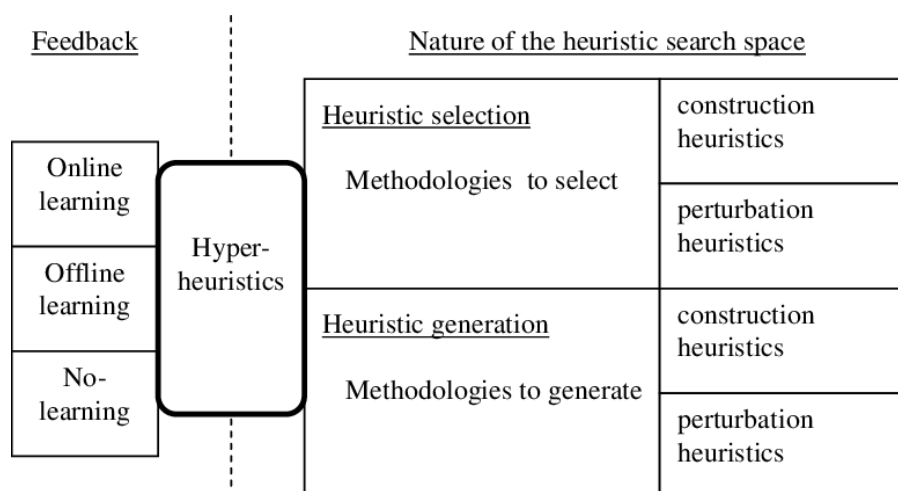


Figure 4.2: A classification of HH approaches, according to two dimensions: (i) the source of feedback used during learning, and (ii) the nature of the heuristic search space.

4.3.1 Source of Feedback

HHs make use of feedback from the search process to adapt and guide the search process. Some HHs implement a learning mechanism that utilises this feedback. Learning HHs can implement a form of *online learning* or *offline learning*.

HHs that implement online learning, implement a form of learning that continues to take place while the algorithm is solving an instance of the underlying optimisation problem. Task-dependent, local properties can be used by the high-level heuristic to determine the appropriate low-level heuristic to apply at various points in the search process. Examples of online learning approaches within HHs include the use of *reinforcement learning* (RL) and *meta-heuristics* (MHs) as high-level search strategies.

HHs that implement offline learning, implement a form of learning where knowledge, in the form of rules or programs, is gathered from a set of training instances and is then applied independently from the search process itself. Examples of offline learning approaches within hyper-heuristics include *learning classifier systems* and *case-based reasoning*.

4.3.2 Heuristic Search Space

According to the classification proposed by Burke et al. [19], the second dimension of classification involves the nature of the heuristic search space. Distinction is made between heuristic *selection* and heuristic *generation*.

Selection vs. Generation

Selection HHs implement a high-level search mechanism that is used to determine which heuristic to apply to the underlying optimisation problem at a given point in the optimisation process. Selection mechanisms can include probabilistic approaches or high level *meta-heuristics* (MHs). On the contrary, heuristic generation methodologies implement a mechanism that generate new heuristics from a pool of *components* of various heuristics. Generation HHs often make use of *evolutionary algorithms* (EAs) [19].

Selection HHs consist of two components, including a low-level heuristic selection strategy and a move acceptance strategy. Low-level heuristic selection can be done in a simple, *non-adaptive* way. No learning is involved in these approaches. For non-adaptive techniques, heuristic selection is based on a predefined heuristic ordering that is generated either randomly or in an ordered cycle that is repeated throughout the optimisation process [27]. As an alternative, heuristic selection may incorporate

an *adaptive (online learning)* mechanism, based on the probabilistic weighting of the low-level heuristics [18] or some type of performance statistics [27].

A move acceptance strategy is the mechanism by which the application of a low-level heuristic is either accepted or rejected. In general, a move is accepted or rejected based on the quality of the move and the current solution during a single point search. Burke et al. [18] mention that many move acceptance strategies have been explored within HHs.

Move acceptance strategies can be divided into two categories. These include *deterministic* and *non-deterministic* move acceptance strategies. Deterministic move acceptance strategies generate the same result for the same candidate solution(s) used for the move acceptance test. Non-deterministic move acceptance strategies can involve other parameters, such as the current time, or a sampling operation that yields possibly different outcomes when repeated for the move acceptance test [18].

Construction vs. Perturbation

Selection and generation HHs can be further classified in terms of *construction* and *perturbation* mechanisms. According to Burke et al. [19], construction approaches build a solution incrementally. Starting with an empty solution, the goal is to intelligently select and use construction heuristics to gradually build a complete solution. As such, the HH framework is provided with a set of pre-existing construction heuristics.

Perturbation approaches refer to approaches that start with a complete solution, generated either randomly or using simple construction heuristics. Perturbation heuristics try to iteratively improve the current solution. As such, the hyper-heuristic framework is provided with a set of neighbourhood structures and/or simple local searchers.

4.4 Summary

This chapter provided the reader with the necessary background information on meta-learning and HHs. Formal definitions were given, followed by detailed discussions. A modern classification scheme for HH as proposed by Burke et al. [19] was presented in detail. Distinction is made between the source of feedback information used during learning and the nature of the heuristic search space. Detailed discussions on selection and generation heuristics are provided, followed by a further classification of construction and perturbation mechanisms.

This chapter concludes the background information of heuristics in general.

The following chapter aims to provide the reader with the necessary background information on statistics and probability theory.

Chapter 5

Probability

“Probability theory is nothing but common sense reduced to calculation.”

- Pierre-Simon Laplace

Probability theory and statistics can be traced back to as early as the 18th century. In 1718, De Moivre [33] published the *Doctrine of Chance*; a book that is widely regarded as the first published book on probability theory. In 1763, Thomas Bayes [6] published an article titled *An Essay towards solving a Problem in the Doctrine of Chances* where the first version of Bayes’ theorem was introduced.

Probability theory, statistics and *machine learning* (ML) are transdisciplinary fields with many shared concepts. There are many examples of how probability theory has been incorporated into ML research. In 1991, Denker et al. [35] proposed a way to transform *artificial neural network* (ANN) outputs to probability distributions. In 1993, Neal [114] developed a *Markov Chain Monte Carlo* (MCMC) sampling algorithm for *Bayesian neural networks* (BNNs). These are but a few examples of the role that probability theory has played in ML research in the past.

Chapter 4 provided the concept of a *hyper-heuristic* (HH). This dissertation aims to develop a selection HH that makes use of probability theory to select the best HH to solve a given problem. This chapter aims to provide the necessary background information on probability theory and statistics. These are large fields and focus is put on the elements that are required to formulate the proposed *Bayesian hyper-heuristic* (BHH). The remainder of the chapter is structured as follows:

- **Section 5.1** provides a brief overview of what probability is and how it is used.
- **Section 5.2** presents the concept of conditional probability.

- **Section 5.3** presents the two laws of probability related to the intersection and union of multiple events.
- **Section 5.4** introduces Bayes' theorem, the fundamental theorem upon which the BHH is built.
- **Section 5.5** presents the reader with relevant probability distributions.
- **Section 5.6** presents the reader with relevant conjugate prior probability distributions.
- **Section 5.7** presents *Bayesian* statistics. Brief discussions follow on the *frequentist* view and *Bayesian* view of probability. Detailed discussions follow on Bayesian optimisation methods such as *Bayesian analysis*.
- **Section 5.8** provides a brief summary of the chapter.

5.1 Overview of Probability

In everyday conversation, the term *probability* is a measure of belief in the occurrence of a future event [175]. Probability is a necessary tool used in many fields including physics, biology, chemistry and computer science. These fields contain many cases that generate observations that cannot be predicted with absolute certainty [175]. Probability can be inferred and confirmed through past events. These events are referred to as *random* or *stochastic* events. The probability that a certain event, A , might occur is denoted by $P(A)$. Although these random events cannot be predicted with absolute certainty, the relative frequency with which they occur over many trials, is often remarkably stable.

Consider flipping an unbiased, fair coin. The coin has two possible outcomes. It can conclude that each side has a $\frac{1}{2}$ or 50% chance of occurring. In statistics, the decimal probability notation is used, where $0 \leq P(A) \leq 1$. Suppose the fair coin is thrown 10 times, there is no guarantee of observing 0.5 heads and 0.5 tails. There is some probability (although small) that the coin might fall on heads 0/10 times. The probability of such an event occurring is 0.0009765625. In the coin flip example, the *central limit theorem* (CLT) shows that the normalised sum of events tends toward a normal distribution with a mean value of 0.5, if the number of events observed, N , is large [175]. The larger the value of N , the higher the confidence of mean probability and relative frequency of the event. The stable long-term relative frequency by which a random event occurs, provides an intuitive and meaningful measure of belief that a certain event will occur again at some point in the future [175].

Probability can also be expressed over multiple random events. Multiple random events can be considered together, dependently or conditionally. The following sections provide insight into conditional and joint probabilities of multiple random events.

5.2 Conditional Probability and Independence

The occurrence of a given random event, A , can often be conditional on the occurrence of another event, B . In the field of medicine, an example of this is to calculate the probability of a certain diagnosis of a sick patient given his/her symptoms. The aforementioned relationship between events is referred to as the conditional probability between two events. The conditional probability is expressed as $P(A|B)$ and is read *the probability of A given B*. On the contrary, the *unconditional* probability is the probability of an event, not dependent on any other. The conditional probability of A given B can be expressed as is given in Definition 1 below [175].

Definition 1 (Conditional Probability). *The conditional probability of an event A , subject to the occurrence of event B is expressed as*

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (5.1)$$

where $P(B) > 0$. Note that conditional probability does not suggest causation. If an event A has a high probability of occurring after observing event B , it does not necessarily mean that A is caused by B . Conditional probability simply expresses the dependence amongst events.

It could also be the case that the outcome of observing event A is not affected by the occurrence of B . In this case, it is said that events A and B are independent. The independence between two events are expressed in Definition 2 below.

Definition 2 (Independence of Events). *Two events, A and B , are said to be independent of each other if, and only if the following criteria hold:*

- $P(A|B) = P(A)$
- $P(B|A) = P(B)$
- $P(A \cap B) = P(A)P(B)$

Otherwise, events A and B are said to be dependent random events.

5.3 Two Laws of Probability for Multiple Events

Suppose there are two random events, A and B , then one can calculate the probability of the union and intersection of these events. From the aforementioned concept, two laws of probability can be formulated. These are referred to as the *multiplicative* and *additive* laws of probability [175] and is given below in Theorems 1 and 2 respectively.

Theorem 1 (The Multiplicative Law of Probability). *The probability of the intersection of two events, A and B , is given as*

$$\begin{aligned} P(A \cap B) &= P(A)P(B|A) \\ &= P(B)P(A|B) \end{aligned} \tag{5.2}$$

If A and B are independent, then

$$P(A \cap B) = P(A)P(B) \tag{5.3}$$

Proof. Proof is given from Definition 1. □

Theorem 2 (The Additive Law of Probability). *The probability of the union of two events, A and B , is given as*

$$P(A \cup B) = P(A) + P(B) - P(B \cap A) \tag{5.4}$$

Proof. The geometric proof of the additive law of probability is given by the Venn Diagram presented in Figure 5.1. Note that $A \cup B = A \cup (\bar{A} \cap B)$, where A and $\bar{A} \cap B$ are mutually exclusive events. Furthermore, consider that $B = (\bar{A} \cap B) \cup (A \cap B)$, where $\bar{A} \cap B$ and $A \cap B$ are mutually exclusive. Then $P(A \cup B) = P(A) + P(\bar{A} \cap B)$ and $P(B) = P(\bar{A} \cap B) + P(A \cap B)$. The equality on the right implies that $P(\bar{A} \cap B) = P(B) - P(A \cap B)$. By substituting the expression for $P(\bar{A} \cap B)$ into the expression for $P(A \cup B)$, the resulting expression $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ is obtained. □

5.4 Bayes' Theorem

Bayes' theorem, named after Thomas Bayes, describes the probability of an event, A , based on prior knowledge of conditions that might be related to A [181]. In order

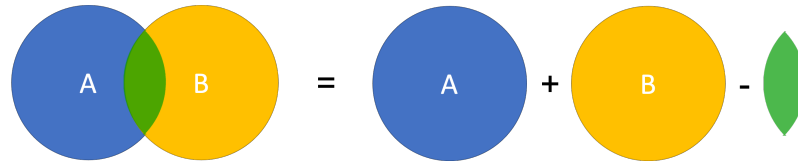


Figure 5.1: A Venn-Diagram showing the proof of the additive law of probability for multiple events.

to derive the formal theorem and proof, first consider Definition 3 below [181].

Definition 3. For some positive integer, K , let the sets B_1, B_2, \dots, B_K be such that

1. $S = B_1 \cup B_2 \cup \dots \cup B_K$
2. $B_i \cap B_j = \emptyset$, for $i \neq j$

Then the collection of sets $\{B_1, B_2, \dots, B_K\}$ is said to be a partition of S , the union of mutually exclusive subsets.

Theorem 3 (Bayes' theorem). Assume that $\{B_1, B_2, \dots, B_K\}$ is a partition of S such that $P(B_i) > 0$, for $i = 1, 2, \dots, K$ then

$$P(B_j|A) = \frac{P(A|B_j)P(B_j)}{\sum_{i=1}^K P(A|B_i)P(B_i)} \quad (5.5)$$

Proof. The proof follows from the definition of conditional probability as was presented in Section 5.2:

$$\begin{aligned} P(B_j|A) &= \frac{P(A \cap B_j)}{P(A)} \\ &= \frac{P(A|B_j)P(B_j)}{\sum_{i=1}^K P(A|B_i)P(B_i)} \end{aligned} \quad (5.6)$$

□

One of the many applications of Bayes' theorem is to do statistical inference. Bayes' theorem expresses how a degree of belief, expressed as a probability, should rationally change to account for the availability of related evidence.

5.5 Probability Distributions

Probability distributions are mathematical functions that give the probabilities of the occurrences of different possible outcomes in the experiment. The theory and equations presented in the following sections were all taken from Wackerly et al. [175].

5.5.1 Beta Probability Distribution

The Beta probability distribution is a family of univariate, continuous, probability distributions over some x , with support on the interval $[0, 1]$ [175]. It is parameterised by two shape parameters $\alpha > 0$, $\alpha \in \mathbb{R}$ and $\beta > 0$, $\beta \in \mathbb{R}$. The Beta probability distribution is denoted as $Beta(\alpha, \beta)$. The *probability density function* (PDF) of the Beta probability distribution is given as

$$P(x|\alpha, \beta) = f_{Beta}(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (5.7)$$

The normalising constant, $B(\alpha, \beta)$, is defined as

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)} \quad (5.8)$$

where Γ is the Gamma function defined as

$$\Gamma(n) = (n-1)! \quad (5.9)$$

It should be noted that the Gamma function can also be written as

$$\Gamma(n+1) = n! \quad (5.10)$$

The control parameters α and β determine the shape of the distribution. There exists a special case where $\alpha = \beta$. This is referred to as the *symmetric* Beta probability distribution. In the case where $\alpha = \beta = 1$ the distribution is equivalent to the uniform distribution over all points in its support. The Beta probability distribution for various values of α and β , including the symmetric version is illustrated in Figure 5.2.

The expected value of x is given as

$$E[x] = \frac{\alpha}{\alpha + \beta} \quad (5.11)$$

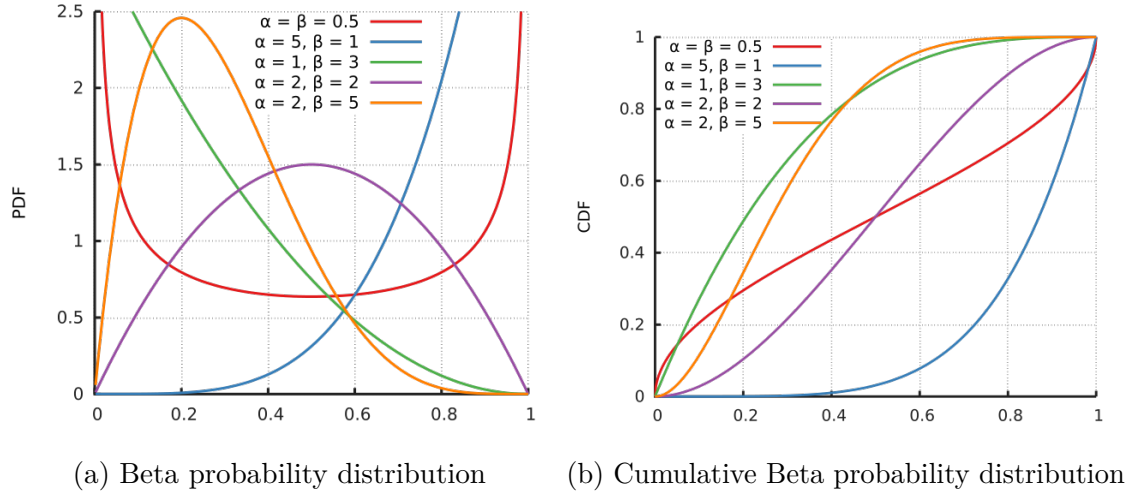


Figure 5.2: An illustration of the Beta probability distribution (left) [77] as well as the cumulative Beta probability distribution (right) [78] for various values of α and β .

Similarly, the expected value of the natural logarithm of x is calculated as

$$E[\ln(x)] = \psi(\alpha) - \psi(\alpha + \beta) \quad (5.12)$$

where ψ is the logarithmic derivative of the Gamma function, called the *Digamma* function. The Digamma function is given as

$$\psi(n) = \frac{d}{dn} \ln(\Gamma(n)) = \frac{\Gamma'(n)}{\Gamma(n)} \quad (5.13)$$

Finally, the mode of the distribution is given as

$$M[x] = E[x] - 1 = \frac{\alpha - 1}{\alpha + \beta - 2} \quad (5.14)$$

5.5.2 Dirichlet Probability Distribution

The Dirichlet probability distribution is a family of multivariate continuous probability distributions over some \mathbf{x} in K dimensions [175]. The Dirichlet probability distribution is a multivariate generalisation of the Beta probability distribution and is thus sometimes referred to by its alternative name, i.e. the multivariate Beta probability distribution. The Dirichlet probability distribution is parameterised by some vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$, $\forall_{k=1}^K \alpha_k > 0, \alpha_k \in \mathbb{R}$. The parameters, $\boldsymbol{\alpha}$, are referred to as the concentration parameters. The Dirichlet probability distribution of order $K \geq 2$ with parameters $\boldsymbol{\alpha}$, denoted $Dir(\boldsymbol{\alpha})$, has a PDF given as

$$P(\mathbf{x}|\boldsymbol{\alpha}) = f_{Dir}(\mathbf{x}; K, \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^K x_k^{\alpha_k - 1} \quad (5.15)$$

The normalising constant, $B(\boldsymbol{\alpha})$, is defined as

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\alpha_0)} \quad (5.16)$$

where α_0 is defined as

$$\alpha_0 = \sum_{k=1}^K \alpha_k \quad (5.17)$$

Importantly, the set $\{x_k\}_{k=1}^K$ belongs to the standard $K - 1$ probability simplex S , meaning that $x_K = 1 - \sum_{k=1}^{K-1} x_k$ with support $\forall_{k=1}^K x_k \in [0, 1]$. Under the simplex S , this means that the sum over all values of the vector \mathbf{x} must be 1. The simplex can thus be rewritten as $\sum_{k=1}^K x_k = 1$.

Similar to the Beta probability distribution, $\boldsymbol{\alpha}$ determines the shape of the distribution in K dimensions and thus, there also exist a special case, referred to as the *symmetric* distribution when $\forall_{k=1}^K \alpha_k = c$, where c is some constant. In the case where $c = 1$, the distribution is referred to as a *flat* distribution and yields the uniform distribution over all points in S . The Dirichlet probability distribution of order $K = 3$ for various values of $\boldsymbol{\alpha}$, including the symmetric version, is presented in Figure 5.3.

The expected value of x_k is calculated as

$$E[x_k] = \frac{\alpha_k}{\alpha_0} \quad (5.18)$$

Similarly, the expected value of the natural logarithm of x_k is calculated as

$$E[\ln(x_k)] = \psi(\alpha_k) - \psi(\alpha_0) \quad (5.19)$$

where ψ is the Digamma function as defined in Equation (5.13). Finally, the mode of the distribution is given as

$$\begin{aligned} M[x_k] &= E[x_k] - K^{-1} \\ &= \frac{\alpha_k - 1}{\alpha_0 - K} \end{aligned} \quad (5.20)$$

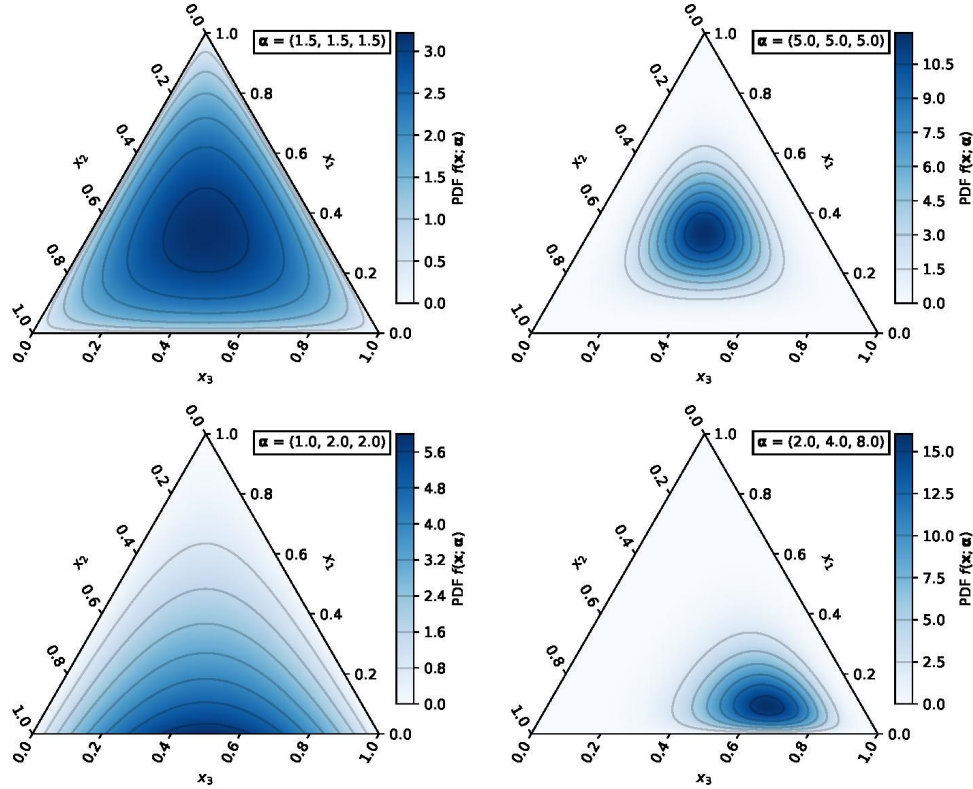


Figure 5.3: The *probability density functions* (PDFs) for the Dirichlet probability distribution over the 2-simplex. The concentration parameters, α , are varied. The values of the PDF are shown by the colour maps with contour lines at equal values as indicated in the colour bars [117].

5.5.3 Bernoulli Probability Distribution

The Bernoulli probability distribution is a discrete probability distribution over some random variable x that takes the value of 1 with probability θ and 0 with probability $1 - \theta$ [175]. The Bernoulli probability distribution is denoted as $Ber(\theta)$ with support $x \in \{0, 1\}$. In probability theory, the Bernoulli probability distribution is often used to explain the possible outcomes of a single experiment that asks a *yes-no* question such as flipping a coin. The outcome of such an experiment is a Boolean value. The Bernoulli probability distribution has a PMF, given as

$$P(x|\theta) = f_{Ber}(x; \theta) = \begin{cases} \theta & \text{if } x = 1 \\ 1 - \theta & \text{if } x = 0 \end{cases} \quad (5.21)$$

Equation (5.21) can also be expressed as

$$f_{Ber}(x; \theta) = \theta^x (1 - \theta)^{1-x} \quad (5.22)$$

The mean of the Bernoulli probability distribution approaches θ over many samples according to the CLT [63]. Figure 5.4 illustrates a fair-coin flipping simulation. The mean converges to 0.5 for various sample sizes.

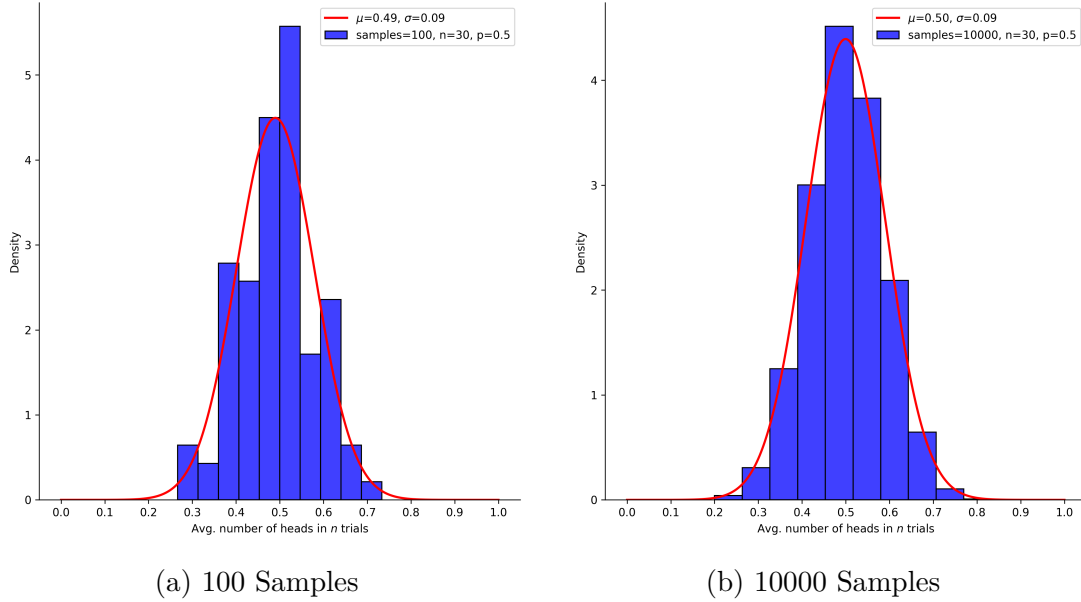


Figure 5.4: An illustration of the coin-flip simulation for different sample sizes that show the convergence of the mean as per the *central limit theorem* (CLT).

The expected value of the distribution is thus given as

$$E[x] = \theta \quad (5.23)$$

The mode of the distribution is given as

$$M[x] = \begin{cases} 0 & \text{if } \theta < 0.5 \\ 0, 1 & \text{if } \theta = 0.5 \\ 1 & \text{if } \theta > 0.5 \end{cases} \quad (5.24)$$

5.5.4 Binomial Probability Distribution

The Binomial probability distribution is a discrete probability distribution over a random variable x taking on a number of successes in N sequential independent experiments that each ask a *yes-no* question [175]. The probability of a single independent experiment yielding a success is given as θ and the Binomial probability distribution is denoted as $\text{Bin}(N, \theta)$, with support $x \in \{0, 1, \dots, N\}$. It should be noted that the Binomial probability distribution is an extension of the Bernoulli

probability distribution over N independent sequential experiments, and thus each experiment also yields some Boolean outcome. When $N = 1$, the experiment is referred to as a Bernoulli trial, and the distribution is just a Bernoulli probability distribution. When $N > 1$, the sequence of outcomes is referred to as a Bernoulli process. The PMF of the Binomial probability distribution is given as

$$P(x|\theta; N) = f_{Bin}(x; N, \theta) = \binom{N}{x} \theta^x (1 - \theta)^{1-x} \quad (5.25)$$

Similar to the Bernoulli probability distribution, the mean of the Binomial probability distribution is $N\theta$ given the CLT as was shown in Figure 5.4. The expected value of the Binomial probability distribution is thus given as

$$E[x] = N\theta \quad (5.26)$$

The mode of the distribution is given as

$$\begin{aligned} M[x] &= E[x] + \theta \\ &= N\theta + \theta \\ &= (N + 1)\theta \end{aligned} \quad (5.27)$$

5.5.5 Categorical Probability Distribution

The Categorical probability distribution is a discrete probability distribution over some random variable, x , taking on any one of K possible categories [175]. There is no ordering to these categories and therefore, for simplicity, each category is assigned a numerical representative value such that $k \in \{1, 2, \dots, K\}$. The probabilities for all outcomes is given by the probability vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_K)$. This means that the probability $P(x = k) = \theta_k$ has support $x \in \{1, \dots, K\}$. The Categorical probability distribution, denoted $Cat(\boldsymbol{\theta})$, is a generalisation of the Bernoulli probability distribution and is sometimes referred to it by its alternative names, i.e. the generalised Bernoulli probability distribution or the Multinoulli probability distribution. In probability theory, the Categorical probability distribution is often used to explain the outcome of a single experiment with more than two possible outcomes, such as rolling a six-sided die [175]. The PMF of the Categorical probability distribution is given as

$$P(x|\boldsymbol{\theta}; K) = f_{Cat}(x; K, \boldsymbol{\theta}) = \prod_{k=1}^K \theta_k^{[x=k]} \quad (5.28)$$

where the term, $[x = k]$, is the *Inversion Bracket* [83], yielding 1 if $x = k$ and 0 otherwise. Given class k , the categorical distribution simply yields θ_k as follows:

$$f_{Cat}(x = k; K, \boldsymbol{\theta}) = \theta_k \quad (5.29)$$

The random variable x can also be encoded in binary format, yielding a vector $\mathbf{x} = (x_1, \dots, x_K)$ of Bernoulli probability distributions such that the support is $\forall_{k=1}^K x_k \in \{0, 1\}$. Importantly, if the outcome of the random event is of category k , then $x_k = 1$ and $\forall_{j=1}^K x_j = 0, j \neq k$ so that the standard $K - 1$ probability simplex S still holds. The PMF of the Categorical probability distribution is rewritten as

$$f_{Cat}(\mathbf{x}; K, \boldsymbol{\theta}) = \prod_{k=1}^K \theta_k^{\mathbb{1}(x_k)} \quad (5.30)$$

where $\mathbb{1}(x_k)$ is the *indicator function*, yielding 1 if $x_k = 1$ and 0 otherwise.

Since there is no order to the underlying categories, the mean of the distribution does not yield any relevant information. The mode of the distribution is given as

$$M[x] = \arg \max_k (\theta_1, \dots, \theta_K) \quad (5.31)$$

5.5.6 Multinomial Probability Distribution

The Multinomial probability distribution is a discrete probability distribution over some random variable $\mathbf{x} = (x_1, \dots, x_K)$ that takes on the counts for each occurrence of K possible classes in N independent trials [175]. The probabilities for all possible outcomes in a single trial is given by the probability vector, $\boldsymbol{\theta} = (\theta_1, \dots, \theta_K)$. The Multinomial probability distribution, denoted $Mul(N, K, \boldsymbol{\theta})$, is thus a generalisation of the Binomial probability distribution to K dimensions. Consider the following special cases:

- When K is 2 and $N = 1$, the Multinomial probability distribution is the Bernoulli probability distribution.
- When K is 2 and $N > 1$, the Multinomial probability distribution is the Binomial probability distribution.
- When $K > 2$ and $N = 1$, the Multinomial probability distribution is the Categorical probability distribution.

The support for the Multinomial is $\forall_{i=1}^K x_k \in \{1, \dots, N\}, \sum_{k=1}^K x_k = N$ and the PMF for the Multinomial probability distribution is given as

$$P(\mathbf{x}|\boldsymbol{\theta}; N; K) = f_{Mul}(\mathbf{x}; N, K, \boldsymbol{\theta}) = \frac{N!}{\prod_{k=1}^K x_k!} \prod_{k=1}^K \theta_k^{x_k} \quad (5.32)$$

Similar to the Categorical probability distribution, the random variable x can also be encoded in binary format, yielding an $N \times K$ matrix \mathbf{X} of Bernoulli probability distributions. The support is then given as $\mathbf{X} \in \{0, 1\}^{N \times K}$, $\forall_{i=1}^N \sum_{k=1}^K x_{i,k} = 1$ so that the standard $K - 1$ probability simplex S still holds for each trial. The PMF of the Multinomial probability distribution is then rewritten as

$$\begin{aligned} f_{Mul}(\mathbf{X}; N, K, \boldsymbol{\theta}) &= \frac{N!}{\prod_{k=1}^K (\sum_{i=1}^N x_{i,k})!} \prod_{i=1}^N \prod_{k=1}^K \theta_k^{\mathbb{1}_1(x_{i,k})} \\ &= \frac{N!}{\prod_{k=1}^K (\sum_{i=1}^N x_{i,k})!} \prod_{k=1}^K \theta_k^{\sum_{i=1}^N \mathbb{1}_1(x_{i,k})} \\ &= \frac{N!}{\prod_{k=1}^K (\sum_{i=1}^N x_{i,k})!} \prod_{k=1}^K \theta_k^{N_k} \end{aligned} \quad (5.33)$$

where N_k is a summary variable, denoting the number of times a category k occurs over all trials in N .

5.6 Conjugate Priors

Wackerly et al.[175] state that conjugate priors are prior probability distributions that result in posterior distributions that are of the same functional form, $\mathcal{A}(v)$, as the prior, but with different parameter values. This section considers the conjugate priors that are used with the Binomial likelihood and Categorical/Multinomial likelihood.

5.6.1 Binomial Likelihood

The conjugate prior to a Bernoulli probability distribution is the Beta probability distribution [175]. This is shown by demonstrating that the posterior distribution has the same functional form, $\mathcal{A}(v)$, as the prior distribution as follows.

Setup:

- Let I be a number of independent, identical (iid) random events.
- Let $\alpha \in \mathbb{R}, \alpha > 0$ and $\beta \in \mathbb{R}, \beta > 0$ be the shape parameters to the Beta probability distribution.
- Let θ be the probability of a success. With $\theta|\alpha, \beta \sim \text{Beta}(\alpha, \beta)$.

- $P(\theta)$ is the prior probability distribution with the functional form $\mathcal{A}(v)$.
- Let $\mathbf{X} = (x_1, \dots, x_I)$ be the outcomes of I independent, identical random events, each with Boolean outcome. That is $x_i|\theta \stackrel{\text{iid}}{\sim} \text{Ber}(\theta)$ and $\mathcal{L}(x_i|\theta)$ is the Bernoulli likelihood.
- Let \mathcal{D} denote all the prior data of \mathbf{X} , parameterised by α, β .
- Let $N_1 = \sum_{i=1}^I \mathbb{1}(x_i = 1)$ and $N_0 = \sum_{i=1}^I \mathbb{1}(x_i = 0)$.

The Bernoulli likelihood is given as

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= P(\mathcal{D}|\theta) \\ &\propto \theta^{N_1} (1 - \theta)^{N_0} \end{aligned} \quad (5.34)$$

By Bayes' theorem, the posterior distribution with given prior data \mathcal{D} is given as

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})} \quad (5.35)$$

Since the denominator sums to 1, the denominator and constants for the Bernoulli likelihood and the Beta prior can be removed by expressing the posterior as proportional to the likelihood times the prior as follows:

$$\begin{aligned} P(\theta|\mathcal{D}) &\propto [\theta^{N_1} (1 - \theta)^{N_0}] [\theta^{\alpha-1} (1 - \theta)^{\beta-1}] \\ &\propto \theta^{(N_1+\alpha)-1} (1 - \theta)^{(N_0+\beta)-1} \\ &\propto \text{Beta}(N_1 + \alpha, N_0 + \beta) \end{aligned} \quad (5.36)$$

The posterior distribution has the same functional form, $\mathcal{A}(v)$, as the prior, but with updated prior parameters $\alpha' = N_1 + \alpha$ and $\beta' = N_0 + \beta$. This shows that the Beta probability distribution is the conjugate prior used with the Bernoulli likelihood.

5.6.2 Categorical and Multinomial Likelihood

The conjugate prior to a Categorical and Multinomial probability distribution is the Dirichlet probability distribution [175]. The proof of the conjugate prior for the Bernoulli probability distribution is similar to the proof presented in Section 5.6.1. This means that the posterior distribution must have the same functional form, $\mathcal{A}(v)$, as the prior distribution. The proof is shown as follows.

Setup:

- Let I be a number of independent, identical (iid) random events.
- Let K be a number of possible outcomes for each event, with $K \geq 2$.
- Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$, $\forall_{k=1}^K \alpha_k \in \mathbb{R}, \alpha_k > 0$ be the concentration parameters to the Dirichlet probability distribution.
- Let $\boldsymbol{\theta} = (\theta_1, \dots, \theta_K)$, $\forall_{k=1}^K \theta_k \in (0, 1), \sum_{k=1}^K \theta_k = 1$ be the probability of each class in K and $\boldsymbol{\theta}$ belongs to the standard $K - 1$ probability simplex S . With $\boldsymbol{\theta}|\boldsymbol{\alpha} \sim \text{Dir}(K, \boldsymbol{\alpha})$.
- $P(\boldsymbol{\theta})$ is the prior probability distribution with the functional form $\mathcal{A}(v)$.
- Let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_I)$ be the outcomes of I independent, identical random events, each with K possible outcomes. That is $\mathbf{x}_i|\boldsymbol{\theta} \stackrel{\text{iid}}{\sim} \text{Cat}(\boldsymbol{\theta})$ and $\mathcal{L}(\mathbf{x}_i|\boldsymbol{\theta})$ is the Categorical likelihood.
- Let \mathcal{D} denote all the prior data of \mathbf{X} , parameterised by $\boldsymbol{\alpha}$.
- Let $\mathbf{N} = (N_1, \dots, N_K)$, $N_k = \sum_{i=1}^I \mathbb{1}(x_{i,k} = 1)$, denote the counts for each occurrence of a class k .

The likelihood of the Categorical and Multinomial probability distributions is given as

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= P(\mathcal{D}|\boldsymbol{\theta}) \\ &\propto \prod_{k=1}^K \theta_k^{N_k} \end{aligned} \quad (5.37)$$

By Bayes' theorem, the posterior distribution with given prior data \mathcal{D} is given as

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \quad (5.38)$$

Since the denominator sums to 1, the denominator and constants for the Dirichlet prior can be removed by expressing the posterior as proportional to the likelihood times the prior as follows:

$$\begin{aligned} P(\boldsymbol{\theta}|\mathcal{D}) &\propto \prod_{k=1}^K \theta_k^{N_k} \prod_{k=1}^K \theta_k^{\alpha_k-1} \\ &\propto \prod_{k=1}^K \theta_k^{(N_k+\alpha_k)-1} \\ &\propto \text{Dir}(K, \mathbf{N} + \boldsymbol{\alpha}) \end{aligned} \quad (5.39)$$

The posterior distribution has the same form, $\mathcal{A}(v)$, as the prior, but with updated prior parameters $\boldsymbol{\alpha}' = \mathbf{N} + \boldsymbol{\alpha}$. This shows that the Dirichlet probability distribution is the conjugate prior used with the Categorical/Multinomial likelihood.

5.7 Bayesian Statistics

This section provides detailed discussions on various aspects related to Bayesian statistics and probability. Brief discussion follow on the frequentist and the Bayesian approach to statistics and probability. Finally, the concept of Bayesian analysis is presented in detail.

5.7.1 Frequentist vs. Bayesian Statistics

In general, there are two main views to probability and statistics. These include the *frequentist* and the *Bayesian* view of statistics. Naturally, Bayesian statistics is based on Bayes' theorem as was presented in Section 5.4. Bayesian statistics describe the probability of an event in terms of some belief, based on previous knowledge of the event and the conditions under which the event happened [68]. To introduce the concept of Bayesian inference and Bayesian analysis, the differences between the frequentist and the Bayesian view of statistics need to be presented.

Bayesian statistics out-date the frequentist approach, but lacked interest in the early days, partly because of the limited applications where the conjugate priors were known [68]. More recent advancements in mathematical methods popularised the Bayesian approach again. A notable contribution to this switch was the development of the *Markov Chain Monte Carlo* (MCMC) algorithm in the 1950s. This family of algorithms allowed for the construction of random sampling algorithms from a probability distribution, which allows for the calculation of Bayesian hierarchical models [68]. Soon after followed one of the earliest papers that use Bayesian statistics in the field of medicine in 1982 [4].

The difference between the frequentist approach and the Bayesian approach can be illustrated using an example. Hackenberger [68] suggests an experiment that investigates whether the gender ratio in some hypothetical mice population is 1 : 1. Two experiments were designed. In the first experiment, mice are randomly selected until the first male is chosen. The result in this experiment will then be the total number of mice chosen by gender. For the second experiment, exactly seven mice are randomly selected. The result of the second experiment would be the number of males and females in a sample of seven. Suppose the outcome of the experiment was *FFFFFFM*, where *F* represents a female and *M* represents a male. If the

experimental design is not known ahead of time, the result is useless. Consider the p -value for each of these experiments. The p -value is the probability of obtaining results at least as extreme as the observed results of a statistical hypothesis test [163]. For the first experiment, the p -value is 0.031 and for the second experiment, the p -value is 0.227. Using a confidence level of $\alpha = 0.05$, opposite outcomes could be concluded for these two experiments when it comes to rejecting the null hypothesis, despite using the same data. The reason for the difference in outcomes, is due to the difference in their null distributions, which represent the probability distribution of the test statistic when the null hypothesis is true. The first approach uses a geometrical approach, and the second used a binomial approach as illustrated in Figure 5.5.

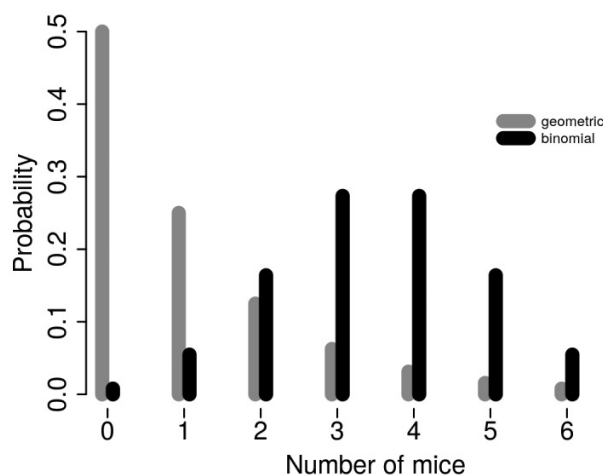


Figure 5.5: The experimental outcomes for the mice-population experiments as were taken from [68] .

If Bayesian statistics is used, the experimental design that was chosen does not matter. In Bayesian statistics it is common to use a Beta probability distribution as a prior distribution. If the prior distribution is sampled from $Beta(3, 3)$, then using Bayesian analysis, the posterior distribution, according to the outcomes of this experiment, would yield $Beta(9, 4)$.

Hackenberger [68] mentions that the *Beta* probability distribution can be seen as a probability distribution of the occurrence of specific parameters. From the information that is now known about the *Beta* probability distribution, it is possible to calculate the probability that the gender ratio in this mice population is not 1 : 1, with the *Beta* probability distribution as a prior, yielding a p -value of 0.92. This means that there is a probability of 92% that the mice population is not 1 : 1, regardless of experimental design. An illustration of this is given in Figure 5.6.

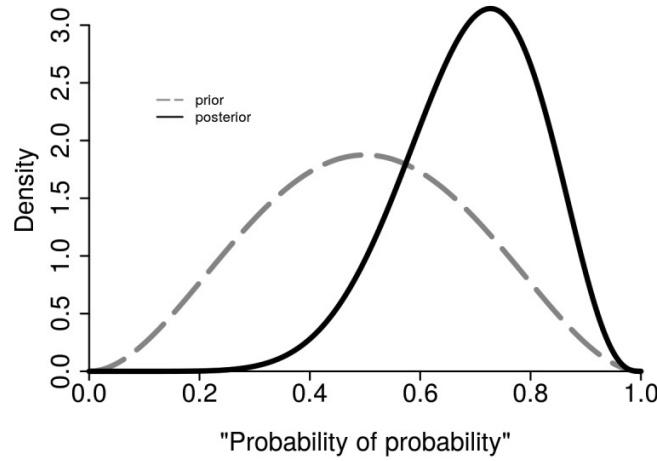


Figure 5.6: An illustration of the prior and posterior probability distributions for the outcomes of the mice-population experiment, using a *Beta* prior, as was taken from [68].

Figure 5.6 shows how the posterior distribution differs from the prior distribution.

5.7.2 Bayesian Analysis

Bayesian analysis is the process by which prior beliefs are updated as a result of observing new data/evidence. Similar to the approaches followed above to explain Bayesian statistics, a proposal is made to explain Bayesian analysis by means of an example taken from [175]. Let $\mathbf{Y} = (Y_1, Y_2, \dots, Y_N)$ denote the random variable that is observed over a sample size of N . Then the likelihood of the sample is given as $\mathcal{L}(y_1, y_2, \dots, y_n|\theta)$. In the discrete case, the likelihood function is defined to be the joint probability, $P(Y_1 = y_1, Y_2 = y_2, \dots, Y_N = y_n)$, and for the continuous case, yields the joint density of Y_1, Y_2, \dots, Y_N , evaluated at y_1, y_2, \dots, y_n . The Bayesian view models the parameter θ as a random variable with a probability distribution, referred to as the prior distribution of θ . The symbol θ , is included in the notation of \mathcal{L} as an argument to illustrate that this function is dependent, explicitly, on the value of θ . Importantly, the prior distribution is specified before any data is collected and represents the theoretical prior knowledge about θ . Assume that the parameter θ has a continuous distribution with density $g(\theta)$ that has no unknown parameters. Considering the likelihood of the data and the prior on θ , then the joint likelihood of $Y_1, Y_2, \dots, Y_N, \theta$ is given as

$$f(y_1, y_2, \dots, y_n, \theta) = \mathcal{L}(y_1, y_2, \dots, y_n|\theta)g(\theta) \quad (5.40)$$

The marginal density or mass function of Y_1, Y_2, \dots, Y_N is given as

$$m(y_1, y_2, \dots, y_n) = \int_{-\infty}^{\infty} \mathcal{L}(y_1, y_2, \dots, y_n | \theta) g(\theta) d\theta \quad (5.41)$$

Finally, the posterior density of $\theta | y_1, y_2, \dots, y_n$ denoted by $g^*(\theta | y_1, y_2, \dots, y_n)$, according to Bayes' theorem, is given as

$$g^*(\theta | y_1, y_2, \dots, y_n) = \frac{\mathcal{L}(y_1, y_2, \dots, y_n | \theta) g(\theta)}{\int_{-\infty}^{\infty} \mathcal{L}(y_1, y_2, \dots, y_n | \theta) g(\theta) d\theta} \quad (5.42)$$

Wackerly et al.[175] mention that the posterior density summarises all the pertinent information about the parameter θ by making use of the information contained in the prior for θ , as well as the information in the observed data/evidence.

Consider now how Bayesian analysis can be used to update the priors on θ based on newly observed data. As with the example above, the discussion below is taken from [175]. Let Y_1, Y_2, \dots, Y_N denote a random sample from a Bernoulli probability distribution, where $P(Y_i = 1) = \theta$ and $P(Y_i = 0) = 1 - \theta$ and the prior distribution for θ is $Beta(\alpha, \beta)$. The posterior distribution for θ can be formulated as follows. The Bernoulli probability function is written as

$$P(y_i | \theta) = \theta^{y_i} (1 - \theta)^{1-y_i} \quad (5.43)$$

where $y_i = \{0, 1\}$ and $0 < \theta < 1$. The likelihood $\mathcal{L}(y_1, y_2, \dots, y_n | \theta)$ is presented as follows:

$$\begin{aligned} \mathcal{L}(y_1, y_2, \dots, y_n | \theta) &= P(y_1, y_2, \dots, y_n | \theta) \\ &= \theta^{y_1} (1 - \theta)^{1-y_1} \times \theta^{y_2} (1 - \theta)^{1-y_2} \times \dots \times \theta^{y_n} (1 - \theta)^{1-y_n} \\ &= \theta^{\sum y_i} (1 - \theta)^{n - \sum y_i} \end{aligned} \quad (5.44)$$

Then the joint likelihood of $Y_1, Y_2, \dots, Y_N, \theta$ from Equation (5.40) is formulated as

$$\begin{aligned} f(y_1, y_2, \dots, y_n, \theta) &= \mathcal{L}(y_1, y_2, \dots, y_n | \theta) g(\theta) \\ &= \theta^{\sum y_i} (1 - \theta)^{n - \sum y_i} \times \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \Gamma(\beta)} \theta^{\alpha-1} (1 - \theta)^{\beta-1} \\ &= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \Gamma(\beta)} \theta^{\sum y_i + \alpha - 1} (1 - \theta)^{n - \sum y_i + \beta - 1} \end{aligned} \quad (5.45)$$

The marginal density of Y_1, Y_2, \dots, Y_N is then given as

$$\begin{aligned}
m(y_1, y_2, \dots, y_n) &= \int_0^1 \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\sum y_i + \alpha - 1} (1 - \theta)^{n - \sum y_i + \beta - 1} d\theta \\
&= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\sum y_i + \alpha)\Gamma(n - \sum y_i + \beta)}{\Gamma(n + \alpha + \beta)}
\end{aligned} \tag{5.46}$$

Finally, the posterior density of θ , denoted by $g^*(\theta|y_1, y_2, \dots, y_n)$, is given as

$$\begin{aligned}
g^*(\theta|y_1, y_2, \dots, y_n) &= \frac{\frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\sum y_i + \alpha - 1} (1 - \theta)^{n - \sum y_i + \beta - 1}}{\frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\sum y_i + \alpha)\Gamma(n - \sum y_i + \beta)}{\Gamma(n + \alpha + \beta)}} \\
&= \frac{\Gamma(n + \alpha + \beta)}{\Gamma(\sum y_i + \alpha)\Gamma(n - \sum y_i + \beta)} \theta^{\sum y_i + \alpha - 1} (1 - \theta)^{n - \sum y_i + \beta - 1}
\end{aligned} \tag{5.47}$$

The posterior density has the same functional form, $\mathcal{A}(v)$, as the prior, yielding the update on prior parameters as $\alpha' = \sum y_i + \alpha$ and $\beta' = n - \sum y_i + \beta$.

5.8 Summary

This chapter provided all the necessary background information on probability theory and statistics. The origins of statistical analysis were discussed and the differences between the frequentist and Bayesian view of statistics were discussed in detail. Probability distributions and proofs of conjugate priors were provided with detailed mathematical descriptions. Finally, a detailed description of Bayesian analysis was provided with detailed mathematical descriptions.

This chapter concludes all the relevant background information that is needed to formulate the BHH. The proposed BHH is presented in the following chapter.

Chapter 6

Bayesian Hyper-Heuristic

“The result is a posterior distribution, which the agent may use as its new prior in the next step.”

- Pedro Domingos, The Master Algorithm

The above quote was the inspiration for the development of a novel *hyper-heuristic* (HH) that uses Bayesian probability concepts as a selection mechanism to drive the heuristic selection process. Thus far the reader has been presented with all of the necessary background information on ANNs (Chapter 2), low-level heuristics (Chapter 3), HHs (Chapter 4) and lastly, probability theory (Chapter 5). These elements form the fundamental components of the proposed *Bayesian hyper-heuristic* (BHH). This chapter provides the details around the implementation of the BHH and explains how it is used to train FFNNs. The remainder of the chapter is structured as follows:

- **Section 6.1** provides a brief overview of the BHH.
- **Section 6.2** provides the general architecture and HH framework implemented by the BHH.
- **Section 6.3** presents the *heuristic pool*, a collection of low-level heuristics. Discussions follow on the importance of diversity amongst heuristics in the heuristic pool.
- **Section 6.4** presents details on entity (local) and population (global) memory (state).
- **Section 6.5** presents detailed discussions on performance measurement. The *performance log*, implemented by the BHH, is presented.

- **Section 6.6** presents detailed discussions on credit assignment strategies.
- **Section 6.7** presents the Bayesian probabilistic model that is used as the heuristic selection mechanism.
- **Section 6.8** presents the learning mechanisms by which the probabilistic model can be optimised.
- **Section 6.9** summarises and discusses the associated hyper-parameters and default values.
- **Section 6.10** provides the pseudo-code algorithm for the BHH.
- **Section 6.11** provides a brief summary of the chapter.

6.1 Overview

This section provides an overview of the workings of the *Bayesian hyper-heuristic* (BHH). The general concept of the BHH can be summarised as follows: The BHH implements a high-level heuristic selection mechanism that learns to select the best heuristic from a pool of low-level heuristics, to apply to a population of entities, each implementing a candidate solution to a FFNN, with the intent of both optimising the underlying FFNN and FFNN training process. The BHH does so by learning the probability that a given heuristic will perform well at a given stage in the FFNN training process. These probabilities are then used as heuristic selection probabilities in the next step of the training process.

Formal classification of the BHH is needed. Chapter 4 presented the reader with a proposed classification scheme for HHs by Burke et al. [19]. According to the aforementioned classification scheme, the BHH is a population-based, meta-hyper-heuristic that utilises selection and perturbation of low-level heuristics in an online learning fashion. A breakdown of the classification is given as follows:

- **Population-Based:** The BHH implements a population-based approach, where a collection of different candidate solutions, referred to as *entities*, work together to yield a global best solution.
- **Meta-Hyper-Heuristic:** There exists a domain barrier, where the BHH searches through the *heuristic space*, using only heuristic performance information, while lower level heuristics search through the *solution space* using information from the search domain itself. Furthermore, the heuristic pool

implemented by the [BHH](#) supports both gradient-based low level heuristics, as well as [MHs](#).

- **Selection and Perturbation of Low-Level Heuristics:** The [BHH](#) implements a heuristic selection mechanism that selects from a collection of lower level heuristics, called a *heuristic pool*. Selection is biased towards good performing heuristics. A credit assignment strategy is used to *reward* good heuristic performance. The [BHH](#) maintains entity and population state through operations that *proxy* update steps from different heuristics (perturbation), as is required. These proxy operations ensure that heuristic requirements are satisfied even when different heuristics are used for different entities throughout the training process.
- **Online Learning:** The [BHH](#) applies learning, at specified intervals, throughout the [FFNN](#) training process.

6.2 Architecture

This section aims to present the reader with all the high level components in the architecture of the [BHH](#). Burke et al. [19] propose an initial framework for [HHs](#) and Grobler [64] further propose a framework for a heterogeneous meta-[HH](#). The aforementioned frameworks are adapted for the implementation of the [BHH](#). An illustration of the high-level architecture of the [BHH](#) is given in Figure 6.1.

With reference to Figure 6.1, the components are briefly given below in the order of information flow during the [FFNN](#) training process steps:

- **Initialisation Step:** The initialisation step refers to the initialisation strategy implemented by the [BHH](#).
- **Heuristic Pool:** The heuristic pool contains a collection of low-level heuristics.
- **Entity Pool:** The entity pool represents a collection of candidate solutions for the underlying [FFNN](#), referred to as entities in a population.
- **Selection Mechanism:** The selection mechanism refers to the Bayesian probabilistic model.
- **Heuristic-Entity Selections:** Every entity in the entity pool is assigned a selected heuristic.

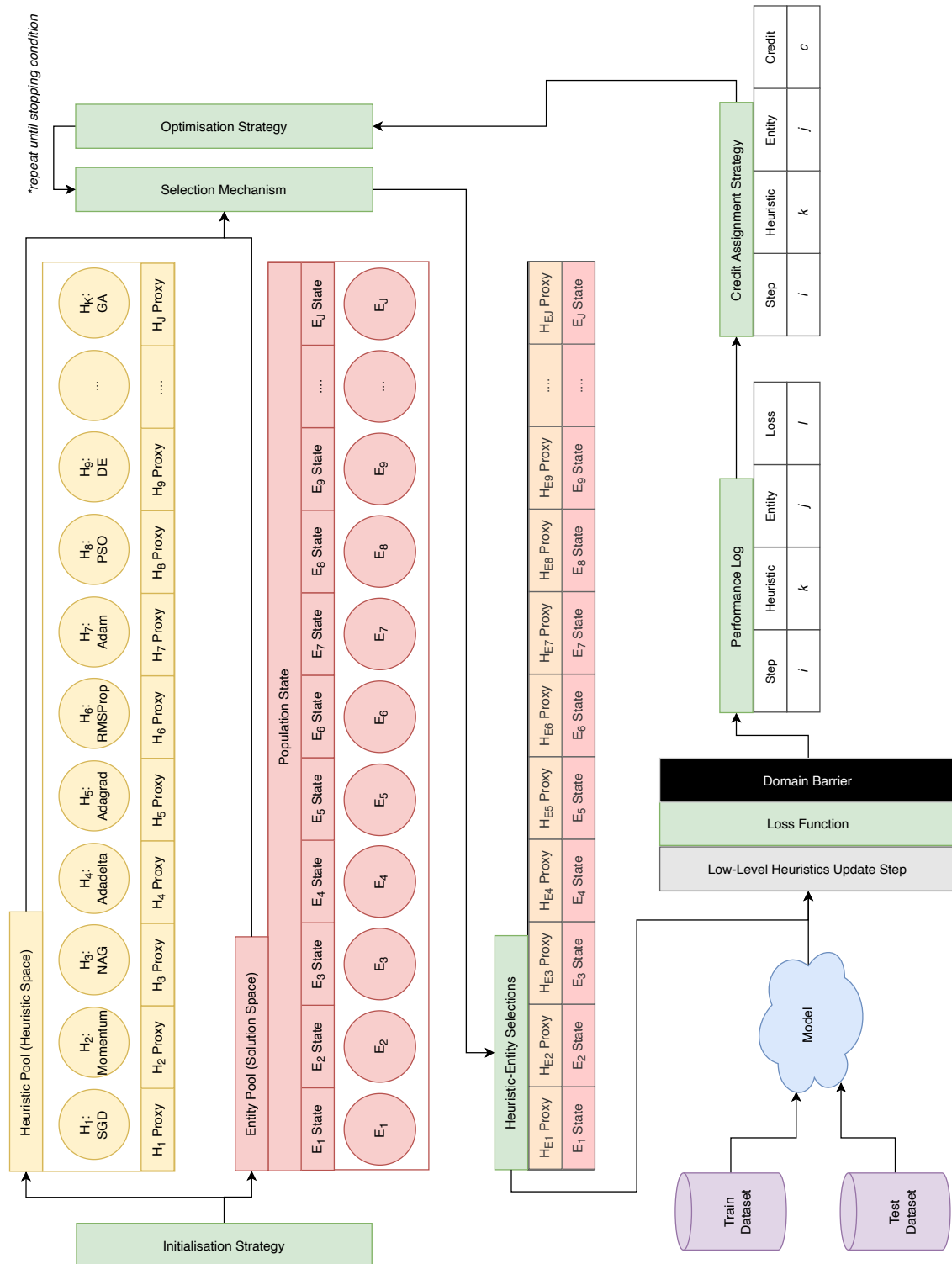


Figure 6.1: An illustration of the architecture and high level components of the *Bayesian hyper-heuristic* (BHH).

- **Train and Test Datasets:** The training set is used to train the model while the test set is used to evaluate the model's generalisation capabilities.
- **Model:** The underlying [FFNN](#) to be trained.
- **Low-Level Heuristics Update Step:** The low-level heuristics' update step as it applies to its allocated entity.
- **Loss Function:** In supervised learning, the loss function is a measure of distance between predicted output and the ground truth, and is the mechanism used to evaluate the model's performance.
- **Domain Barrier:** The domain barrier is the logical separation of information available in the heuristic space and the solution space. The [BHH](#) searches in the heuristic space, while low-level heuristics search in the solution space.
- **Performance Log:** Contains a record of heuristic-entity performance at each step of the training process.
- **Credit Assignment Strategy:** The strategy used to assign credit to heuristics for their performance.
- **Update Step:** The high-level heuristic update step as implemented by the [BHH](#).

6.3 Heuristic Pool

Generally speaking, the heuristic pool is a collection of low-level heuristics under consideration by the [BHH](#). The heuristic pool contains the set of low-level heuristics that, together with their performance information, make up the heuristic space. Importantly, the [BHH](#) searches in heuristic space. The heuristic pool is defined in terms of the diversity of heuristics, as well as the number of heuristics in the heuristic pool. Each of these is discussed in the following sections.

6.3.1 Heuristic Diversity

A heuristic's eligibility for inclusion in the heuristic pool is determined by its ability to solve the underlying problem and its search behaviour and characteristics. The heuristic pool should contain a variety of different heuristics that have different search behaviours, so that a trade-off can be made between exploration and exploitation

of solutions, as is required during the training process. Heuristics that explore a lot should be selected when exploration is needed, and heuristics that exploit a lot should be selected accordingly. The trade-off between exploration and exploitation is managed by the high-level heuristic. The BHH learns to select and apply the appropriate heuristic at the appropriate time, throughout the training process. In doing so, a balance is achieved between exploration and exploitation.

6.3.2 Heuristic Pool Size

Another design decision with respect to the heuristic pool is heuristic pool size. Since the BHH is a *learning HH*, every heuristic that is included in the heuristic pool increases the requirement of more statistical evidence of heuristic performance. Not only does a large heuristic pool drastically complicate the learning process that is required by the BHH, but it also drastically complicates the process of maintaining *state*. State is the term used to describe data that is relevant to the training process, and is maintained between training steps. An example of *local* state is an entity's position, while an example of *global* state is the global best solution found thus far.

6.3.3 Proxies

The concept of proxies arise from the sparsity of state as maintained by different heuristics. Since heuristics maintain (possibly) different states, there is an uncertainty of state transition when switching between heuristics. Consider an example where the heuristic pool consists of just two heuristics. One heuristic is a gradient-based heuristic that maintains momentum, such as Adam [94]. The other is a meta-heuristic that does not require a gradient, such as PSO [146]. Both these heuristics track different parameters in their state. For Adam, the expected gradient mean and variance is maintained, while the PSO maintains record of the gbest and pbest solutions of all of entities. A solution to state indifference is to proxy heuristic state update operations. State is then maintained in two parts:

- **Primary State:** Refers to the state that is originally maintained by a heuristic. The selected heuristic simply applies the normal state update operations to its state.
- **Proxied State:** Refers to the state that is not directly maintained by the heuristic, but can be updated by outsourcing the required state update operation to another heuristic.

Primary and proxied state parameters must be maintained together. Since entities represent candidate solutions, which are forms of state, entities are ideal candidates to store and maintain these state parameters. Entities are extended to include the primary state elements of *all* the underlying low-level heuristics, as well as the candidate solution (weights) for the [FFNN](#). At each heuristic-entity application step, all state parameters, per entity, are updated either by primary method or by proxied method. The [BHH](#) thus incorporates a mapping of proxied state update operation as given in the example in Table 6.1.

Table 6.1: An example of a mapping of proxied state update operation maintained by the [BHH](#).

		State Parameter		
		1	2	3
Heuristic	A	n/a	B	n/a
	B	n/a	n/a	A
	C	n/a	B	A

From the example given in Table 6.1, when heuristic A is selected, it will outsource state update operations from heuristic B for state parameter 2. Heuristic B will outsource from heuristic A for state parameter 3. Finally, heuristic C will outsource from heuristic A and B for state parameters 2 and 3 respectively. In this way, all heuristics maintain all the state parameters.

Proxied state update operations is a simple concept in principle, but requires detailed decomposition of the heuristics included in the heuristic pool. Overlapping and unique state parameters must be identified so that a proxy mapping, such as the one given in Table 6.1, can be constructed. A suggestion to simplify this process is to borrow concepts from the equations of motion from physics. These include *position*, *velocity*, *acceleration*, and *momentum*. Expressing heuristic update steps according to these parameters drastically simplify the process. However, it is possible that heuristics implement unique state parameters that do not overlap. These have to be catered for in the proxy mapping.

State parameters and update operations should not be considered in isolation. An example is *position* and *velocity*. Consider, for example, heuristics such as [DE](#) [128] and [GAs](#) [76]. These heuristics recombine entities. In the aforementioned case, the concept of an equation of motion does not entirely make sense, since the displacement of its position is not a result of maintaining velocity or momentum, but rather by displacement through recombination. In this particular case, a solution is to

apply the recombination operation to *all* applicable state parameters as well, or to nullify the necessary state parameters. Unfortunately, there is no general, automated solution to formulate these heuristic state overlaps. Each heuristic must be carefully considered.

6.4 Entity Pool

The entity pool refers to a collection of *entities* that each represent a candidate solution to the underlying [FFNN](#) being trained. The entities contain information of the solution space. A common naming convention for such a collection is a *population* of entities. The [BHH](#) is a population-based [HH](#) and as such, the entity pool size or population size is an important design choice. The population size is defined as a hyper-parameter of the [BHH](#) and can be empirically evaluated.

The entity pool maintains two different types of state. These include entity (local) and population (global) state. Each of these is discussed in more detail next.

6.4.1 Entity State

Entities represent candidate solutions to the model's trainable parameters (weights) and other heuristic-specific state parameters, as was discussed in Section [6.3.3](#). It can be said that entities implement *local* state. It was mentioned that these entities can be treated as physical *particles* in a hyper-dimensional search environment. Entities model concepts from physics. For example, the candidate solution is represented as the entity's position, and an entity's velocity and acceleration are analogous to the gradient and momentum of the entity respectfully. Examples of entity state parameters, as derived from various low-level heuristics, is given as follows:

- **position:** A general parameter that represents the actual candidate solution and is thus a primary state parameter for all heuristics.
- **velocity:** Directly implemented by heuristics such as [PSOs](#) and is analogous to the gradient for gradient-based heuristics such as [Momentum](#).
- **gradient:** The last known gradient, as derived from gradient-based heuristics.
- **position delta:** The last computed position delta between the current time step and the previous time step.
- **sum of the gradients squared:** As required and maintained by heuristics such as [Adagrad](#).

- **expected position delta variance:** As required and maintained by heuristics such as [Adadelta](#).
- **expected gradient mean:** As required and maintained by heuristics such as [Momentum](#), [NAG](#) and [Adam](#).
- **expected gradient variance:** As required and maintained by heuristics such as [RMSProp](#), [Adadelta](#) and [Adam](#).
- **personal best position:** A parameter that tracks that best known position by the entity thus far, as required and maintained by heuristics such as [PSO](#).
- **personal best loss:** A parameter that tracks that best known loss by the entity thus far, as required and maintained by heuristics such as [PSO](#).
- **loss:** A parameter that tracks the loss as achieved by the entity throughout training.

From the list above, it should be clear that entity state becomes increasingly complicated with the increase of the number of distinct heuristics in the heuristic pool with unique state parameters.

6.4.2 Population State

The population state refers to a collection of parameters that are shared between the entities in the population. Population state is also referred to as *global* state and represents the population's memory. The population state generally contains state parameters that are of importance to multiple heuristics, and usually tracks the state of the population and not individual heuristic. Some examples of population state that can arise from different heuristics are given below:

- **entities:** Naturally, the population state contains the list of entities in the population.
- **ibest** and **ibest loss:** Refers to the best position and loss achieved by the population for the current iteration/step.
- **rbest** and **rbest loss:** Refers to the best position and loss achieved by the population for a number of steps, referred to as the replay window size. The replay window size defines a window of memory for tracking historical performance data of heuristics.

- **gbest** and **gbest loss**: Refers to the overall/global best position and loss achieved by the population for the entire training process. This parameter is introduced by heuristics such as [PSOs](#).

Entities in the entity pool are each assigned an associated heuristic. The [BHH](#) selects from the heuristic pool a low-level heuristic to be applied to an individual entity. The outcome of this selection process is a mapping table that tracks which heuristic has been selected for which entity. The selection process is executed by the selection mechanism of the [BHH](#). These heuristic-entity combinations are applied to the underlying [FFNN](#). The [BHH](#) tracks the performance of each of these combinations throughout the training process in a performance log.

6.5 Performance Log

The [BHH](#) incorporates a form of probabilistic modelling in its selection mechanism. Probability is calculated based on heuristic-entity performance over time. Evidence of heuristic-entity performance is thus required for the [BHH](#) to learn. Since the performance log can become very big, only a sliding window of the performance history is maintained at each step in the learning process. The sliding window is also referred to as a *replay* buffer, a term borrowed from the field of *reinforcement learning* ([RL](#)). The replay window size is defined as a hyper-parameter of the [BHH](#). The performance log is then simply a table of events and metrics. The [BHH](#) tracks the following metrics in the performance log:

- **step**: The current mini-batch step.
- **heuristic**: The selected heuristic's index.
- **entity**: The entity's index to whom the heuristic is applied.
- **loss**: The heuristic-entity performance metric. The loss is retrieved from the loss function.
- **ibest loss**: Keeps track of the *iteration* best loss. Thus, the best loss value achieved by all entity-heuristic combinations, for a single mini-batch step.
- **rbest loss**: Keeps track of the *replay* best loss. Thus, the best loss value achieved by all entity-heuristic combinations, over all mini-batch steps currently in the performance log/replay window.

- **gbest loss:** Keeps track of the *global* best loss. Thus, the best loss value achieved by all entity-heuristic combinations over all mini-batch steps thus far.

An example of the performance log implemented by the BHH is given in Table 6.2.

Table 6.2: An example of the performance log implemented by the BHH, showing the first five entities, their allocated heuristics and their resulting performance measurements for the first step of the training process.

step	entity	heuristic	loss	ibest loss	pbest loss	rbest loss	gbest loss
1	1	1	0.016444	0.016444	0.016444	0.016444	0.016444
1	2	2	0.337965	0.016444	0.337965	0.016444	0.016444
1	3	1	0.134781	0.016444	0.134781	0.016444	0.016444
1	4	1	0.998719	0.016444	0.998719	0.016444	0.016444
1	5	3	0.708702	0.016444	0.708702	0.016444	0.016444

The performance log itself introduces a number of design considerations as well. The larger the performance log, the longer memory is retained throughout the training process. The performance log size, referred to as the *replay window size*, controls the recency of evidence from which the BHH must learn. If the replay window size is too small, it does not accumulate enough samples/evidence to statistically make accurate selections. If the replay window size is too big, a bias could exist for selecting heuristics that performed well in the past. Furthermore, it is not guaranteed that past performance is indicative of future performance during the training process. The performance log should be considered along with the selection of a credit assignment strategy.

6.6 Credit Assignment Strategy

The credit assignment strategy is a mechanism that assigns a discrete credit indicator to heuristics that perform well, based on their performance metrics such as loss. The credit assignment strategy implements the “move acceptance” process as proposed by Özcan et al. [121, 122] and addresses the credit assignment problem as discussed by Burke et al. [19]. A good credit assignment strategy will correctly allocate credit to the appropriate heuristic-entity combination. The credit assignment strategy to use is defined as a hyper-parameter. The BHH implements the following credit assignment strategies to choose from:

- **ibest**: Credit is assigned to the heuristic-entity combination that set the *ibest* loss value, meaning that it is the entity-heuristic combination that achieved the best performance in the current mini-batch iteration.
- **pbest**: Credit is assigned to the heuristic-entity combination that set the *pbest* loss value, meaning that it is the entity-heuristic combination that was able to improve on its personal best past performance loss.
- **rbest**: Credit is assigned to the heuristic-entity combination that set the *rbest* loss value, meaning that it is the entity-heuristic combination that achieved the best performance in the current replay window.
- **gbest**: Credit is assigned to the heuristic-entity combination that set the *gbest* loss value, meaning that it is the entity-heuristic combination that achieved the overall best performance so far.
- **symmetric**: Credit is assigned to all entity-heuristic combinations, regardless of their performance. The symmetric credit assignment strategy does not randomly assign credit, but rather assigns credit to all events. In effect, no learning and performance-bias is achieved with the symmetric credit assignment strategy and thus the symmetric credit assignment strategy is implemented as a basis for comparison. Comparing a credit assignment strategy to the symmetric credit assignment indicates the ability of the other credit assignment strategy to have an effect on the learning process and performance of the BHH.

The implementation of a credit assignment strategy is simply a function that translates the performance log from Table 6.2 into Table 6.3.

Table 6.3: Credit assignment strategy output table showing *ibest* credit assignment for the first five entities and their selected heuristics for step 1 of the training process.

step	entity	heuristic	credit
1	1	1	true
1	2	2	false
1	3	1	false
1	4	1	false
1	5	3	false

Credit is used by the selection mechanism of the BHH.

6.7 Selection Mechanism

This section provides the detail around the selection mechanism as it is implemented by the BHH. Detail is provided around random events as observed by the BHH. An argument for independence between random events is made. Bayes' theorem is briefly reviewed in the context of the BHH implementation. It is shown how the BHH implements a probabilistic, predictive model based on the fundamentals of the Naïve Bayes algorithm. Brief discussions follow on numerical stability and mode collapse.

6.7.1 Random Events

Observation of random events is treated as evidence. In the context of a Bayesian approach, this evidence is used to update prior beliefs. The BHH distinguishes between the following events:

- ***H***: The event of observing *heuristics*.
- ***E***: The event of observing *entities*.
- ***C***: The event of observing *credit assignments* that indicate that the credit assignment *performance criteria* are met.

It should be noted that event information is stored and observed directly from the performance log as presented in Table 6.3. From the above list of events, the event ***C*** is dependent on the occurrence of ***H*** and ***E*** and a credit assignment strategy.

6.7.2 Independence

The dependence between random events can have an impact on the probabilistic model that is implemented. For simplicity, the BHH assumes independence between events, although the event ***C*** is clearly dependent on the occurrence of ***H*** and ***E***. Furthermore, the BHH assumes independence between steps and treats each training step as if training has restarted. The BHH implements a form of Naïve Bayes classifier, where the appropriate heuristic to assign to an entity is a classification problem. Domingos et al. [36] mention that although the probability estimates of Bayesian classifiers are only optimal under quadratic loss if the independence assumption holds, the classifier itself can still be optimal under zero-one loss (misclassification rate), even when this assumption is violated by a wide margin. This means that independence can be assumed when the probabilistic model is used as a classifier.

6.7.3 Bayes' Theorem

Chapter 5 presented Bayes' theorem, but for convenience, it is given below:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (6.1)$$

Since the BHH implements a Bayesian classifier, the BHH is not concerned with actual probabilities. Equation (6.1) can thus be evaluated for proportionality. The resulting proportionality is expressed as

$$P(A|B) \propto P(B|A)P(A) \quad (6.2)$$

The conditionality implemented by Bayes' theorem can be extended to include multiple criteria. Bayes' theorem is used by the selection mechanism of the BHH.

6.7.4 Predictive Model

This section presents the predictive model as implemented by the selection mechanism of the BHH. The predictive model is arguably the most important component of the BHH, because it drives the heuristic selection process. The BHH implements a predictive model that predicts which heuristic to select, given the conditionality that a particular entity is used and a particular credit assignment criteria is met. The predictive model can be derived as follows.

Setup:

Let

- I denote the maximum number of instances in the replay window.
- J denote the entity pool (population) size.
- K denote the heuristic pool size.
- L denote the number of credit assignment output classes. Since the output of credit assignment is Boolean, $L = 2$.
- $\alpha = (\alpha_1, \dots, \alpha_K)$ denote the concentration parameters for the heuristic probability distribution.
- $\beta = (\beta_1, \dots, \beta_K)^J$ denote the concentration parameters for the entity-heuristic probability distributions.

- $\gamma = (\gamma_1, \dots, \gamma_K)^L$ denote the concentration parameters for the credit-heuristic probability distribution.
- $\theta|\alpha \sim \text{Dir}(\alpha; K)$ denote the heuristic probability distribution, implementing a Dirichlet probability distribution parameterised by α and K . Heuristic selection probabilities are sampled from this distribution.
- $\phi|\beta \sim \text{Dir}(\beta; K)^J$ denote the entity-heuristic probability distribution, implementing a Dirichlet probability distribution parameterised by β and K for each entity in the entity pool with population size J . Entity-heuristic probabilities are sampled from this distribution.
- $\psi|\gamma_1, \gamma_0 \sim \text{Beta}(\gamma_1, \gamma_0)$ denote the credit-heuristic probability distribution, implementing a Beta probability distribution parameterised by γ_1 and γ_0 . Credit-heuristic probabilities are sampled from this distribution.
- $\mathbf{H}|\theta \sim \text{Mult}(\theta; I, K)$ denote the distribution of heuristics (event \mathbf{H}), implementing a Multinomial distribution, parameterised by the sampled heuristic selection probabilities θ , the heuristic pool size K , and maximum number of instances I .
- $\mathbf{E}|\phi \sim \text{Mult}(\phi; I, K)^J$ denote the distribution of entity-heuristic combinations (event \mathbf{E}), implementing a Multinomial distribution, parameterised by the sampled entity-heuristic selection probabilities ϕ , the heuristic pool size K , and maximum number of instances I for each entity in J .
- $\mathbf{C}|\psi \sim \text{Bin}(\psi, I)$ denote the distribution of credit-heuristic combinations (event \mathbf{C}), implementing a Binomial probability distribution, parameterised by the sampled credit-heuristic selection probabilities ψ and the maximum number of instances I .

The parameterised predictive model, as derived from Bayes' theorem, is then given as

$$\begin{aligned}
 P(\mathbf{H}|\mathbf{E}, \mathbf{C}; \theta, \phi, \psi) &= \frac{P(\mathbf{E}, \mathbf{C}|\mathbf{H}; \phi, \psi)P(\mathbf{H}|\theta)}{P(\mathbf{E}, \mathbf{C}|\theta, \phi, \psi)} \\
 &= \frac{P(\mathbf{E}|\mathbf{H}; \phi)P(\mathbf{C}|\mathbf{H}; \psi)P(\mathbf{H}|\theta)}{P(\mathbf{E}|\theta, \psi)P(\mathbf{C}|\theta, \phi)} \\
 &= \frac{P(\mathbf{E}|\mathbf{H}; \phi)P(\mathbf{C}|\mathbf{H}; \psi)P(\mathbf{H}|\theta)}{\left[\sum_k^K P(\mathbf{E}, \mathbf{H} = k|\theta, \phi) \right] \left[\sum_k^K P(\mathbf{C}, \mathbf{H} = k|\theta, \psi) \right]} \\
 &= \frac{P(\mathbf{E}|\mathbf{H}; \phi)P(\mathbf{C}|\mathbf{H}; \psi)P(\mathbf{H}|\theta)}{\left[\sum_k^K P(\mathbf{E}|\mathbf{H} = k, \phi)P(\mathbf{H} = k|\theta) \right] \left[\sum_k^K P(\mathbf{C}|\mathbf{H} = k, \psi)P(\mathbf{H} = k|\theta) \right]}
 \end{aligned} \tag{6.3}$$

Notice the joint probability over events \mathbf{E} and \mathbf{C} , given the selection of a heuristic \mathbf{H} , denoted by the product of the sums of the separate parts for \mathbf{E} and \mathbf{C} , in the denominator. The calculation in the denominator can be intractable. Since the BHH selection mechanism uses the predictive model as a classifier, the posterior distribution as given in Equation (6.1) can be evaluated for proportionality as follows:

$$P(\mathbf{H}|\mathbf{E}, \mathbf{C}; \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}) \propto P(\mathbf{E}|\mathbf{H}; \boldsymbol{\phi})P(\mathbf{C}|\mathbf{H}; \boldsymbol{\psi})P(\mathbf{H}|\boldsymbol{\theta}) \quad (6.4)$$

The predictive model thus models the *proportional* probability of the event (selection of) heuristic \mathbf{H} , given allocation to entity \mathbf{E} and credit \mathbf{C} , parameterised by sampled $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$. These parameters are in turn parameterised by concentrations $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, γ_1 and γ_0 , which denote the prior beliefs of the BHH.

6.7.5 Naïve Bayes

This section aims to dissect the probabilistic model that is presented in Equation (6.4). The BHH implements a form of Naïve Bayes classifier, and thus independence between events can be assumed. The following derived PMFs are provided as fundamental building blocks to show the mechanism by which the BHH learns.

The independence between events for class label \mathbf{H} , simply yields the PMF of the Multinomial distribution as presented below:

$$\begin{aligned} P(\mathbf{H}|\boldsymbol{\theta}) &\propto \prod_{i=1}^I \prod_{k=1}^K P(h_{i,k}|\theta_k) \\ &\propto \prod_{i=1}^I \prod_{k=1}^K \theta_k^{\mathbb{1}_1(h_{i,k})} \\ &\propto \prod_{k=1}^K \theta_k^{\sum_{i=1}^I \mathbb{1}_1(h_{i,k})} \\ &\propto \prod_{k=1}^K \theta_k^{N_k} \end{aligned} \quad (6.5)$$

where N_k is a summary variable such that $N_k = \sum_{i=1}^I \mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the event h_i taking on class k in I independent, identical runs.

The independence between events \mathbf{E} , given class label \mathbf{H} , is denoted by the likelihood of \mathbf{E} , conditional to the occurrence of heuristic k and model parameter $\boldsymbol{\phi}$ as follows:

$$\begin{aligned}
P(\mathbf{E}|\mathbf{H}; \phi) &\propto \prod_{i=1}^I \prod_{j=1}^J \prod_{k=1}^K P(e_{i,j,k}|h_{i,k}; \phi_{j,k}) \\
&\propto \prod_{i=1}^I \prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{\mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})} \\
&\propto \prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{\sum_{i=1}^I [\mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})]} \\
&\propto \prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{N_{j,k}}
\end{aligned} \tag{6.6}$$

where $N_{j,k}$ is a summary variable such that $N_{j,k} = \sum_{i=1}^I \mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the events e_i taking on class j and h_i taking on class k , i.e. the count of occurrences of both entity j and heuristic k occurring together in I independent, identical runs.

Finally, the independence between events for the performance criteria \mathbf{C} , given class label \mathbf{H} , is denoted by the likelihood of \mathbf{C} , conditional to the occurrence of heuristic k and model parameter ψ as given below:

$$\begin{aligned}
P(\mathbf{C}|\mathbf{H}; \psi) &\propto \prod_{i=1}^I \prod_{k=1}^K P(c_{i,k}|h_{i,k}; \psi_k) \\
&\propto \prod_{i=1}^I \prod_{k=1}^K \psi_k^{\mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})} (1 - \psi_k)^{\mathbb{1}_0(c_{i,k})\mathbb{1}_1(h_{i,k})} \\
&\propto \prod_{k=1}^K \psi_k^{\sum_{i=1}^I \mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})} (1 - \psi_k)^{\sum_{i=1}^I \mathbb{1}_0(c_{i,k})\mathbb{1}_1(h_{i,k})} \\
&\propto \prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{N_{0,k}} \\
&\propto \prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})}
\end{aligned} \tag{6.7}$$

where N_k is the same summary variable as described for Equation (6.5). $N_{1,k}$ is a summary variable such that $N_{1,k} = \sum_{i=1}^I \mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the events c_i taking on a success (i.e. $c_i = 1$) and h_i taking on class k , i.e. the count of occurrences of both succeeding in the performance criteria and heuristic k occurring together in I independent, identical runs. Similarly, $N_{0,k} = N_k - N_{1,k}$ denotes the count of occurrences of the events c_i taking on a failure (i.e. $c_i = 0$) and h_i taking on class k .

Equations (6.5), (6.6) and (6.7) can be substituted into the proportional evaluation of the predictive model as given in Equation (6.4), resulting in

$$\begin{aligned}
P(\mathbf{H}|\mathbf{E}, \mathbf{C}; \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}) &\propto P(\mathbf{E}|\mathbf{H}; \boldsymbol{\phi})P(\mathbf{C}|\mathbf{H}; \boldsymbol{\psi})P(\mathbf{H}|\boldsymbol{\theta}) \\
&\propto \left[\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{N_{j,k}} \right] \left[\prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})} \right] \left[\prod_{k=1}^K \theta_k^{N_k} \right] \quad (6.8)
\end{aligned}$$

Consider the practical implementation of the predictive model as shown in Equation (6.8). Computationally, the equation presented in Equation (6.8) will underflow on a real computer if the resulting probabilities are very small.

6.7.6 Numerical Stability

When Equation (6.8) is evaluated, the numerical stability is shown to underflow if the resulting probabilities from its parts are very small. Multiplication of multiple fractional parameters leads to an even smaller fractional number. Probabilities might be very low at some points during training. Consider an example where training has stagnated, effectively leading to a scenario where a credit assignment strategy never fulfils a credit assignment, yielding an extremely small probability for $\boldsymbol{\psi}$. A solution to the aforementioned problem is to apply the *log-sum-exp* trick. The transformation of Equation (6.8) using the log-sum-exp trick is given as

$$LSE(P(h_k|e_j, c_1; \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi})) = \ln(\exp(\phi_{j,k}) + \exp(\psi_k) + \exp(\theta_k)) \quad (6.9)$$

The log-sum-exp trick as shown above caters for very small probabilities. However, there might be a situation where a random event is never seen, purely by chance. This results in a scenario referred to as *mode collapse*.

6.7.7 Mode Collapse

Mode collapse is the situation that occurs where the selective pressure towards a heuristic is close to zero as a result of random sampling, low initial bias, or bad performance. This leads to a situation where the probabilistic model continually decreases the selective pressure, until the selection probability of that heuristic is 0, yielding no further observations of that particular random event. The BHH addresses the aforementioned issue by

- using *symmetric* initialisation of the concentration parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, γ_1 and γ_0 ;
- setting the lower-bound of the concentration parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, γ_1 and γ_0 to 1, so that a selection probability of 0 is highly improbable; and

- continuously resampling heuristics throughout the training process, before and after optimisation, eliminating mode collapse as a result of just random sampling.

Another suggestion is to ensure that at least one of every heuristic is always selected during training. The [BHH](#) does not incorporate this approach, because this could result in a situation where a bad choice of heuristic is continually used throughout the training process, possibly resulting in bad update steps/selections. Instead, the [BHH](#) relies on the underlying learning process to control the selective pressure according to performance and nothing else. If this leads to a situation where mode collapse occurs, the [BHH](#) accepts the outcome.

6.8 Optimisation Step

Thus far all the fundamental components of the [BHH](#) have been presented. The final step that is missing is to present the optimisation step by which the learning process takes place. This section provides a solid mathematical explanation to show exactly how the [BHH](#) is able to learn. This section provides a brief overview of the role of concentration parameters and pseudo counts. Detailed discussions and mathematical descriptions follow for two techniques, known as [MLE](#) and [MAP](#), that are used to train Naïve Bayes classifiers.

6.8.1 Concentration Parameters and Pseudo Counts

The intent of the [BHH](#) is to gather evidence that can be used to update prior beliefs about which heuristics perform well during training. These beliefs are represented by the concentration parameters α , β , γ_1 and γ_0 . A change in prior beliefs is represented by a change in these concentration parameters. Specifically, it can be said that the optimisation process implemented by the [BHH](#) updates *pseudo counts* of events that are observed in the performance logs. These pseudo counts track the occurrence of a heuristic, an entity, and resulting performance of these two elements. Through the credit assignment strategy, these pseudo counts are biased towards entity-heuristic combinations that meet performance requirements and yield credit allocations.

The [BHH](#) is an adaptive [HH](#), meaning that it implements online learning. Online learning refers to a process where learning happens during the training process. Throughout the training process, concentration parameters are updated strategically to guide the selection mechanism towards heuristics that perform well. The learning capability of the [BHH](#) lies in carefully updating these concentration parameters. An-

other possibility is to introduce *a priori* information in the form of expert knowledge. A priori biases can be introduced by carefully setting the concentration parameters to values that are known to be good.

Generally, there are two different techniques that are used to train Naïve Bayes classifiers. The frequentist approach implements *maximum likelihood estimation* (MLE) and the Bayesian approach implements *maximum a posteriori estimation* (MAP). These methods are provided in Sections 6.8.2 and 6.8.3 respectively and show how the concentration parameters are updated throughout the training process, yielding the mechanism by which optimisation and learning takes place. It should be mentioned that the BHH makes use of MAP to optimise the predictive model. However, the details of MLE is provided as well as it contains fundamental mathematical building blocks that are required to present MAP.

6.8.2 Maximum Likelihood Estimation

In probability theory and statistics, *maximum likelihood estimation* (MLE) is a method that estimates the parameters of a prior probability distribution and is based on newly observed data and evidence. This section shows that the values for θ , ϕ and ψ can be estimated by MLE as follows:

$$\hat{\theta}_k = E[\theta_k] = \frac{N_k}{N} \quad (6.10)$$

$$\hat{\phi}_{j,k} = E[\phi_{j,k}] = \frac{N_{j,k}}{N_j} \quad (6.11)$$

$$\hat{\psi}_k = E[\psi_k] = \frac{N_{1,k}}{N_k} \quad (6.12)$$

The derivations of the MLE for $\hat{\theta}_k$, $\hat{\phi}_{j,k}$ and $\hat{\psi}_k$ are presented as follows: The log likelihood of $\hat{\theta}$, $\hat{\phi}$ and $\hat{\psi}$ as derived from the Equation (6.4) is given as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}) &= \ln \left(\left[\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{N_{j,k}} \right] \left[\prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})} \right] \left[\prod_{k=1}^K \theta_k^{N_k} \right] \right) \\
&= \ln \left(\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{N_{j,k}} \right) + \ln \left(\prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})} \right) + \ln \left(\prod_{k=1}^K \theta_k^{N_k} \right) \quad (6.13) \\
&= \left(\sum_{j=1}^J \sum_{k=1}^K N_{j,k} \ln(\phi_{j,k}) \right) \\
&\quad + \left(\sum_{k=1}^K N_{1,k} \ln(\psi_k) + (N_k - N_{1,k}) \ln(1 - \psi_k) \right) + \left(\sum_{k=1}^K N_k \ln(\theta_k) \right)
\end{aligned}$$

Equation (6.13) is broken down into each of its components. Consider the log likelihood of $\boldsymbol{\psi}$ as denoted by

$$\mathcal{L}(\boldsymbol{\psi}) = \sum_{k=1}^K N_{1,k} \ln(\psi_k) + (N_k - N_{1,k}) \ln(1 - \psi_k) \quad (6.14)$$

The MLE for $\hat{\psi}_k$ is calculated by taking the partial derivative of Equation (6.14) with respect to ψ_k and equating to zero as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{\psi})}{\partial \psi_k} &= N_{1,k} \frac{1}{\psi_k} + (N_k - N_{1,k}) \frac{-1}{(1 - \psi_k)} \\
\therefore 0 &= \frac{N_{1,k}(1 - \psi_k) + (N_{1,k} - N_k) \psi_k}{\psi_k(1 - \psi_k)} \\
\therefore 0 &= N_{1,k}(1 - \psi_k) + (N_{1,k} - N_k) \psi_k \\
\therefore 0 &= N_{1,k} - N_{1,k} \psi_k + N_{1,k} \psi_k - N_k \psi_k \\
\therefore 0 &= N_{1,k} - N_k \psi_k \\
\therefore N_k \psi_k &= N_{1,k} \\
\therefore \hat{\psi}_k &= \frac{N_{1,k}}{N_k}
\end{aligned} \quad (6.15)$$

The MLE for $\hat{\theta}_k$ can be calculated similarly. However, the $K - 1$ simplex S has to be compensated for by adding error factor, ϵ , to correct values for $\boldsymbol{\theta}$, where $\sum_k^K \theta_k \neq 1$. The new log likelihood function is then given as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \epsilon) &= \left(\sum_{k=1}^K N_k \ln(\theta_k) \right) + \epsilon \left(1 - \sum_{k=1}^K \theta_k \right) \\
&= \left(\sum_{k=1}^K N_k \ln(\theta_k) \right) + \left(\epsilon - \epsilon \sum_{k=1}^K \theta_k \right)
\end{aligned} \quad (6.16)$$

Solving first for ϵ yields

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \epsilon)}{\partial \epsilon} &= 1 - \sum_{k=1}^K \theta_k \\ \therefore \sum_{k=1}^K \theta_k &= 1\end{aligned}\tag{6.17}$$

Then solving for θ_k yields

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \epsilon)}{\partial \theta_k} &= N_k \frac{1}{\theta_k} + \epsilon(-1) \\ \therefore \frac{N_k}{\theta_k} &= \epsilon \\ \therefore N_k &= \theta_k \epsilon \\ \therefore \sum_{k=1}^K N_k &= \sum_{k=1}^K \theta_k \epsilon \\ \therefore N &= \epsilon \sum_{k=1}^K \theta_k \\ \therefore N &= \epsilon\end{aligned}\tag{6.18}$$

Substitution of $N = \epsilon$ back into Equation (6.18) yields

$$\begin{aligned}N_k &= \theta_k \epsilon \\ N_k &= \theta_k N \\ \hat{\theta}_k &= \frac{N_k}{N}\end{aligned}\tag{6.19}$$

The MLE for $\hat{\phi}_{j,k}$ is calculated similarly. To compensate for the $K - 1$ simplex S , an error, $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_J)$ is added as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\lambda}) &= \left(\sum_{j=1}^J \sum_{k=1}^K N_{j,k} \ln(\phi_{j,k}) \right) + \sum_{j=1}^J \lambda_j \left(1 - \sum_{k=1}^K \phi_{j,k} \right) \\ &= \left(\sum_{j=1}^J \sum_{k=1}^K N_{j,k} \ln(\phi_{j,k}) \right) + \sum_{j=1}^J \lambda_j - \sum_{j=1}^J \lambda_j \sum_{k=1}^K \phi_{j,k}\end{aligned}\tag{6.20}$$

Solving first for λ_j yields

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\lambda})}{\partial \lambda_j} &= 1 - \sum_{k=1}^K \phi_{j,k} \\ \therefore \sum_{k=1}^K \phi_{j,k} &= 1\end{aligned}\tag{6.21}$$

Then solving for $\phi_{j,k}$ yields

$$\begin{aligned}
 \frac{\partial \mathcal{L}(\phi, \lambda)}{\partial \phi_{j,k}} &= N_{j,k} \frac{1}{\phi_{j,k}} - \lambda_j \\
 \therefore \frac{N_{j,k}}{\phi_{j,k}} &= \lambda_j \\
 \therefore N_{j,k} &= \phi_{j,k} \lambda_j \\
 \therefore \sum_{k=1}^K N_{j,k} &= \sum_{k=1}^K \phi_{j,k} \lambda_j \\
 \therefore N_j &= \lambda_j \sum_{k=1}^K \phi_{j,k} \\
 \therefore N_j &= \lambda_j
 \end{aligned} \tag{6.22}$$

Substitution of $N_j = \lambda_j$ back into Equation (6.22) yields

$$\begin{aligned}
 N_{j,k} &= \phi_{j,k} \lambda_j \\
 N_{j,k} &= \phi_{j,k} N_j \\
 \hat{\phi}_{j,k} &= \frac{N_{j,k}}{N_j}
 \end{aligned} \tag{6.23}$$

6.8.3 Maximum A Posteriori Estimation

Another approach to optimise the values for $\hat{\theta}_k$, $\hat{\phi}_{j,k}$ and $\hat{\psi}_k$ is to optimise the parameters by their probability distributions' parameters. This process is referred to as *Bayesian analysis*. This section provides the mathematical details of Bayesian analysis and *maximum a posteriori estimation* (MAP) as it is implemented in the BHH.

Bayesian analysis makes use of the *posterior* probability distribution. The posterior distribution is simply expressed proportionally as

$$POSTERIOR \propto LIKELIHOOD \times PRIOR \tag{6.24}$$

The event \mathbf{H} is a multinomial distribution with probability parameter $\boldsymbol{\theta}$. It is known that the conjugate prior to a multinomial distribution is a Dirichlet probability distribution. The *prior* probability distribution for $\boldsymbol{\theta}$ is thus presented as

$$P(\boldsymbol{\theta}|\boldsymbol{\alpha}) \propto \prod_{k=1}^K \theta_k^{\alpha_k-1} \tag{6.25}$$

Furthermore, the event \mathbf{E} is also a multinomial distribution with parameter $\boldsymbol{\phi}$.

The *prior* probability distribution for ϕ is thus presented as

$$P(\phi|\beta) \propto \prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{\beta_{j,k}-1} \quad (6.26)$$

Finally, the event \mathbf{C} is a Binomial probability distribution with probability parameter ψ . It is known that the conjugate prior to a Binomial probability distribution is a Beta probability distribution. The prior probability distribution for ψ is thus presented as

$$P(\psi|\gamma_1, \gamma_0) \propto \prod_{k=1}^K \psi_k^{\gamma_{1,k}} (1 - \psi_k)^{\gamma_{2,k}} \quad (6.27)$$

Putting the likelihood and priors together yields the posterior distribution as given below:

$$\begin{aligned} P(\theta, \phi, \psi | \mathbf{H}, \mathbf{E}, \mathbf{C}; \alpha, \beta, \gamma_1, \gamma_0) &\propto P(\mathbf{H} | \mathbf{E}, \mathbf{C}; \theta, \phi, \psi) P(\theta, \phi, \psi | \alpha, \beta, \gamma_1, \gamma_0) \\ &= P(\mathbf{E} | \mathbf{H}; \phi) P(\mathbf{C} | \mathbf{H}; \psi) P(\mathbf{H} | \theta) P(\phi | \beta) P(\psi | \gamma_1, \gamma_0) P(\theta | \alpha) \\ &= \left[\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{N_{j,k}} \right] \left[\prod_{k=1}^K \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})} \right] \left[\prod_{k=1}^K \theta_k^{N_k} \right] \\ &\times \left[\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{\beta_{j,k}-1} \right] \left[\prod_{k=1}^K \psi_k^{\gamma_{1,k}-1} (1 - \psi_k)^{\gamma_{2,k}-1} \right] \\ &\times \left[\prod_{k=1}^K \theta_k^{\alpha_k-1} \right] \\ &= \left[\prod_{j=1}^J \prod_{k=1}^K \phi_{j,k}^{(N_{j,k} + \beta_{j,k})-1} \right] \\ &\times \left[\prod_{k=1}^K \psi_k^{(N_{1,k} + \gamma_{1,k})-1} (1 - \psi_k)^{(N_{0,k} + \gamma_{2,k})-1} \right] \\ &\times \left[\prod_{k=1}^K \theta_k^{(N_k + \alpha_k)-1} \right] \end{aligned} \quad (6.28)$$

From Equation (6.28) above, it can be seen that the posterior distribution has the same form, $\mathcal{A}(v)$, as the prior distribution. The concentration update operations are then be given as

$$\alpha_k(t+1) = N_k + \alpha_k(t) \quad (6.29)$$

$$\beta_{j,k}(t+1) = N_{j,k} + \beta_{j,k}(t) \quad (6.30)$$

$$\gamma_{1,k}(t+1) = N_{1,k} + \gamma_{1,k}(t) \quad (6.31)$$

$$\gamma_{2,k}(t+1) = N_{0,k} + \gamma_{2,k}(t) \quad (6.32)$$

It can be seen that the prior and posterior probability distributions are of the same form, $\mathcal{A}(v)$. It can therefore be said that the [BHH](#) implements a Gaussian process [62]. Since the reselection of heuristics happens at regular intervals, the outcome of a selection in one iteration may influence the outcome of another in the next iteration, making the implementation of the [BHH](#) a [HMM](#) [130].

6.9 Hyper-Parameters

This section provides a breakdown of all the hyper-parameters that were identified and briefly mentioned in this chapter. The following hyper-parameters are presented: *heuristic pool*, *population size*, *credit assignment strategy*, *reselection interval*, *replay window size*, *reanalysis interval*, *burn in window size*, *discounted rewards* and *normalisation*. Some logical arguments can be made as to what their values should be. However it is still up to empirical analysis to determine the effects of certain hyper-parameter design decisions. Where possible, a range of possible values are provided.

6.9.1 Heuristic Pool

Section 6.3 provided the reader with the context and detail of the heuristic pool as it is included in the architecture of the [BHH](#). A detailed discussion was given around the importance of diversity and balancing exploration and exploitation capabilities. The heuristic pool hyper-parameter refers to the heuristic-pool configuration that is used. The configuration of the heuristic pool should account for the following factors:

- The type of low-level heuristics to include in the heuristic pool. This dissertation investigates two main groups, including gradient-based heuristics and [MHs](#).
- The low-level heuristics' hyper-parameters. At a lower-level, each of these heuristics has their own set of hyper-parameters.

- The heuristic pool size. This refers to the number of heuristics in the heuristic pool.

The design choices at a high level are not as strict for the BHH as it is for low-level heuristics. If the choice of hyper-parameters is uncertain, multiple configurations of a low-level heuristic, each with its own unique hyper-parameter configurations, can be included in the heuristic pool.

6.9.2 Population Size

Population size refers to the number of entities included in the entity pool. The population size to use is to be determined empirically. Naturally, an assumption is made that a larger population could lead to better results, assuming that entities are initialised uniformly across the solution search space. However, larger population sizes are computationally more expensive. Oldewage [120] mentions that a large number of particles (entities) may be able to traverse a greater portion of the search space, because every particle provides additional information about the search space. However, it is to be determined if this is indeed the case for the BHH. The following factors need to be considered when picking a population size:

- When considering the lower bound of possible population sizes, take into account the highest, minimum population size required by all low-level heuristics. For example, if DE is included into the heuristic pool, a population size of at least four is required. For PSO, the minimum population size is three.
- When considering the upper bound of possible population sizes, take into account that a bigger population size results in a more computationally expensive training process. Furthermore, there could exist a point of *diminishing returns* where an increase in population size does not yield any better outcome.

6.9.3 Credit Assignment Strategy

Five different credit assignment strategies were presented in Section 6.6. The choice of credit assignment is assumed to be problem dependent. However, this is to be tested empirically. Along with the credit assignment strategy itself, the following factors related to the credit assignment strategy should be considered:

- When to apply credit assignment during the training process.
- What should the credit assignment score/value be (default = 1).

- To what degree should credit assignment be allocated to past results in the performance log.

A number of other hyper-parameters are specifically included to address the list of considerations as given above. These include the *reselection interval*, *replay window size* and *reanalysis interval*, *burn in window size*, *discounted rewards* and *normalisation*.

6.9.4 Reselection Interval

The reselection interval is a hyper-parameter that controls how often heuristic selections are resampled from the heuristic distribution, parameterised by the heuristic selection probability that is learnt. The choice of reselection should be influenced by the following considerations: If reselection happens too frequently, heuristics are not allowed sufficient time to achieve their goals and to smooth out update steps. If reselection happens too infrequently, it does not allow for sufficient collection of samples/evidence from which learning is done. Naturally, the correct reselection interval to use is to be determined empirically.

6.9.5 Replay Window Size

The replay window size is a hyper-parameter that controls how much historical performance information is kept in the performance log for the BHH to learn from and was introduced in Section 6.5. The replay window size is a parameter that is borrowed from *reinforcement learning* (RL) and aims to control the effect that past performances have on the BHH. A replay window size that is too small includes little to no memory, resulting in a situation where the BHH simply looks at the most recent evidence that is collected and nothing more. On the contrary, if the replay window size is large, the BHH tracks a longer history of past performances, leading to a situation where previously well-performing heuristics that are not relevant later in the training process, are still selected often.

6.9.6 Reanalysis Interval

The reanalysis interval is a hyper-parameter that controls how often the BHH updates its past beliefs (priors). The reanalysis interval goes hand in hand with the reselection interval and replay window size. If the reanalysis interval is too small, the BHH prematurely updates its priors. Similar to other hyper-parameters discussed thus far, the correct value to use for the reanalysis interval is to be determined empirically.

However, some logical exclusions can be made. A reanalysis interval that is smaller than the reselection interval does not make sense, since the effects of reanalysis are only realised during reselection. A reanalysis interval that is too big could lead to a situation where the delay in updating priors could be detrimental to the outcomes of the [BHH](#).

6.9.7 Burn In

Burn in is a hyper-parameter that is borrowed from *Markov Chain Monte Carlo* ([MCMC](#)) and aims to delay the learning process of the [BHH](#). The intent of the burn in hyper-parameter is to determine if heuristics need time to execute before reanalysis starts. Notice that a delay in the learning process does not mean that reselection does not occur or that performance information is not collected in the replay window. Therefore, it should be noted that the burn in window size, reselection interval, replay window size, and reanalysis interval should be considered together. If the replay window size is smaller than the burn in window size, performance information is discarded and an opportunity to learn is lost.

6.9.8 Discounted Rewards

Discounted rewards is another hyper-parameter that is borrowed from [RL](#) and is a flag that determines if discounted rewards are applied or not. Discounted rewards refer to an implementation where credit assignments for past performances are exponentially decreased further into the past. An exponential decay factor of 0.5 is proposed. The intent of the discounted rewards flag is to scale the effect of past performances that are further back in the performance log, such that they have a smaller impact than more recent evidence on the learning process of the [BHH](#).

6.9.9 Normalisation

Normalisation is a hyper-parameter that provides a mechanism of control for the degree to which exploration is allowed. Figure [6.2](#) shows the effect of different concentrations (α and β) on the Beta probability distribution. By normalising the concentration parameters for the [BHH](#) (α , β , γ_1 and γ_0), the variance of the selection probability distributions increases, allowing for sampled probabilities further from the mean and thus allowing for more exploration.

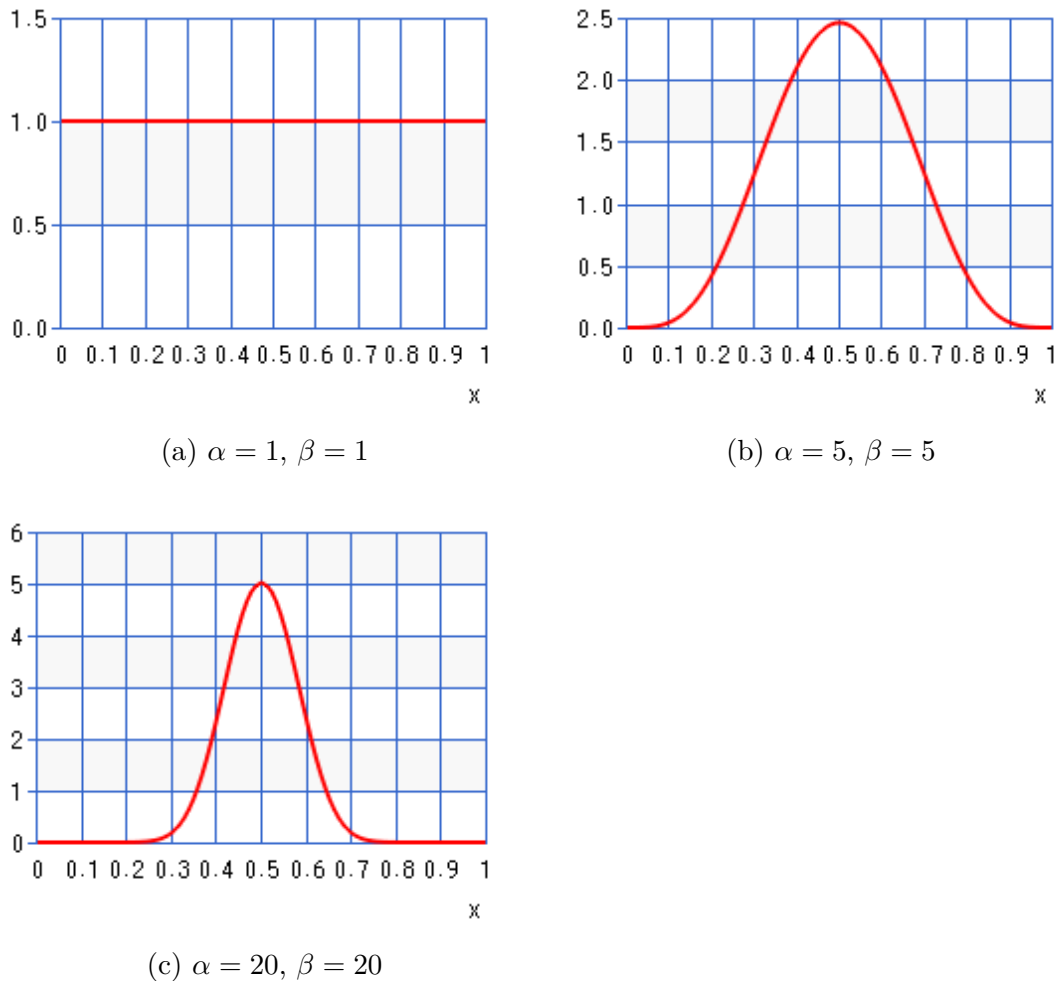


Figure 6.2: The Beta probability distribution with varying α and β values.

6.9.10 Defaults

Section 6.3.3 provided the reader with the concept of proxied heuristic update step operations. As was mentioned in Section 6.9.1, low-level heuristics each have their own set of hyper-parameters as well. A set of default low-level parameters must be allocated specifically for these proxies. Consider a scenario where two instances of *Adam* is included in the heuristic pool. Each has its own set of hyper-parameters that differ from each other. In the case of *Adam*, there are four hyper-parameters that include learning rate, η , β_1 , β_2 and ϵ . If another heuristic needs to proxy *Adam*'s expected gradient mean operation, a default β_1 parameter must be supplied that is applied by the proxy mechanism. This is only the case when multiple instances of a heuristic is included and there is uncertainty about which instance's hyper-parameters to use for the proxied heuristic update steps. If there is only one instance of a particular heuristic, that instance's hyper-parameters are used.

6.10 The BHH Algorithm

The high level pseudo-code implementation of the BHH is given in Algorithm 9.

6.11 Summary

This chapter provided extensive detail on the inner workings and design of the BHH. The BHH was formally classified and the details around various components of the BHH's architecture was presented. Formal mathematical descriptions of the Bayesian selection method have been provided. The selection mechanism was presented as a probabilistic, predictive classifier. The optimisation process, by means of Bayesian analysis, has also been presented. Hyper-parameters were discussed in detail and a pseudo-code implementation of the BHH was presented.

This methodology for the empirical process is provided in the following chapter.

Algorithm 9 The pseudo-code for the implementation of the *Bayesian hyper-heuristic* (BHH)

```

step  $\leftarrow$  0
select initial heuristics
initialise population and entities
evaluate entities' initial position
update population state
while stopping condition not met do
  for all entities in entity pool do
    if selected heuristic is gradient-based then
      get gradients
    end if
    apply low-level heuristic and proxy operations
    update population state
    log performance metrics to performance log
    if step < burn-in window size then
      select heuristic
    else
      if step % reanalysis interval = 0 then
        apply Bayesian analysis
      end if
      if step % reselection interval = 0 then
        select heuristic
      end if
      if step > replay window size then
        prune performance log
      end if
    end if
  end for
  step  $\leftarrow$  step + 1
end while

```

Chapter 7

Methodology

“Observation, reason and experimentation make up what we call the scientific method.”

- Richard P. Feynman

So far, this dissertation has provided background information on *artificial neural networks* (ANNs), heuristics, HHs and probability theory in Chapters 2 to 5. Chapter 6 provided the detail of the implementation of the proposed BHH with all its components and hyper-parameters. Scientific experimentation can now be conducted. Chapter 1 identified a set of empirical goals for this dissertation. The goals of this dissertation include an empirical evaluation of the BHH, its behaviour and performance, and a comparison to state of the art low-level heuristics. This chapter provides the detailed specification of the methodology that is followed for the empirical process. Details are provided on the implementation of experiments, datasets, selection of hyper-parameters and design decisions. The process by which the results are statistically analysed are also presented. The remainder of the chapter is structured as follows:

- **Section 7.1** provides a brief overview of the overall empirical process.
- **Section 7.2** presents the detail around the datasets that are used.
- **Section 7.3** provides the details of the models (FFNNs) that are trained.
- **Section 7.4** presents the detail around the different heuristics that are used along with their hyper-parameters.
- **Section 7.5** provides the detail of the configuration of the BHH baseline.
- **Section 7.6** sheds light into the performance evaluation measures that are used.

- **Section 7.7** discusses the stopping conditions that are used.
- **Section 7.8** presents the different experimental groups that are executed.
- **Section 7.9** presents the procedures that are followed for the statistical analysis of the results.
- **Section 7.10** sheds light on the implementation and execution of the empirical process.
- **Section 7.11** presents a brief summary of the chapter.

7.1 Overview of Empirical Process

The purpose of the empirical process is to conduct a carefully crafted set of experiments that produce data that can be used to reject or verify a hypothesis about the element under investigation. Chapter 1 identified a number of empirical tests to execute. These include:

- An empirical study to show that the **BHH** can effectively be used to train **FFNNs**.
- An empirical study to investigate the behavioural characteristics of the **BHH** as it is used to train **FFNNs** on an example problem.
- An empirical study to critically evaluate the performance of the **BHH** compared to individual low-level heuristics in the heuristic space as they are used to train **FFNNs** on a number of different problems.

These empirical tests represent a set of questions, related to the **BHH**, that need to be answered. The empirical process is designed to answer these questions. The empirical process is structured as follows.

Each empirical test starts with a question to be answered. A hypothesis is formulated for the outcome of the empirical test. The empirical tests are then designed around the implementation of the components under evaluation. Each empirical test defines the configuration of elements and generally include a set of parameters that are altered between experiments. Each experiment is evaluated by means of a performance measurement. In the context of training **FFNNs**, the underlying model is trained across a number of datasets. Each dataset is split into a training set comprising of 80% of the data, and a test set comprising of 20% of the data. The training set is used to train the model, while the test set is used to

evaluate the model's performance on unseen data. Training epochs are split into mini-batch iterations with a specified mini-batch size per dataset. Evaluation takes place at every mini-batch step. Due to the stochastic nature of the experiments, each experiment is repeated over 30 independent runs to provide sufficient sample for statistical certainty about the outcome of the results. Each run contains a different random seed, such that each run is unique. The evaluation data forms the results of the empirical test. These results are analysed for statistical significance, from which findings and conclusions are then made. The null hypothesis is then either rejected or accepted based on these findings.

Details around each of these elements is provided in the following sections.

7.2 Datasets

This section provides the detail around the different datasets that are used throughout the empirical process. These datasets originate from the UCI Machine Learning Repository [40]. Datasets are grouped by problem type and include seven classification and seven regression datasets. The classification datasets are given in Table 7.1 and the regression datasets are given in Table 7.2 below.

Table 7.1: Classification datasets

dataset	output	types	attributes	classes	instances	batch	steps	citation
iris	multivariate	real	4	3	150	16	10	[51]
car	multivariate	categorical	6	4	1728	128	14	[14]
abalone	multivariate	categorical, integer, real	8	28	4177	256	17	[176]
wine quality	multivariate	real	12	11	4898	256	20	[26]
mushroom	multivariate	categorical	22	2	8214	512	17	[143]
bank	multivariate	real	17	2	45211	512	89	[112]
diabetic	multivariate	integer	55	3	100000	1024	98	[156]

Table 7.2: Regression datasets

dataset	output	types	attributes	instances	batch	steps	citation
fish toxicity	multivariate	real	7	908	64	15	[23]
housing	univariate	real	13	506	32	16	[72]
forest fires	multivariate	real	13	517	32	17	[24]
student performance	multivariate	integer	33	649	32	21	[25]
parkinsons	multivariate	integer, real	26	5875	256	23	[166]
air quality	multivariate, time series	real	15	9358	256	37	[34]
bike	univariate	integer, real	16	17389	256	68	[48]

Details on how the datasets where preprocessed and prepared is given in Appendix C.

7.2.1 Class Balancing

A number of classification datasets given in Table 7.1 contain an unbalanced representation of classes. The empirical process defined in this dissertation does not apply mechanisms to cater for class balancing, in order to eliminate as many variables and factors in the empirical process as possible. It is therefore suggested that the BHH first be studied under the assumption of balanced classes, before applying class balancing techniques. In future research opportunities, class balancing should be utilised.

7.3 Models

All models that are trained in this dissertation follow implementations of shallow FFNNs, meaning that they only have one hidden layer. The architecture of a model is dependent on the dataset being trained on, the type of optimisation problem, the number of input dimensions and the number of output dimensions. The number of hidden layers used were determined empirically. Weights are initialised by means of Glorot uniform sampling [58]. The models and their configuration, as it is used for each dataset, are given in Table 7.3.

Table 7.3: Model configurations

dataset	inputs	hidden	output	biases	parameters	topology	l1 activation	l2 activation
fish toxicity	6	3	1	yes	25	dense	LReLU ($\alpha = 0.3$)	sigmoid
iris	4	5	3	yes	43	dense	LReLU ($\alpha = 0.3$)	softmax
air quality	12	8	1	yes	113	dense	LReLU ($\alpha = 0.3$)	sigmoid
housing	13	8	1	yes	121	dense	LReLU ($\alpha = 0.3$)	sigmoid
wine quality	13	10	7	yes	217	dense	LReLU ($\alpha = 0.3$)	softmax
parkinsons	21	10	1	yes	231	dense	LReLU ($\alpha = 0.3$)	sigmoid
car	21	10	4	yes	264	dense	LReLU ($\alpha = 0.3$)	softmax
forest fires	43	16	1	yes	721	dense	LReLU ($\alpha = 0.3$)	sigmoid
abalone	10	36	28	yes	1432	dense	LReLU ($\alpha = 0.3$)	softmax
bank	51	32	1	yes	1697	dense	LReLU ($\alpha = 0.3$)	softmax
bike	61	32	1	yes	2017	dense	LReLU ($\alpha = 0.3$)	sigmoid
student performance	99	32	1	yes	3233	dense	LReLU ($\alpha = 0.3$)	sigmoid
adult	108	64	1	yes	7041	dense	LReLU ($\alpha = 0.3$)	softmax
mushroom	117	64	1	yes	7617	dense	LReLU ($\alpha = 0.3$)	softmax
diabetic	2369	32	3	yes	75939	dense	LReLU ($\alpha = 0.3$)	softmax

For the classification problems presented in Tables 7.1 and 7.3, the softmax activation function is added after training and is not included in the model during training. The loss functions, *sparse categorical cross entropy* (SparseCatXE) and *binary cross entropy* (BinXE), that are used, contain a softmax function.

7.4 Heuristics

This section provides the details of the low-level heuristics that are used in the empirical process. Each of these low-level heuristics is implemented as standalone heuristics and is also included in the heuristic pool of the BHH. Table 7.4 contains a list of all the standalone, low-level heuristics that are used as well as their hyper-parameter configurations.

Note from Table 7.4 that values that are configured to make use of a decay schedule are presented with the initial value first and the decay rate in brackets next to it. The number of steps is the total number of mini-batch training steps that are executed for that particular dataset.

Furthermore, it should be noted that a learning rate was added to PSO as an attempt to avoid overshooting solutions later in the training process. A learning rate parameter does not traditionally form part of PSO, but was found to be useful. The position update step for PSO using a learning rate is then defined as

$$x_{ij}(t+1) = x_{ij}(t) + \eta v_{ij}(t+1) \quad (7.1)$$

where η is the added learning rate with $0 \leq \eta \leq 1, \eta \in \mathbb{R}$.

Section 6.3.3 presented the concept of a mapping of proxied heuristic state update operations. The mapping of proxied heuristic state update operations implemented by the BHH in the empirical process is given in Figure 7.1.

In Figure 7.1, cells containing **x** indicate that the associated heuristic implements that particular state parameter explicitly, and cells containing **o** indicate that the state parameter is implicitly implemented as part of the BHH algorithm. The required mapping of proxied heuristic state operations is then implemented as a lookup of the table presented in Figure 7.1.

7.5 BHH Baseline

The BHH baseline is a name given to a specific configuration of the BHH which, during development, has been found to provide a reasonable baseline performance. The baseline configuration is used as the cornerstone configuration from which all other heuristics and their configurations are evaluated. The BHH baseline is also used for the behavioural study of the BHH. The BHH baseline configuration is given in Table 7.5.

In Table 7.5, the heuristic pool configuration that is used, is referred to as *all*. The *all* configuration refers to a configuration where the heuristic pool contains

Table 7.4: Low-level heuristics and their hyper-parameter configurations.

heuristic	configuration	value	citation
sgd	learning rate	0.1 (0.01)	[157]
momentum	learning rate	0.1 (0.01)	[157]
	momentum	0.9	
nag	learning rate	0.1 (0.01)	[157]
	momentum	0.9	
adagrad	learning rate	0.1 (0.01)	[41]
	epsilon	1E-07	
rmsprop	learning rate	0.1 (0.01)	[74]
	rho	0.95	
	epsilon	1E-07	
adadelta	learning rate	1.0 (0.95)	[182]
	rho	0.95	
	epsilon	1E-07	
adam	learning rate	0.1 (0.01)	[94]
	beta1	0.9	
	beta2	0.999	
	epsilon	1E-07	
pso	population size	10	[170]
	learning rate	1.0 (0.9)	
	inertia weight (w)	0.729844	
	cognitive control (c1)	1.49618	
	social control (c2)	1.49618	
	velocity clip min	-1.0	
	velocity clip max	1.0	
de	population size	10	[107]
	selection strategy	best	
	xo strategy	exp	
	recombination probability	0.9 (0.1)	
	beta	2.0 (0.1)	
ga	population size	10	[98]
	selection strategy	rand	
	xo strategy	bin	
	mutation rate	0.2 (0.05)	

notes			Entity State					Population State				
heuristic	hyper-parameter	position	velocity	gradient	sum of gradients squared	expected position variance	expected gradient mean (hp1)	expected gradient variance (hp2)	pbest	ibest	rbest	gbest
sgd momentum	learning rate	x	x	x	-	-	-	-	-	-	-	x
	learning rate, momentum (maps to hp1)	x	x	x	-	-	x	-	-	-	-	x
nag	learning rate, epsilon	x	x	x	-	-	x	-	-	-	-	-
adagrad	learning rate, rho (maps to hp2), momentum (maps to hp1), epsilon	x	x	x	x	-	-	-	-	-	-	x
rmsprop	learning rate, rho (maps to hp2), momentum (maps to hp1), epsilon	x	x	x	-	-	-	x	-	-	-	x
adadelat	rho (maps to hp2), epsilon	x	x	x	-	x	-	x	-	-	-	x
adam	learning rate, momentum (maps to hp1), rho (maps to hp2)	x	x	x	-	-	x	x	-	-	-	x
PSO	W, C1, C2	x	x	o	-	-	-	-	x	o	-	x
DE		x	o	o	-	-	-	-	-	o	-	x
GA	mutation rate	x	o	o	-	-	-	-	-	-	-	x
BHH	burn in, replay window size, population size, reselection and reanalysis window size, normalisation, discounted rewards	x	x	x	x	x	x	x	x	x	x	x

Figure 7.1: Mapping of proxied heuristic state update operations as implemented by the BHH

Table 7.5: The [BHH](#) baseline configuration as it is used in the empirical study.

hyper-heuristic	variant	configuration	value
bhh	baseline	heuristic pool	all
		population	5
		credit	ibest
		burn in	0
		reselection	10
		replay	10
		reanalysis	10
		normalise	false
		discounted rewards	false

all the low-level heuristics as presented in Section 7.4, including all gradient-based heuristics and meta-heuristics.

7.6 Performance Measures

This section provides the performance measures that are used to evaluate the different experimental runs during the empirical process. Chapter 2 provided a number of loss functions. These loss functions are used to measure the performance of the [FFNNs](#) being trained. In this dissertation, [BinXE](#) is used for classification problems with two classes and [SparseCatXE](#) is used for classification problems with more than two classes. For the classification problems, accuracy is also measured. Accuracy refers to the proportion of correct classifications. For regression problems, the [RMSE](#) is used as a performance metric.

The test set is used as a validation set during training. All performance metrics are measured for the training set as well as the test/validation set during training. As such, a divergence in performance metrics between the training and test/validation set is used to evaluate for overfitting during the training process.

After training has been completed, the *average rank*, based on test loss, for all configurations, is calculated at each mini-batch step. A total of 30 independent runs with different random seed configurations are executed for each empirical test. The test loss metric and average rank are then statistically analysed across all runs, and are used to derive findings and conclusions.

7.7 Stopping Conditions

This section provides the stopping conditions that are used during the empirical process. For this dissertation, the *maximum epoch* stopping condition is used, where a fixed, maximum number of training steps and epochs are executed and training is not halted until the maximum number of epochs have been reached. [FFNNs](#) are trained for a maximum of 30 epochs. Early stopping is a technique where training is stopped when the heuristic is not able to improve the performance of the [FFNN](#) for a number of steps. If the performance has not improved, the training is halted and the last known best model weights are restored. Early stopping is not implemented for the empirical process in this dissertation. This design decision is made so that the behaviour of the [BHH](#) can be studied beyond the point of optimal results.

7.8 Experiments

This section presents the experimental groups that are executed in the empirical process. Three main experimental groups are formulated. Each of these is discussed in more detail in the following sections.

7.8.1 Behavioural Case Study

The behavioural case study analyses the behaviour of the [BHH](#) baseline configuration as it relates to an example problem dataset, across 30 independent runs. The example problem dataset is the iris dataset, presented in Table 7.1. The behavioural case study is meant to provide an introductory analysis of the inner workings of the [BHH](#) and includes analysis of the selection mechanism, as well as the perturbative nature of the [BHH](#). The behavioural case study is used to determine if the [BHH](#) is learning and that selection is indeed biasing towards better performance. These observations also provide an opportunity to observe the outcome of the perturbative nature of the [BHH](#), which includes proxied heuristic update step operations.

The behavioural case study provides an implementation of the [BHH](#) baseline which, by default, has a replay window size of 10. The baseline configuration is provided and compared to an implementation of a [BHH](#) configuration where the replay window size is large (250), as well as an implementation of the [BHH](#) using the *symmetric* credit assignment strategy. The large replay window size configuration is used to show the behaviour of the [BHH](#) when it has access to a large number of performance log samples, which increases the statistical certainty of the learning outcome. The [BHH](#) configuration that uses the *symmetric* credit assignment strategy

is used to illustrate the behaviour of the BHH where no performance bias takes place and thus, no learning occurs.

7.8.2 Standalone Heuristics

For the standalone heuristics experimental group, the heuristics, as presented in Section 7.4, are used along with their specified hyper-parameters. Each of these standalone low-level heuristics is compared to that of the BHH baseline configuration, across all datasets.

The intent of the standalone heuristics experimental group is to determine if the BHH baseline configuration can generalise to multiple problems in comparison to individual low-level heuristics.

Additional to the BHH baseline configuration, two more BHH configurations are included. These include BHH configurations where the heuristic pool only makes use of gradient-based heuristics, and a configuration where the heuristic pool only makes use of MHs. The intent behind the inclusion of these configurations is to determine the effectiveness of multi-method approaches in the heuristic pool as it applies to training FFNNs.

7.8.3 BHH Variants

This section provides the details of the experimental group that focuses on BHH variants and the effect that different hyper-parameter configurations can have on the outcome of the BHH. The different variants of the BHH and their possible configurations are given in Table 7.6.

Table 7.6: BHH variants and their configuration

hyper-heuristic	variant	values
bhh	heuristic pool	all, gd, mh
	population	5, 10, 15, 20, 25
	credit	ibest, pbest, rbest, gbest, symmetric
	reselection	1, 5, 10, 15, 20
	replay	1, 5, 10, 15, 20
	reanalysis	1, 5, 10, 15, 20
	burn in	0, 5, 10, 15, 20
	normalise	false, true
	discounted rewards	false, true

In Table 7.6, the heuristic pool configuration *gd* refers to the heuristic pool

configuration where only gradient-based heuristics are included, and *mh* refers to the heuristic pool configuration where only [MHs](#) are included.

7.9 Statistical Analysis

This section provides the detail of the process that is used to execute the statistical analysis of the results.

Various experimental groups and configurations have been presented in Sections [7.8.1](#), [7.8.2](#) and [7.8.3](#). The results from each of these experimental groups is statistically analysed. To ensure that there are sufficient samples for statistical analysis, each experiment and configuration is trained for 30 epochs and is repeated over 30 runs, for each of the fourteen datasets. The experimental results contain performance evaluation data for the training and testing datasets, and includes loss and accuracy measurements. Statistical analysis is executed on the results from the testing datasets. An average rank is calculated across all 30 runs, for all experimental groups and configurations, at each step, for every epoch of training. Each run is executed using a unique random seed, such that each run is identical in its setup and configuration, but unique between runs.

The evaluation of outcome is based on the aggregated statistical results as mentioned above. A descriptive analysis is executed on the spread of the data. The results are analysed and checked for overfitting and outliers are identified. The skewness of the results is evaluated per dataset and the Shapiro-Wilk test for normality [\[145\]](#) ($\alpha = 0.001$) is used to determine if the results are normally distributed. Furthermore, the Levene's test for equality of variance [\[102\]](#) ($\alpha = 0.001$) is used. The output of the full statistical analysis is presented in [Appendix D](#).

Dependent on the outcomes of the above statistical tests, the appropriate statistical significance test is then executed. For experiments where there are only two configurations, the Mann-Whitney U independent samples t-Test [\[106\]](#) ($\alpha = 0.001$) is executed. For experiments with three or more configurations, the [ANOVA](#) statistical test [\[52\]](#) ($\alpha = 0.001$) is used. The Kruskal-Wallis ranked non-parametric test [\[95\]](#) for statistical significance ($\alpha = 0.001$) is used for cases where data is not normally distributed.

Regardless of the statistical test that is used, a post-hoc Tukey honest significant difference test [\[167\]](#) ($\alpha = 0.001$) is used from which significant ranking is retrieved. Descriptive and critical difference plots are then retrieved from these results to provide visual aid.

7.10 Implementation and Execution

All implementations are done from first principles in Python 3.9 using Tensorflow 2.7 and Tensorflow Probability 0.15.0. Most underlying mathematical functions are reused from the Tensorflow library, however, all heuristics are implemented from first principles to fit the HH framework that was developed. The source code and data for this dissertation is provided and made public at <https://github.com/arneschreuder/masters>.

It should be noted that this implementation makes heavy use of CPU processing, due to flattening of the FFNN weights by the heuristics. For this reason, execution is much more timely and costly than with GPU training.

All experiments were run on the Centre for High Performance Computing (CHPC) cluster. A total of 14 different server were used, each running 24 to 56 cores and 256GB of memory. The entire empirical process took six days.

7.11 Summary

This chapter provided the detail around the methodology that was used to execute the empirical process. The datasets, FFNNs and heuristics that are used during the empirical process have been presented in detail, along with details around various BHH configurations. A baseline BHH has been formulated. The empirical process was defined in terms of a number of different experimental groups and finally, the process of statistical analysis of the results was provided.

The results and findings of the empirical process are presented in the following chapter.

Chapter 8

Results

“Without data, you are just another person with an opinion.”

- W. Edwards Deming

The last step in the scientific process requires the presentation of the outcomes of the empirical process. Objective discussions based on statistical analysis of the findings are required. The design and methodology of the empirical process was presented in Chapter 7. The methodology presented a number of experimental groups to conduct. The experimental groups that were identified include a case study on the behaviour of the BHH during training, an empirical comparison to the performance of state-of-the-art, standalone, low-level heuristics, and a number of experiments related to the empirical testing of the effects of hyper-parameters on the outcomes of the BHH.

This chapter provides the outcomes of the empirical processes and provides results on all the experiments that have been conducted. Detailed discussions follow on the outcomes of each experiment. Discussions are accompanied by figures and plots to help provide visual aid for discussions. The output of the statistical analysis that yielded the results is provided in Appendix D. The remainder of the chapter is structured as follows:

- **Section 8.1** provides a brief overview of the outcomes of the empirical process and highlights key aspects to observe in the results.
- **Section 8.2** provides detailed discussions on the outcomes of the case study on the behaviour of the BHH. Illustrations are provided to show the change in parameter values to illustrate the outcomes of the learning process of the BHH, while training the underlying *feedforward neural network* (FFNN).

- **Section 8.3** provides the results of the performance of the BHH compared to individual low-level heuristics on all datasets. Three variants of the BHH are included in these results. These include the baseline configuration, as well as a configuration that only includes gradient-based heuristics in the heuristic pool, and a configuration that only includes MHs in the heuristic pool.
- **Section 8.4** provides the results for the experimental group that analyses the effects of the *heuristic pool* hyper-parameter on the outcomes of the BHH.
- **Section 8.5** provides the results for the experimental group that analyses the effects of the *population size* hyper-parameter on the outcomes of the BHH.
- **Section 8.6** provides the results for the experimental group that analyses the effects of the *credit assignment strategy* hyper-parameter on the outcomes of the BHH.
- **Section 8.7** provides the results for the experimental group that analyses the effects of the *reselection interval* hyper-parameter on the outcomes of the BHH.
- **Section 8.8** provides the results for the experimental group that analyses the effects of the *replay window size* hyper-parameter on the outcomes of the BHH.
- **Section 8.9** provides the results for the experimental group that analyses the effects of the *reanalysis interval* hyper-parameter on the outcomes of the BHH.
- **Section 8.10** provides the results for the experimental group that analyses the effects of the *burn in window size* hyper-parameter on the outcomes of the BHH.
- **Section 8.11** provides the results for the experimental group that analyses the effects of the *normalisation* hyper-parameter on the outcomes of the BHH.
- **Section 8.12** provides the results for the experimental group that analyses the effects of the *discounted rewards* hyper-parameter on the outcomes of the BHH.
- **Section 8.13** provides a brief summary of the chapter.

8.1 Overview

This section provides a brief discussion on the general outcome of the empirical process as a whole. Overall, the BHH is shown to successfully train the underlying

FFNNs for all datasets. In general, the BHH performs well and the empirical process provides key insights into the workings of the BHH. Where possible, a number of improvements to the BHH are identified and recommended as it relates to the outcomes of the results.

Given the nature of the BHH, it is recommended that the following aspects be kept in mind when observing the results. The BHH applies a form of online learning. As such, the BHH applies the learning mechanism during training in a single run of the training process. The training process is not repeated iteratively. All experiments conducted are executed for 30 epochs. The number of training steps executed is dependent on the batch size applied for each dataset.

All of the underlying FFNNs trained in the empirical process are relatively small. As such, most of the training progress is observed to occur within the first five epochs. As a result, the BHH should apply most learning at the early stages of the training process. After five epochs, the training of most of the underlying FFNNs converges and little performance gain is observed after that point. Since this empirical process does not apply early stopping of the training process, the BHH will continue to explore the heuristic space beyond the five epoch mark.

The BHH does not implement a move-acceptance strategy, where the application of a heuristic to an entity is only accepted if it leads to a better solution. In some cases the BHH then selects heuristics that yield sub-optimal results, but is shown to mostly return to optimal solutions over a number of steps.

Given the stochastic nature of the heuristic selection mechanism, sufficient samples of the performance of each heuristics-entity combination in the performance log is required for the BHH to learn. This requirement is further strengthened by the Bayesian nature of the probabilistic model implemented by the BHH. The probabilistic model implements *probability distributions of heuristic selection probabilities* and as such, insufficient samples in the performance log could render a form of random search.

The *reanalysis interval* defines intervals at which the BHH reanalyses the performance log, in effect, resetting the concentration parameters to their default values and reapplying the underlying Bayesian analysis process on the performance log. Furthermore, the *replay window size* defines the amount of performance evidence in the performance log. By default, the BHH baseline configuration has a reanalysis interval of 10, and a replay window size of 10, which is a small window to learn from. Despite the small reanalysis interval and the small replay window size, it should be observed that the BHH exploits small performance biases regardless and does find small performance gains throughout.

8.2 Behavioural Case Study

This section provides the empirical results of the case study on the behaviour of the BHH as it is used to train a FFNN on the iris dataset. As a reminder, the iris dataset contains 150 records, split into a training dataset containing 120 records, and a test dataset that contains 30 records. The test dataset is treated like a validation dataset, and is evaluated at every training step to illustrate the outcomes of the BHH throughout the training process. Mini-batch training is executed with a batch size of 30. The results that are presented are averaged over 30 independent runs.

Furthermore, the case study focuses on three variants of the BHH. These include the baseline configuration, as was provided in Chapter 6 and is denoted as `bhh_baseline` (illustrated throughout in red). Furthermore, the case study also includes a configuration of the BHH that has a “long memory” by setting the *replay window size* hyper-parameter to 250, and is denoted as `bhh_replay_250` (illustrated throughout in green). Finally, the case study includes a configuration that makes use of the *symmetric* credit assignment strategy hyper-parameter, and is denoted as `bhh_credit_symmetric` (illustrated throughout in blue).

These configurations were specifically chosen for the following reasons: the large replay window size configuration is chosen to illustrate a case where the BHH does not “forget” past performances of the low-level heuristics in the heuristic pool, and the configuration that utilises the symmetric credit assignment strategy is chosen to illustrate a case where there BHH does not reward good performance, and does not yield a bias towards heuristics that perform well at all. This section is structured as follows:

- **Section 8.2.1** provides illustrations and discussions around the outcomes of the performance metrics as obtained by the BHH.
- **Section 8.2.2** provides illustrations and discussions around the changes in concentration parameter values as a result of the learning mechanism implemented by the BHH.
- **Section 8.2.3** provides illustrations and discussions around the changes in the *probability distribution of heuristic selection probabilities* as a result of the learning mechanism implemented by the BHH.
- **Section 8.2.4** provides illustrations and discussions around the changes in the *prior heuristic selection probabilities* as a result of the learning mechanism implemented by the BHH.

- **Section 8.2.5** provides illustrations and discussions around the changes in the *posterior heuristic selection probabilities* as a result of the learning mechanism implemented by the BHH.

8.2.1 Performance Metrics

This section provides illustrations and discussions around the outcomes of the train and test performance metrics as obtained by the BHH. Figures 8.1a to 8.1d provide illustrations of the train and test loss and accuracy plots of the BHH over 30 epochs, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

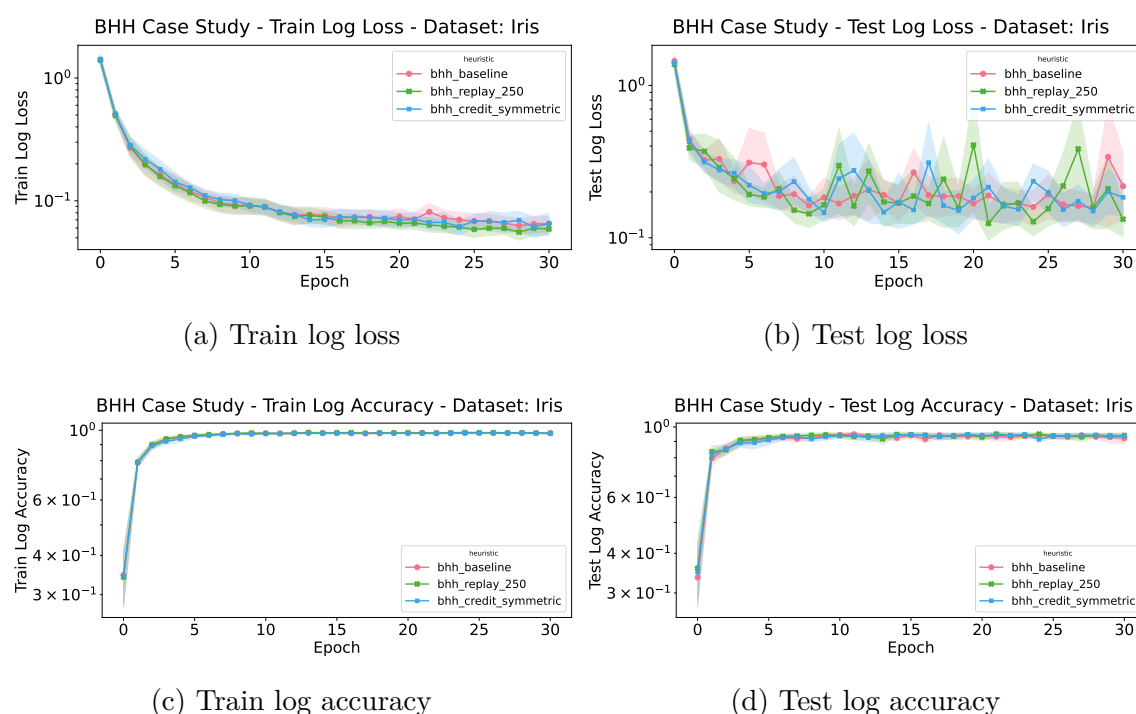


Figure 8.1: The average train and test loss and accuracy plots over 30 epochs, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

The first logical observation that can be made is that the BHH was indeed able to successfully train the underlying FFNN, observed by the convergence of the training process, yielding good results. Figures 8.1c and 8.1d show that the trained FFNN achieved an accuracy of almost 100%.

Although the different configurations of the BHH are all able to successfully train the underlying FFNN, in this particular case, there is no clear distinction between the performance of any of the configurations under consideration.

The volatility and minor divergence of the test loss compared to the training loss, observed in Figure 8.1b, is due to overfitting and partly due to the noisiness that results from mini-batch training. Early stopping can be applied to the training process to halt training before the model overfits on the training data. The training dataset is also very small, with just 120 samples. Furthermore, a very small batch size of 16 is used. As such, noisiness and overfitting can be expected.

From Figure 8.1a, at the 22nd epoch, a small divergence of the train loss can be observed. This could simply be a result of momentum that is maintained when switching between low-level heuristics in an attempt to find better solutions. For example, momentum can be built up by one of the gradient-based heuristics, after which the BHH switches to another heuristic in an attempt to find better solutions. The BHH does not incorporate a *move-acceptance strategy*, whereby a heuristic's outcome is rejected if it does not improve on previous solutions, yielding a possibly worse loss measurement as it relates to the test set. A move-acceptance strategy can be utilised on a validation set as a mechanism to accept or reject heuristic progressions from one step to the other.

Finally, it should be noted that the BHH implements learning at every mini-batch step, while Figure 8.1 only provides the outcomes of performance metrics at the end of each epoch. Further investigation is required at a mini-batch step level and is provided in the following sections.

8.2.2 Concentration Parameters

To illustrate the learning process undergone by the BHH, further investigation is required. Consider the concentration parameter α that parameterises the Dirichlet probability distribution, denoted $P(\theta|\alpha)$. The probability distribution, $P(\theta|\alpha)$, is used to sample prior *heuristic selection probabilities*. Figures 8.2a to 8.2d provide illustrations that show that change in values of the concentration parameter α , are at indices 0, 6, 7, and 8 respectively. Indices 0, 6, 7, and 8 represent the concentration parameters for the SGD, Adam, PSO and GA low-level heuristics respectively, and only represent a subset of the elements in α and thus, the heuristic pool. Similar illustrations for the other elements in α are left out for brevity as they contain similar illustrations.

The first logical observation to be made from Figures 8.2a to 8.2d is the step-wise, increasing nature of α for the bhh_replay_250 configuration, illustrated in green. Since the replay window size is sufficiently large to contain the performance log for all the steps executed in the training process, the BHH does not forget past performances of low-level heuristics at all. The value for α is never reset to its

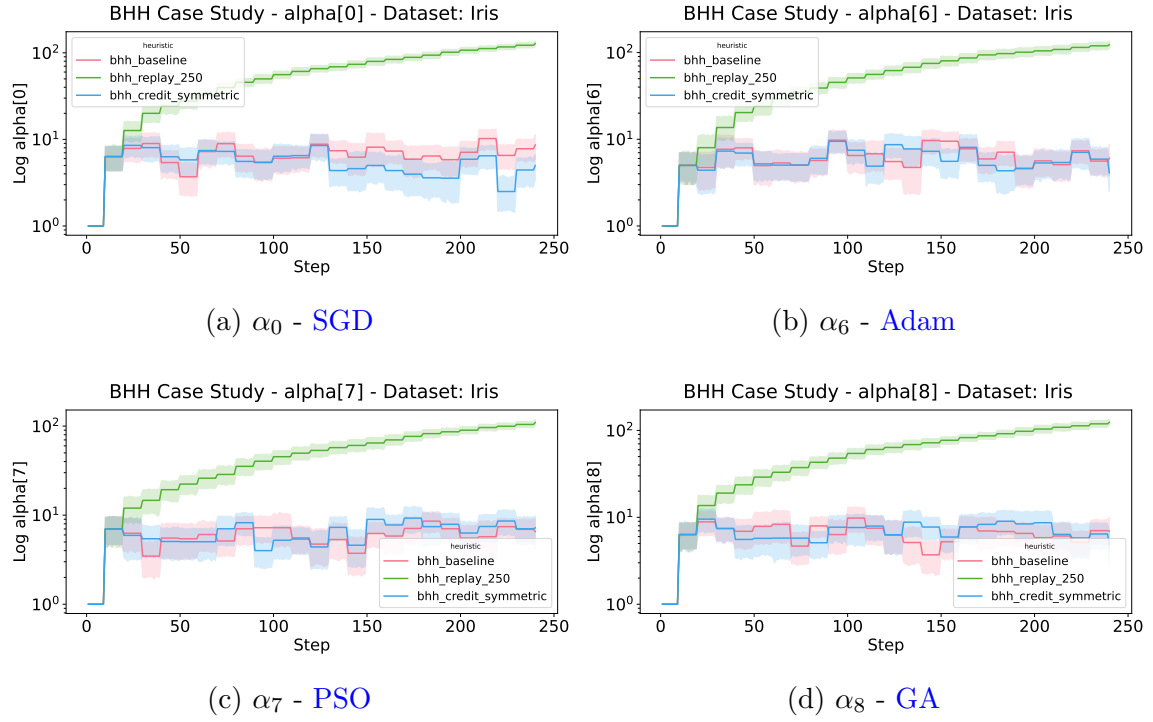


Figure 8.2: The average value of the concentration parameter α , are at indices 0, 6, 7, and 8 over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

initial value of 1.0, and thus, α continues to increase throughout the training process. However, it should be noted that the rate and degree of change is different for different indices of α . The aforementioned observation is the first indicator of the learning process yielded by the BHH. Heuristics that perform well will see their corresponding element in α increase more rapidly than other heuristics that do not perform well.

The next observations that can be made are for the *bhh_baseline* configuration, illustrated in red, and the *bhh_credit_symmetric* configuration in blue. Both these configurations see α being reset to its initial value of 1.0 as regular intervals. This interval is defined by the *reanalysis interval* hyper-parameter. The reanalysis interval dictates the frequency at which Bayesian analysis is conducted on the performance log, maintained by the BHH. Bayesian analysis is used to update α only at the reanalysis interval. As a result, small plateaus appear where α does not change.

Notice from Figure 8.2 that $\alpha \geq 1.0$ for multiple elements. This is due to the accumulation of pseudo counts for α by the Bayesian analysis process. Furthermore, this is also because the same heuristic can be allocated to multiple entities, each contributing a minimum pseudo-count gain of 1.0 to the corresponding element in α through the Bayesian analysis process. Consider a case where all five entities in the

entity pool are allocated the same heuristic and that a the replay window size of 10 is used. Then for all ten samples defined in the performance log, $\alpha_i = 50$, where i is the index of the i -th heuristic in the heuristic pool.

Furthermore, it is important to mention that a change in α between reanalysis windows does not yet necessarily indicate that the BHH is learning. The concentration parameter α tracks the *occurrence* of the random event of observing heuristics (denoted \mathbf{H}). Heuristics can be observed as a result of good performance, by which the BHH then learns to frequently reselect these heuristics again, or just by chance through the stochastic nature of probabilistic sampling as implemented by the BHH. Further investigation is required to illustrate the learning mechanism of the BHH.

8.2.3 Probability Distribution of Heuristic Selection Probabilities

This section provides a detailed investigation into the *probability distribution of heuristic selection probabilities* as it changes throughout the training process. As a reminder, the BHH implements a Bayesian view of probabilistic modelling and thus, heuristic selection probabilities are defined by an underlying probabilistic distribution themselves. The probability distribution of heuristic selection probabilities is denoted by $P(\boldsymbol{\theta}|\boldsymbol{\alpha})$, where $\boldsymbol{\theta} \sim \text{Dir}(\boldsymbol{\alpha}, \mathbf{K})$ are the sampled heuristic selection probabilities.

Figures 8.3a to 8.3d provide illustrations of the distribution of heuristic selection probabilities, $\boldsymbol{\theta}$, sampled from the probability distribution $P(\boldsymbol{\theta}|\boldsymbol{\alpha})$ throughout the training process, and averaged over all 30 runs. These illustrations are presented in log scale. Indices 0, 6, 7, and 8 represent the distribution of heuristic selection probabilities for the SGD, Adam, PSO and GA low-level heuristics respectively, and only represent a subset of the distribution of heuristic selection probabilities in $\boldsymbol{\theta}$. Similar illustrations for the other elements in $\boldsymbol{\theta}$ are left out for brevity as they contain similar illustrations.

From the illustrations presented in Figures 8.3a to 8.3d, a clearer picture of the learning process of the BHH is formed. The first important observation to make is for the `bhh_replay_250` configuration, presented in green. As a reminder, ten low-level heuristics are included in the heuristic pool, yielding an expected mean heuristics selection probability of 0.1, for each heuristic, by the frequentist view of probabilistic modelling.

Towards the end of the training process, the heuristic selection probability converges back to the symmetrical, uniform probability distribution, yielding a heuristic selection probability of 0.1, for all heuristics. This can be explained as

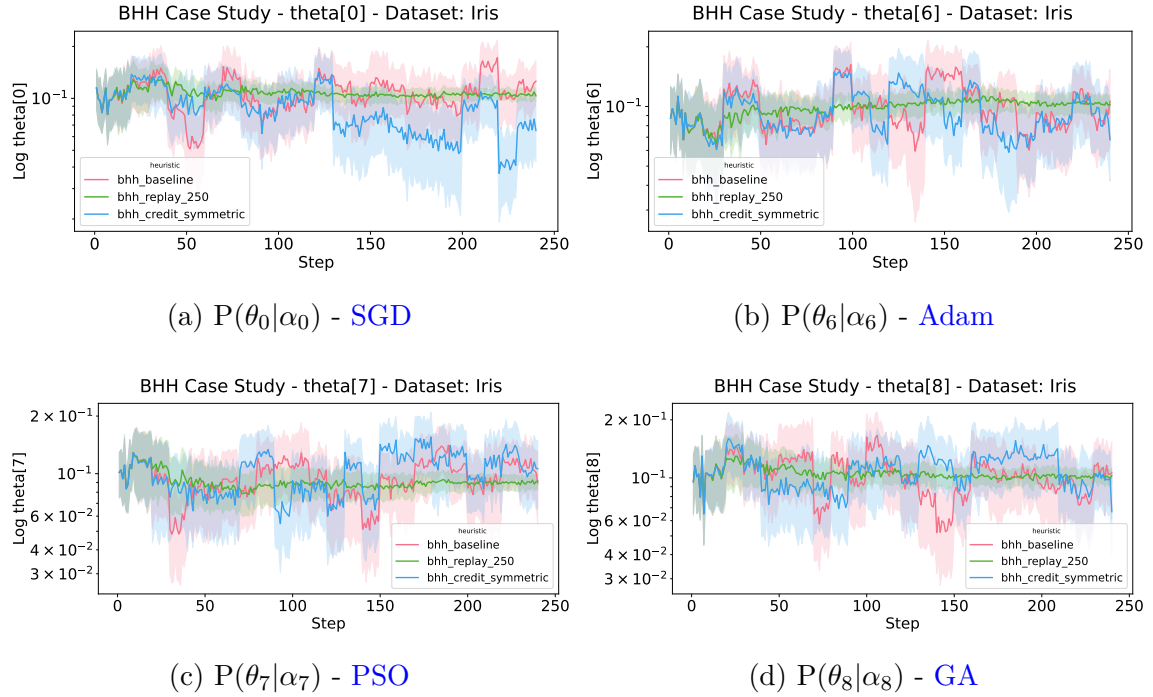


Figure 8.3: The average sampled heuristic selection probabilities, denoted θ , are at indices 0, 6, 7, and 8. The heuristic selection probabilities are sampled from the probability distribution, denoted $P(\theta|\alpha)$, over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

follows: most of the training progress is made in the early stages of the training process, and training converges towards the end of the training process. Since training converges, all heuristics, no matter their past performances, fail to yield better solutions towards the end of the training process. As a result of training convergence, heuristics then fail to meet the performance criteria and credit allocations by means of the credit assignment strategy.

Both the `bhh_baseline` (red) and the `bhh_replay_250` (green) configurations make use of the *ibest* credit assignment strategy. The *ibest* credit assignment strategy allocates credit to the heuristic that yields the best performance for the current iteration/step and thus, towards the end of the training process, any random heuristic can yield the best iteration performance. However, since the `bhh_baseline` is configured with a small replay window size of 10, and a reanalysis interval of 10, the concentration parameter α is reset to its default value of 1.0 more often than the `bhh_replay_250` configuration, resulting in a probability distribution that is broader, and thus, explaining the larger variance of θ throughout.

Another observation to make occurs in the first 30 steps of the training process. In Figure 8.3, notice how all three configurations mostly yield the same heuristic

selection probabilities in these first 30 steps. This can be explained as follows: all three configurations use a different random seed per run, but use the same random seed across configurations for the same run number. This is done so that any difference in the behaviour of the different BHH configurations are then not a result of random sampling, but solely because of differences in their approaches. This is especially applicable to the initialisation process, where entities are randomly placed in the search space, as well as the early stages of training, where most of the training progress is made.

It should be noted that, despite using the same random seed across configurations for the same run number, the behavioural changes between the configurations start to show after about 30 steps. As a reminder, the `bhh_credit_symmetric` (blue) configuration, does not bias towards best performing heuristics. Where the `bhh_baseline` and `bhh_replay_250` configurations then diverge from the behaviour of the `bhh_credit_symmetric` configuration is proof of the effect of performance bias.

Furthermore, it can be observed for small windows, at various steps for multiple runs, that the variance of θ , for the `bhh_baseline` configuration and the `bhh_credit_symmetric` configuration, do not yield means that are equal to the expected heuristic selection probabilities of 0.1. This is proof that the BHH does not just implement a form of random search, despite having small reanalysis interval and replay window size configurations. This is also true for the `bhh_credit_symmetric` configuration, as the `bhh_credit_symmetric` configuration biases towards heuristics that happen to be sampled, despite not biasing towards good performance.

Finally, the `bhh_baseline` configuration and the `bhh_credit_symmetric` configuration both yield similar volatile behaviour, much more so than with the `bhh_replay_250` configuration. This can be attributed to a very small reanalysis window combined with a small replay window size of 10, that contains very few samples to learn from. A small reanalysis interval and a small replay window size allows for more exploration of the heuristic space, but can also yield greater variance of the heuristic selection probabilities. Once again, any differences then in the behaviour of the `bhh_baseline` compared to the `bhh_credit_symmetric` configurations is proof of small performance exploitations and biases by the `bhh_baseline` configuration.

8.2.4 Prior Heuristic Selection Probabilities

This section provides a brief investigation into the *prior heuristic selection probabilities* that result from the probabilistic model implemented by the BHH. The prior heuristic selection probability distribution is denoted $P(\mathbf{H}|\theta)$. As such, $P(\mathbf{H}|\theta) = \theta$ and

heuristics are initially sampled such that $\mathbf{H} \sim \text{Cat}(\boldsymbol{\theta})$.

Figures 8.4a to 8.4d provide illustrations of the prior heuristic selection probabilities, denoted by $P(\mathbf{H}|\boldsymbol{\theta})$, are at indices 0, 6, 7, and 8, throughout the training process, and averaged over all 30 runs. Similar to before, these illustrations are also presented in log scale.

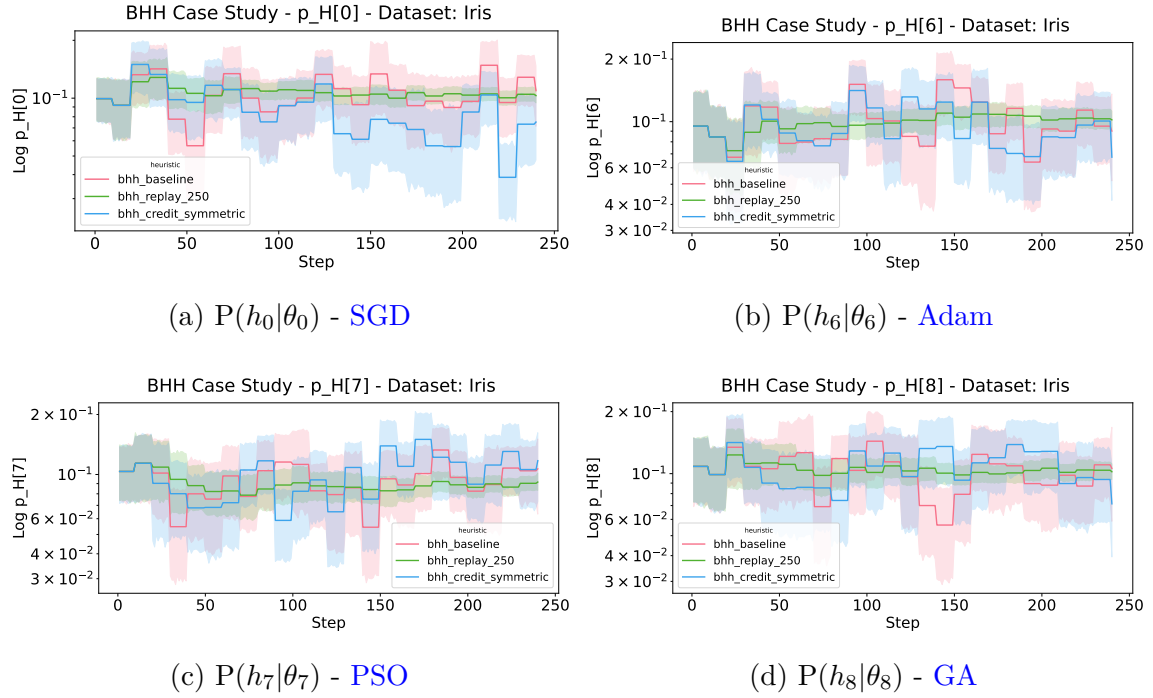


Figure 8.4: The average prior heuristic selection probabilities, $P(\mathbf{H}|\boldsymbol{\theta})$, are at indices 0, 6, 7, and 8. The prior heuristic selection probabilities are sampled from the probability distribution of heuristic selection probabilities, denoted by $P(\boldsymbol{\theta}|\boldsymbol{\alpha})$, over 240 steps, obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

The main observation to make from these figures is that they are much less volatile and noisy than the figures presented for the distribution of heuristic selection probabilities, $\boldsymbol{\theta}$, presented in the previous section in Figure 8.3. This is because of the *reselection* interval hyper-parameter. The reselection interval hyper-parameter is implemented as a way to control the frequency by which new heuristics are selected and allocated to each entity. Since the default *reselection* interval is set to 10, the heuristic selection probabilities are only resampled at intervals of 10. These illustrations then simply yield rough approximations of the illustrations provided in Figures 8.3a to 8.3d for the distribution of heuristic selection probabilities. As such, the same observations and conclusions that are made in Section 8.2.3 also apply in this section.

Finally, it should be mentioned that at each step, the goal of the BHH is to update these prior “beliefs” based on newly observed evidence of heuristic performances. Since these prior heuristic selection probabilities change over time, it can be concluded that the change in prior heuristic selection probabilities is a result of the learning mechanism of the BHH. Furthermore, the prior heuristic selection probability distribution provides an opportunity to utilise prior knowledge by some expert before training starts, by injecting heuristic selection biases, by means of the initial values for the concentration parameter α .

8.2.5 Posterior Heuristic Selection Probabilities

This section provides a detailed discussion on the outcomes of the *posterior heuristic selection probabilities*. These posterior heuristic selection probabilities form the basis of the probabilistic model implemented by the BHH. The posterior heuristic selection probability distribution is defined as $P(\mathbf{H}|\mathbf{E}, \mathbf{C}; \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi})$, where \mathbf{E} represents the vector of entities in the entity pool, and \mathbf{C} represents the vector of credit allocation outcomes as implemented by the credit assignment strategy. Furthermore $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ represent the probability distributions of heuristic selection probabilities and the entity selection probabilities respectively. Finally, $\boldsymbol{\psi}$ represents the probability distribution of successful credit allocation probabilities.

Figures 8.5a to 8.5d provide illustrations of the calculated posterior heuristic selection probabilities at indices 0, 6, 7, and 8, throughout the training process, averaged over 30 runs. Similar to before, these illustrations are also presented in log scale. As before, illustrations for the other indices are left out for brevity as they yield similar illustrations.

The main observation to make from Figures 8.5a to 8.5d is that the implemented posterior heuristic selection distribution, defined by $P(\mathbf{H}|\mathbf{E}, \mathbf{C}; \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi})$, does not yield normalised probabilities, but rather yield unnormalised *logits*, which are used to parameterise a Categorical probability distribution from which new heuristic selections are sampled. The reasons for the aforementioned is because the probabilistic model is evaluated proportionally as was discussed in Chapter 6. As a reminder, the log-sum-exp trick is used in order to maintain numerical stability, yielding *logits* instead of probabilities.

Another observation to make is for the `bhh_replay_250` configuration (*green*). Figures 8.5a to 8.5d show that the posterior heuristic selection probabilities converge to the expected heuristic selection probabilities later in the training stages. The aforementioned suggests that the BHH is not able to find further performance biases and cannot exploit better solutions. At this point, the BHH starts to explore more

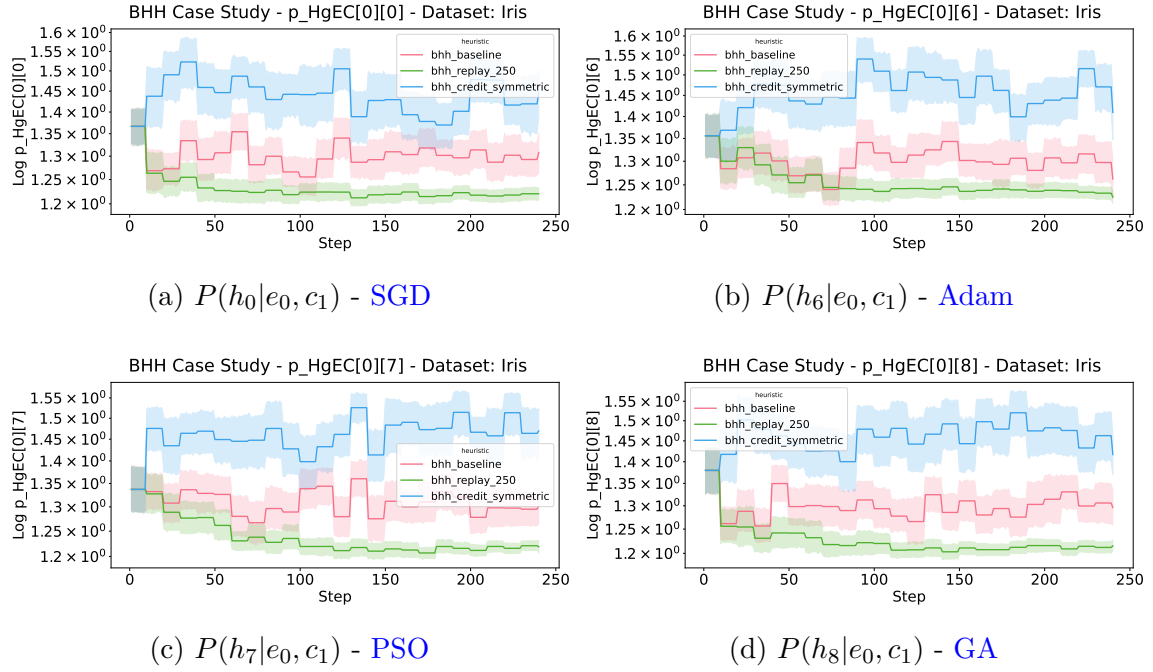


Figure 8.5: The average calculated proportional posterior heuristic selection probabilities for heuristics (H) at indices 0, 6, 7, and 8, given the application to entity e_0 and requiring a successful credit allocation, c_1 , from the *ibest* credit assignment strategy. The proportional posterior heuristic selection probabilities are calculated from the probabilistic model, denoted $P(H|E, C; \theta, \phi, \psi)$, over 240 steps, and obtained from 30 runs of the case study on the behaviour of the BHH on the iris dataset, illustrated in log scale.

as the concentration parameters are reanalysed more uniformly, resolving more and more to a random search in attempt to find better solutions.

Furthermore, the posterior heuristic selection probability distribution is conditional on the occurrence of a specific entity that the potential heuristic will be applied to, as well as a specific performance criterion enforced by a specific credit assignment strategy. This means that heuristic selection is specific to each entity. A particular heuristic might be good for one entity, but not for another. This is a strong characteristic of the BHH, as it learns to apply the correct heuristic to the correct entity at the correct time in the training process.

Finally, the posterior heuristic selection probabilities are much less volatile than their prior heuristic selection probability equivalents. This is a result of the information that is added in the performance log by tracking the entity and credit allocation as well.

It can be concluded from Sections 8.2.1 to 8.2.5 that the BHH is able to successfully train the underlying FFNN for the case study on the iris dataset. Furthermore it can be concluded that the learning mechanism implemented by the BHH is able to

exploit minor performance biases, and thus the BHH is able to correctly allocate the correct heuristic to the correct entity at the correct time in the training process.

8.3 BHH vs. Low-Level Heuristics

This section provides the empirical results for the experimental group that compares the performance of the BHH to the performance of the individual, standalone, low-level heuristics. Detailed discussions and illustrations follow. As a reminder, the set of low-level heuristics includes a number of gradient-based heuristics and a number of MHs. Three variants of the BHH is included in the experiment, including the BHH baseline configuration with a heuristic pool that contains all the low-level heuristics (denoted `bhh_all`), the BHH configuration with a heuristic pool that contains only gradient-based heuristics (denoted `bhh_gd`), and finally, the BHH configuration with a heuristics pool that contains only meta-heuristics (denoted `bhh_mh`).

Table 8.1 presents the empirical results for this experimental group, showing the average test loss and statistics for all the low-level heuristics, compared to the three variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.1 shows that the `bhh_gd` configuration produced the best results of the BHH variants and managed to perform well, producing generally good results across all datasets. The `bhh_gd` configuration managed to produce results that are comparable to the top three heuristics for each dataset, while the `bhh_all` and `bhh_mh` produced average results compared to all the heuristics.

Table 8.2 provides the empirical results from Table 8.1 in ranked format. The performance rank is calculated as the average rank produced by each heuristic, across all datasets, for all independent runs and all epochs. The average rank across all epochs produces a view on the performance of the heuristics as it relates to the entire training process. Finally, a normalised average rank is provided for the overall performance of all heuristics at the bottom of the table. The normalised average rank is calculated as a discrete normalisation of the average rank achieved across all datasets, for all independent runs and epochs.

From the normalised average ranks provided in Table 8.2, it can be seen that the `bhh_gd` configuration ranked fourth, while the `bhh_all` and `bhh_mh` configurations ranked sixth and eighth amongst all thirteen heuristic implementations respectively. These results show that the BHH generally performs well, but is not able to outperform the best heuristic for each dataset.

Figure 8.6 provides an illustration showing a descriptive plot of the average ranks

Table 8.1: Empirical results showing test loss and statistics for different low-level heuristics compared to three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs, measured at the last epoch.

BHH vs. Low-Level Heuristics - Average Test Loss													
dataset	adagrad	adam	rmsprop	bhh_gd	nag	bhh_all	adadelta	bhh_mh	ga	pso	sgd	momentum	de
abalone	1.9678 (±0.032)	1.9651 (±0.035)	1.9455 (±0.035)	2.0125 (±0.068)	2.0156 (±0.035)	2.0587 (±0.088)	2.0199 (±0.034)	2.2942 (±0.055)	2.8155 (±0.055)	3.0384 (±0.294)	2.428 (±0.029)	2.4942 (±0.029)	3.2959 (±0.016)
air quality	0.2569 (±0.007)	0.2606 (±0.009)	0.2513 (±0.008)	0.2642 (±0.011)	0.2568 (±0.006)	0.2729 (±0.016)	0.2568 (±0.006)	0.2637 (±0.007)	0.2759 (±0.008)	0.2923 (±0.023)	0.2852 (±0.015)	0.2929 (±0.016)	0.3097 (±0.017)
bank	0.2096 (±0.005)	0.2065 (±0.004)	0.2058 (±0.006)	0.2164 (±0.005)	0.2164 (±0.006)	0.2456 (±0.065)	0.2186 (±0.004)	0.2562 (±0.011)	0.2881 (±0.013)	0.3454 (±0.034)	0.2382 (±0.006)	0.2386 (±0.005)	0.3437 (±0.028)
bike	0.0458 (±0.002)	0.0682 (±0.068)	0.1123 (±0.103)	0.0545 (±0.005)	0.0995 (±0.002)	0.0651 (±0.02)	0.0679 (±0.002)	0.1179 (±0.006)	0.1531 (±0.005)	0.1503 (±0.025)	0.1599 (±0.003)	0.1614 (±0.003)	0.2399 (±0.04)
car	0.2027 (±0.018)	0.0972 (±0.024)	0.1039 (±0.031)	0.169 (±0.025)	0.2454 (±0.029)	0.1572 (±0.034)	0.2859 (±0.027)	0.4891 (±0.077)	0.7374 (±0.063)	0.5632 (±0.158)	0.7035 (±0.052)	0.7322 (±0.043)	0.8332 (±0.074)
diabetic	0.8895 (±0.004)	0.9193 (±0.01)	0.8957 (±0.004)	0.8976 (±0.011)	0.8809 (±0.004)	1.2983 (±0.66)	0.8839 (±0.005)	0.9137 (±0.008)	0.9613 (±0.007)	0.9661 (±0.016)	0.8974 (±0.004)	0.898 (±0.003)	1.0774 (±0.039)
fish toxicity	0.0991 (±0.008)	0.0972 (±0.007)	0.0963 (±0.008)	0.1056 (±0.008)	0.1023 (±0.008)	0.1046 (±0.008)	0.1016 (±0.009)	0.1043 (±0.009)	0.1093 (±0.009)	0.1099 (±0.012)	0.1383 (±0.012)	0.1441 (±0.012)	0.1193 (±0.013)
forest fires	0.0643 (±0.04)	0.0527 (±0.039)	0.0626 (±0.039)	0.0586 (±0.034)	0.0592 (±0.032)	0.081 (±0.069)	0.0488 (±0.032)	0.0621 (±0.038)	0.0526 (±0.026)	0.0692 (±0.047)	0.1806 (±0.008)	0.1885 (±0.01)	0.3598 (±0.09)
housing	0.0889 (±0.012)	0.0848 (±0.011)	0.0842 (±0.015)	0.0951 (±0.015)	0.0923 (±0.016)	0.0941 (±0.017)	0.106 (±0.019)	0.1169 (±0.019)	0.1279 (±0.02)	0.1452 (±0.03)	0.1735 (±0.023)	0.1713 (±0.018)	0.2057 (±0.036)
iris	0.2161 (±0.059)	0.1203 (±0.098)	0.0753 (±0.046)	0.3729 (±1.119)	0.085 (±0.042)	0.2411 (±0.295)	0.426 (±0.072)	0.1809 (±0.164)	0.2895 (±0.117)	0.8965 (±0.844)	0.4694 (±0.088)	0.5028 (±0.072)	0.8027 (±0.684)
mushroom	0.0026 (±0.001)	0.0013 (±0.005)	0.0001 (±0)	0.0009 (±0.001)	0.0094 (±0.002)	0.0052 (±0.012)	0.0042 (±0.001)	0.0774 (±0.02)	0.4892 (±0.025)	0.0611 (±0.06)	0.1797 (±0.009)	0.2418 (±0.013)	0.687 (±0.016)
parkinsons	0.0563 (±0.001)	0.0545 (±0.002)	0.054 (±0.002)	0.0587 (±0.003)	0.0655 (±0.002)	0.0587 (±0.003)	0.0597 (±0.002)	0.0658 (±0.003)	0.0669 (±0.003)	0.0671 (±0.005)	0.0923 (±0.008)	0.0954 (±0.009)	0.0843 (±0.009)
student performance	0.1656 (±0.011)	0.4929 (±0.105)	0.5724 (±0.054)	0.2454 (±0.134)	0.1697 (±0.011)	0.2359 (±0.102)	0.1708 (±0.01)	0.1947 (±0.014)	0.1962 (±0.01)	0.4565 (±0.049)	0.193 (±0.011)	0.1925 (±0.011)	0.2014 (±0.012)
wine quality	1.0651 (±0.024)	1.0395 (±0.018)	1.0514 (±0.021)	1.1028 (±0.039)	1.0718 (±0.025)	1.0827 (±0.026)	1.0809 (±0.021)	1.1666 (±0.03)	1.2516 (±0.046)	1.3046 (±0.114)	1.1648 (±0.023)	1.1774 (±0.019)	1.4896 (±0.093)

Table 8.2: Empirical results showing normalised average rank and statistics for different low-level heuristics compared to three heuristic pool variants of the **BHH** baseline configuration, across multiple datasets, for all independent runs and epochs.

dataset	BHH vs. Low-Level Heuristics - Average Rank (Based on Test Loss)												
	adagrad	adam	rmsprop	bhb_gd	nag	bhb_all	adadelta	bhb_mh	ga	pso	sgd	momentum	de
abalone	2.2215 (± 1.591)	2.3689 (± 1.887)	4.6172 (± 2.65)	4.7032 (± 2.108)	4.2731 (± 1.542)	5.9376 (± 2.399)	5.3129 (± 1.478)	8.1882 (± 1.195)	11.1108 (± 1.102)	11.2559 (± 1.826)	8.628 (± 1.019)	9.8151 (± 1.16)	12.5376 (± 1.329)
air quality	3.6409 (± 2.259)	5.4312 (± 2.62)	3.4452 (± 2.57)	5.0817 (± 2.762)	3.8194 (± 2.229)	6.086 (± 3.061)	5.2441 (± 3.162)	6.357 (± 2.303)	7.8204 (± 2.265)	9.9151 (± 2.288)	10.6613 (± 1.606)	11.7559 (± 1.473)	11.1419 (± 2.236)
bank	2.5495 (± 1.598)	2.0796 (± 1.587)	3.4645 (± 2.209)	4.8828 (± 1.702)	4.2871 (± 1.732)	6.2419 (± 2.157)	5.672 (± 1.241)	9.7495 (± 1.048)	10.9817 (± 1.216)	11.8376 (± 1.464)	8.2774 (± 1.03)	8.4774 (± 1.068)	12.4989 (± 1.224)
bike	1.7204 (± 1.384)	3.6925 (± 4.004)	6.2624 (± 4.58)	3.8441 (± 1.398)	6.4516 (± 1.02)	4.2151 (± 1.361)	5.3602 (± 1.155)	7.4108 (± 1.008)	9.2269 (± 1.183)	9.3086 (± 1.761)	10.3355 (± 1.419)	10.7086 (± 1.423)	12.4634 (± 1.465)
car	4.7684 (± 0.938)	1.6226 (± 1.405)	2.3269 (± 1.409)	3.3473 (± 1.35)	6.0785 (± 0.799)	3.5624 (± 1.315)	7.7344 (± 1.746)	8.8505 (± 1.413)	10.7763 (± 1.471)	8.3613 (± 1.622)	10.2452 (± 1.447)	10.9226 (± 1.349)	12.4086 (± 1.492)
diabetic	2.7796 (± 1.639)	7.1484 (± 2.227)	6.7376 (± 2.577)	5.2269 (± 2.186)	1.8118 (± 1.413)	9.3968 (± 3.022)	2.6753 (± 1.629)	8.537 (± 1.17)	11.2011 (± 1.413)	10.7022 (± 1.067)	5.9215 (± 1.542)	6.3355 (± 1.612)	12.5065 (± 1.242)
fish toxicity	4.2645 (± 2.614)	3.6022 (± 2.445)	3.5946 (± 2.329)	5.4118 (± 2.665)	5.8914 (± 2.629)	5.829 (± 2.856)	7.914 (± 3.429)	6.3849 (± 2.944)	6.7043 (± 2.82)	7.5731 (± 2.982)	11.5785 (± 1.459)	12.2301 (± 1.382)	10.0215 (± 2.358)
forest fires	5.1559 (± 2.922)	4.2688 (± 2.984)	5.0355 (± 3.143)	4.6935 (± 2.759)	5.6882 (± 2.215)	5.4839 (± 3.107)	6.5161 (± 3.082)	5.4591 (± 2.668)	7.3667 (± 2.37)	6.4796 (± 3.354)	10.8129 (± 1.207)	11.7065 (± 1.325)	12.3333 (± 1.923)
housing	3.4484 (± 2.025)	3.3344 (± 1.819)	3.6946 (± 2.166)	4.4742 (± 2.312)	4.6839 (± 2.658)	4.3763 (± 2.438)	7.5903 (± 2.748)	7.5441 (± 1.736)	7.8839 (± 2.099)	9.9409 (± 2.317)	11.4075 (± 1.528)	11.2731 (± 1.506)	11.3484 (± 2.096)
iris	6.3946 (± 1.6)	3.5839 (± 2.511)	2.6968 (± 1.912)	4.7473 (± 2.275)	3.5548 (± 2.125)	5.2204 (± 3.041)	11.3527 (± 1.779)	6.6075 (± 2.555)	8.2473 (± 1.765)	8.2731 (± 4.384)	10.3796 (± 1.294)	11.0548 (± 1.409)	8.8871 (± 3.251)
mushroom	4.4656 (± 1.053)	2.1344 (± 1.883)	2.4656 (± 1.359)	3.4484 (± 1.602)	6.3323 (± 0.891)	3.6688 (± 2.469)	6.5538 (± 1.071)	9.0452 (± 1.093)	11.5731 (± 1.193)	7.872 (± 0.92)	9.7527 (± 1.083)	10.9785 (± 1.108)	12.7097 (± 1.478)
parkinsons	2.4677 (± 1.497)	2.2333 (± 1.742)	3.5656 (± 2.492)	4.572 (± 1.934)	7.5355 (± 1.44)	4.3839 (± 1.861)	6.472 (± 2.423)	8.3161 (± 1.644)	7.7968 (± 1.719)	8.4892 (± 1.901)	11.7516 (± 1.155)	12.5419 (± 1.351)	10.8742 (± 1.317)
student performance	2.5684 (± 1.912)	11.3978 (± 2.178)	12.4312 (± 1.34)	5.6024 (± 3.57)	3.1935 (± 2.12)	5.8634 (± 3.159)	3.4194 (± 2.006)	6.9333 (± 2.44)	7.1032 (± 1.989)	11.0624 (± 1.067)	6.6366 (± 2.023)	6.7011 (± 2.242)	8.0323 (± 1.935)
wine quality	3.2806 (± 1.931)	2.1118 (± 1.666)	3.6301 (± 1.731)	4.7882 (± 2.105)	4.1505 (± 1.916)	5.1925 (± 1.951)	6.0011 (± 2.404)	9.5935 (± 1.494)	10.3387 (± 1.62)	11.1602 (± 1.773)	8.6344 (± 1.118)	9.5269 (± 1.341)	12.5903 (± 1.352)
avg rank	3.5512 (± 2.25)	3.9314 (± 3.423)	4.5691 (± 3.517)	4.6346 (± 2.364)	4.8394 (± 2.384)	5.4327 (± 2.9)	6.2727 (± 3.004)	7.7855 (± 2.271)	9.1522 (± 2.48)	9.4451 (± 2.75)	9.6445 (± 2.214)	10.2877 (± 2.346)	11.4538 (± 2.354)
normalised avg rank	1	2	3	4	5	6	7	8	9	10	11	12	13

achieved over all independent runs, for each heuristic, per dataset. The heuristics are ordered according to the normalised ranks presented in Table 8.2.

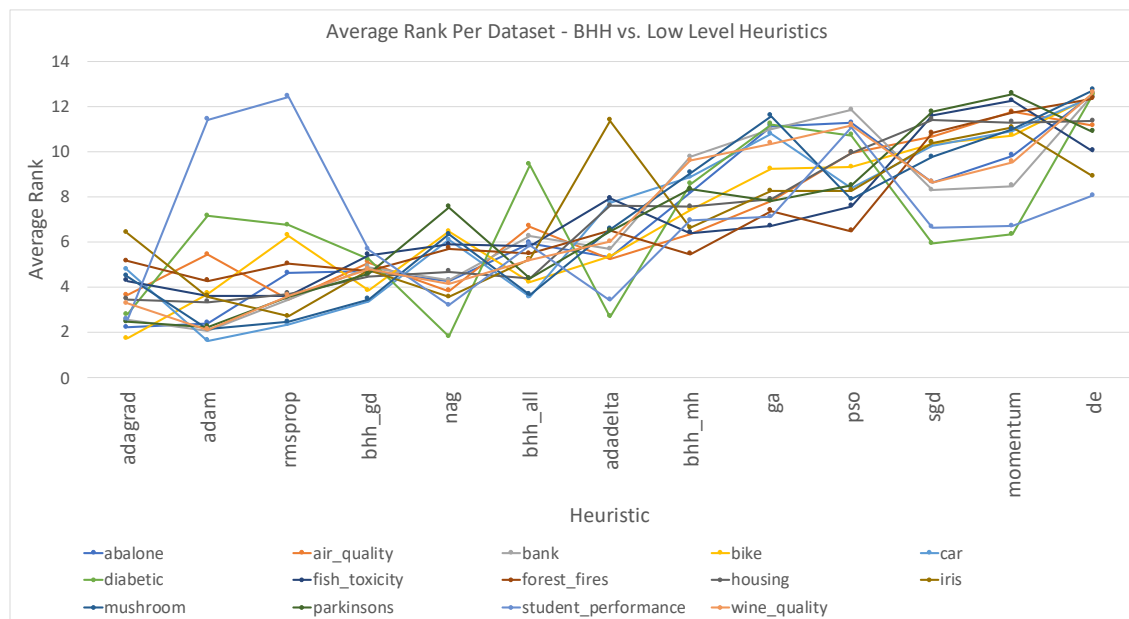


Figure 8.6: Descriptive plots for the average ranks of all low-level heuristics compared to three heuristic pool variants of the BHH baseline configuration, per dataset, across all independent runs and epochs.

From Figure 8.6 a few observations can be made. Firstly, both Adam and RMSProp achieved radically different average ranks for the student performance dataset, compared to the other datasets. This can also be seen in Table 8.2. Further investigation into the reasons for this is required as both these heuristics perform well on all other datasets, and the other heuristics perform well for the student performance dataset. A suggestion is that an invalid learning rate or learning rate schedule leads to a scenario where good solutions are overshoot, causing Adam and RMSProp to struggle to resolve back to good solutions. Furthermore, the bhh_gd heuristic achieved the lowest variance in average rank across all datasets, compared to the other heuristics. The aforementioned shows the generalisation capabilities of the BHH to multiple problems.

Figure 8.7 provides an illustration of the overall critical difference plots that illustrate the statistically significant differences in ranked performance for each heuristic as it relates to all datasets, across all independent runs and epochs.

Although the outcomes of the bhh_all and bhh_mh configurations seem to produce average performance results, it should be noted that the performance difference between all heuristics is very small. Furthermore, the best configuration of the BHH, namely the bhh_gd configuration, is statistically only outperformed overall

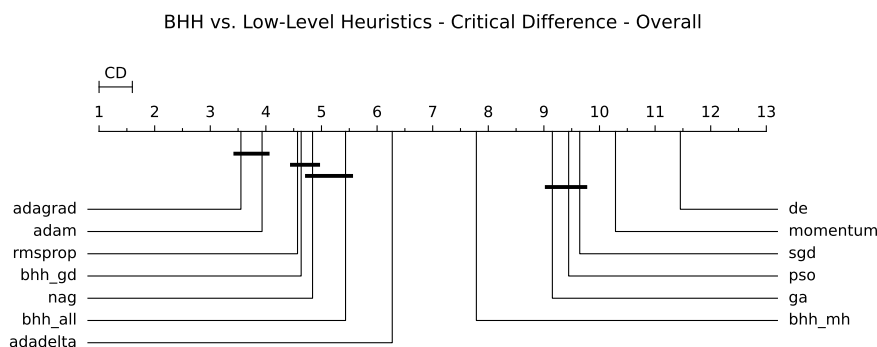


Figure 8.7: Critical difference plots for the average ranks of all low-level heuristics compared to three heuristic pool variants of the baseline [BHH](#), across all datasets, runs and epochs.

by [Adagrad](#) and [Adam](#), yielding statistically comparable results to [RMSProp](#) and [NAG](#). It should be noted that the standalone low-level heuristics already produce good results in general across all datasets. In this particular case, producing better performance outcomes can be hard to achieve. However, as mentioned previously, the [BHH](#) provides a generalisation capability across all datasets that is advantageous to the [BHH](#).

Another observation that can be made is that the gradient-based heuristics generally performed much better than the meta-heuristics on all datasets. State of the art methods for training [FFNNs](#), such as [Adam](#), utilise gradient-based approaches that have been proven to work well on many occasions [94]. Exploration of the heuristic space leads the [BHH](#) to consider other heuristics during the training process, which could possibly result in worse performances at times. As previously mentioned, a suggestion to improve on these results is to include a move-acceptance strategy, where heuristic progressions are discarded if they fail to produce better results.

Figures 8.8 to 8.10 provide example illustrations of the train and test loss and accuracy plots for each heuristic as it relates to a selection of classification datasets. The train and test loss and accuracy plots provided are illustrated in log scale. The other classification datasets are left out for brevity as they produce similar illustrations.

Figures 8.11 to 8.12 provide example illustrations of the train and test loss plots for each heuristic as it relates to a selection of regression datasets. The train and test loss plots provided are illustrated in log scale. Similar to before, the other regression datasets are left out for brevity as they produce similar illustrations.

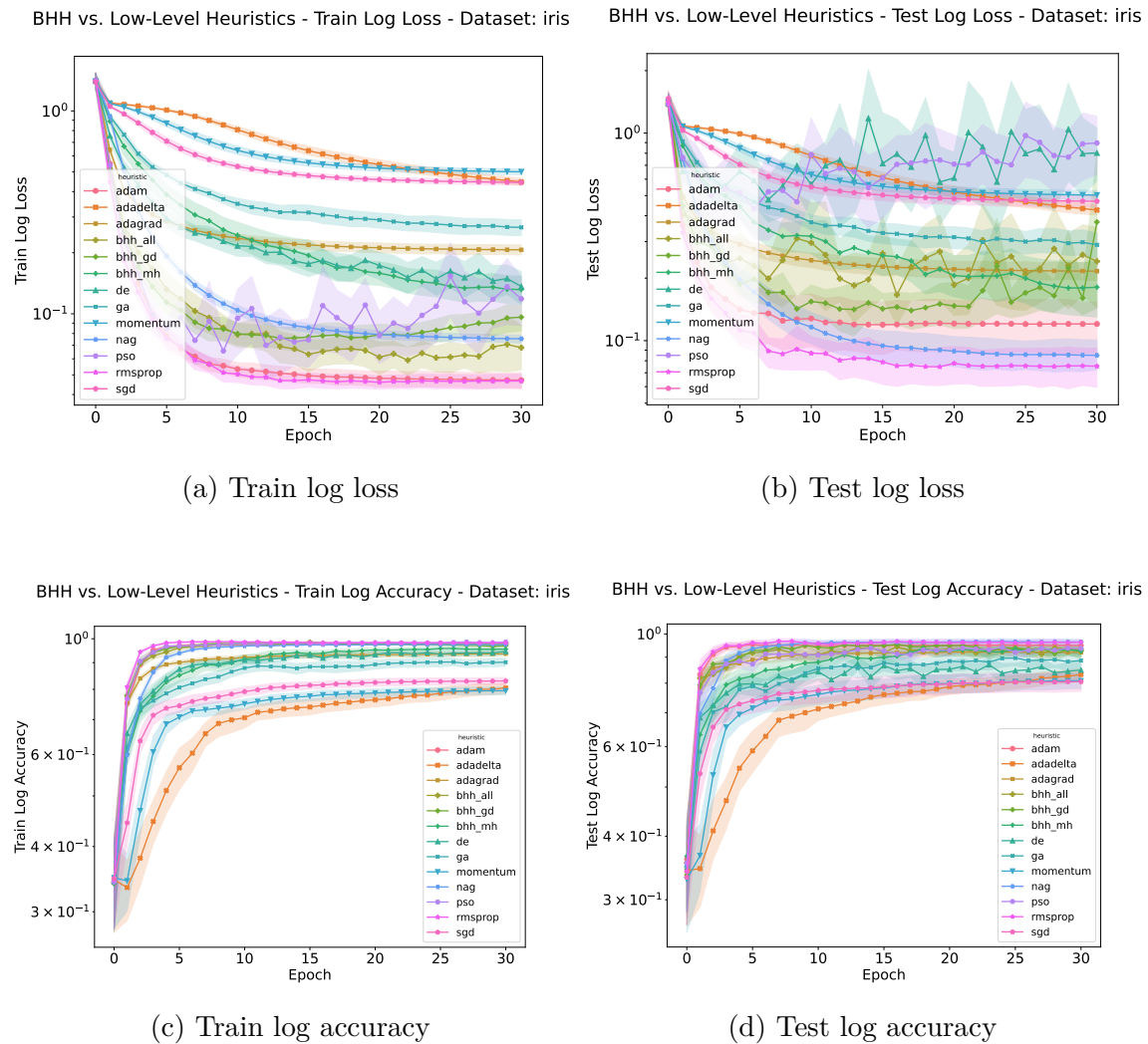


Figure 8.8: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the iris dataset over 30 epochs, illustrated in log scale.

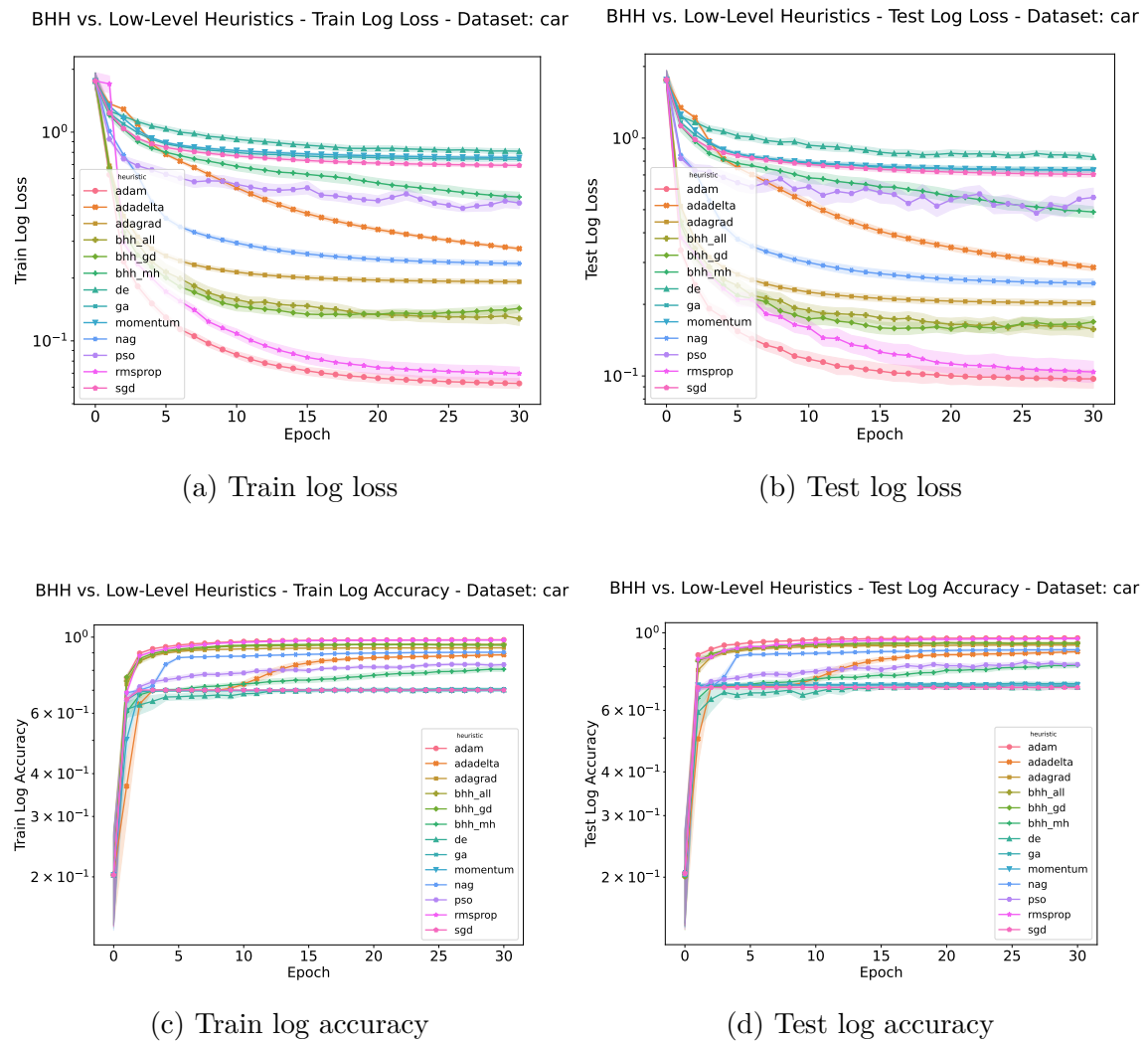


Figure 8.9: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the car dataset over 30 epochs, illustrated in log scale.

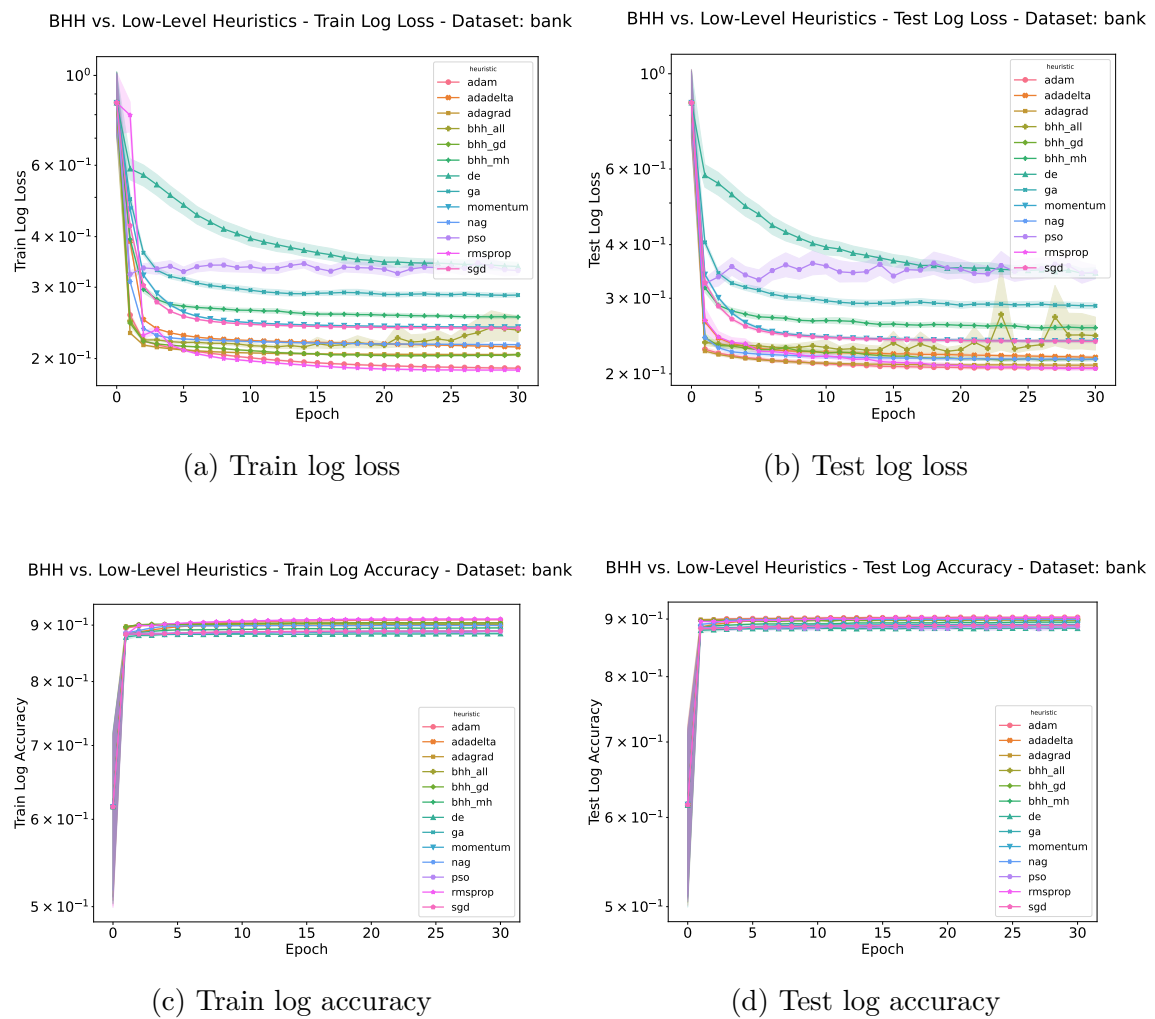


Figure 8.10: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the bank dataset over 30 epochs, illustrated in log scale.

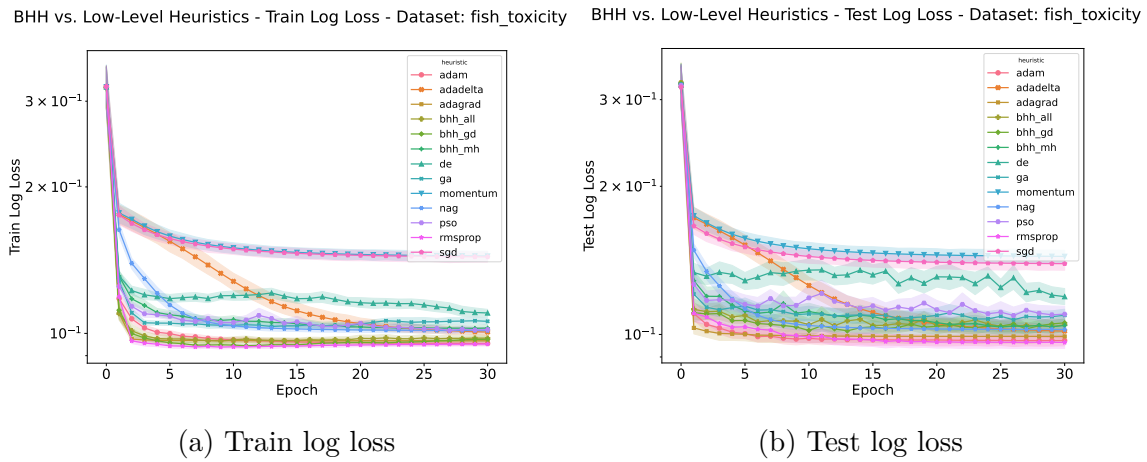


Figure 8.11: The train and test loss plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the fish toxicity dataset over 30 epochs, illustrated in log scale.

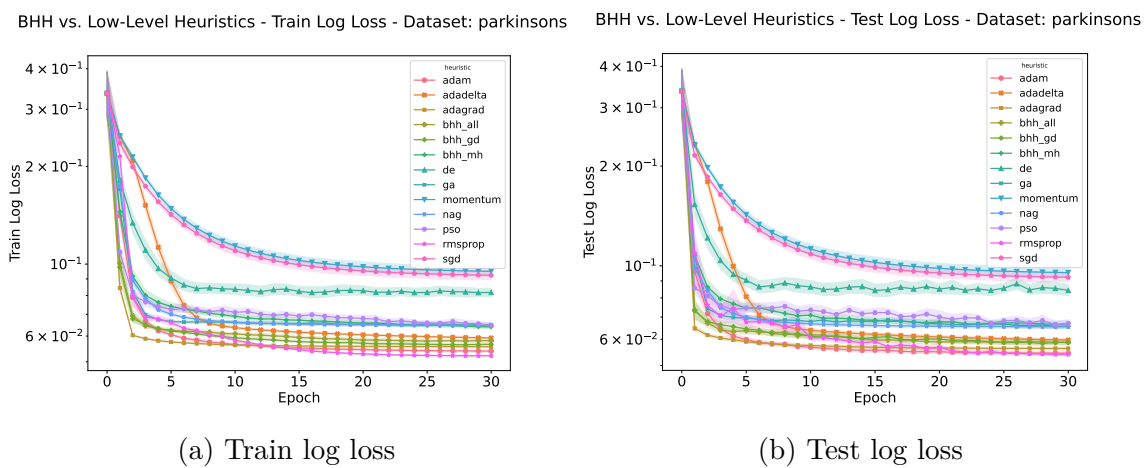


Figure 8.12: The train and test loss plots for the experimental group comparing the performance of the BHH to individual, standalone, low-level heuristics on the parkinsons dataset over 30 epochs, illustrated in log scale.

8.4 Heuristic Pool

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *heuristic pool* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. This experimental group utilises the same three variants of the BHH as was utilised in Section 8.3, but provides an opportunity to look at the BHH heuristic pool hyper-parameter in more depth. These variants are denoted as follows: The BHH baseline configuration with a heuristic pool that contains all the low-level heuristics is denoted as *all*, the BHH configuration with a heuristic pool that contains only gradient-based heuristics is denoted as *gd*, and finally, the BHH configuration with a heuristics pool that contains only meta-heuristics is denoted as *mh*.

Table 8.3 presents the empirical results for this experimental group, showing the average test loss and statistics for all the heuristic pool variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.3: Empirical results showing average test loss and statistics for all heuristic pool configurations used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Heuristic Pool - Average Test Loss		
	all	gd	mh
abalone	2.0587 (± 0.088)	2.0125 (± 0.068)	2.2942 (± 0.055)
air quality	0.2729 (± 0.016)	0.2642 (± 0.011)	0.2637 (± 0.007)
bank	0.2456 (± 0.065)	0.2164 (± 0.005)	0.2562 (± 0.011)
bike	0.0651 (± 0.02)	0.0545 (± 0.005)	0.1179 (± 0.006)
car	0.1572 (± 0.034)	0.169 (± 0.025)	0.4891 (± 0.077)
diabetic	1.2983 (± 0.66)	0.8976 (± 0.011)	0.9137 (± 0.008)
fish toxicity	0.1046 (± 0.008)	0.1056 (± 0.008)	0.1043 (± 0.009)
forest fires	0.081 (± 0.069)	0.0586 (± 0.034)	0.0621 (± 0.038)
housing	0.0941 (± 0.017)	0.0951 (± 0.015)	0.1169 (± 0.019)
iris	0.2411 (± 0.295)	0.3729 (± 1.119)	0.1809 (± 0.164)
mushroom	0.0052 (± 0.012)	0.0009 (± 0.001)	0.0774 (± 0.02)
parkinsons	0.0587 (± 0.003)	0.0587 (± 0.003)	0.0658 (± 0.003)
student performance	0.2359 (± 0.102)	0.2454 (± 0.134)	0.1947 (± 0.014)
wine quality	1.0827 (± 0.026)	1.1028 (± 0.039)	1.1666 (± 0.03)

Similar to before, Table 8.4 provides the average ranked performance, per dataset, of each of the BHH heuristic pool configurations. The performance rank is calculated as the average rank produced by each heuristic pool configuration, for all datasets,

over all independent runs and epochs.

Table 8.4: Empirical results showing average rank and statistics for all heuristic pool configurations used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Heuristic Pool - Average Rank		
	all	gd	mh
abalone	1.7946 (± 0.641)	1.3828 (± 0.56)	2.8226 (± 0.42)
air quality	2.1871 (± 0.815)	1.6828 (± 0.751)	2.1301 (± 0.788)
bank	1.7731 (± 0.562)	1.3215 (± 0.492)	2.9054 (± 0.334)
bike	1.6247 (± 0.553)	1.4387 (± 0.532)	2.9366 (± 0.281)
car	1.5903 (± 0.511)	1.4409 (± 0.518)	2.9688 (± 0.228)
diabetic	2.4516 (± 0.735)	1.2806 (± 0.576)	2.2677 (± 0.58)
fish toxicity	1.9903 (± 0.849)	1.8344 (± 0.768)	2.1753 (± 0.796)
forest fires	2.0667 (± 0.834)	1.8269 (± 0.781)	2.1065 (± 0.807)
housing	1.6043 (± 0.687)	1.6602 (± 0.665)	2.7355 (± 0.524)
iris	1.8581 (± 0.8)	1.7806 (± 0.727)	2.3613 (± 0.796)
mushroom	1.5204 (± 0.581)	1.5484 (± 0.527)	2.9312 (± 0.289)
parkinsons	1.557 (± 0.624)	1.6011 (± 0.591)	2.8419 (± 0.445)
student performance	1.9828 (± 0.822)	1.7903 (± 0.848)	2.2269 (± 0.715)
wine quality	1.5785 (± 0.55)	1.4935 (± 0.563)	2.928 (± 0.294)
avg rank	1.8271 (± 0.743)	1.5773 (± 0.671)	2.5955 (± 0.659)
normalised avg rank	2	1	3

Similar to the outcomes provided in Section 8.3, by ranked performance over all epochs, it is found that the *gd* configuration yielded the best overall performance compared to the *all* and *mh* configurations. However, when considering the test loss measured at the last epoch, there is no clear difference in performance overall. This illustrates the reasoning behind the use of the rank metric over all epochs to evaluate the overall training process, as the test loss outcome differs at different epochs and early stopping of the training process was not used.

Figure 8.13 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the performance of the different heuristic pool configurations, per dataset. From this illustration it can clearly be seen that, by ranked performance, the *gd* configuration produced better results overall, considering the entire training process.

Figure 8.14 provides an illustration of the overall critical difference plots that illustrate the statistically significant differences in ranked performance for each heuristic pool configuration as it relates to all datasets, across all independent runs and epochs.

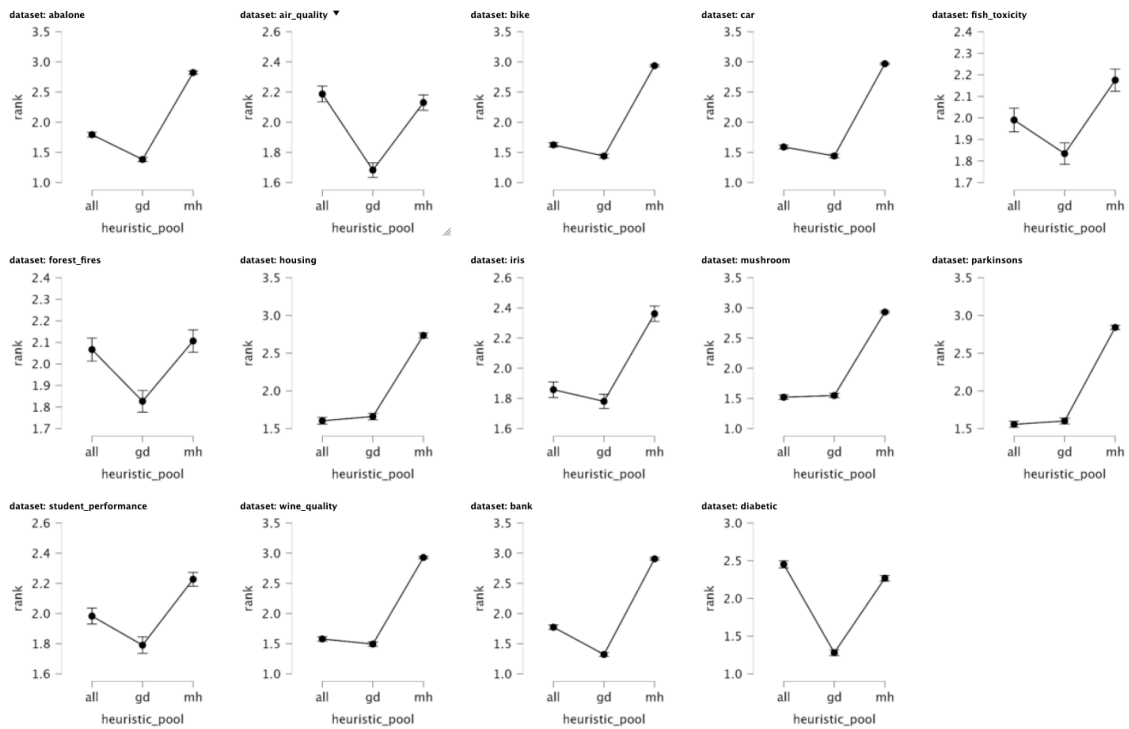


Figure 8.13: Descriptive plots for the average ranks of the BHH with varying heuristic pools per dataset, across all independent runs and epochs.

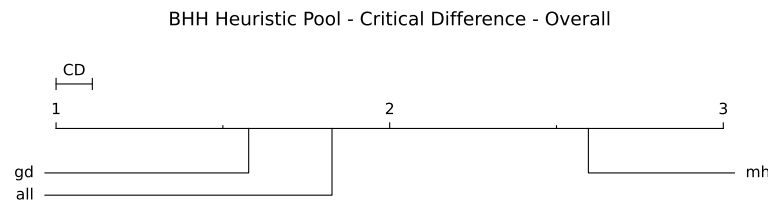


Figure 8.14: Critical difference plots for the average ranks of the BHH with varying heuristic pools across all datasets, runs and epochs.

From Figures 8.13 and 8.14, it is clear that the *gd* configuration performs best overall, with exceptions to the parkinsons and housing datasets, where the *all* and *gd* configurations performed equally well with statistically insignificant differences in their outcomes.

From the results shown in this section and Section 8.3, the gradient-based heuristics outperformed the meta-heuristics in almost all cases. It can then logically be concluded that the BHH baseline configuration with only gradient-based heuristics in the heuristic pool yields the best performance. The inclusion of meta-heuristics is done as an attempt to provide a mechanism that could provide even better performance, as well as generalisation capabilities to other datasets. The benefits of using meta-heuristics in the heuristic pool is not realised in this dissertation,

since the gradient-based heuristics provided the best overall performance across all datasets. Furthermore, the standalone gradient-based heuristics already produce good performance results, and improvement on those results are hard. Faster convergence is also difficult to achieve, since the BHH needs sufficient time to explore the heuristic space. The benefit that the BHH then brings is that it provides a mechanism whereby prior expert knowledge can be injected, before training starts. Since gradient-based heuristics perform well, future research can exploit this knowledge and provide a significant bias towards these gradient-based heuristics through the initialisation of the concentration parameters related to these heuristics as mentioned previously.

Similar to before, Figure 8.15 provides an illustration of the train and test loss and accuracy plots for an example classification dataset (abalone) as it relates to the heuristic pool experimental group. As before, the illustrations are provided in log scale and illustrations for the other classification datasets are left out for brevity as they yield similar illustrations.

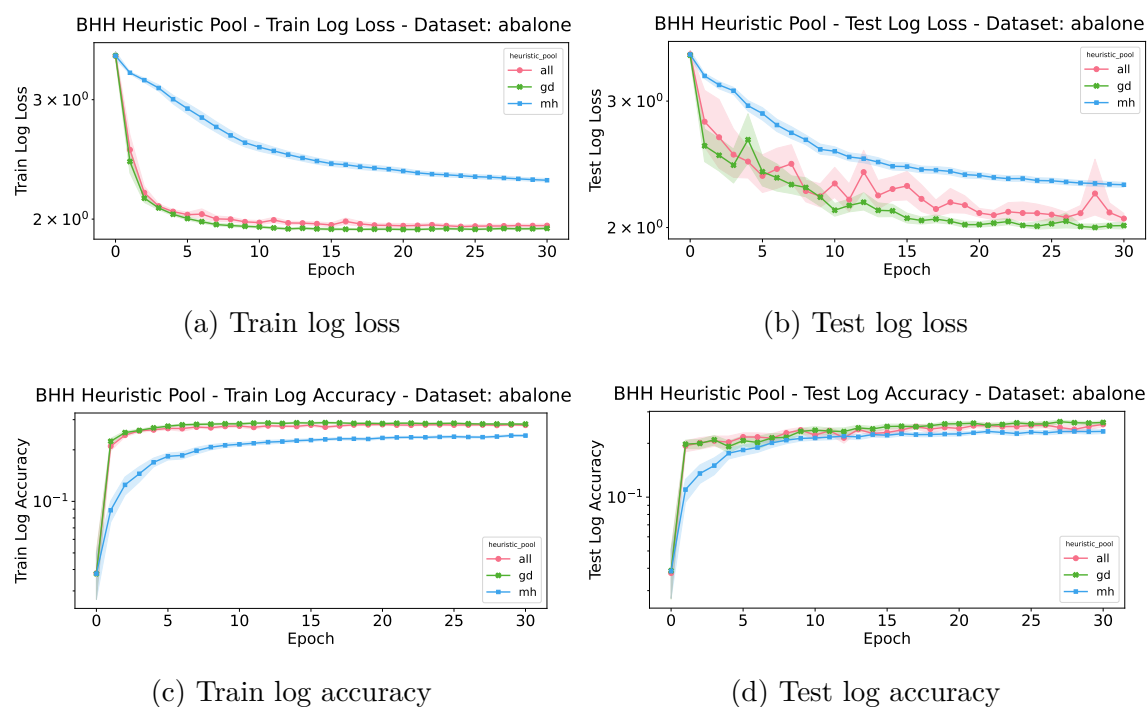


Figure 8.15: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the heuristic pool hyper-parameter on the abalone dataset over 30 epochs, illustrated in log scale.

Figure 8.16 provides the train and test loss plots for an example regression dataset (forest fires) as it relates to the heuristic pool experimental group. As before, the illustrations are provided in log scale and illustrations for the other regression datasets are left out for brevity.

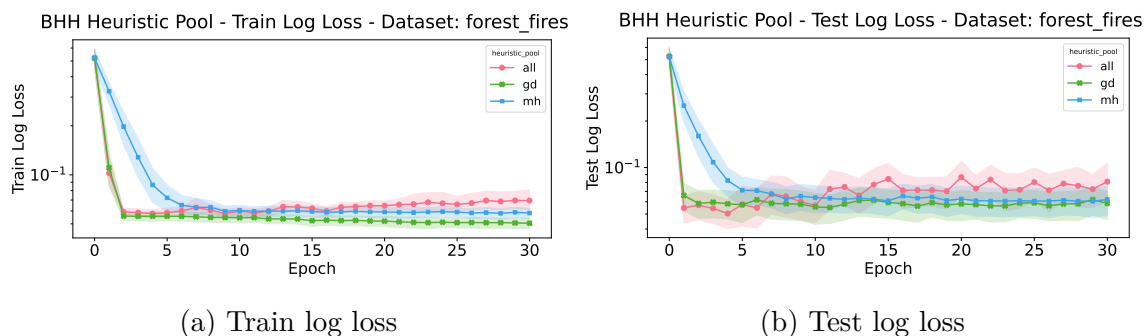


Figure 8.16: The train and test loss plots for the experimental group comparing the performance of the BHH with different configurations of the heuristic pool hyper-parameter on the forest fires dataset over 30 epochs, illustrated in log scale.

8.5 Population Size

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *population size* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, five different population sizes are considered. These include population sizes of 5, 10, 15, 20, and 25, and experiments are denoted as such.

Table 8.5 presents the empirical results for this experimental group, showing the average test loss and statistics for all the population size variants of the BHH that was implemented. Similar to before, the test loss is measured at the last epoch for each dataset, over all independent runs.

From Table 8.5 it can be seen that the lowest population size produced mostly the best performance outcomes, yielding the best average test loss for eight of the fourteen datasets. However, the lowest population size also produced the worst performance outcomes for four of the fourteen datasets. From the empirical results that show the average test loss, measured at the last epoch, it is not clear which population size configuration produced the overall best results.

As before, the performance of the different BHH population size configurations need to be considered for the entire training process. As such, Table 8.6 provides the average ranked performance, per dataset, of each of the BHH population size configurations. Similar to before, the performance rank is calculated as the average rank produced by each population size configuration, for all datasets, over all independent runs and epochs.

From Table 8.6 it can be seen that a lower population size of five, mostly produced the best results, with exception to the bike and car datasets, for which the lowest population size configuration produced the worst performance, yielding statistically

Table 8.5: Empirical results showing average test loss and statistics for all population size configurations used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Population - Average Test Loss				
	5	10	15	20	25
abalone	2.0587 (± 0.088)	2.1592 (± 0.278)	2.3321 (± 0.446)	2.3345 (± 0.357)	2.4188 (± 0.473)
air quality	0.2729 (± 0.016)	0.2701 (± 0.011)	0.2736 (± 0.014)	0.2808 (± 0.026)	0.2715 (± 0.011)
bank	0.2456 (± 0.065)	0.2356 (± 0.029)	0.2268 (± 0.015)	0.2294 (± 0.019)	0.2278 (± 0.012)
bike	0.0651 (± 0.02)	0.0527 (± 0.002)	0.0535 (± 0.003)	0.0541 (± 0.003)	0.0535 (± 0.003)
car	0.1572 (± 0.034)	0.1595 (± 0.035)	0.161 (± 0.039)	0.1636 (± 0.04)	0.17 (± 0.048)
diabetic	1.2983 (± 0.66)	0.9954 (± 0.103)	1.0826 (± 0.3)	0.9893 (± 0.105)	0.9875 (± 0.078)
fish toxicity	0.1046 (± 0.008)	0.1077 (± 0.009)	0.1071 (± 0.011)	0.1104 (± 0.017)	0.1079 (± 0.008)
forest fires	0.081 (± 0.069)	0.0694 (± 0.059)	0.0761 (± 0.056)	0.084 (± 0.07)	0.0854 (± 0.065)
housing	0.0941 (± 0.017)	0.0985 (± 0.026)	0.0943 (± 0.016)	0.0987 (± 0.017)	0.1038 (± 0.017)
iris	0.2411 (± 0.295)	0.4212 (± 0.671)	0.5905 (± 1.043)	0.2784 (± 0.261)	0.327 (± 0.381)
mushroom	0.0052 (± 0.012)	1.9037 (± 10.345)	0.3366 (± 1.559)	0.1311 (± 0.486)	0.2602 (± 1.115)
parkinsons	0.0587 (± 0.003)	0.0598 (± 0.003)	0.0606 (± 0.003)	0.0602 (± 0.003)	0.0612 (± 0.003)
student performance	0.2359 (± 0.102)	0.2007 (± 0.032)	0.2092 (± 0.039)	0.2033 (± 0.037)	0.2091 (± 0.054)
wine quality	1.0827 (± 0.026)	1.0864 (± 0.034)	1.1083 (± 0.054)	1.0961 (± 0.027)	1.1071 (± 0.039)

significant differences in outcomes from the other datasets. A larger population size configuration is preferred for the bike and car datasets. Furthermore, a population size configuration of ten, slightly higher than the default of five, is preferred for the forest fires and housing datasets. Finally, the overall normalised average rank is provided at the bottom of the table, showing that a population size of five produced the best performance outcome across all datasets on average.

Figure 8.17 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the performance of the different population size configurations, per dataset.

Figure 8.17 shows the correlation of performance with population size for each dataset. From these illustrations, it can be seen that the correlation between the population size configuration and performance is different for each dataset, suggesting that the population size hyper-parameter is problem specific. However, the overall performance related to each population configuration, across all datasets is not clear from these illustrations.

Figure 8.18 provides an illustration of the overall critical difference plots for ranked performance for each population size configuration as it relates to all datasets, across all independent runs and epochs. It is shown that there is no overall statistical significant difference between population sizes, and thus it can be concluded that the population size hyper-parameter is problem specific.

Similar to before, Figure 8.19 provides the train and test loss and accuracy plots

Table 8.6: Empirical results showing average rank and statistics for different population sizes used by the BHH across multiple datasets, for all independent runs and all epochs.

dataset	Population - Average Rank				
	5	10	15	20	25
abalone	2.4387 (± 1.34)	2.7452 (± 1.392)	3.0129 (± 1.386)	3.4065 (± 1.351)	3.3968 (± 1.351)
air quality	2.7527 (± 1.428)	2.9258 (± 1.39)	2.9656 (± 1.379)	3.314 (± 1.379)	3.0419 (± 1.437)
bank	2.8129 (± 1.41)	3.1591 (± 1.435)	2.9989 (± 1.42)	3.0505 (± 1.444)	2.9785 (± 1.341)
bike	3.8957 (± 1.335)	2.8688 (± 1.285)	2.9645 (± 1.345)	2.743 (± 1.376)	2.528 (± 1.328)
car	3.1892 (± 1.391)	3.0409 (± 1.394)	3.0398 (± 1.452)	2.8559 (± 1.394)	2.8742 (± 1.415)
diabetic	2.7978 (± 1.508)	2.9387 (± 1.413)	3.2129 (± 1.452)	2.9613 (± 1.304)	3.0892 (± 1.353)
fish toxicity	2.8151 (± 1.425)	3.0409 (± 1.429)	3.0925 (± 1.323)	3.128 (± 1.407)	2.9237 (± 1.463)
forest fires	3.0172 (± 1.403)	2.9215 (± 1.38)	2.9806 (± 1.383)	3.0968 (± 1.412)	2.9839 (± 1.488)
housing	2.8366 (± 1.404)	2.8258 (± 1.38)	2.9753 (± 1.439)	2.9849 (± 1.411)	3.3774 (± 1.368)
iris	2.8301 (± 1.304)	2.8731 (± 1.397)	3.0505 (± 1.413)	3.186 (± 1.425)	3.0602 (± 1.499)
mushroom	2.7989 (± 1.205)	2.9441 (± 1.325)	3.0462 (± 1.452)	3.1366 (± 1.494)	3.0538 (± 1.566)
parkinsons	2.6645 (± 1.441)	3.1065 (± 1.327)	3.0247 (± 1.427)	3.0495 (± 1.442)	3.1548 (± 1.381)
student performance	2.6376 (± 1.436)	2.8645 (± 1.411)	3.1548 (± 1.404)	3.2387 (± 1.397)	3.1043 (± 1.34)
wine quality	2.5505 (± 1.317)	2.7817 (± 1.415)	3.1581 (± 1.384)	3.1806 (± 1.394)	3.329 (± 1.414)
avg rank	2.8598 (± 1.424)	2.9312 (± 1.389)	3.0484 (± 1.406)	3.0952 (± 1.412)	3.064 (± 1.428)
normalised avg rank	1	2	3	5	4

for an example classification dataset (mushroom) as it relates to the population size experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

As a reminder, the experimental group for population sizes only varies the population size hyper-parameter and all other hyper-parameters remain the same between configurations. As such, all population size configurations make use of the heuristic pool configuration that includes all the low-level heuristics, including gradient-based heuristics and meta-heuristics.

Divergence of the training loss can be observed in Figure 8.19b. Note that, despite the divergence of training loss, the accuracy is still almost 100% and that these divergent behaviours are simply a result of an attempt to further improve performance, by exploring the heuristic space more. For example, heuristics could have momentum as a result of good training progression thus far, but then overshoot good solutions after heuristic reselection. Overshooting good solutions then causes the heuristics to struggle to converge back to good solutions, especially if many different heuristics are selected in quick succession. A suggestion to this problem is to implement a move-acceptance strategy as was mentioned before. At the ninth epoch, the BHH fails to produce better solutions on the training set. Since there is not much room for more improvement, the BHH is bound to try different heuristics

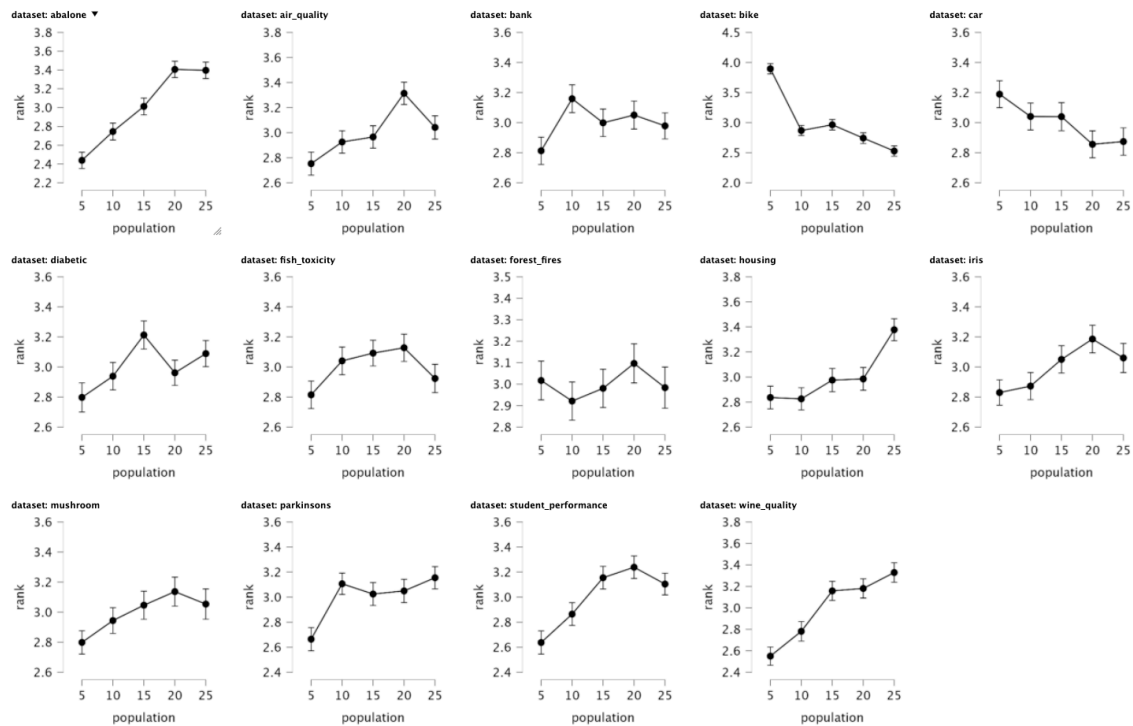


Figure 8.17: Descriptive plots for the average ranks of BHH with varying population sizes per dataset, across all independent runs and epochs.

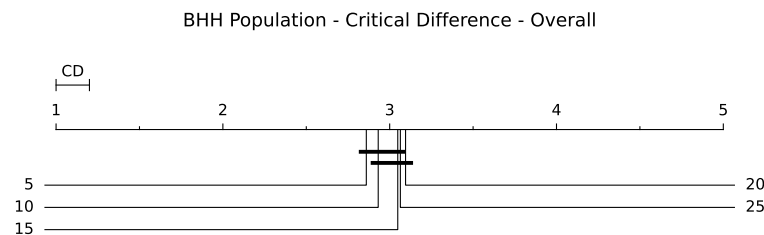


Figure 8.18: Critical difference plots for the average ranks of BHH with varying population sizes across all datasets, runs and epochs.

that yield sub-optimal results. From this point onwards, the BHH finds slightly better results on the train set, and as a result, produces volatile performance on the test set.

Figure 8.20 provides the train and test loss plots for an example regression dataset (student performance) as it relates to the population size experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

From Figure 8.20a it can be seen that the BHH with a low population size of five starts to diverge away from optimal results as it explores different heuristics in an attempt to provide better solutions. Divergence can be eliminated by means of an early stopping strategy as well as a move-acceptance strategy as mentioned before.

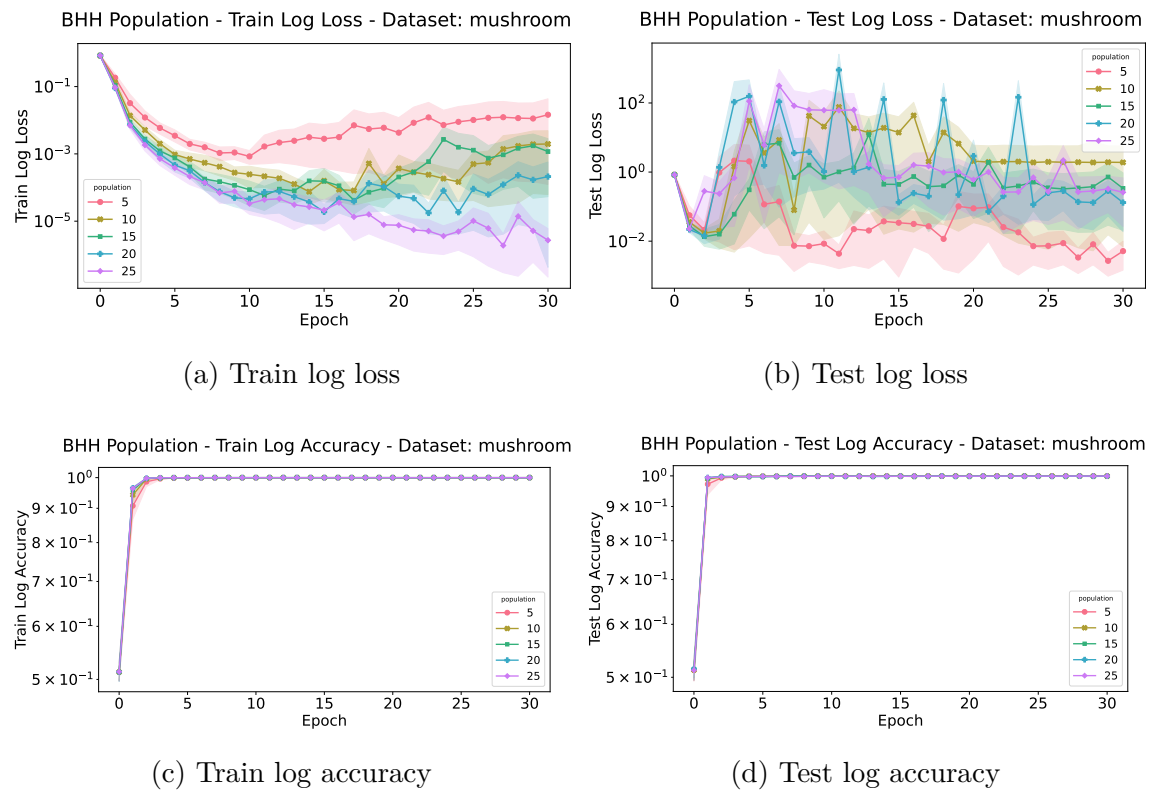


Figure 8.19: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the population size hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.

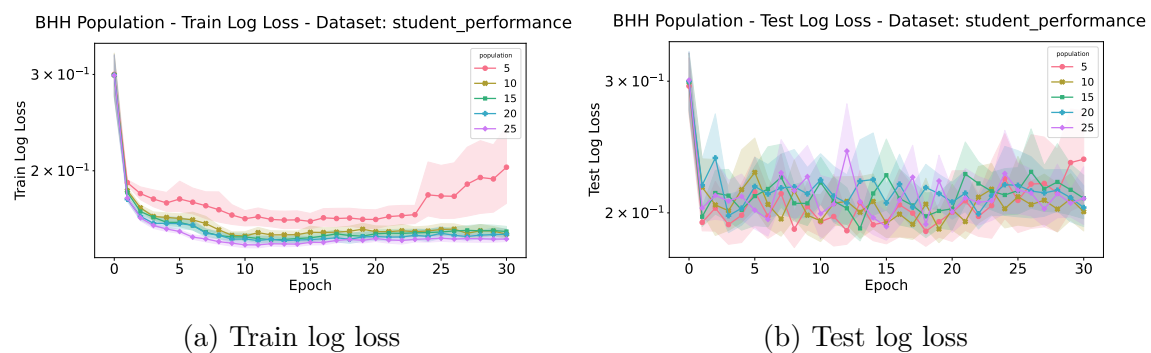


Figure 8.20: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the population size hyper-parameter on the student performance dataset over 30 epochs, illustrated in log scale.

8.6 Credit Assignment Strategy

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *credit assignment strategy* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, five different credit assignment strategies are considered. These include the *ibest*, *pbest*, *rbest*, *gbest*, and *symmetric* credit assignment strategies as presented in Chapter 6 and experiments are denoted as such.

Table 8.7 presents the empirical results for this experimental group, showing the average test loss and statistics for all the credit assignment strategy variants of the BHH that was implemented. As before, the test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.7: Empirical results showing average test loss and statistics for different credit assignment strategies used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Credit - Average Test Loss				
	ibest	pbest	rbest	gbest	symmetric
abalone	2.0587 (+0.088)	2.0422 (+0.094)	2.0922 (+0.152)	2.0833 (+0.136)	2.1177 (+0.259)
air quality	0.2729 (+0.016)	0.2678 (+0.015)	0.2644 (+0.015)	0.2684 (+0.013)	0.2688 (+0.017)
bank	0.2456 (+0.065)	0.2645 (+0.171)	0.238 (+0.047)	0.2361 (+0.031)	0.2498 (+0.047)
bike	0.0651 (+0.02)	0.0634 (+0.023)	0.0633 (+0.02)	0.063 (+0.021)	0.0666 (+0.021)
car	0.1572 (+0.034)	0.1569 (+0.029)	0.1665 (+0.03)	0.1504 (+0.03)	0.1564 (+0.037)
diabetic	1.2983 (+0.66)	1.4981 (+2.41)	1.6172 (+1.527)	1.0347 (+0.231)	1.3627 (+1.145)
fish toxicity	0.1046 (+0.008)	0.1028 (+0.008)	0.1024 (+0.009)	0.1061 (+0.009)	0.1026 (+0.009)
forest fires	0.081 (+0.069)	0.0819 (+0.062)	0.0591 (+0.046)	0.0845 (+0.077)	0.0822 (+0.07)
housing	0.0941 (+0.017)	0.0974 (+0.021)	0.0961 (+0.018)	0.0995 (+0.016)	0.0969 (+0.018)
iris	0.2411 (+0.295)	0.1441 (+0.131)	0.1547 (+0.111)	0.1616 (+0.187)	0.1507 (+0.142)
mushroom	0.0052 (+0.012)	0.0289 (+0.107)	0.0044 (+0.011)	0.1835 (+0.949)	5.3926 (+29.503)
parkinsons	0.0587 (+0.003)	0.0594 (+0.002)	0.0602 (+0.002)	0.0592 (+0.002)	0.0593 (+0.003)
student performance	0.2359 (+0.102)	0.2303 (+0.095)	0.2551 (+0.109)	0.2251 (+0.088)	0.2777 (+0.116)
wine quality	1.0827 (+0.026)	1.0799 (+0.031)	1.0794 (+0.021)	1.0757 (+0.027)	1.0835 (+0.029)

From Table 8.7, it is not clear which credit assignment strategy yields the best performance for the entire training process. As such, Table 8.8 provides the average ranked performance, per dataset, of each of the BHH credit assignment strategy configurations. Similar to before, the performance rank is calculated as the average rank produced by each credit assignment strategy configuration, for all datasets, over all independent runs and epochs.

Table 8.8 shows that different credit assignment strategies perform best for different datasets, from which it can be concluded that the credit assignment strategy hyper-parameter is problem specific. Furthermore, the overall normalised average rank across all datasets is provided at the bottom of the table. It should be noted

Table 8.8: Empirical results showing average rank and statistics for different credit assignment strategies used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Credit - Average Rank				
	ibest	pbest	rbest	gbest	symmetric
abalone	3.1677 (± 1.319)	2.8613 (± 1.459)	2.9968 (± 1.399)	2.9667 (± 1.455)	3.0075 (± 1.421)
air quality	3.1312 (± 1.347)	3.0376 (± 1.357)	2.5914 (± 1.465)	3.128 (± 1.336)	3.1118 (± 1.487)
bank	2.8591 (± 1.344)	2.9903 (± 1.432)	2.8882 (± 1.41)	3.0441 (± 1.415)	3.2183 (± 1.442)
bike	3.0527 (± 1.34)	3.0086 (± 1.393)	3.0667 (± 1.483)	2.8742 (± 1.339)	2.9978 (± 1.503)
car	3.1151 (± 1.356)	3.0312 (± 1.483)	3.2516 (± 1.402)	2.6892 (± 1.354)	2.9129 (± 1.411)
diabetic	2.8914 (± 1.349)	2.6269 (± 1.417)	3.4151 (± 1.365)	2.8925 (± 1.362)	3.1742 (± 1.449)
fish toxicity	3.1516 (± 1.436)	2.9903 (± 1.452)	2.7581 (± 1.475)	3.2043 (± 1.299)	2.8957 (± 1.358)
forest fires	3.0968 (± 1.277)	3.1806 (± 1.304)	2.8559 (± 1.356)	3.1215 (± 1.502)	2.7452 (± 1.563)
housing	2.9011 (± 1.494)	2.8527 (± 1.317)	2.9022 (± 1.397)	3.3108 (± 1.324)	3.0333 (± 1.483)
iris	3.1892 (± 1.428)	3.0839 (± 1.441)	3.0591 (± 1.376)	2.8237 (± 1.409)	2.8441 (± 1.384)
mushroom	2.8075 (± 1.459)	3.0183 (± 1.411)	2.8957 (± 1.398)	3.0839 (± 1.418)	3.172 (± 1.381)
parkinsons	2.5645 (± 1.484)	2.8925 (± 1.343)	3.5065 (± 1.219)	3.0796 (± 1.392)	2.957 (± 1.455)
student performance	2.6624 (± 1.312)	3.029 (± 1.407)	3.1892 (± 1.382)	2.7978 (± 1.47)	3.3215 (± 1.394)
wine quality	3.1871 (± 1.308)	2.6366 (± 1.471)	3.014 (± 1.371)	2.9419 (± 1.411)	3.2204 (± 1.43)
avg rank	2.9841 (± 1.39)	2.9457 (± 1.415)	3.0279 (± 1.414)	2.997 (± 1.402)	3.0437 (± 1.449)
normalised avg rank	2	1	4	3	5

that the *symmetric* credit assignment strategy yielded the best results for the forest fires dataset. This does not necessarily suggest that random search in the heuristic space yields the best performance for this dataset. As a reminder, the *symmetric* credit assignment strategy does not bias towards performance, but rather uniformly assigns credit to any heuristic that happens to be selected. For the particular case where the *symmetric* credit assignment strategy yielded the best results, it could be the case that the initial selection of heuristics is good enough and that it is difficult to find a performance bias that results in better performance, as all heuristics in the heuristic pool provide good results. For all other datasets, it is found that a particular non-*symmetric* credit assignment strategy yields better results.

Figure 8.21 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the performance of the different credit assignment strategy configurations, per dataset.

Figure 8.22 provides an illustration of the overall critical difference plots for ranked performance for each credit assignment strategy configuration as it relates to all datasets, across all independent runs and epochs. It is shown that there is no overall statistical significant difference between credit assignment strategies and that the credit assignment strategy hyper-parameter is problem specific. At this point it should be mentioned that the credit assignment strategies implemented in this

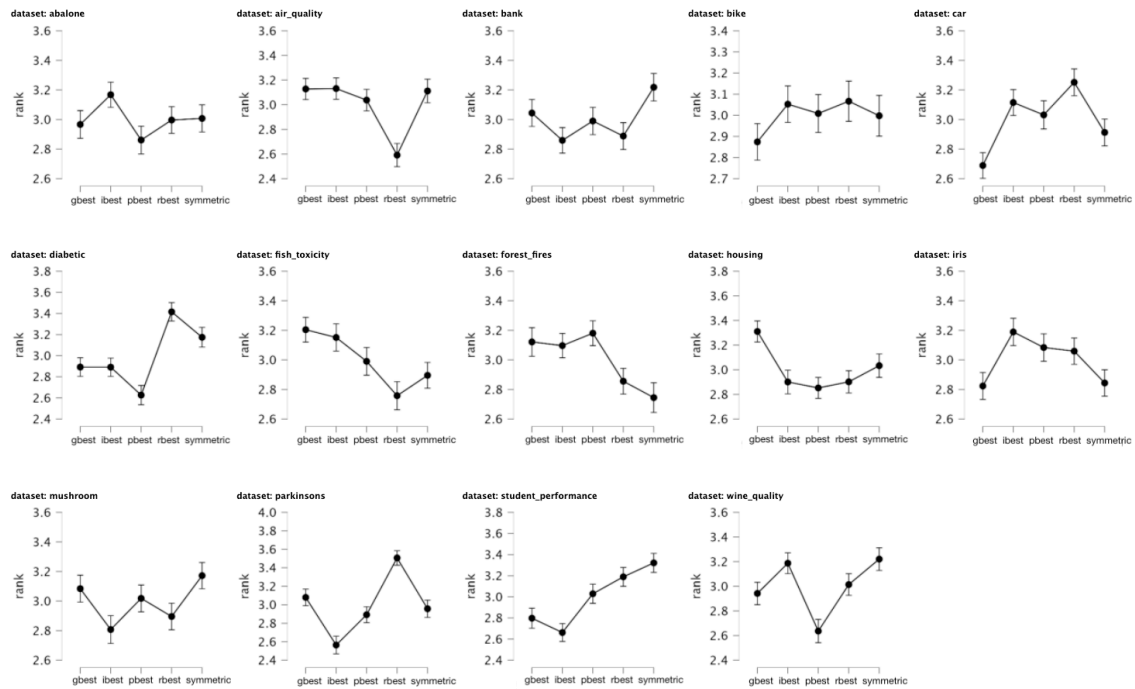


Figure 8.21: Descriptive plots for the average ranks of the BHH with varying credit assignment strategies per dataset, across all independent runs and epochs.

dissertation yield a discrete credit value and that future research should consider a continuous credit value in an attempt to provide a more fine grained indicator of performance, which should be easier to exploit.

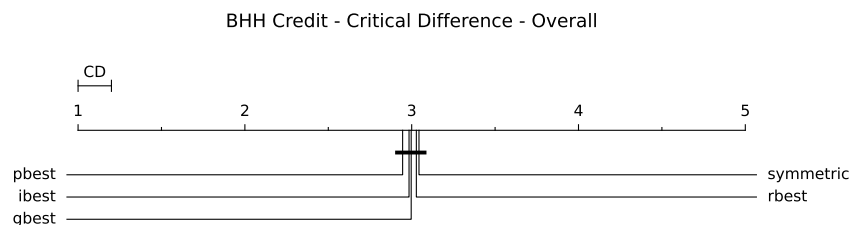


Figure 8.22: Critical difference plots for the average ranks of the BHH with varying credit assignment strategies across all datasets, runs and epochs.

Similar to before, Figure 8.23 provides the train and test loss and accuracy plots for an example classification dataset (bank) as it relates to the credit assignment strategy experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

From Figure 8.23 it can be seen that the *pbest* credit assignment strategy diverged away from the current best solution, but was able to return to the current best solution. Since the solution found by the BHH with the *pbest* credit assignment strategy was

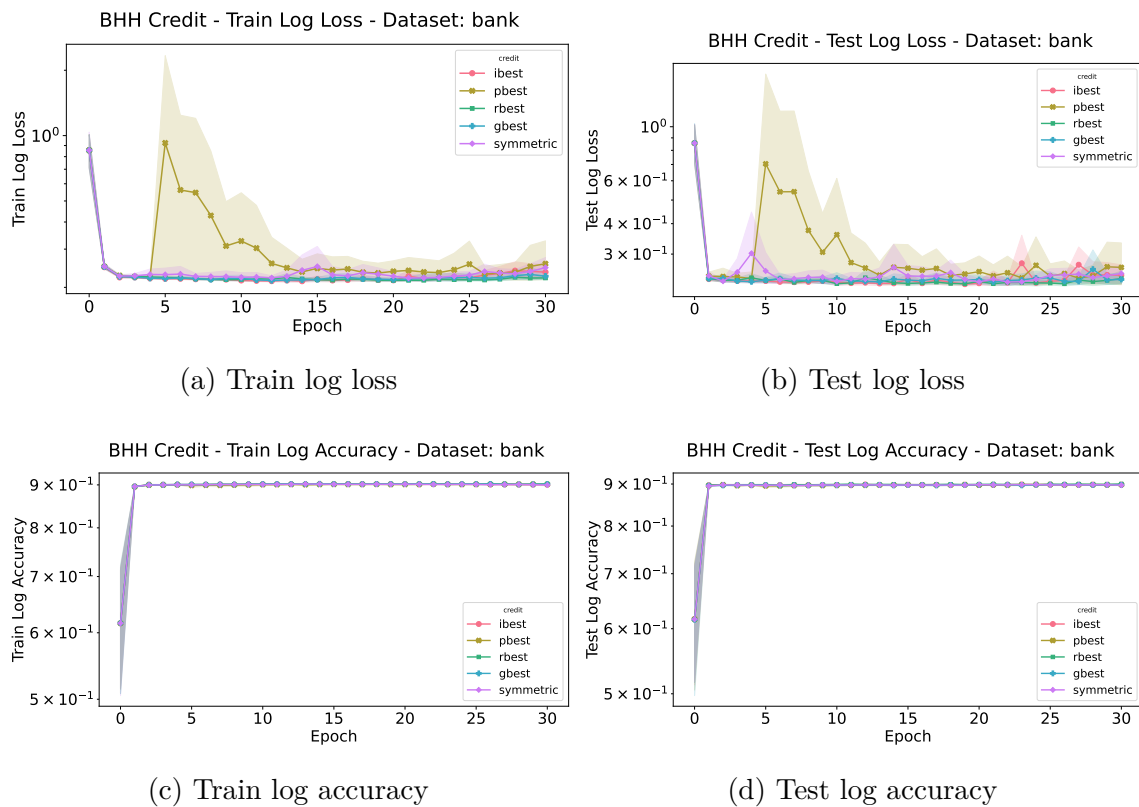


Figure 8.23: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the credit assignment strategy hyper-parameter on the bank dataset over 30 epochs, illustrated in log scale.

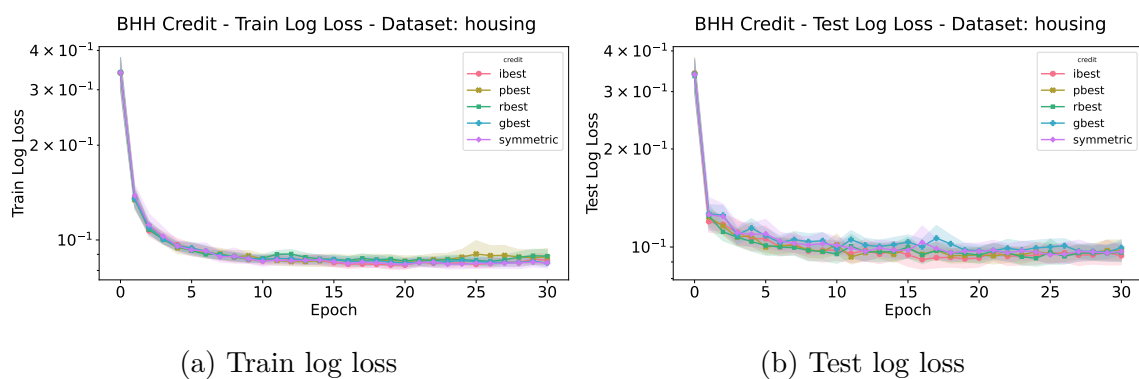


Figure 8.24: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the credit assignment strategy hyper-parameter on the housing dataset over 30 epochs, illustrated in log scale.

already optimal before divergence, it stands to reason that this divergence is a result of the BHH exploring other heuristics in the heuristic space that yield sub-optimal solutions. As mentioned before, a move-acceptance strategy can be incorporated to counteract this effect.

Figure 8.24 provides the train and test loss plots for an example regression dataset (housing) as it relates to the credit assignment strategy experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

8.7 Reselection Interval

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *reselection interval* hyper-parameter. Detailed discussions follow and illustrations are provided for visual aid. As a reminder, five different reselection intervals are considered. These include reselection intervals of 1, 5, 10, 15, and 20. Experiments are denoted as such.

Table 8.9 presents the empirical results for this experimental group, showing the average test loss and statistics for all the reselection interval variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.9: Empirical results showing average test loss and statistics for different reselection intervals used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Reselection - Average Test Loss				
	1	5	10	15	20
abalone	2.737 (± 0.507)	2.0907 (± 0.113)	2.0587 (± 0.088)	2.0475 (± 0.103)	2.1139 (± 0.336)
air quality	0.3003 (± 0.046)	0.2835 (± 0.021)	0.2729 (± 0.016)	0.2619 (± 0.013)	0.2637 (± 0.016)
bank	0.2933 (± 0.033)	0.3198 (± 0.18)	0.2456 (± 0.065)	0.2299 (± 0.032)	0.224 (± 0.018)
bike	0.1524 (± 0.023)	0.1084 (± 0.025)	0.0651 (± 0.02)	0.0566 (± 0.01)	0.052 (± 0.004)
car	0.2157 (± 0.031)	0.1591 (± 0.026)	0.1572 (± 0.034)	0.1659 (± 0.064)	0.1456 (± 0.03)
diabetic	0.9624 (± 0.043)	3.7821 (± 4.993)	1.2983 (± 0.66)	1.2541 (± 0.868)	1.0085 (± 0.251)
fish toxicity	0.1091 (± 0.01)	0.1043 (± 0.009)	0.1046 (± 0.008)	0.1013 (± 0.008)	0.1021 (± 0.008)
forest fires	0.1253 (± 0.087)	0.1002 (± 0.064)	0.081 (± 0.069)	0.0681 (± 0.055)	0.0586 (± 0.056)
housing	0.1268 (± 0.025)	0.0974 (± 0.017)	0.0941 (± 0.017)	0.0961 (± 0.026)	0.0952 (± 0.02)
iris	0.1633 (± 0.135)	0.1376 (± 0.138)	0.2411 (± 0.295)	0.2373 (± 0.361)	0.1853 (± 0.235)
mushroom	1.8461 (± 5.829)	0.0283 (± 0.099)	0.0052 (± 0.012)	0.0388 (± 0.111)	0.0335 (± 0.158)
parkinsons	0.0872 (± 0.018)	0.0609 (± 0.004)	0.0587 (± 0.003)	0.0584 (± 0.002)	0.0585 (± 0.003)
student performance	0.4376 (± 0.108)	0.397 (± 0.13)	0.2359 (± 0.102)	0.223 (± 0.098)	0.1991 (± 0.064)
wine quality	1.1183 (± 0.031)	1.081 (± 0.023)	1.0827 (± 0.026)	1.0711 (± 0.028)	1.0729 (± 0.022)

From Table 8.9, a clear pattern can be observed. The empirical results show

that, in general, a larger reselection interval is preferred, with the largest reselection intervals yielding the best performance outcomes in the majority of cases, and the lowest reselection interval of one, yielding the worst performance in the majority of cases.

To illustrate the effects of the reselection interval hyper-parameter on the entire training process, Table 8.10 provides the average ranked performance, per dataset, of each of the BHH reselection interval configurations. Similar to before, the performance rank is calculated as the average rank produced by reselection interval configuration, for all datasets, over all independent runs and epochs.

Table 8.10: Empirical results showing average rank and statistics for different reselection intervals used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Reselection - Average Rank				
	1	5	10	15	20
abalone	4.3548 (± 0.976)	2.8473 (± 1.288)	2.7505 (± 1.262)	2.5602 (± 1.299)	2.4871 (± 1.318)
air quality	3.7355 (± 1.405)	3.4075 (± 1.379)	2.9688 (± 1.278)	2.4237 (± 1.228)	2.4645 (± 1.291)
bank	4.5677 (± 0.704)	3.6323 (± 1.127)	2.4237 (± 1.133)	2.2462 (± 1.114)	2.1301 (± 1.096)
bike	4.7935 (± 0.612)	3.9204 (± 0.648)	2.5161 (± 0.863)	2.0763 (± 0.901)	1.6935 (± 0.891)
car	4.6409 (± 0.785)	2.9527 (± 1.158)	2.6548 (± 1.175)	2.5312 (± 1.244)	2.2204 (± 1.217)
diabetic	3.0129 (± 1.048)	4.4624 (± 0.869)	2.7957 (± 1.175)	2.6065 (± 1.439)	2.1226 (± 1.264)
fish toxicity	3.7484 (± 1.238)	3.1376 (± 1.317)	3.0344 (± 1.406)	2.4656 (± 1.304)	2.614 (± 1.432)
forest fires	3.8194 (± 1.433)	3.272 (± 1.248)	2.8656 (± 1.346)	2.8194 (± 1.305)	2.2237 (± 1.219)
housing	4.1892 (± 1.099)	3.0172 (± 1.288)	2.5871 (± 1.354)	2.5312 (± 1.265)	2.6753 (± 1.34)
iris	2.9839 (± 1.424)	2.971 (± 1.317)	3.3462 (± 1.418)	3.014 (± 1.405)	2.6849 (± 1.429)
mushroom	4.2065 (± 1.236)	2.9688 (± 1.299)	2.586 (± 1.234)	2.7882 (± 1.34)	2.4505 (± 1.226)
parkinsons	4.7645 (± 0.756)	3.1882 (± 1.021)	2.2957 (± 1.209)	2.4968 (± 1.128)	2.2548 (± 1.097)
student performance	3.971 (± 1.094)	4.0226 (± 1.12)	2.4774 (± 1.221)	2.2495 (± 1.2)	2.2796 (± 1.132)
wine quality	4.0688 (± 1.122)	2.9204 (± 1.293)	2.9559 (± 1.383)	2.4032 (± 1.378)	2.6516 (± 1.28)
avg rank	4.0612 (± 1.229)	3.3372 (± 1.277)	2.7327 (± 1.283)	2.5151 (± 1.282)	2.3538 (± 1.266)
normalised avg rank	5	4	3	2	1

Table 8.10 shows that larger reselection intervals are mostly preferred. The overall normalised average rank is given at the bottom of the table and supports this finding. This outcome can be explained as follows. A larger reselection interval gives the low-level heuristics more time to execute, resulting in smoother update steps and provides an opportunity for the BHH to gather more evidence of the current selection of heuristics' performance in the performance log. Both of these effects allow the BHH to select better heuristics during training. This is further supported by the findings from Sections 8.3 and 8.4, that show that gradient-based heuristics generally yield the best performance. Most of the gradient-based heuristics that are included in the heuristic pool implement exponentially averaged state parameters in order to produce smooth update steps. If reselection occurs too frequently, these

exponentially averaged state parameters do not yield smooth update steps. This is further supported by the inclusion of [MHs](#) such as [GAs](#) and [DE](#), which recombine state parameters.

Finally, it should be noted that the default reanalysis interval is 10 and the default replay window size is also ten. At every reanalysis interval, the concentrations for low-level heuristics are reset to the default, symmetrical value of 1.0. The replay window size then determines the number of samples in the performance log from which the reanalysis can be done. As such, [BHH](#) configurations with reselection intervals that are larger than the replay window size, will only consider performance samples from the last replay window, which should contain less entries than the reselection interval. The reselection interval then directly influences what goes into the performance log. Since it is found that reselection intervals larger than the default reanalysis interval and replay window size yield better performance, it can be suggested that exploitation of low-level heuristics yields better results than exploration of low-level heuristics.

Furthermore, early in the training process, a low reselection interval yields the highest variance of heuristic selection. Since the default population size is five and the default number of heuristics in the heuristic pool is thirteen, not all heuristics are represented at every training step, causing an unwanted bias towards the heuristics that happen to be selected.

Figure [8.21](#) provides an illustration of the descriptive plots for the different [BHH](#) configurations as it relates to the performance of different reselection interval configurations, per dataset. It can be seen that a general pattern occurs for almost all datasets that show an inverse correlation between performance and the reselection interval. Some exceptions to this relationship can be observed for the iris and diabetic datasets where an average reselection interval yielded the best results.

Figure [8.26](#) provides an illustration of the overall critical difference plots for ranked performance for each reselection interval configuration as it relates to all datasets, across all independent runs and epochs.

Figure [8.26](#) shows the statistically significant difference in overall results for the various reselection intervals across all datasets. It can be seen that larger reselection intervals statistically perform better than smaller reselection intervals. The largest reselection intervals (15 and 20) yielded the best overall results with statistically insignificant differences between them.

Similar to before, Figure [8.27](#) provides the train and test loss and accuracy plots for an example classification dataset (car) as it relates to the reselection interval experimental group. As before, the illustrations are provided in log scale and

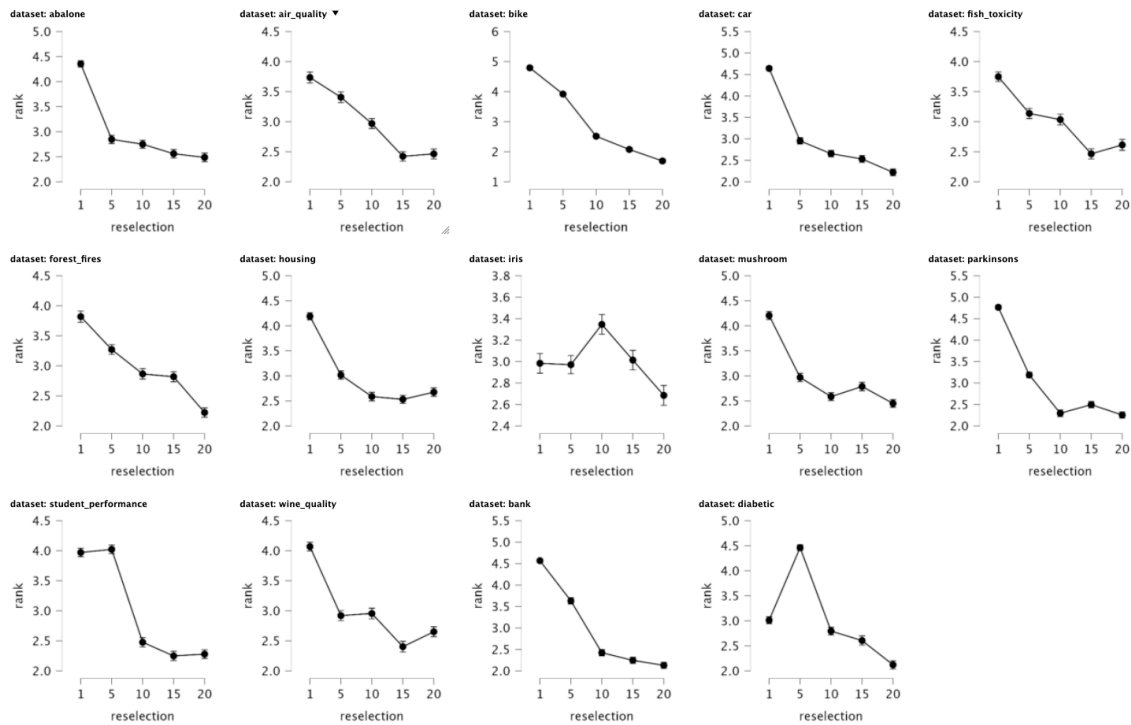


Figure 8.25: Descriptive plots for the average ranks of the BHH with varying reselection interval values per dataset, across all independent runs and epochs.

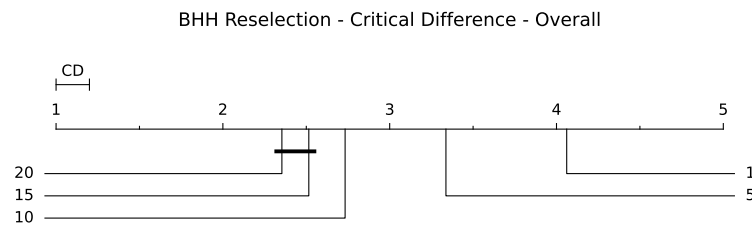


Figure 8.26: Critical difference plots for the average ranks of the BHH with varying reselection interval values across all datasets, runs and epochs.

illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

Figure 8.28 provides the train and test loss plots for an example regression dataset (bike) as it relates to the reselection interval experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

Figures 8.27a and 8.28a both show a smooth decline in training loss for all reselection interval configurations of the BHH, but larger reselection interval configurations are able to provide better results than lower reselection intervals. This further supports the suggestion that exploitation of low-level heuristics is more important than exploration of low-level heuristics in this particular case.

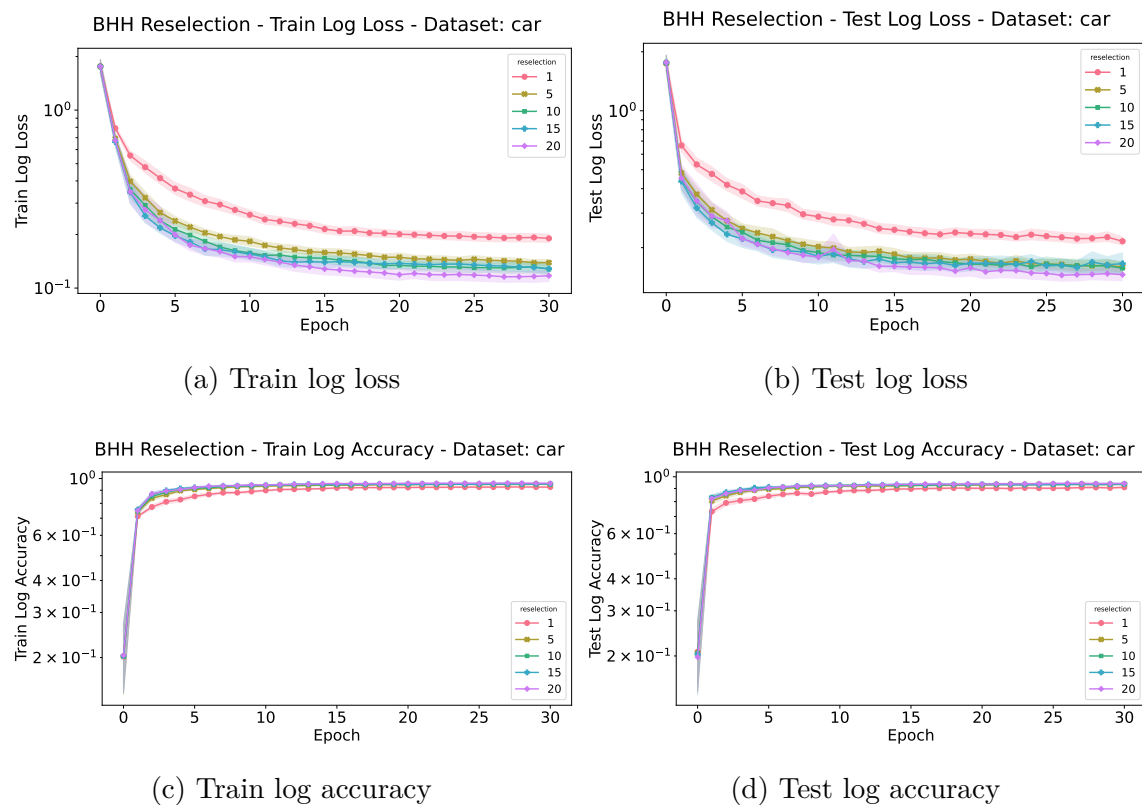


Figure 8.27: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the reselection interval hyper-parameter on the car dataset over 30 epochs, illustrated in log scale.

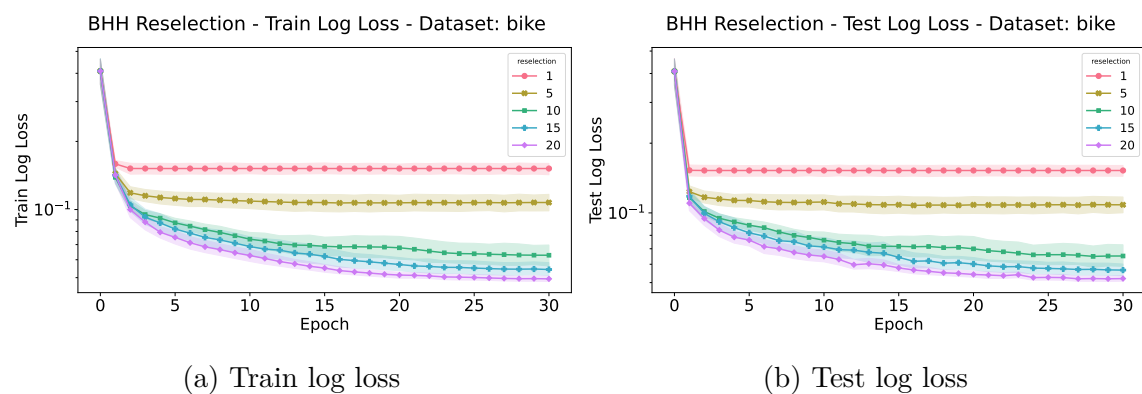


Figure 8.28: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the reselection interval hyper-parameter on the bike dataset over 30 epochs, illustrated in log scale.

8.8 Replay Window Size

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *replay window size* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, five different replay window sizes are considered. These include replay window sizes of 1, 5, 10, 15, and 20. Experiments are denoted as such.

As before, Table 8.11 presents the empirical results for this experimental group, showing the average test loss and statistics for all the replay window size variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.11: Empirical results showing average test loss and statistics for different replay window sizes used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

	Replay - Average Test Loss				
dataset	1	5	10	15	20
abalone	2.136 (± 0.088)	2.0424 (± 0.47)	2.0587 (± 0.135)	2.0551 (± 0.159)	2.1299 (± 0.292)
air quality	0.2676 (± 0.016)	0.2687 (± 0.014)	0.2729 (± 0.016)	0.2703 (± 0.015)	0.2673 (± 0.011)
bank	0.2358 (± 0.065)	0.2347 (± 0.022)	0.2456 (± 0.019)	0.2374 (± 0.037)	0.2358 (± 0.025)
bike	0.0568 (± 0.02)	0.0552 (± 0.007)	0.0651 (± 0.004)	0.0625 (± 0.019)	0.065 (± 0.018)
car	0.1605 (± 0.034)	0.1537 (± 0.05)	0.1572 (± 0.026)	0.1671 (± 0.056)	0.1599 (± 0.028)
diabetic	1.3315 (± 0.66)	1.2672 (± 0.691)	1.2983 (± 0.534)	1.0647 (± 0.22)	1.4342 (± 1.399)
fish toxicity	0.1028 (± 0.008)	0.1055 (± 0.008)	0.1046 (± 0.012)	0.1034 (± 0.011)	0.1074 (± 0.012)
forest fires	0.0935 (± 0.069)	0.0814 (± 0.078)	0.081 (± 0.073)	0.0794 (± 0.063)	0.105 (± 0.068)
housing	0.0979 (± 0.017)	0.0944 (± 0.015)	0.0941 (± 0.015)	0.0967 (± 0.022)	0.1002 (± 0.024)
iris	0.1577 (± 0.295)	0.295 (± 0.16)	0.2411 (± 0.546)	0.1514 (± 0.192)	0.2208 (± 0.349)
mushroom	0.0504 (± 0.012)	0.0881 (± 0.239)	0.0052 (± 0.336)	0.4733 (± 2.385)	0.139 (± 0.62)
parkinsons	0.0596 (± 0.003)	0.0589 (± 0.003)	0.0587 (± 0.003)	0.0596 (± 0.003)	0.0594 (± 0.003)
student performance	0.2529 (± 0.102)	0.2456 (± 0.103)	0.2359 (± 0.093)	0.2354 (± 0.083)	0.2739 (± 0.126)
wine quality	1.0876 (± 0.026)	1.0756 (± 0.035)	1.0827 (± 0.022)	1.0866 (± 0.039)	1.0767 (± 0.022)

Table 8.12 provides the average ranked performance, per dataset, for each of the BHH replay window size configurations. Similar to before, the performance rank is calculated as the average rank produced by the replay window size configurations, for all datasets, over all independent runs and epochs. The overall normalised average rank is provided at the end of the table.

Tables 8.11 and 8.12 show that there is no clear overall best performing replay window size configuration of the BHH for all datasets. In some cases, a replay window size of one yielded the best results. A replay window of one simply considers the last step in the training process. This means that memory of past performance actually had a negative impact on the training process for those particular cases. It should

Table 8.12: Empirical results showing average rank and statistics for different replay window sizes used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Replay - Average Rank				
	1	5	10	15	20
abalone	3.0258 (± 1.382)	2.6785 (± 1.455)	3.1968 (± 1.346)	3.0903 (± 1.385)	3.0086 (± 1.45)
air quality	2.8742 (± 1.385)	3.0151 (± 1.497)	2.9978 (± 1.429)	2.9505 (± 1.297)	3.1624 (± 1.443)
bank	2.9484 (± 1.46)	3.057 (± 1.473)	2.8602 (± 1.344)	3.0172 (± 1.42)	3.1172 (± 1.359)
bike	2.9602 (± 1.274)	2.7366 (± 1.34)	3.1452 (± 1.448)	2.9226 (± 1.417)	3.2355 (± 1.527)
car	2.7054 (± 1.357)	2.8376 (± 1.39)	3.0763 (± 1.372)	3.3269 (± 1.463)	3.0538 (± 1.409)
diabetic	3.2473 (± 1.369)	3.3247 (± 1.406)	2.5935 (± 1.372)	2.8075 (± 1.384)	3.0269 (± 1.411)
fish toxicity	2.7269 (± 1.419)	3.0785 (± 1.49)	3.1774 (± 1.398)	2.9892 (± 1.386)	3.028 (± 1.337)
forest fires	2.8054 (± 1.493)	2.9151 (± 1.381)	2.9914 (± 1.408)	3.0591 (± 1.411)	3.229 (± 1.342)
housing	3.1548 (± 1.301)	2.9774 (± 1.407)	2.8548 (± 1.45)	3.0355 (± 1.49)	2.9774 (± 1.404)
iris	3.072 (± 1.376)	3.1581 (± 1.419)	3.2204 (± 1.392)	2.6183 (± 1.396)	2.9312 (± 1.41)
mushroom	3.0538 (± 1.324)	3.143 (± 1.437)	3.0419 (± 1.414)	2.8613 (± 1.382)	2.8957 (± 1.497)
parkinsons	3.172 (± 1.372)	2.843 (± 1.244)	2.6731 (± 1.453)	3.1172 (± 1.419)	3.1946 (± 1.498)
student performance	3.0398 (± 1.368)	2.9226 (± 1.411)	2.6548 (± 1.403)	3.0935 (± 1.323)	3.2892 (± 1.487)
wine quality	3.1742 (± 1.43)	2.7323 (± 1.456)	3.1581 (± 1.418)	3.0419 (± 1.356)	2.8935 (± 1.362)
avg rank	2.9972 (± 1.389)	2.9585 (± 1.426)	2.9744 (± 1.418)	2.9951 (± 1.404)	3.0745 (± 1.43)
normalised avg rank	4	1	2	3	5

also be observed that the largest replay window size yielded the worst performance overall and also for the most datasets. This suggests that heuristic selection is more likely to benefit from short, recent observations in the training process. From these observations it can be concluded that past performances of the heuristics are not always beneficial to the training process, and that the replay window size is problem specific. Furthermore, it can be concluded that some problems require more exploration of the heuristic space, while other problem sets require more exploitation of low-level heuristics.

Figure 8.29 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the performance of different replay window size configurations per dataset.

Figure 8.30 provides an illustration of the overall critical difference plots for ranked performance for each replay window size configuration as it relates to all datasets, across all independent runs and epochs.

Figure 8.30 shows no overall statistically significant difference in results for the various replay window size configurations as it relates to all datasets.

Similar to before, Figure 8.31 provides the train and test loss and accuracy plots for an example classification dataset (diabetic) as it relates to the replay window size experimental group. As before, the illustrations are provided in log scale and

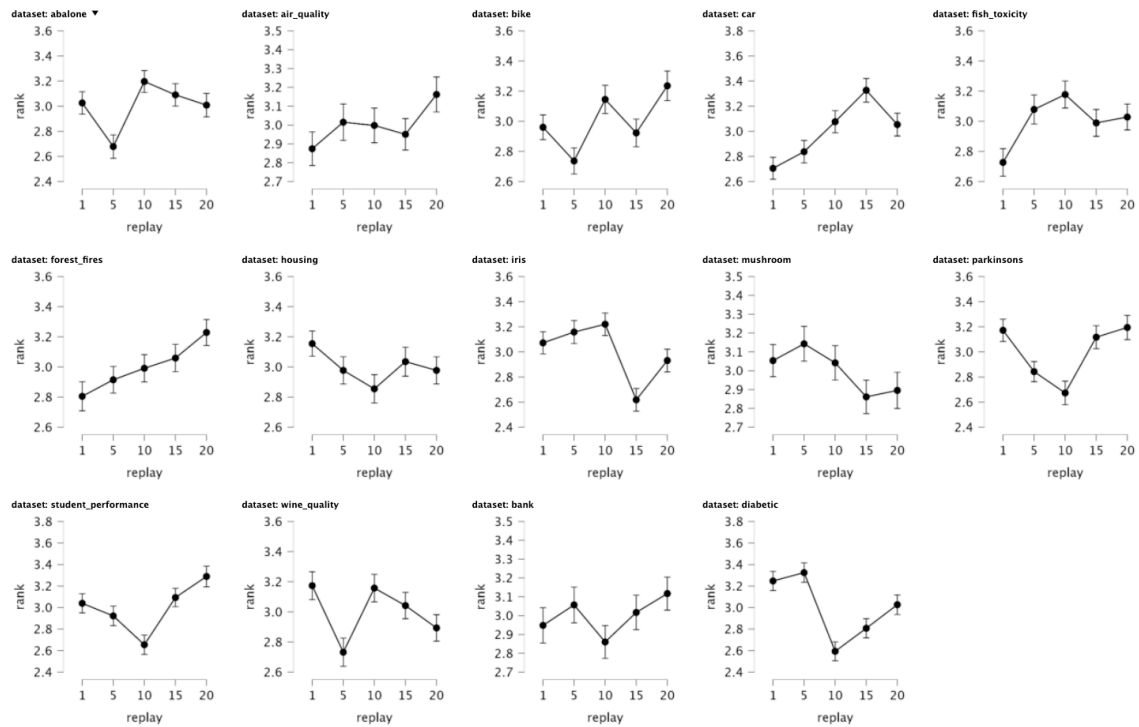


Figure 8.29: Descriptive plots for the average ranks of the BHH with varying replay window sizes per dataset, across all independent runs and epochs.

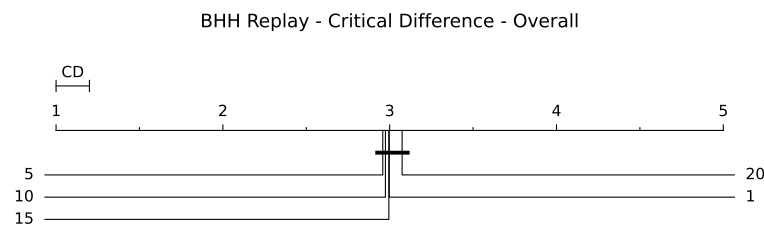


Figure 8.30: Critical difference plots for the average ranks of the BHH with varying replay window sizes across all datasets, runs and epochs.

illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

Figure 8.32 provides the train and test loss plots for an example regression dataset (fish toxicity) as it relates to the replay window size experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

The divergence that can be seen in Figure 8.31a and 8.31b is explained in previous sections and is simply a result of the BHH considering new heuristics that yield sub-optimal results, when an optimal solution has already been found. As mentioned before, an early stopping strategy as well as a move-acceptance strategy can be incorporated to avoid this effect.

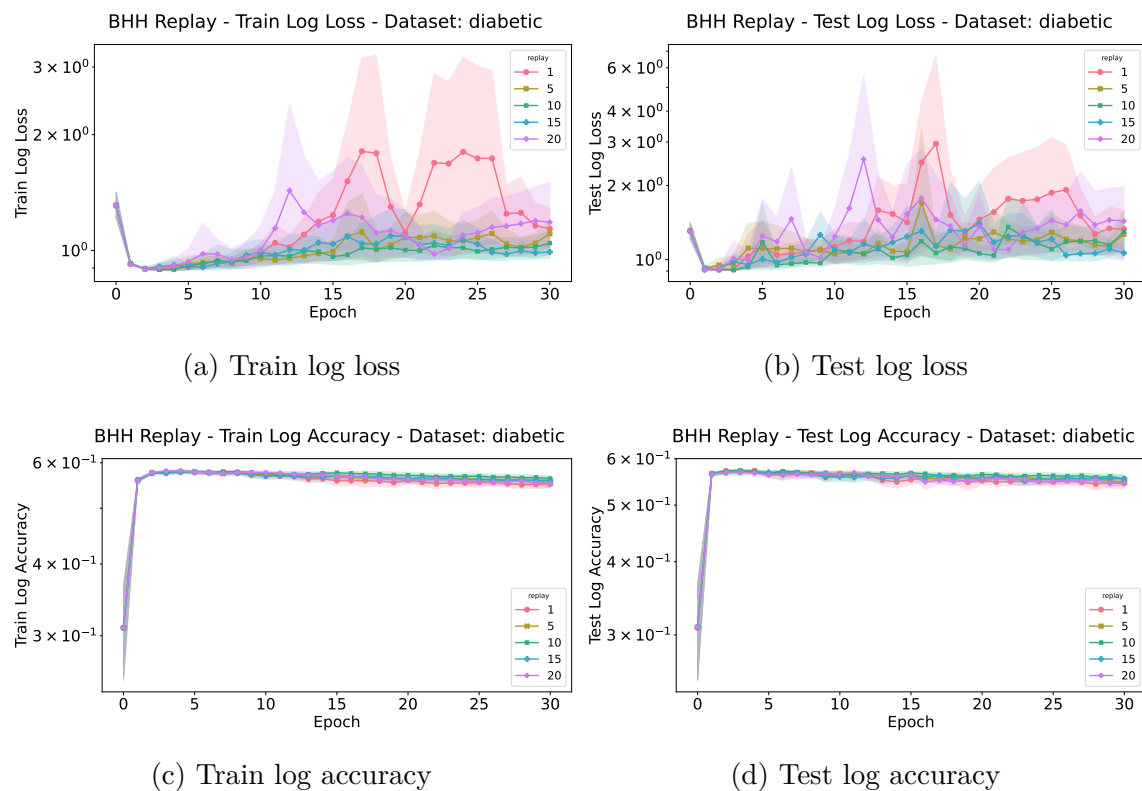


Figure 8.31: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the replay window size hyper-parameter on the diabetic dataset over 30 epochs, illustrated in log scale.

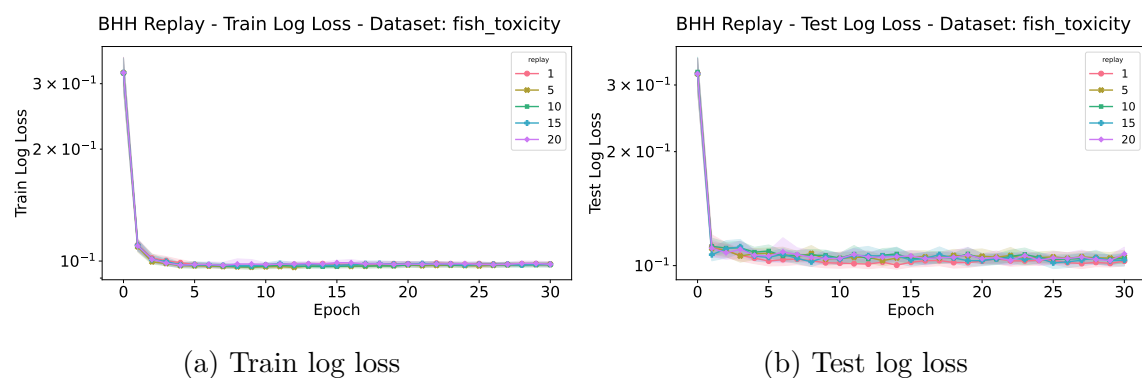


Figure 8.32: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations of the replay window size hyper-parameter on the fish toxicity dataset over 30 epochs, illustrated in log scale.

8.9 Reanalysis Interval

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *reanalysis interval* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, five different reanalysis intervals are considered. These include reanalysis intervals of 1, 5, 10, 15, and 20. Experiments are denoted as such.

As before, Table 8.13 presents the empirical results for this experimental group, showing the average test loss and statistics for all the reanalysis interval variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.13: Empirical results showing average test loss and statistics for different reanalysis intervals used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Reanalysis - Average Test Loss				
	1	5	10	15	20
abalone	2.0571 (± 0.102)	2.0424 (± 0.071)	2.0587 (± 0.088)	2.0879 (± 0.226)	2.0456 (± 0.079)
air quality	0.2682 (± 0.014)	0.2637 (± 0.012)	0.2729 (± 0.016)	0.2718 (± 0.019)	0.2651 (± 0.012)
bank	0.2599 (± 0.114)	0.2406 (± 0.023)	0.2456 (± 0.065)	0.254 (± 0.047)	0.2353 (± 0.025)
bike	0.0621 (± 0.016)	0.0627 (± 0.021)	0.0651 (± 0.02)	0.0663 (± 0.023)	0.0625 (± 0.02)
car	0.1597 (± 0.042)	0.1535 (± 0.038)	0.1572 (± 0.034)	0.1667 (± 0.036)	0.1644 (± 0.04)
diabetic	1.482 (± 1.563)	1.4074 (± 1.558)	1.2983 (± 0.66)	1.222 (± 0.6)	1.2944 (± 0.832)
fish toxicity	0.1031 (± 0.01)	0.1033 (± 0.008)	0.1046 (± 0.008)	0.1062 (± 0.009)	0.1004 (± 0.009)
forest fires	0.1083 (± 0.1)	0.1003 (± 0.069)	0.081 (± 0.069)	0.0927 (± 0.069)	0.0822 (± 0.059)
housing	0.0998 (± 0.017)	0.0953 (± 0.017)	0.0941 (± 0.017)	0.0963 (± 0.029)	0.0962 (± 0.011)
iris	0.582 (± 2.084)	0.164 (± 0.154)	0.2411 (± 0.295)	0.1218 (± 0.11)	0.2674 (± 0.545)
mushroom	0.0548 (± 0.241)	0.1165 (± 0.597)	0.0052 (± 0.012)	0.0344 (± 0.102)	0.0181 (± 0.059)
parkinsons	0.0597 (± 0.003)	0.0593 (± 0.002)	0.0587 (± 0.003)	0.06 (± 0.004)	0.0597 (± 0.003)
student performance	0.2331 (± 0.083)	0.2297 (± 0.09)	0.2359 (± 0.102)	0.2315 (± 0.063)	0.2274 (± 0.098)
wine quality	1.0728 (± 0.026)	1.0818 (± 0.028)	1.0827 (± 0.026)	1.0747 (± 0.026)	1.0909 (± 0.042)

Table 8.14 provides the average ranked performance, per dataset, for each of the BHH reanalysis interval configurations. Similar to before, the performance rank is calculated as the average rank produced by the reanalysis interval configurations, for all datasets, over all independent runs and epochs. The overall normalised average ranks are provided at the bottom of the table.

Tables 8.13 and 8.14 show that there is no clear overall best performing reanalysis interval configuration of the BHH for all datasets.

It should be noted that the configuration and results for the reanalysis interval configurations should be considered along with the reselection interval configurations as reselection is based on the outcome of the reanalysis step. Furthermore, the

Table 8.14: Empirical results showing average rank and statistics for different reanalysis intervals used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Reanalysis - Average Rank				
	1	5	10	15	20
abalone	2.9849 (± 1.397)	3.0699 (± 1.442)	3.171 (± 1.32)	2.9441 (± 1.445)	2.8301 (± 1.443)
air quality	2.9613 (± 1.344)	2.8387 (± 1.438)	3.2559 (± 1.402)	3.0538 (± 1.42)	2.8903 (± 1.43)
bank	3.1161 (± 1.432)	3.0452 (± 1.383)	2.7355 (± 1.398)	2.928 (± 1.403)	3.1753 (± 1.415)
bike	3.1409 (± 1.258)	2.7581 (± 1.391)	3.1946 (± 1.365)	3.0978 (± 1.498)	2.8086 (± 1.491)
car	3.1237 (± 1.423)	2.7237 (± 1.358)	2.8763 (± 1.299)	3.1538 (± 1.385)	3.1226 (± 1.547)
diabetic	3.2452 (± 1.305)	3.072 (± 1.391)	2.7785 (± 1.407)	3.029 (± 1.425)	2.8753 (± 1.494)
fish toxicity	3.1452 (± 1.362)	3.0032 (± 1.445)	3.2172 (± 1.364)	3.1 (± 1.409)	2.5344 (± 1.388)
forest fires	2.8398 (± 1.369)	3.1215 (± 1.471)	3.0043 (± 1.318)	2.886 (± 1.354)	3.1484 (± 1.526)
housing	3.3129 (± 1.311)	2.9892 (± 1.393)	2.6871 (± 1.483)	2.6968 (± 1.496)	3.314 (± 1.236)
iris	3.1204 (± 1.301)	3.0968 (± 1.391)	3.1269 (± 1.432)	2.6366 (± 1.449)	3.0194 (± 1.435)
mushroom	2.9312 (± 1.379)	3.1538 (± 1.403)	2.8473 (± 1.384)	3.1215 (± 1.453)	2.9032 (± 1.471)
parkinsons	2.9688 (± 1.385)	3.0086 (± 1.356)	2.6129 (± 1.481)	3.1032 (± 1.38)	3.3065 (± 1.378)
student performance	2.9409 (± 1.412)	3.1849 (± 1.38)	2.771 (± 1.41)	3.1258 (± 1.366)	2.9774 (± 1.467)
wine quality	2.8032 (± 1.444)	2.9204 (± 1.304)	3.2022 (± 1.41)	2.9247 (± 1.349)	3.1495 (± 1.517)
avg rank	3.0453 (± 1.374)	2.999 (± 1.403)	2.9629 (± 1.409)	2.9858 (± 1.425)	3.0039 (± 1.462)
normalised avg rank	5	3	1	2	4

reanalysis interval configurations should also be considered along with the replay window size, as the replay window size determines the number of performance log samples to reanalyse. For the BHH baseline configuration, the reselection and reanalysis interval hyper-parameters have default values of 10, and the replay window size also has a default value of 10. These values are deliberately chosen so that there is a balance between the different hyper-parameters from which empirical conclusions can be made.

Furthermore, the smaller the reanalysis interval, the more frequently the BHH resets concentration parameters. The smaller the replay window size, the less performance samples in the performance log are considered for reanalysis. A replay window size of one yields a case where only the last executed step is considered for reanalysis and thus provides a short-sighted view on the relevance of past performance. As such, reselection is based only on the most recent performance of each selected heuristic. However, if the reanalysis interval is larger than the replay window size, the BHH does not update its beliefs fast enough, and as a result, heuristic performance evidence is lost.

Figure 8.33 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the performance of different reanalysis interval configurations, per dataset.

Figure 8.34 provides an illustration of the overall critical difference plots for

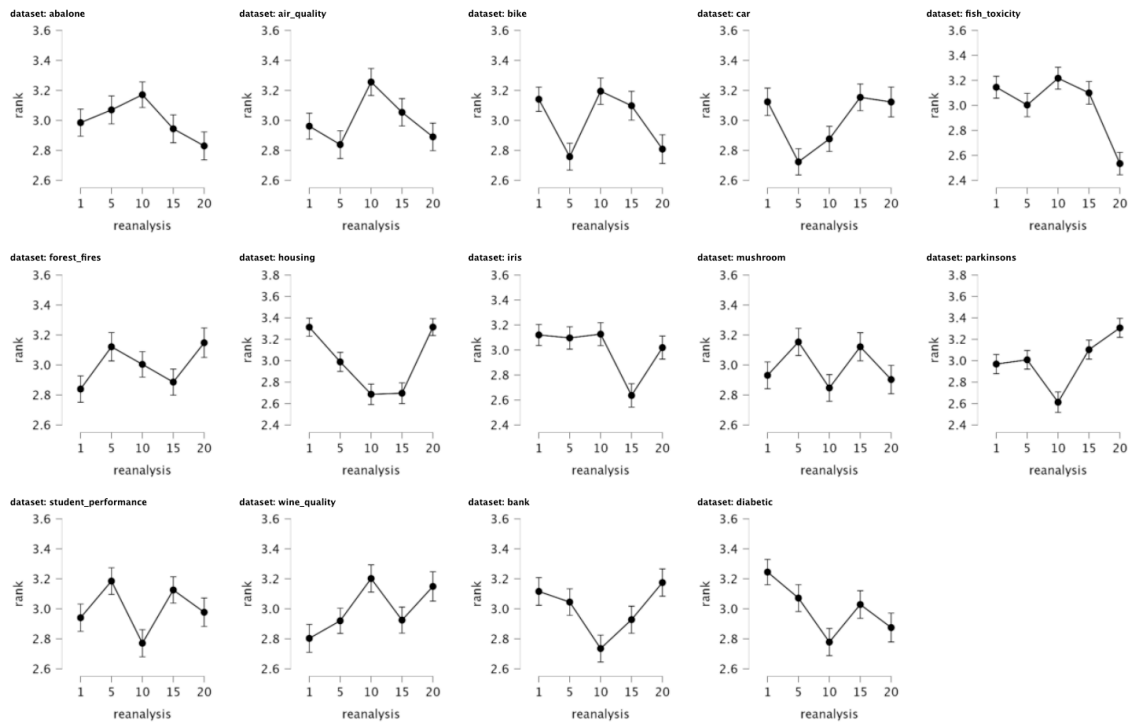


Figure 8.33: Descriptive plots for the average ranks of the BHH with varying reanalysis intervals per dataset, across all independent runs and epochs.

ranked performance for each reanalysis interval configuration as it relates to all datasets, across all independent runs and epochs. From Figure 8.34, it can be concluded that the best reanalysis interval is problem specific.

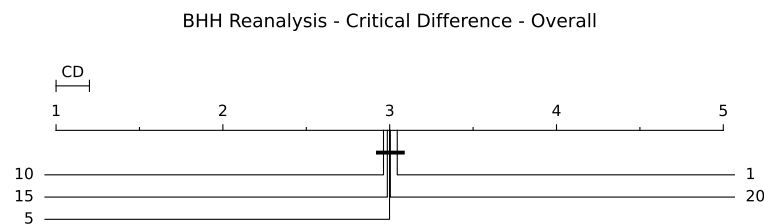


Figure 8.34: Critical difference plots for the average ranks of the BHH with varying reanalysis intervals across all datasets, runs and epochs.

Figure 8.34 shows no overall statistically significant difference in results for the various reanalysis interval configurations as it relates to all datasets.

Similar to before, Figure 8.35 provides the train and test loss and accuracy plots for an example classification dataset (mushroom) as it relates to the reanalysis interval experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

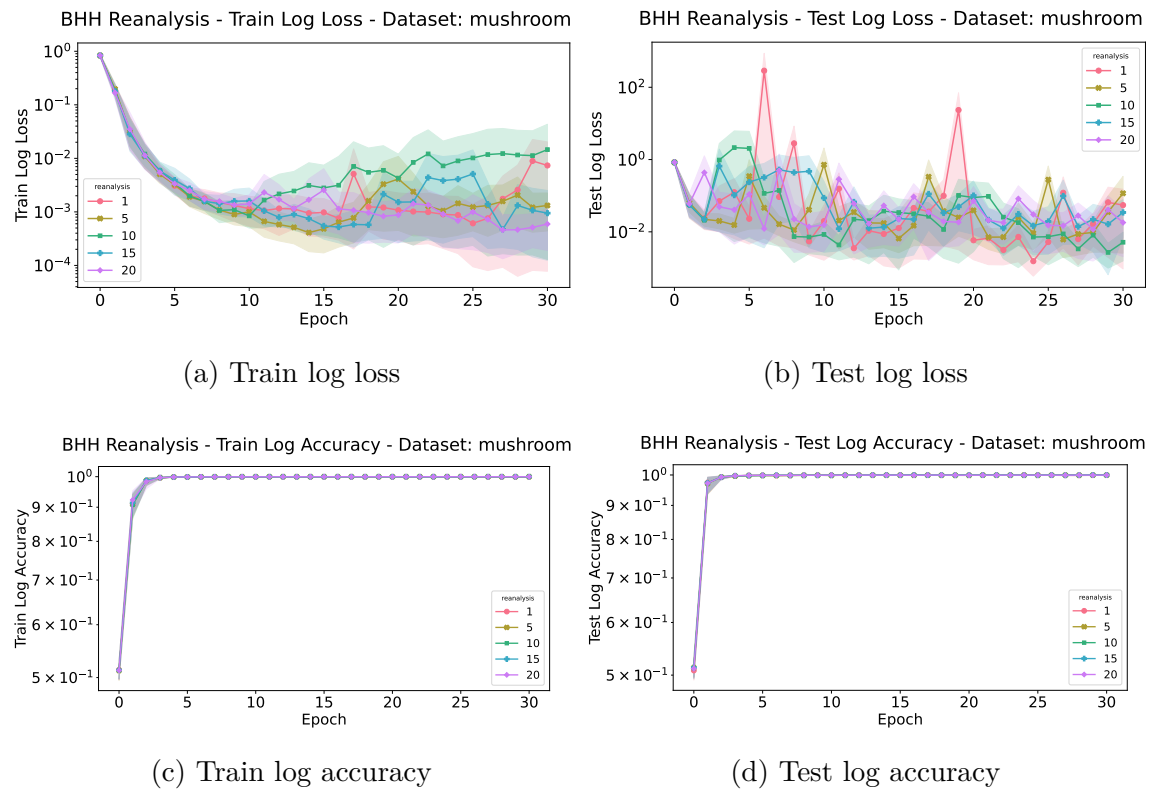


Figure 8.35: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the reanalysis interval hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.

Figure 8.36 provides the train and test loss plots for an example regression dataset (parkinsons) as it relates to the reanalysis interval experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

The divergence that can be seen in Figure 8.35a and 8.35b is explained in previous sections and is simply a result of the BHH considering new heuristics that yield sub-optimal results when an optimal solution has already been found. As mentioned before, an early stopping strategy as well as a move-acceptance strategy can be incorporated to avoid this effect.

8.10 Burn In

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *burn in* window size hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. Five different burn in window sizes are considered. These include burn in

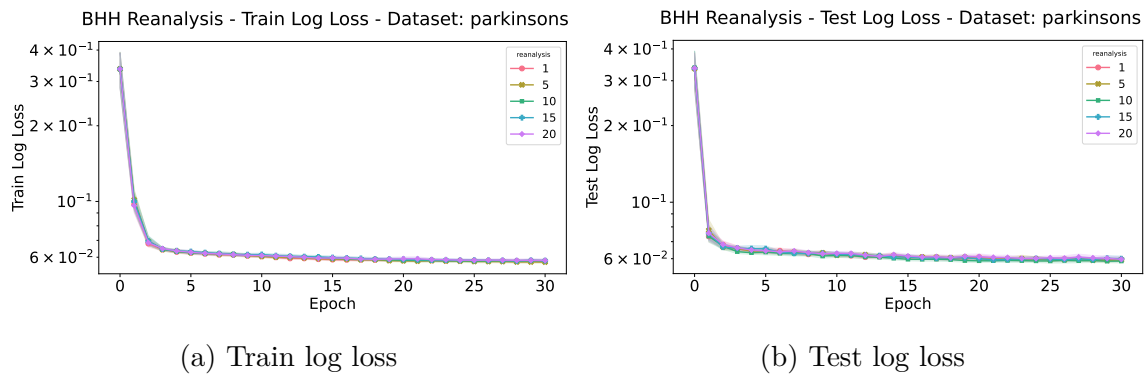


Figure 8.36: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the reanalysis interval hyper-parameter on the parkinsons dataset over 30 epochs, illustrated in log scale.

window sizes of 0, 5, 10, 15, and 20. Experiments are denoted as such.

As a reminder, the burn in window size is a hyper-parameter that is borrowed from the MCMC algorithm, and provides a window whereby exploration is encouraged in order to obtain sufficient samples of heuristic performance evidence for statistical inference.

As before, Table 8.15 presents the empirical results for this experimental group, showing the average test loss and statistics for all the burn in window size variants of the BHH that was implemented. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.15: Empirical results showing average test loss and statistics for different burn in window sizes used by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Burn In - Average Test Loss				
	0	5	10	15	20
abalone	2.0587 (+0.088)	2.0583 (+0.181)	2.0961 (+0.168)	2.1454 (+0.365)	2.6753 (+2.18)
air quality	0.2729 (+0.016)	0.2719 (+0.02)	0.2688 (+0.014)	0.2784 (+0.017)	0.2851 (+0.029)
bank	0.2456 (+0.065)	0.2373 (+0.033)	0.2514 (+0.109)	0.2427 (+0.033)	0.2703 (+0.056)
bike	0.0651 (+0.02)	0.0678 (+0.024)	0.0929 (+0.047)	0.1109 (+0.043)	0.1383 (+0.034)
car	0.1572 (+0.034)	0.164 (+0.039)	0.1745 (+0.037)	0.1883 (+0.067)	0.1803 (+0.039)
diabetic	1.2983 (+0.66)	1.1216 (+0.582)	1.4701 (+1.161)	1.9918 (+3.645)	2.2996 (+3.589)
fish toxicity	0.1046 (+0.008)	0.1026 (+0.009)	0.1009 (+0.01)	0.1043 (+0.007)	0.1052 (+0.009)
forest fires	0.081 (+0.069)	0.0772 (+0.057)	0.0714 (+0.045)	0.1136 (+0.059)	0.1044 (+0.093)
housing	0.0941 (+0.017)	0.096 (+0.015)	0.1036 (+0.037)	0.0997 (+0.021)	0.1116 (+0.034)
iris	0.2411 (+0.295)	0.1472 (+0.134)	0.2717 (+0.462)	0.2291 (+0.376)	0.1842 (+0.189)
mushroom	0.0052 (+0.012)	0.1068 (+0.3)	0.0262 (+0.057)	0.1075 (+0.256)	0.0612 (+0.108)
parkinsons	0.0587 (+0.003)	0.0599 (+0.003)	0.0594 (+0.003)	0.0644 (+0.013)	0.0676 (+0.015)
student performance	0.2359 (+0.102)	0.2801 (+0.127)	0.314 (+0.137)	0.3623 (+0.139)	0.3988 (+0.133)
wine quality	1.0827 (+0.026)	1.0791 (+0.021)	1.079 (+0.03)	1.086 (+0.029)	1.0897 (+0.034)

From Table 8.15 a noticeable pattern can be observed. Generally, it is found that a small burn in window size produces better results than larger burn in window sizes. The largest burn in window size of 20 produced the worst results in nine out of thirteen datasets. Some exceptions include the mushroom and the iris datasets. Notice that burn in window sizes smaller than the default reanalysis interval, reselection interval and replay window size of 10, generally produced the best results. However, Table 8.15 does not provide a clear view of the overall performance produced by different burn in window size configurations for the entire training process.

In order to obtain a better view of the overall performance produced by different burn in window size configurations for the entire training process, Table 8.16 provides the average ranked performance, per dataset, for each of the BHH burn in configurations. Similar to before, the performance rank is calculated as the average rank produced by the burn in window size configurations, for all datasets, over all independent runs and epochs. The overall normalised average rank is provided at the bottom of the table.

Table 8.16: Empirical results showing average rank and statistics for different burn in window sizes used by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Burn In - Average Rank				
	0	5	10	15	20
abalone	2.9151 (± 1.395)	2.6935 (± 1.393)	2.9172 (± 1.384)	2.8398 (± 1.352)	3.6344 (± 1.357)
air quality	2.9097 (± 1.441)	2.6817 (± 1.427)	2.7699 (± 1.344)	3.372 (± 1.314)	3.2667 (± 1.412)
bank	2.7301 (± 1.321)	2.8194 (± 1.335)	2.857 (± 1.485)	2.9312 (± 1.384)	3.6624 (± 1.339)
bike	1.9237 (± 0.981)	2.1806 (± 1.024)	2.9237 (± 1.274)	3.5054 (± 1.186)	4.4667 (± 0.851)
car	2.3624 (± 1.231)	2.5882 (± 1.425)	3.2978 (± 1.392)	3.3796 (± 1.271)	3.372 (± 1.401)
diabetic	2.4054 (± 1.358)	2.7011 (± 1.358)	3.2398 (± 1.372)	3.2323 (± 1.326)	3.4215 (± 1.396)
fish toxicity	3.0634 (± 1.459)	2.7935 (± 1.402)	2.8204 (± 1.513)	3.1022 (± 1.346)	3.2204 (± 1.295)
forest fires	2.8161 (± 1.33)	2.8495 (± 1.383)	2.8645 (± 1.409)	3.3774 (± 1.398)	3.0925 (± 1.471)
housing	2.6903 (± 1.384)	2.9516 (± 1.318)	2.9968 (± 1.442)	3.0581 (± 1.344)	3.3032 (± 1.51)
iris	2.9473 (± 1.461)	2.972 (± 1.367)	3.1903 (± 1.449)	2.771 (± 1.369)	3.1194 (± 1.387)
mushroom	2.2065 (± 1.249)	2.9376 (± 1.298)	3.1613 (± 1.297)	3.1441 (± 1.401)	3.5226 (± 1.501)
parkinsons	2.2796 (± 1.282)	2.8892 (± 1.247)	2.7065 (± 1.314)	3.3968 (± 1.417)	3.728 (± 1.33)
student performance	2.1452 (± 1.205)	2.6946 (± 1.403)	3.0968 (± 1.352)	3.4581 (± 1.354)	3.6054 (± 1.231)
wine quality	2.9699 (± 1.437)	3.0108 (± 1.304)	2.6634 (± 1.419)	3.1247 (± 1.398)	3.2312 (± 1.447)
avg rank	2.5975 (± 1.376)	2.7688 (± 1.353)	2.9647 (± 1.404)	3.1923 (± 1.367)	3.4747 (± 1.402)
normalised avg rank	1	2	3	4	5

Table 8.16 shows that, in general, a configuration where there is no burn in produces the overall best ranked performance across all datasets. Furthermore, it can be observed that performance degrades with larger burn in window sizes. This can be explained from the fact that most of the training progress that is made, is

done early in the training process. The faster the BHH is able to start learning and exploiting heuristics and solutions, the better.

Note that the BHH is configured with a reselection interval of one, until the burn in window has been exhausted, after which the actual reselection interval applies. This means that reselection occurs at each step during the burn in window, but the reanalysis interval and replay window size remain intact. This observation agrees with previous results, where the BHH prefers to start exploiting heuristics and solutions as soon as possible. Furthermore, this agrees with previous results, where large reselection intervals are found to produce the best results, giving heuristics sufficient time to progress and smooth out training steps.

Recall that the gradient-based heuristics generally produce the best results overall. Most of the gradient-based heuristics included in the heuristic pool implement some form of momentum and exponentially average state parameters. When reselection occurs frequently in succession, these gradient-based heuristics struggle to produce smooth update steps. As a result, the BHH can produce delayed convergence and thus leads to worse performance when considering the entire training process.

Figure 8.37 provides an illustration of the descriptive plots for the different BHH configurations as it relates to the burn in window size configurations, per dataset. Figure 8.37 shows that there is mostly a linear correlation between performance results and burn in window sizes, with minor exceptions to some datasets.

Figure 8.38 provides an illustration of the overall critical difference plots for ranked performance for each burn in window size configuration as it relates to all datasets, across all independent runs and epochs.

Figure 8.38 shows with statistical certainty that a small burn in configuration of the BHH generally produce overall better results than larger burn in configurations, as it relates to all datasets.

Similar to before, Figure 8.39 provides the train and test loss and accuracy plots for an example classification dataset (mushroom) as it relates to the burn in window size experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

Figure 8.40 provides the train and test loss plots for an example regression dataset (parkinsons) as it relates to the burn in window size experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

The divergence that can be seen in Figures 8.39a and 8.39b is explained in previous sections and is simply a result of the BHH considering new heuristics that

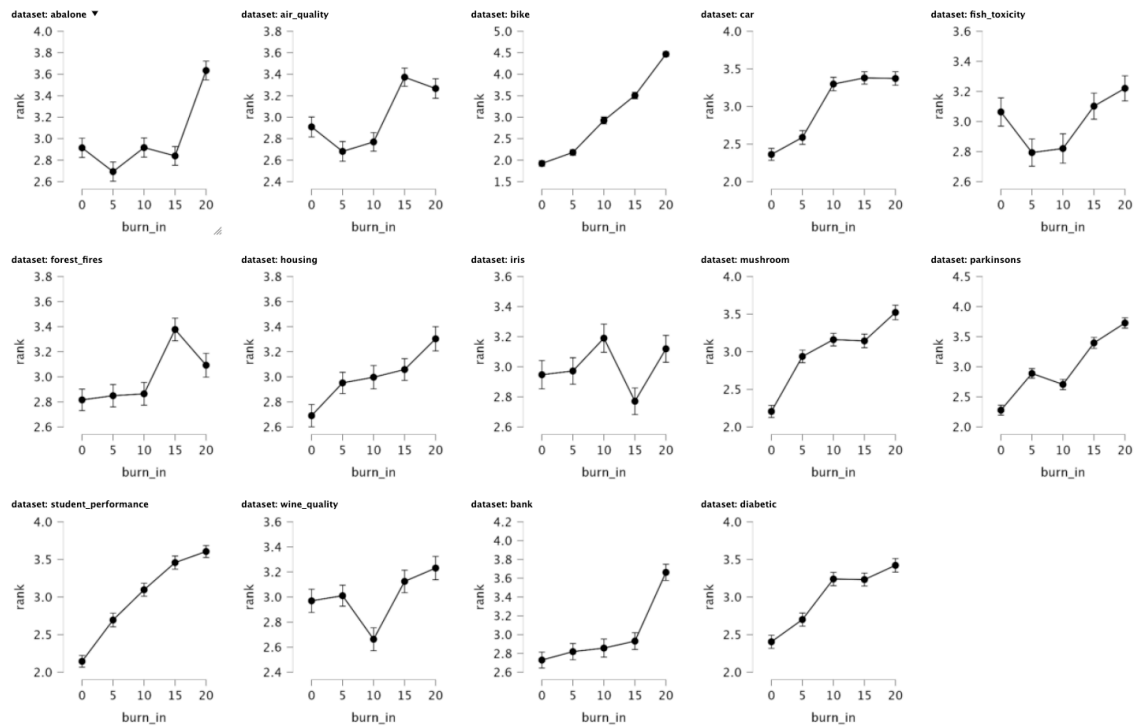


Figure 8.37: Descriptive plots for the average ranks of the BHH with varying burn in values per dataset, across all independent runs and epochs.

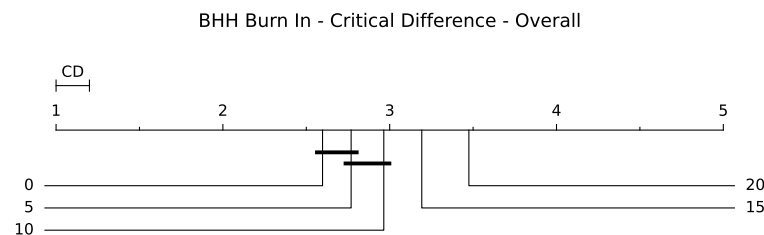


Figure 8.38: Critical difference plots for the average ranks of the BHH with varying burn in values across all datasets, runs and epochs.

yield sub-optimal results, when an optimal solution has already been found. As mentioned before, an early stopping strategy as well as a move-acceptance strategy can be incorporated to avoid this effect.

8.11 Normalisation

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *normalisation* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, the normalisation hyper-parameter is a flag that enables or disables normalisation of the concentration parameters in an attempt to provide

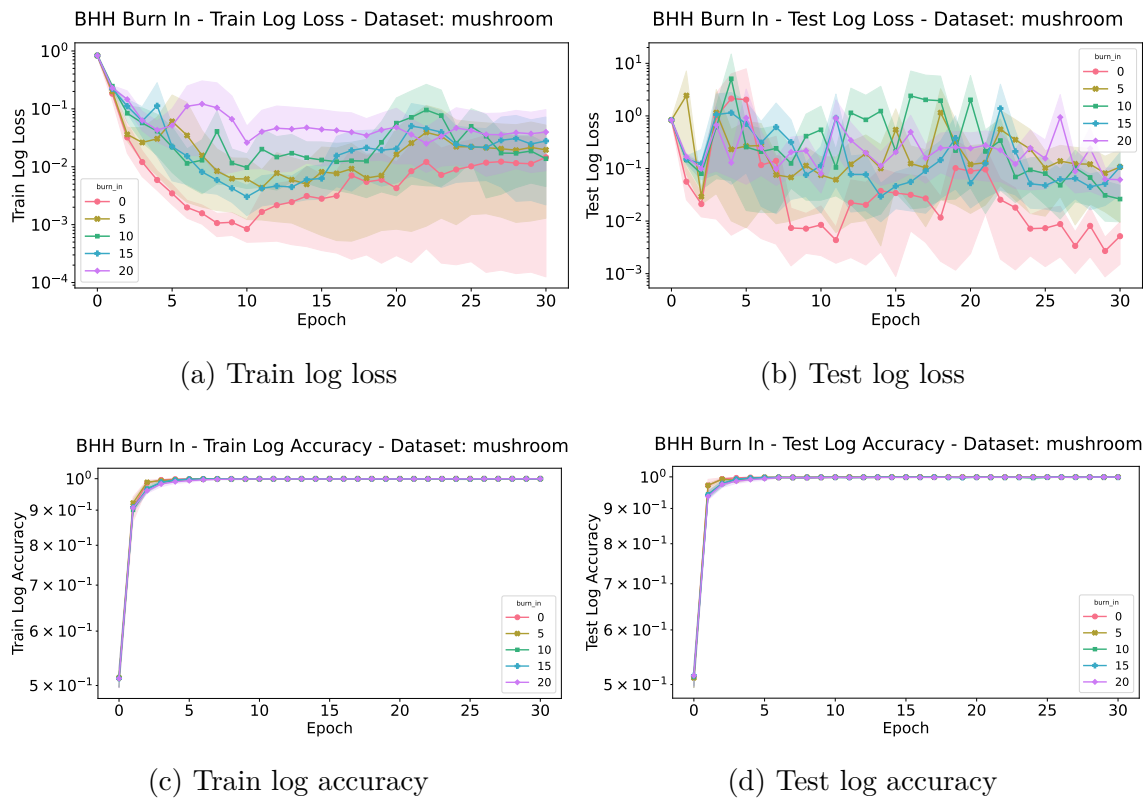


Figure 8.39: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the burn in window size hyper-parameter on the mushroom dataset over 30 epochs, illustrated in log scale.

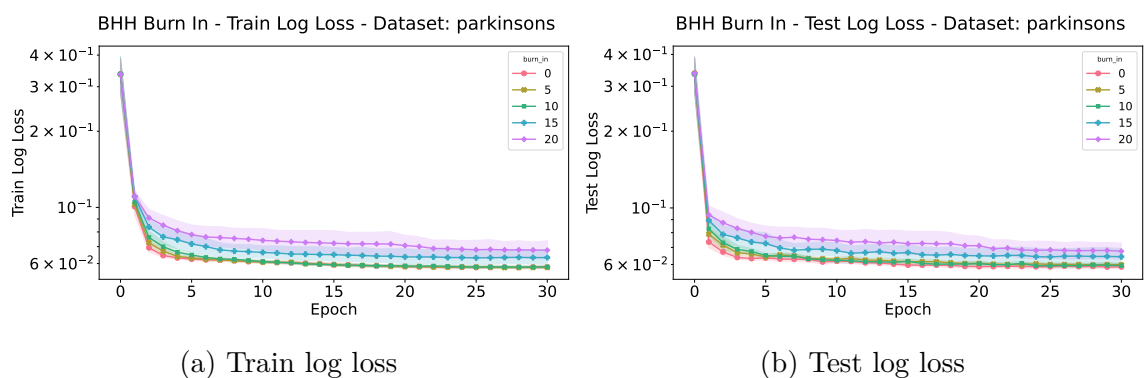


Figure 8.40: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with different configurations for the burn in window size hyper-parameter on the parkinsons dataset over 30 epochs, illustrated in log scale.

more exploration in the heuristic space. The experiments are denoted as *true* where normalisation is enabled, and *false* where normalisation is disabled.

As before, Table 8.17 presents the empirical results for this experimental group, showing the average test loss and statistics for variants of the BHH that was implemented where normalisation is enabled and disabled. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.17: Empirical results showing average test loss and statistics for normalisation toggled by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Normalisation - Average Test Loss	
	false	true
abalone	2.0587 (± 0.088)	2.0936 (± 0.231)
air quality	0.2729 (± 0.016)	0.2704 (± 0.017)
bank	0.2456 (± 0.065)	0.2374 (± 0.042)
bike	0.0651 (± 0.02)	0.0689 (± 0.03)
car	0.1572 (± 0.034)	0.1721 (± 0.038)
diabetic	1.2983 (± 0.66)	1.5051 (± 1.93)
fish toxicity	0.1046 (± 0.008)	0.1021 (± 0.009)
forest fires	0.081 (± 0.069)	0.0868 (± 0.069)
housing	0.0941 (± 0.017)	0.0931 (± 0.019)
iris	0.2411 (± 0.295)	0.2358 (± 0.333)
mushroom	0.0052 (± 0.012)	0.0455 (± 0.216)
parkinsons	0.0587 (± 0.003)	0.0598 (± 0.004)
student performance	0.2359 (± 0.102)	0.2522 (± 0.098)
wine quality	1.0827 (± 0.026)	1.0831 (± 0.045)

Table 8.18 provides the average ranked performance, per dataset, with normalisation enabled and disabled. Similar to before, the performance rank is calculated as the average rank produced by the normalisation configurations, for all datasets, over all independent runs and epochs. The overall normalised average ranks are provided at the bottom of the table.

Table 8.18 shows that, in general, there is no obvious difference in the outcomes of the BHH when normalisation is enabled or disabled. This suggests that the normalisation flag is problem specific. Since the normalisation hyper-parameter provides a mechanism for the BHH to explore more in the heuristic space, with the intent of finding better solutions, it can be concluded that some datasets require more exploration, while others require more exploitation. However, a better suggestion can be made from the observation of previous results. The BHH has a default replay window size of 10, yielding a small window of memory to maintain heuristic performance evidence in the performance log. Normalisation of the concentration

Table 8.18: Empirical results showing average rank and statistics for normalisation toggled by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Normalisation - Average Rank	
	false	true
abalone	1.4935 (± 0.5)	1.5065 (± 0.5)
air quality	1.4892 (± 0.5)	1.5108 (± 0.5)
bank	1.5484 (± 0.498)	1.4516 (± 0.498)
bike	1.4978 (± 0.5)	1.5022 (± 0.5)
car	1.4258 (± 0.495)	1.5742 (± 0.495)
diabetic	1.4753 (± 0.5)	1.5247 (± 0.5)
fish toxicity	1.5505 (± 0.498)	1.4495 (± 0.498)
forest fires	1.5054 (± 0.5)	1.4946 (± 0.5)
housing	1.472 (± 0.499)	1.528 (± 0.499)
iris	1.5645 (± 0.496)	1.4355 (± 0.496)
mushroom	1.4323 (± 0.496)	1.5667 (± 0.496)
parkinsons	1.4 (± 0.49)	1.6 (± 0.49)
student performance	1.3892 (± 0.488)	1.6108 (± 0.488)
wine quality	1.5473 (± 0.498)	1.4527 (± 0.498)
avg rank	1.4851 (± 0.5)	1.5148 (± 0.5)
normalised avg rank	1	2

parameters then yield an insignificant difference in the outcome as the concentration parameters never reach high values. Furthermore, previous results have shown that the best results are obtained from exploitation of heuristics and solutions, in comparison to exploration.

Figure 8.41 provides an illustration of the descriptive plots for the BHH configurations with the normalisation hyper-parameter enabled and disabled, per dataset.

Figure 8.42 provides an illustration of the overall critical difference plots for ranked performance for normalisation configurations that are enabled and disabled, as it relates to all datasets, across all independent runs and epochs.

Figure 8.42 shows no statistical difference in the outcomes of the normalisation configurations of the BHH. This further supports the suggestion that the use of the normalisation hyper-parameter is problem specific or that the effects of the normalisation hyper-parameter is not realised for small replay window sizes.

Similar to before, Figure 8.43 provides the train and test loss and accuracy plots for an example classification dataset (abalone) as it relates to the normalisation experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss and accuracy plots for the other classification datasets are left out for brevity as they yield similar illustrations.

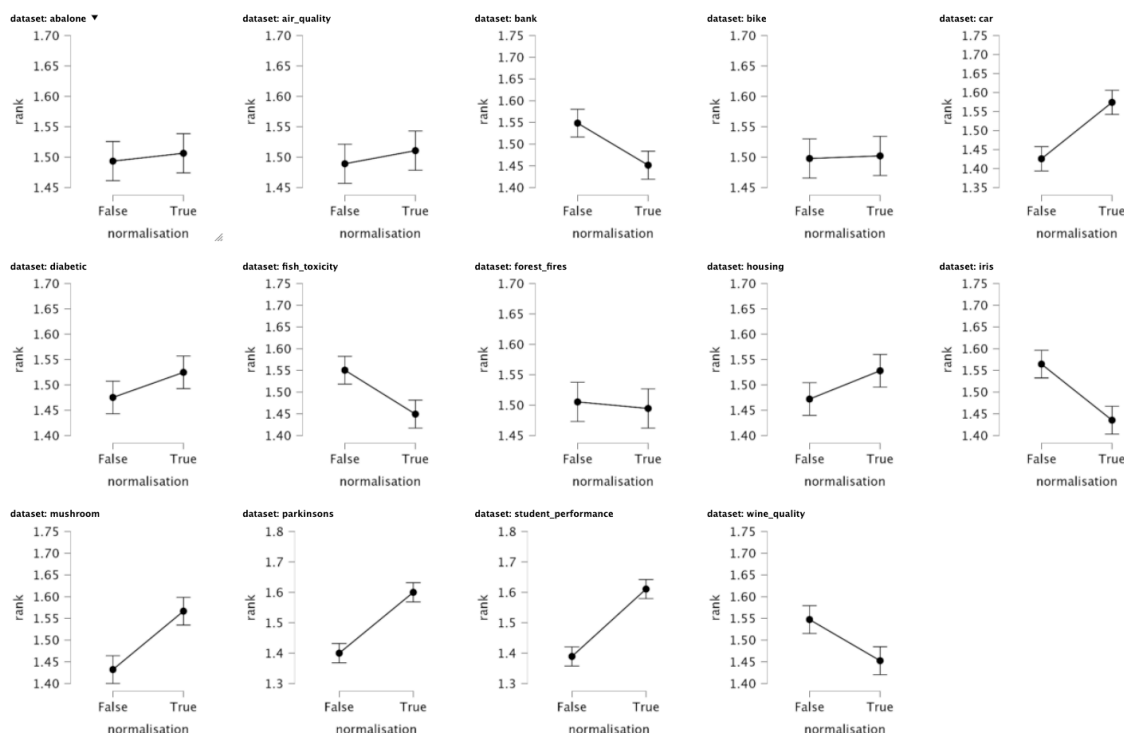


Figure 8.41: Descriptive plots for the average ranks of the BHH with normalisation toggled per dataset, across all independent runs and epochs.

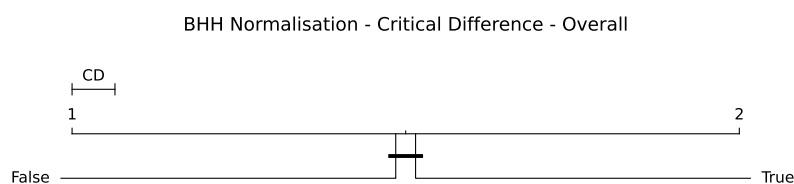


Figure 8.42: Critical difference plots for the average ranks of the BHH with normalisation toggled across all datasets, runs and epochs.

Figure 8.44 provides the train and test loss plots for an example regression dataset (bike) as it relates to the normalisation experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for the other regression datasets are left out for brevity.

8.12 Discounted Rewards

This section provides the empirical results for the experimental group that compares the performance of different variants of the BHH as it relates to the *discounted rewards* hyper-parameter. Brief discussions follow and illustrations are provided for visual aid. As a reminder, the discounted rewards hyper-parameter is a flag

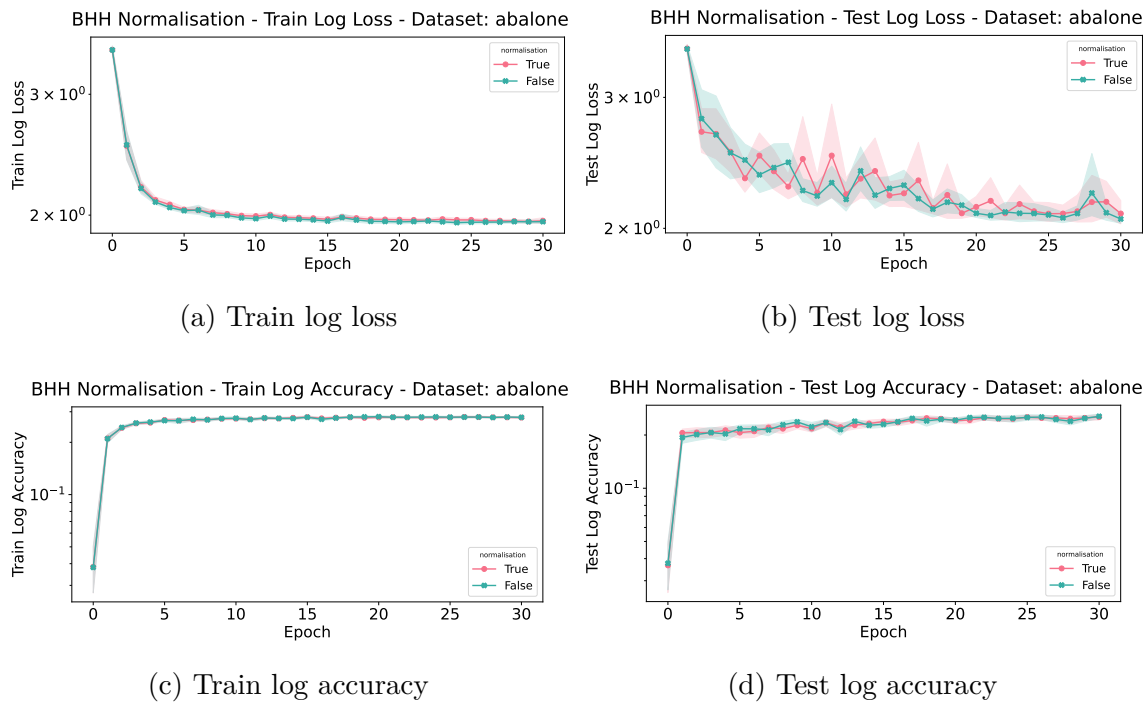


Figure 8.43: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the normalisation hyper-parameter toggled on the abalone dataset over 30 epochs, illustrated in log scale.

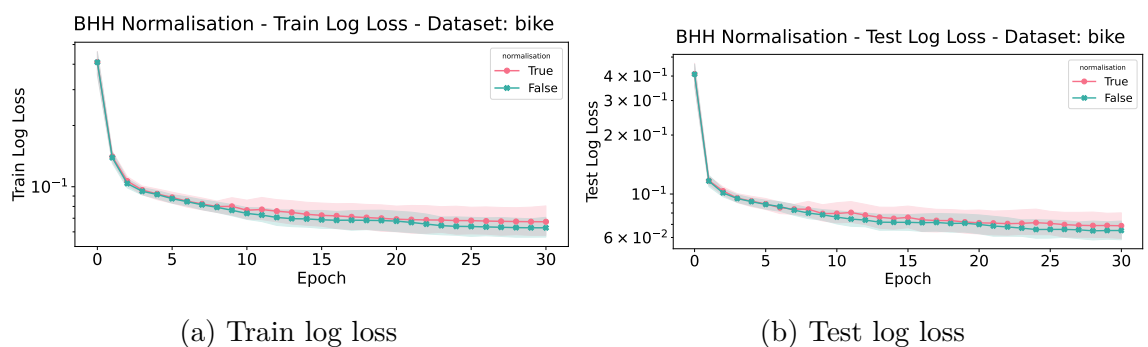


Figure 8.44: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the normalisation hyper-parameter toggled on the bike dataset over 30 epochs, illustrated in log scale.

that enables or disables discounted rewards of the pseudo counts that are added to concentration parameters in an attempt to provide control over the effects of past performances of the low-level heuristics.

As a reminder, the discounted rewards mechanism is implemented such that pseudo counts for past performances that achieved successful credit allocations are exponentially decayed into the past. The default discounted rewards decay rate is set to 0.5.

As before, Table 8.19 presents the empirical results for this experimental group, showing the average test loss and statistics for variants of the BHH that was implemented where discounted rewards is enabled and disabled. The test loss is measured at the last epoch for each dataset, over all independent runs.

Table 8.19: Empirical results showing average test loss and statistics for discounted rewards toggled by the BHH across multiple datasets, for all independent runs and is measured at the last epoch.

dataset	Discounted Rewards - Average Test Loss	
	false	true
abalone	2.0587 (± 0.088)	2.0789 (± 0.102)
air quality	0.2729 (± 0.016)	0.2693 (± 0.011)
bank	0.2456 (± 0.065)	0.2349 (± 0.041)
bike	0.0651 (± 0.02)	0.0676 (± 0.027)
car	0.1572 (± 0.034)	0.1672 (± 0.037)
diabetic	1.2983 (± 0.66)	1.3457 (± 1.419)
fish toxicity	0.1046 (± 0.008)	0.1053 (± 0.008)
forest fires	0.081 (± 0.069)	0.0928 (± 0.079)
housing	0.0941 (± 0.017)	0.0972 (± 0.017)
iris	0.2411 (± 0.295)	0.1833 (± 0.259)
mushroom	0.0052 (± 0.012)	0.0479 (± 0.203)
parkinsons	0.0587 (± 0.003)	0.06 (± 0.002)
student performance	0.2359 (± 0.102)	0.2408 (± 0.086)
wine quality	1.0827 (± 0.026)	1.0792 (± 0.039)

Table 8.20 provides the average ranked performance, per dataset, with discounted rewards enabled and disabled. Similar to before, the performance rank is calculated as the average rank produced by the discounted rewards configurations, for all datasets, over all independent runs and epochs.

Table 8.20 shows that, in general, there is no obvious difference in the outcomes of the BHH when discounted rewards is enabled or disabled. Although this could suggest that the discounted rewards hyper-parameter is problem specific, a better suggestion supports the findings from Section 8.11. These findings suggest that manipulation of the concentration parameters yield insignificant differences in the

Table 8.20: Empirical results showing average rank and statistics for discounted rewards toggled by the BHH across multiple datasets, for all independent runs and epochs.

dataset	Discounted Rewards - Average Rank	
	false	true
abalone	1.4914 (± 0.5)	1.5086 (± 0.5)
air quality	1.4613 (± 0.499)	1.5387 (± 0.499)
bank	1.5161 (± 0.5)	1.4839 (± 0.5)
bike	1.5011 (± 0.5)	1.4989 (± 0.5)
car	1.428 (± 0.495)	1.572 (± 0.495)
diabetic	1.4699 (± 0.499)	1.5301 (± 0.499)
fish toxicity	1.5011 (± 0.5)	1.4989 (± 0.5)
forest fires	1.4742 (± 0.5)	1.5258 (± 0.5)
housing	1.4237 (± 0.494)	1.5763 (± 0.494)
iris	1.5903 (± 0.492)	1.4097 (± 0.492)
mushroom	1.3946 (± 0.489)	1.5806 (± 0.494)
parkinsons	1.371 (± 0.483)	1.629 (± 0.483)
student performance	1.4398 (± 0.497)	1.5602 (± 0.497)
wine quality	1.557 (± 0.497)	1.443 (± 0.497)
avg rank	1.4728 (± 0.499)	1.5254 (± 0.499)
normalised avg rank	1	2

outcomes of performance, as a small replay window size of 10 is used by default.

In general, it is found that the online learning approach, followed by the BHH yields very little room to learn from. This is supported by the fact that most of the significant performance gains are made in the early stages of training after which exploitation of heuristics and solutions later in the training process is shown to produce the best results.

Figure 8.45 provides an illustration of the descriptive plots for the BHH configurations with the discounted rewards hyper-parameter enabled and disabled, per dataset.

Figure 8.46 provides an illustration of the overall critical difference plots for ranked performance for discounted rewards configurations that are enabled and disabled, as it relates to all datasets, across all independent runs and epochs.

Figure 8.46 shows no statistically significant difference in the outcomes of the discounted rewards configurations of the BHH. This further supports the suggestion that the effects of discounted rewards is not realised for small replay window sizes.

Similar to before, Figure 8.47 provides the train and test loss and accuracy plots for an example classification dataset (wine quality) as it relates to the discounted rewards experimental group. As before, the illustrations are provided in log scale

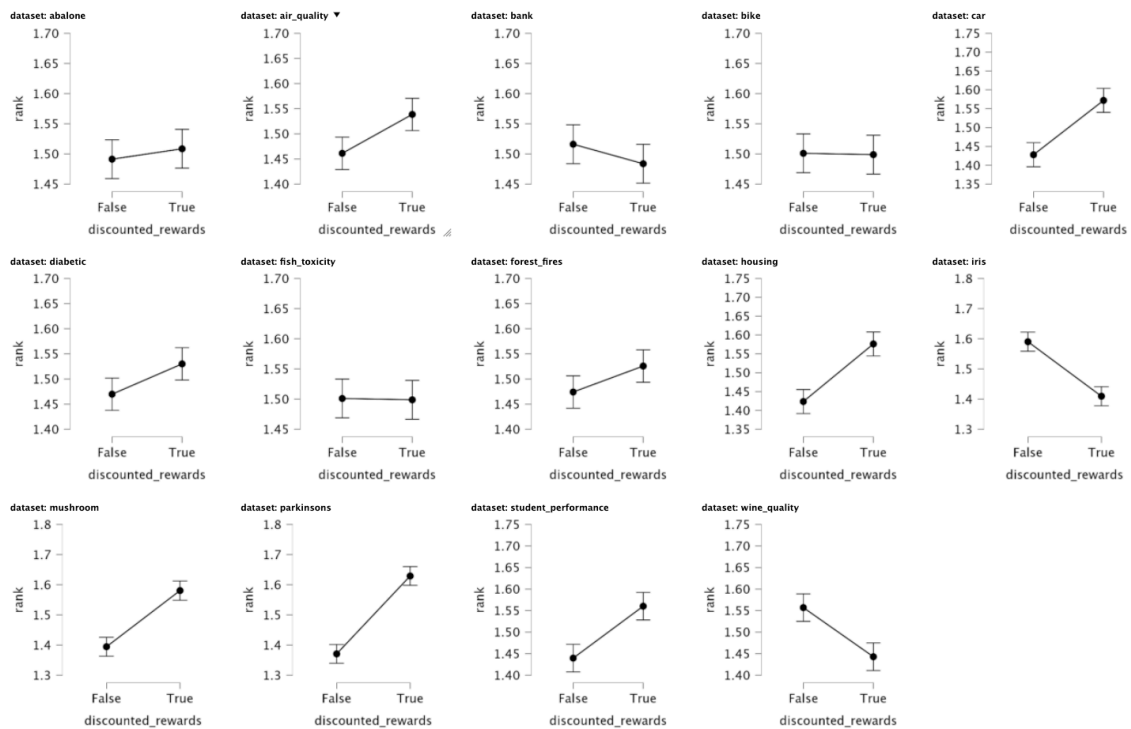


Figure 8.45: Descriptive plots for the average ranks of the BHH with discounted rewards toggled per datasets, across all independent runs and epochs.

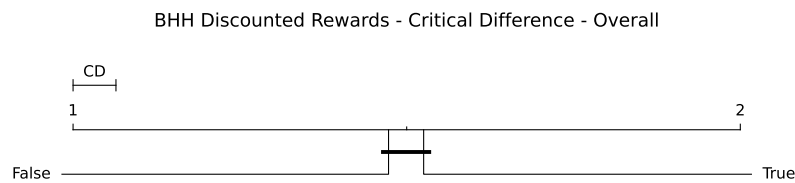


Figure 8.46: Critical difference plots for the average ranks of the BHH with discounted rewards toggled across all datasets, runs and epochs.

and illustrations of the train and test loss and accuracy plots for other datasets are left out for brevity as they yield similar illustrations.

Figure 8.48 provides the train and test loss plots for an example regression dataset (housing) as it relates to the discounted rewards experimental group. As before, the illustrations are provided in log scale and illustrations of the train and test loss plots for other regression datasets are left out for brevity.

8.13 Summary

This section provided the results of the empirical process that was described in Chapter 7. Three experimental groups were designed. These experimental groups include a case study on the behaviour of the BHH as it relates to an example dataset

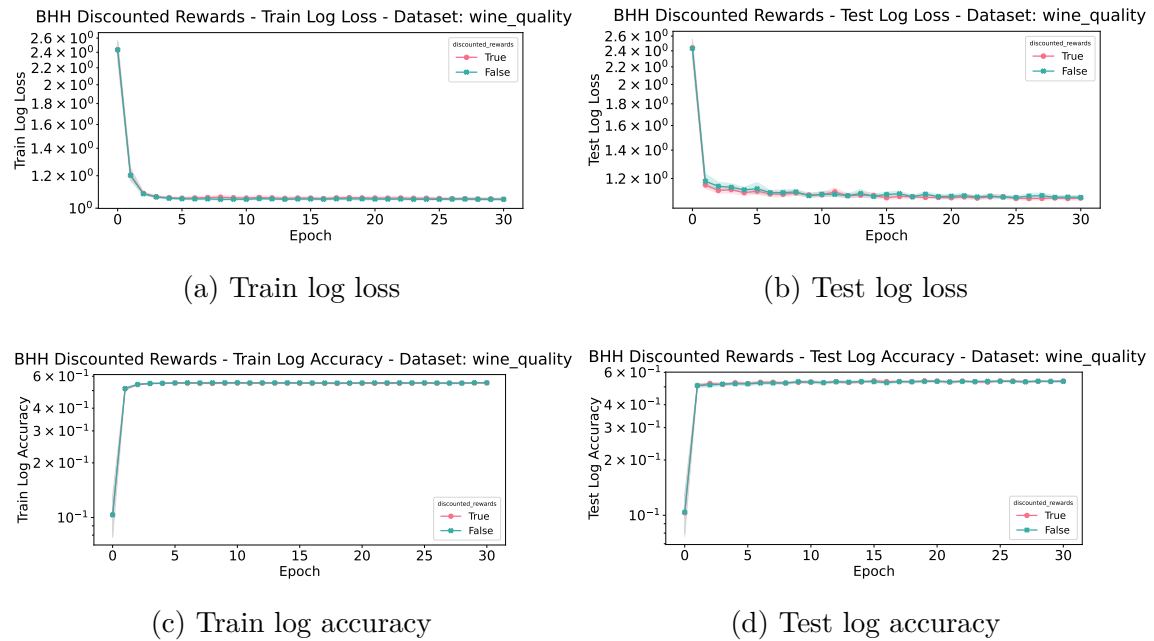


Figure 8.47: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the discounted rewards hyper-parameter toggled on the wine quality dataset over 30 epochs, illustrated in log scale.

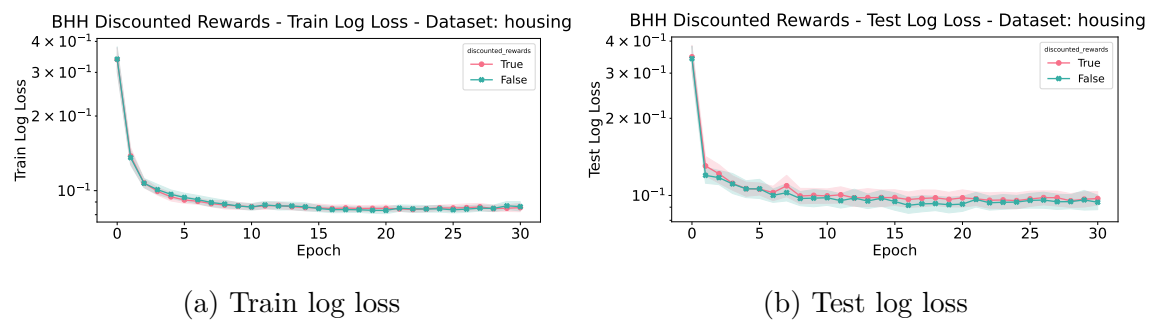


Figure 8.48: The train and test loss and accuracy plots for the experimental group comparing the performance of the BHH with the discounted rewards hyper-parameter toggled on the housing dataset over 30 epochs, illustrated in log scale.

(iris). Furthermore, an experimental group that compares the performance of the BHH to standalone, low-level heuristics was included, and finally, an experimental group that compares the effects of various hyper-parameters on the outcomes of the BHH was also included. The empirical results for each of these experimental groups were presented in detail.

In general, it was found that the BHH is able to successfully train the underlying FFNNs. Detailed analysis and discussions of the empirical results were provided along with illustrations for visual aid. Where possible, detailed discussions were presented that explain the reasons behind the outcomes that were produced. Throughout, various suggestions were made to improve on the implementations made in this dissertation.

Finally, all results that were presented were retrieved from statistical analysis which yield statistical certainty in the results and outcomes observed.

Chapter 9

Conclusion

“I am sorry to have made such a long speech, but I did not have time to make a shorter one.”

- Winston Churchill

This chapter provides a summary of the research that was done in this dissertation. The research objectives were set out in Chapter 1 and are concluded in this chapter. Furthermore, this chapter provides a summary of the results and findings from the empirical process and provides future research opportunities. The remainder of the chapter is structured as follows:

- **Section 9.1** provides a summary of the research intent and provides a review of the problem statement, objectives, and motivations behind the research.
- **Section 9.2** summarises the background information that was covered.
- **Section 9.3** provides a brief summary of the BHH and its implementation.
- **Section 9.4** provides a brief summary of the methodology and the empirical process that was followed.
- **Section 9.5** summarises the research findings. This section is broken down into the various experimental groups that were executed.
- **Section 9.6** provides suggestions for future research opportunities.
- **Section 9.7** provides a brief discussion on supporting material related to the research, as made available by the authors.
- **Section 9.8** provides a brief summary of the chapter.

9.1 Summary of Research Intent

This section briefly reviews the problem statement, motivations, and research objectives that was set out in Chapter 1 of this dissertation.

9.1.1 Review of Problem Statement

The main area of focus for the research done in this dissertation stems from the problem statement, which identified the difficult and tedious problem of selecting the best heuristic for training FFNNs. This process is often non-trivial and time-consuming. There is no easy way to know which heuristic to select to train FFNNs. Traditionally, an iterative approach of trial-and-error was followed. This process involves a tedious process of carefully considering and selecting candidate heuristics, whereby each candidate heuristic is then empirically tested and evaluated. This process is often executed at the expense of the researcher's resources.

9.1.2 Review of Research Motivation

The research presented in this dissertation identified the possibility of using a different approach, referred to as HHs, to automate the heuristic selection process. HHs are high-level search algorithms that search in the heuristic space, whereby low-level heuristics search in the solution space. HHs have been shown to provide a general solution to the heuristic selection process. The benefit that such an approach provides is that it automates the heuristic selection process. This approach saves researchers time and potentially produce a solution that could generalise to multiple problems.

Investigation was done into the landscapes of existing solutions to train FFNNs as well as HHs. Though there exists many applications where HHs are used for optimisation, there are little to no published work on using HHs to train FFNNs. The only work that was found include the work by Nel [115].

9.1.3 Review of Research Objectives

Existing techniques from other problem domains were considered. RL, meta-learning and probabilistic learning approached were considered. Bayesian probability theory showed promise. This research then set out to develop a novel high-level heuristic that utilises probability theory in an online learning setting to drive the automatic heuristic selection process. As such, the BHH was conceptualised. The research objectives were defined and were addressed as follows:

- A literature study was conducted on all related topics that are relevant to the development of the BHH and is provided in Chapters 2 to 5. The literature study included background information on ANN, existing low-level heuristics, meta-learning, HHs, probability theory, and Bayesian statistics. Detailed discussions were provided on every topic and was supported by visual illustrations and mathematical derivations.
- A novel BHH that can be used to automate the heuristic selection process was proposed in detail and was implemented. The BHH follows an approach that includes a selective and perturbative element in an online learning setting. Bayesian statistics and MAP is used as the optimisation technique and detailed mathematical derivations of the optimisation process was provided. Detailed discussions were provided on the mapping of proxied heuristic state update operations. Finally, a BHH baseline configuration was defined as a cornerstone reference for empirical analysis.
- An empirical process was designed and the detail of the implementation for the empirical process and the BHH was provided.
- An empirical study was conducted to show that the BHH is able to train FFNNs effectively on a number of different datasets.
- An empirical study was conducted to study the behavioural characteristics of the BHH.
- An empirical study was conducted to critically evaluate the performance of the BHH compared to traditional, low-level, standalone heuristic when training FFNNs on a number of different datasets.
- An empirical study was conducted to evaluate variants of the BHH to study the impact of various hyper-parameters on the outcomes of the BHH.
- Results from the empirical process were statistically analysed over multiple runs and provided statistical significance. These results were discussed in detail and was supported through visual illustrations.

The research objectives set out in this dissertation were extensive. Each of these research objectives has been successfully executed as indicated in the list above. From the findings made in this dissertation, many different opportunities for future research were identified.

9.2 Summary of Background Information

Chapters 2 to 5 provided the reader with the relevant background information that is necessary to develop the novel BHH. Chapter 2 provided background information on ANNs. All the components that make up the AN were provided and is followed by discussions on FFNNs. Chapter 3 provided background information on various low-level heuristics that are designed to train FFNNs. A number of gradient-based heuristics and MHs were presented in detail. Chapter 4 provided background information and literature reviews on HHs. Meta-learning is discussed and a HH classification scheme, developed by Burke et al. [19], was discussed in detail. Chapter 5 provided background information and mathematical derivations on probability theory and Bayesian statistics. Various probability distributions were provided along with their conjugate priors. MLE and MAP were presented in detail along with mathematical breakdowns.

From the aforementioned chapters, it can be concluded that the necessary background information was provided as set out in the research objectives of this dissertation.

9.3 Summary of The Bayesian Hyper-Heuristic

The main contribution from this dissertation is the development of a novel *Bayesian hyper-heuristic* (BHH). Chapter 6 provided the details on the concept and implementation of the BHH. Various design decisions and hyper-parameters that could affect the outcomes of the BHH were presented.

It was shown that the BHH is a population-based, meta-hyper-heuristic that utilises selection and perturbation of low-level heuristics in an online learning fashion. Heuristics are selected for each entity in a population of entities that each implement a candidate solution to the underlying FFNN. The BHH uses a Bayesian probabilistic model to drive the heuristic selection process. MAP was derived for the probabilistic model and is used as the mechanism by which the BHH is optimised, yielding the learning capability of the BHH.

Furthermore, it is shown that heuristic selection alone is not sufficient and that a mechanism of transition between heuristics is needed as they are selected for each entity in the population. Entity and heuristic state and state-manipulation formed the cornerstone of this topic. Perturbation of heuristics and solutions are achieved through a technique that deconstructs heuristics into their initialisation and update steps. The update operations for each heuristic were derived as movement equations

borrowed from the field of physics. This technique allows for the proxying of update step operations as they are required by different heuristics.

Detailed discussions were provided on a number of hyper-parameters that each contribute to the trade-off between heuristic and solution exploration and exploitation. These hyper-parameters include the *heuristic pool* configuration, the *population size*, the *credit assignment strategy* used, the *reselection interval*, the *replay window size*, the *reanalysis interval*, the *burn in window size*, *normalisation* of concentration parameters and *discounted rewards* as assigned by the credit assignment strategy. A BHH baseline configuration was identified with default values for each of the hyper-parameters.

It was mentioned that the BHH, implemented in this dissertation, does not implement a move-acceptance strategy that rejects heuristic progressions if they do not improve on the current best solution. Throughout the presentation of empirical results, it is recommended that this technique be investigated and implemented in further research.

9.4 Summary of Methodology

The research objectives defined a number of empirical tests to be executed. Each one of the empirical tests is referred to as an experimental group. Various experimental groups have been identified and are listed below:

- An experimental group that provides a case-study on the behaviour of the BHH as it relates to training of a FFNN on an example dataset (iris), in order to understand the workings of the BHH, and to determine if the BHH is able to successfully train FFNNs. Three different configurations of the BHH configuration were implemented. This includes the default BHH baseline configuration, the BHH baseline configuration where the replay window size is set to 250, and the BHH baseline configuration which utilises the *symmetric* credit assignment strategy.
- An experimental group that provides a comparative analysis between the performance of the BHH baseline configuration and low-level, standalone heuristics when training FFNNs on a number of datasets.
- An experimental group that provides a comparative analysis between variants of the BHH that investigate the effects of various hyper-parameters on the outcomes of the BHH. These variants include:

- The BHH baseline configuration with different heuristic pool configurations. Three different heuristic pool configurations were implemented. These configurations include a configuration with all the low-level heuristics included in the heuristic pool, a configuration with only gradient-based heuristics in the heuristic pool, and a configuration with only MHs in the heuristic pool.
- The BHH baseline configuration with different population sizes. Five different population sizes were implemented including 5, 10, 15, 20, and 25.
- The BHH baseline configuration with different credit assignment strategies. Five different credit assignment strategies were implemented including the *ibest*, *pbest*, *rbest*, *gbest*, and *symmetric* credit assignment strategies.
- The BHH baseline configuration with different reselection intervals. Five different reselection intervals were implemented including 1, 5, 10, 15, and 20.
- The BHH baseline configuration with different reanalysis window sizes. Five different reanalysis intervals were implemented including 1, 5, 10, 15, and 20.
- The BHH baseline configuration with different replay window sizes. Five different replay window sizes were implemented including 1, 5, 10, 15, and 20.
- The BHH baseline configuration with different burn in window sizes. Five different burn in window sizes were implemented including 0, 5, 10, 15, and 20.
- The BHH baseline configuration where normalisation of pseudo counts for concentration parameters, as derived from credit allocations in the performance log, are enabled and disabled.
- The BHH baseline configuration where discounted rewards of pseudo counts for concentration parameters, as derived from credit allocations in the performance log, are enabled and disabled.

Each experimental group was executed against a set of fourteen datasets, consisting of regression and classification problems, split between uni- and multimodal problems. Each experimental group was repeated over 30 runs for statistical certainty. The statistical analysis process was discussed and the ANOVA, post hoc Tukey,

independent T-test and Kruskal-Wallis statistical tests formed part of the statistical tests that were conducted.

All experimental groups were run for a maximum of 30 epochs in order to allow for the studying of the BHH even after training has converged and overfitting has occurred. The maximum epoch strategy yielded a training process that is longer than what was empirically found to be necessary. This also meant that no early stopping strategy was used. Future research opportunities should incorporate an early stopping strategy of the training process.

9.5 Summary of Results

This section provides a summary of the results obtained from each of the experimental groups that were executed. This section is structured as follows:

- **Section 9.5.1** provides a summary of the results for the experimental group that executed a case study on the behaviour of the BHH on an example dataset.
- **Section 9.5.2** provides a summary of the results for the experimental group that compares the performance of the BHH baseline configuration with low-level, standalone heuristics, across all datasets.
- **Section 9.5.3** provides a summary of the results for the experimental group that investigates the effects of the *heuristic pool* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.4** provides a summary of the results for the experimental group that investigates the effects of the *population size* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.5** provides a summary of the results for the experimental group that investigates the effects of the *credit assignment strategy* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.6** provides a summary of the results for the experimental group that investigates the effects of the *reselection interval* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.7** provides a summary of the results for the experimental group that investigates the effects of the *replay window size* hyper-parameter on the BHH, across all datasets.

- **Section 9.5.8** provides a summary of the results for the experimental group that investigates the effects of the *reanalysis interval* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.9** provides a summary of the results for the experimental group that investigates the effects of the *burn in window size* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.10** provides a summary of the results for the experimental group that investigates the effects of the *normalisation* hyper-parameter on the BHH, across all datasets.
- **Section 9.5.11** provides a summary of the results for the experimental group that investigates the effects of the *discounted rewards* hyper-parameter on the BHH, across all datasets.

9.5.1 Behavioural Case Study

For this experimental group it was found that the BHH is able to train a *feedforward neural network* (FFNN) relatively well. Detailed analysis was provided for various state parameters maintained by the BHH. These include the concentration parameters for different heuristics, the probability distribution of heuristic selection probabilities, prior heuristic selection probabilities and posterior heuristic selection probabilities. It was shown that the BHH is able to learn and that the BHH is able to exploit small performance biases as it relates to heuristic selection. This enables the BHH to select the correct heuristics to apply to the correct entities at the correct time in the training process.

Various illustrations of the train and test loss plots, as it relates to training of a FFNN on the iris dataset, have been provided. Furthermore, various illustrations on the changes of the BHH state parameters were provided for visual aid.

The main findings that were made for this experimental group are given as follows: Most of the training progression is made in the early stages of the training process. That leads to the conclusion that the BHH has a small window from which it should learn and gain the most in the training process. After training has converged, the BHH resets its concentration parameters and heuristic selection returns to the symmetric heuristic selection case. As such, the BHH explores other heuristics in an attempt to further improve on the current best solution found. The test set was used as a validation set during training. Some overfitting can be observed as the BHH tried finding better solutions on the train set, but at the cost of generalisation on the

test set. Minor divergence of the training loss is observed as the BHH explores other heuristics in an attempt to improve performance. Since no move-acceptance strategy, and no early stopping strategy was used, the BHH could select heuristics that are sub-optimal. Future research opportunities should incorporate these aforementioned strategies.

9.5.2 BHH Baseline vs. Low-Level Heuristics

For this experimental group, three different configurations of the BHH baseline configuration were implemented. These configurations vary in the type of heuristics in the heuristic pool. The *bhh_all* configuration included all low-level heuristics in the heuristic pool. The *bhh_gd* configuration included only gradient-based heuristics in the heuristic pool, and the *bhh_mh* configuration included only MHs in the heuristic pool.

Overall, the *bhh_gd* configuration performed the best out of the BHH variants, achieving an overall rank of fourth amongst thirteen heuristics that were implemented and executed on fourteen datasets. The *bhh_gd* configuration produced performance results close to that of the best low-level heuristics and was only statistically outperformed by the top two low-level heuristics. The *bhh_all* configuration achieved an overall rank of sixth and the *bhh_mh* achieved an overall rank of eighth.

Although the *bhh_gd* configuration produced performance results comparable to the best low-level heuristics, the *bhh_all* and *bhh_mh* configurations produced average results. It was found that, in general, gradient-based heuristics produced the best results, as such, it is understandable that the *bhh_gd* yielded the best performance outcomes between the different BHH variants that were implemented. Although the BHH variants were not able to produce better results than the top low-level heuristics, the BHH variants still effectively trained the underlying FFNNs and produced good training outcomes overall.

It was shown that the *bhh_gd* configuration produced the lowest variance in rank between datasets out of all of the heuristics implemented, giving the BHH the ability to generalise well to other problems.

Furthermore, it was shown that the BHH provides a mechanism whereby prior expert knowledge can be injected, before training starts. Future research can exploit this knowledge and provide a significant bias towards heuristics that are known to perform well on particular problem types.

9.5.3 Heuristic pool

For this experimental group, the same three heuristic pool variants as outlined for the [BHH](#) behavioural case study, was implemented. These variants are denoted as *all*, *gd*, and *mh*. This experimental group then provided an opportunity to look at the effects of the heuristic pool hyper-parameter in more detail.

As with the experimental group that compares the performance of the [BHH](#) baseline configuration with low-level, standalone heuristics, it was found that the *gd* configuration performed significantly better than the *all* and *mh* configurations overall, across all datasets.

The benefits of using meta-heuristics in the heuristic pool was not realised in this dissertation, since the gradient-based low-level heuristics, as well as the *gd* configuration produced the best overall performance, across all datasets.

9.5.4 Population Size

For this experimental group, five different population sizes were evaluated. These included population sizes of 5, 10, 15, 20, and 25. It was found that lower population sizes yielded better results on the majority of datasets, while only some datasets prefer larger population sizes. However, overall it was shown that the population size hyper-parameter is problem specific. Illustrations of the train and test loss and accuracy plots for some example datasets were provided. Some divergence in training loss was observed and was attributed to selection of heuristics that yield sub-optimal results after an optimal solution has already been found. As before, a move-acceptance strategy and an early stopping strategy can be incorporated to avoid this effect.

9.5.5 Credit Assignment Strategy

For this experimental group, five different credit assignment strategies were evaluated. These included the *ibest*, *pbest*, *rbest*, *gbest*, and *symmetric* credit assignment strategies. No clear overall difference in performance was observed for the various credit assignment strategies. For the majority of cases, a particular non-symmetric credit assignment strategy yielded the best performance, while the *symmetric* credit assignment strategy yielded the best performance results in one case. It can therefore be concluded that the credit assignment strategy is problem specific. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.6 Reselection Interval

For this experimental group, five different reselection intervals were evaluated. These included reselection intervals of 1, 5, 10, 15, and 20. Overall, with statistical significance, it was found that a larger reselection interval is generally preferred. It was found that larger reselection intervals lead to longer time frames for the low-level heuristics to progress, smoothing out update steps. Larger reselection intervals also allow the BHH to collect more samples of heuristic performance evidence from which it can learn. Therefore, a conclusion that can be made is that heuristic and solution exploitation is preferred over exploration. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.7 Replay

For this experimental group, five different replay window sizes were evaluated. These included replay window sizes of 1, 5, 10, 15, and 20. Overall, no statistically significant difference was found for the performance of the replay window size configurations, and thus, it was concluded that the replay window size is problem specific, with some datasets preferring short memory and other preferring long memory. The relationship between the reselection interval, replay window size and reanalysis interval was discussed, and it was shown that these hyper-parameters should be considered together. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.8 Reanalysis Interval

For this experimental group, five different reanalysis intervals were evaluated. These included reanalysis intervals of 1, 5, 10, 15, and 20. Overall, no statistically significant difference was found for the performance of the reanalysis intervals across all datasets, and thus, it was concluded that the reanalysis interval is problem specific. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.9 Burn In

For this experimental group, five different burn in window sizes were evaluated. These included burn in window sizes of 1, 5, 10, 15, and 20. Overall, it was found with statistical significance, that lower burn in window sizes are generally preferred. The lower burn in window sizes generally produced the best results in the majority

of cases, with some exceptions that prefer average burn in window sizes. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.10 Normalisation

For this experimental group an empirical test was conducted to determine the effects of normalisation of pseudo counts allocated by the credit assignment strategy in the performance log. A configuration is included with normalisation enabled and a configuration is included with normalisation disabled. It was concluded that the default replay window size of 10 is too small to yield a significant difference in performance outcome for when normalisation is enabled compared to when normalisation is disabled. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.5.11 Discounted Rewards

For this experimental group an empirical test was conducted to determine the effects of discounted rewards of pseudo counts allocated by the credit assignment strategy in the performance log. A configuration is included with discounted rewards enabled and a configuration is included with discounted rewards disabled. Similar to the findings for the experimental group investigating the effects of the normalisation hyper-parameter, it was concluded that the default replay window size of 10 is too small to yield a significant difference in performance outcome for when discounted rewards is enabled compared to when discounted rewards is disabled. As before, illustrations of the train and test loss and accuracy plots for some example datasets were provided.

9.6 Future Research Opportunities

Throughout this dissertation, many different opportunities have been identified for future research. This section contains a summary of each of these topics.

A Priori Biases

By default, the [BHH](#) initialises the values for the concentration parameters, related to each heuristic in the heuristic pool, symmetrically with an initial value of 1.0. This yields an initial probability distribution of the heuristic selection probabilities that is uniform. Symmetric initialisation contains no a priori heuristic selection bias

and assumes uniform sampling at first. It has been identified that expert knowledge can be incorporated into the BHH by carefully providing the initial concentrations for the each heuristic in the heuristic pool. Such research would then evaluate if such prior heuristic selection biases have a positive impact on the outcomes of the BHH.

Move-Acceptance Strategies

Throughout the presentation of the empirical results, it is mentioned that a move-acceptance strategy can be incorporated to reject heuristic progressions that do not improve on the current best solution found. This would eliminate any divergence in the training process as a result of selecting sub-optimal heuristics. Various different move-acceptance strategies can be developed and empirically tested.

Early Stopping Strategies

Similar to the move-acceptance strategy, throughout the presentation of the empirical results, it is mentioned that an early stopping strategy can be used to halt the training process if the performance does not improve for a number of steps. In this dissertation, a number of examples were presented where divergence of training loss occurred. Furthermore, a number of examples were presented where overfitting occurred. An early stopping strategy can help prevent such outcomes.

Entity Search

For this dissertation, even though the predictive model implemented by the BHH includes entity consideration, it does not sample from an entity pool. The BHH keeps the entire entity pool fixed, and just reselects heuristics for each entity. The BHH, as is, can be used to learn which entities to select as well. The hypothesis is that this would result in better combinations of entities and heuristics, which should have a positive effect on the outcomes of the BHH.

Continuous-Valued Credit Allocation

Credit assignment in the context of this dissertation yields a discrete credit allocation value. More specifically, the credit allocation is binary, yielding a zero or a one. Future research can investigate different credit allocation strategies that yield continuous valued outcomes. This should provide more fine-grained indicators of heuristic performance which the BHH can exploit easier.

Credit Search

Similar to entity search, the [BHH](#), as is, can be used to learn which credit assignment strategies to use. This can then be combined with heuristic and entity selection so that the correct heuristic is selected for the correct entity, by means of the correct credit allocation strategy that is applicable, at the correct time in the training process. For example, a particular heuristic-entity combination might benefit from a *pbest* credit allocation, while another heuristic-entity combination might benefit from a *ibest* credit allocation at the same time in the training process.

Models

For this dissertation, focus was put on training [FFNNs](#). However, in theory, other models can also be used. Suggestions include *deep neural networks* ([DNNs](#)) and *recurrent neural networks* ([RNNs](#)). Furthermore, future research can attempt to use the [BHH](#) for dynamic optimisation problems.

Dynamic Hyper-Parameter Values

It was shown that larger reselection intervals are generally preferred. It was shown that there is a relationship between the reselection interval, the replay window size and the reanalysis interval. These hyper-parameters provide a mechanism for trade-off between exploration and exploitation and therefore, investigation into dynamic selection of hyper-parameter values can be considered in future research.

Heuristic Pools

This dissertation only considered three different types of heuristic pool configurations. The following alternative configurations could be considered. A heuristic pool could be considered where multiple instances of the same heuristic is included, but with different hyper-parameter values, effectively implementing dynamic hyper-parameter optimisation. Empirical testing can be done to determine the effects of various heuristic pool sizes. Different types of heuristics can be considered for inclusion in the heuristic pool. A suggestion is to include black-box optimisers such as [CMA-ES](#). Finally, heuristic ensemble pools could be considered, where the heuristic selection is a selection of an ensemble of heuristics.

Parameter Pools

A suggestion for future research is to apply the [BHH](#) to a multitude of parameter types. These could be hyper-parameters of the low-level heuristics, the parameters

for the topology and architecture of models (similar to neural architecture search) or the hyper-parameters of the BHH itself.

9.7 Documentation and Data

All the source code, data, logs and analyses that were used in this dissertation, as well as the dissertation source itself, can be found at <https://github.com/arneschreuder/masters>. The raw empirical data is made available on Google Cloud's BigQuery platform, under the project id *masters-363209*.

9.8 Summary

This chapter concludes the research that is done in this dissertation. This chapter provided a summary of the research intent, providing a brief review of the research problem, motivations and objectives defined for this dissertation. A brief summary of the background information that was covered in this dissertation was provided. A brief summary on the concept and implementation of the novel BHH was provided, followed by a brief summary of the methodology that was used for the empirical process. This chapter also provided brief summaries of the results and findings for each of the experimental groups that were executed in the empirical process. Future research opportunities were identified and reference was made to materials and assets used in this dissertation that was made available online.

Bibliography

- [1] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. “Data mining: A preprocessing engine”. In: *Journal of Computer Science* 2.9 (2006), pp. 735–739.
- [2] David M Allen. “The relationship between variable selection and data augmentation and a method for prediction”. In: *technometrics* 16.1 (1974), pp. 125–127.
- [3] John A Allen and Steven Minton. “Selecting the right heuristic algorithm: Runtime performance predictors”. In: *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 1996, pp. 41–53.
- [4] Deborah Ashby. “Bayesian statistics in medicine: a 25 year review”. In: *Statistics in medicine* 25.21 (2006), pp. 3589–3631.
- [5] Thomas Back. “Selective pressure in evolutionary algorithms: A characterization of selection mechanisms”. In: *Proceedings of the first IEEE conference on evolutionary computation. IEEE World Congress on Computational Intelligence*. IEEE. 1994, pp. 57–62.
- [6] Thomas Bayes. “LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S”. In: *Philosophical transactions of the Royal Society of London* 53 (1763), pp. 370–418.
- [7] Yoshua Bengio. “Gradient-based optimization of hyperparameters”. In: *Neural computation* 12.8 (2000), pp. 1889–1900.
- [8] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.
- [9] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. “Advances in optimizing recurrent networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8624–8628.

- [10] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. “Are artificial neural networks black boxes?” In: *IEEE Transactions on neural networks* 8.5 (1997), pp. 1156–1164.
- [11] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. “A survey on metaheuristics for stochastic combinatorial optimization”. In: *Natural Computing* 8.2 (2009), pp. 239–287.
- [12] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006.
- [13] Christian Blum and Andrea Roli. “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.
- [14] Marko Bohanec and Vladislav Rajkovic. “Knowledge acquisition and explanation for multi-attribute decision making”. In: *8th Intl Workshop on Expert Systems and their Applications*. Citeseer. 1988, pp. 59–78.
- [15] Hans J Bremermann et al. “Optimization through evolution and recombination”. In: *Self-organizing systems* 93 (1962), p. 106.
- [16] Jason Brownlee. “Supervised and unsupervised machine learning algorithms”. In: *Machine Learning Mastery* 16.03 (2016).
- [17] Jason Brownlee. “How to one hot encode sequence data in python”. In: *Machine Learning Mastery* 12 (2017).
- [18] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. “Hyper-heuristics: An emerging direction in modern search technology”. In: *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.
- [19] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. “A classification of hyper-heuristic approaches”. In: *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.
- [20] Edmund K Burke et al. “Hyper-heuristics: A survey of the state of the art”. In: *Journal of the Operational Research Society* 64.12 (2013), pp. 1695–1724.
- [21] James Cannady. “Artificial neural networks for misuse detection”. In: *National information systems security conference*. Vol. 26. Baltimore. 1998.

- [22] Marcio Carvalho and Teresa B Ludermir. “An analysis of PSO hybrid algorithms for feed-forward neural networks training”. In: *2006 Ninth Brazilian Symposium on Neural Networks (SBRN'06)*. IEEE. 2006, pp. 6–11.
- [23] M Cassotti, D Ballabio, R Todeschini, and V Consonni. “A similarity-based QSAR model for predicting acute toxicity towards the fathead minnow (*Pimephales promelas*)”. In: *SAR and QSAR in Environmental Research* 26.3 (2015), pp. 217–243.
- [24] P. Cortez and A. Morais. “A Data Mining Approach to Predict Forest Fires using Meteorological Data”. In: *New Trends in Artificial Intelligence, Proceedings of the 13th EPIA*. Guimarães, Portugal: APPIA, Dec. 2007, pp. 512–523.
- [25] P. Cortez and A. M. G. Silva. “Using data mining to predict secondary school student performance”. In: *Proceedings of 5th Future Business Technology Conference (FUBUTEC)*. EUROSIS-ETI, Jan. 2008, pp. 5–12.
- [26] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. “Modeling wine preferences by data mining from physicochemical properties”. In: *Decision support systems* 47.4 (2009), pp. 547–553.
- [27] Peter Cowling, Graham Kendall, and Eric Soubeiga. “A hyperheuristic approach to scheduling a sales summit”. In: *International conference on the practice and theory of automated timetabling*. Springer. 2000, pp. 176–190.
- [28] Christian Darken, Joseph Chang, and John Moody. “Learning rate schedules for faster stochastic gradient search”. In: *Neural Networks for Signal Processing [1992] II., Proceedings of the 1992 IEEE-SP Workshop*. IEEE. 1992, pp. 3–12.
- [29] C. Darwin. *On the Origin of the Species and The Voyage of the Beagle*. West Margin Press, 2012. ISBN: 9780882408767.
- [30] Charles Darwin. *Charles Darwin's natural selection: being the second part of his big species book written from 1856 to 1858*. Cambridge University Press, 1987.
- [31] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in neural information processing systems* 27 (2014).

- [32] Judith E Dayhoff and James M DeLeo. “Artificial neural networks: opening the black box”. In: *Cancer: Interdisciplinary International Journal of the American Cancer Society* 91.S8 (2001), pp. 1615–1635.
- [33] Abraham De Moivre. *The doctrine of chances: or, A method of calculating the probability of events in play*. W. Pearson, 1718.
- [34] Saverio De Vito, Ettore Massera, Marco Piga, Luca Martinotto, and Girolamo Di Francia. “On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario”. In: *Sensors and Actuators B: Chemical* 129.2 (2008), pp. 750–757.
- [35] John S Denker and Yann LeCun. “Transforming neural-net output levels to probability distributions”. In: *Advances in neural information processing systems*. 1991, pp. 853–859.
- [36] Pedro Domingos and Michael Pazzani. “On the optimality of the simple Bayesian classifier under zero-one loss”. In: *Machine learning* 29.2 (1997), pp. 103–130.
- [37] Kathryn A Dowsland, Eric Soubeiga, and Edmund Burke. “A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation”. In: *European Journal of Operational Research* 179.3 (2007), pp. 759–774.
- [38] John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. “Recent advances in selection hyper-heuristics”. In: *European Journal of Operational Research* 285.2 (2020), pp. 405–428.
- [39] Juan Du. “The frontier of SGD and its variants in machine learning”. In: *Journal of Physics: Conference Series*. Vol. 1229. 1. IOP Publishing. 2019, p. 012046.
- [40] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [41] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7 (2011).
- [42] Russ Eberhart, Pat Simpson, and Roy Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., 1996.

- [43] Russ C Eberhart and Yuhui Shi. “Comparing inertia weights and constriction factors in particle swarm optimization”. In: *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*. Vol. 1. IEEE. 2000, pp. 84–88.
- [44] Russell Eberhart and James Kennedy. “A new optimizer using particle swarm theory”. In: *MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Ieee. 1995, pp. 39–43.
- [45] A.P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007. ISBN: 9780470512500.
- [46] Deniz Erdogmus, Oscar Fontenla-Romero, Jose C Principe, Amparo Alonso-Betanzos, Enrique Castillo, and Robert Jenssen. “Accurate initialization of neural network weights by backpropagation of the desired response”. In: *Proceedings of the International Joint Conference on Neural Networks, 2003*. Vol. 3. IEEE. 2003, pp. 2005–2010.
- [47] Andres Espinal et al. “Comparison of PSO and DE for training neural networks”. In: *2011 10th Mexican International Conference on Artificial Intelligence*. IEEE. 2011, pp. 83–87.
- [48] Hadi Fanaee-T and Joao Gama. “Event labeling combining ensemble detectors and background knowledge”. In: *Progress in Artificial Intelligence* 2.2 (2014), pp. 113–127.
- [49] Mercedes Fernández-Redondo and Carlos Hernández-Espinosa. “Weight initialization methods for multilayer feedforward.” In: *ESANN*. 2001, pp. 119–124.
- [50] Matthias Feurer and Frank Hutter. “Hyperparameter optimization”. In: *Automated machine learning*. Springer, Cham, 2019, pp. 3–33.
- [51] Ronald A Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of eugenics* 7.2 (1936), pp. 179–188.
- [52] Ronald Aylmer Fisher et al. “014: On the “Probable Error” of a Coefficient of Correlation Deduced from a Small Sample”. In: *Metron* 1 (1921), pp. 3–32.
- [53] Ronald Aylmer Fisher et al. “The design of experiments.” In: *The design of experiments*. 2nd Ed (1937).
- [54] Louise Francis. “Neural networks demystified”. In: *Casualty Actuarial Society Forum*. 2001, pp. 253–320.

- [55] Alex S Fraser. “Simulation of genetic systems by automatic digital computers I. Introduction”. In: *Australian journal of biological sciences* 10.4 (1957), pp. 484–491.
- [56] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, Inc., 2017.
- [57] Christophe Giraud-Carrier, Ricardo Vilalta, and Pavel Brazdil. “Introduction to the special issue on meta-learning”. In: *Machine learning* 54.3 (2004), pp. 187–193.
- [58] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.
- [59] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549.
- [60] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [61] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.
- [62] Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. “A visual exploration of Gaussian processes”. In: *Distill* 4.4 (2019), e17.
- [63] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.
- [64] Jacomine Grobler. “The heterogeneous meta-hyper-heuristic: from low level heuristics to low level metaheuristics”. PhD thesis. University of Pretoria.
- [65] Jacomine Grobler, Andries P Engelbrecht, Graham Kendall, and VSS Yadavalli. “Investigating the use of local search for improving meta-hyper-heuristic performance”. In: *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE. 2012, pp. 1–8.
- [66] Venu G Gudise and Ganesh K Venayagamoorthy. “Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks”. In: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS’03 (Cat. No. 03EX706)*. IEEE. 2003, pp. 110–117.

- [67] Jatinder ND Gupta and Randall S Sexton. “Comparing backpropagation with a genetic algorithm for neural network training”. In: *Omega* 27.6 (1999), pp. 679–684.
- [68] Branimir K Hackenberger. “Bayes or not Bayes, is this the question?” In: *Croatian medical journal* 60.1 (2019), p. 50.
- [69] Jacques Hadamard. *Mémoire sur le problème d’analyse relatif à l’équilibre des plaques élastiques encastrées*. Vol. 33. Imprimerie nationale, 1908.
- [70] Boris Hanin. “Which neural net architectures give rise to exploding and vanishing gradients?” In: *Advances in Neural Information Processing Systems*. 2018, pp. 582–591.
- [71] D. Harris and S.L. Harris. *Digital Design and Computer Architecture*. Computer organization bundle, VHDL Bundle. Elsevier Science, 2010. ISBN: 9780080547060.
- [72] David Harrison Jr and Daniel L Rubinfeld. “Hedonic housing prices and the demand for clean air”. In: *Journal of environmental economics and management* 5.1 (1978), pp. 81–102.
- [73] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).
- [74] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent”. In: *Cited on* 14.8 (2012), p. 2.
- [75] Alan L Hodgkin and Andrew F Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4 (1952), pp. 500–544.
- [76] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [77] Horas. *Wikimedia Commons: Beta distribution*. 2014. URL: https://en.wikipedia.org/wiki/Beta_distribution#/media/File:Beta_distribution_.pdf.svg.
- [78] Horas. *Wikimedia Commons: Cumulative Beta distribution*. 2014. URL: https://en.wikipedia.org/wiki/Beta_distribution#/media/File:Beta_distribution_cdf.svg.

- [79] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feed-forward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.
- [80] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. “Meta-Learning in Neural Networks: A Survey”. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 44.09 (Sept. 2022), pp. 5149–5169. ISSN: 1939-3539.
- [81] Peter R Huttenlocher and Arun S Dabholkar. “Regional differences in synaptogenesis in human cerebral cortex”. In: *Journal of comparative Neurology* 387.2 (1997), pp. 167–178.
- [82] Jarmo Ilonen, Joni-Kristian Kamarainen, and Jouni Lampinen. “Differential evolution training algorithm for feed-forward neural networks”. In: *Neural Processing Letters* 17.1 (2003), pp. 93–105.
- [83] Kenneth E Iverson. “A programming language”. In: *Proceedings of the May 1-3, 1962, spring joint computer conference*. 1962, pp. 345–351.
- [84] Anil Jain, Karthik Nandakumar, and Arun Ross. “Score normalization in multimodal biometric systems”. In: *Pattern recognition* 38.12 (2005), pp. 2270–2285.
- [85] Anil K Jain, Jianchang Mao, and KM Mohiuddin. “Artificial neural networks: A tutorial”. In: *Computer* 3 (1996), pp. 31–44.
- [86] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013.
- [87] Kevin Jarrett, Koray Kavukcuoglu, Marc’Aurelio Ranzato, and Yann LeCun. “What is the best multi-stage architecture for object recognition?”. In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.
- [88] Bekir Karlik and A Vehbi Olgac. “Performance analysis of various activation functions in generalized MLP architectures of neural networks”. In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.
- [89] James Kennedy. “The particle swarm: social adaptation of knowledge”. In: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC’97)*. IEEE. 1997, pp. 303–308.
- [90] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

- [91] Mary B Kennedy. “Synaptic signaling in learning and memory”. In: *Cold Spring Harbor perspectives in biology* 8.2 (2016), a016824.
- [92] Javed Khan et al. “Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks”. In: *Nature medicine* 7.6 (2001), p. 673.
- [93] Ahmed Kheiri and Ed Keedwell. “A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems”. In: *Evolutionary computation* 25.3 (2017), pp. 473–501.
- [94] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).
- [95] William H Kruskal and W Allen Wallis. “Use of ranks in one-criterion variance analysis”. In: *Journal of the American statistical Association* 47.260 (1952), pp. 583–621.
- [96] Sandeep Kumar and Eugene H Spafford. “A pattern matching model for misuse intrusion detection”. In: *Proceedings of the 17th National Computer Security Conference*. Vol. 10. Baltimore, MD. 1994, pp. 11–21.
- [97] Krystyna Kuzniar and Maciej Zajac. “Some methods of pre-processing input data for neural networks”. In: *Computer Assisted Methods in Engineering and Science* 22.2 (2017), pp. 141–151.
- [98] Annu Lambora, Kunal Gupta, and Kriti Chopra. “Genetic Algorithm - A Literature Review”. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019, pp. 380–384.
- [99] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.
- [100] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. “A theoretical framework for back-propagation”. In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann. 1988, pp. 21–28.
- [101] Claude Lemaréchal. “Cauchy and the gradient method”. In: *Doc Math Extra* 251.254 (2012), p. 10.

- [102] Howard Levene. “Robust tests for equality of variances”. In: *Contributions to probability and statistics. Essays in honor of Harold Hotelling* (1961), pp. 279–292.
- [103] Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. “Deal or No Deal? End-to-End Learning of Negotiation Dialogues”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP-17)*. Jan. 2017, pp. 2443–2453.
- [104] Che-Wei Lin and Jeen-Shing Wang. “A digital circuit design of hyperbolic tangent sigmoid function for neural networks”. In: *2008 IEEE International Symposium on Circuits and Systems*. IEEE. 2008, pp. 856–859.
- [105] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.
- [106] Henry B Mann and Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The annals of mathematical statistics* (1947), pp. 50–60.
- [107] Efrñn Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A Coello Coello. “A comparative study of differential evolution variants for global optimization”. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 485–492.
- [108] Risto Miikkulainen. “Topology of a Neural Network”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 988–989. ISBN: 978-0-387-30164-8.
- [109] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. “Designing Neural Networks Using Genetic Algorithms.” In: *ICGA*. Vol. 89. 1989, pp. 379–384.
- [110] Liu Mingguang and Li Gaoyang. “Artificial neural network co-optimization algorithm based on differential evolution”. In: *2009 Second International Symposium on Computational Intelligence and Design*. Vol. 1. IEEE. 2009, pp. 256–259.
- [111] David J Montana and Lawrence Davis. “Training Feedforward Neural Networks Using Genetic Algorithms.” In: *IJCAI*. Vol. 89. 1989, pp. 762–767.
- [112] Sérgio Moro, Paulo Cortez, and Paulo Rita. “A data-driven approach to predict the success of bank telemarketing”. In: *Decision Support Systems* 62 (2014), pp. 22–31.

- [113] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *ICML*. 2010.
- [114] Radford M Neal. “Bayesian learning via stochastic dynamics”. In: *Advances in neural information processing systems*. 1993, pp. 475–482.
- [115] Gerrit Stephanus Nel. “A hyperheuristic approach towards the training of artificial neural networks.” PhD thesis. University of Stellenbosch, 2021.
- [116] Filipe V Nepomuceno and Andries P Engelbrecht. “A self-adaptive heterogeneous pso for real-parameter optimization”. In: *2013 IEEE congress on evolutionary computation*. IEEE. 2013, pp. 361–368.
- [117] Nerebur. *Wikimedia Commons: Dirichlet distribution*. 2014. URL: <https://en.wikipedia.org/wiki/File:Dirichlet.pdf>.
- [118] Yurii Nesterov. “A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$ ”. In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.
- [119] Derrick Nguyen and Bernard Widrow. “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights”. In: *1990 IJCNN International Joint Conference on Neural Networks*. IEEE. 1990, pp. 21–26.
- [120] Elre Talea Oldewage. “The Perils of Particle Swarm Optimization in High Dimensional Problem Spaces”. MA thesis. South Africa: Department of Computer Science, School of Information Technology, EBIT Faculty, University of Pretoria, 2003.
- [121] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. “Hill climbers and mutational heuristics in hyperheuristics”. In: *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006, pp. 202–211.
- [122] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. “A comprehensive analysis of hyper-heuristics”. In: *Intelligent data analysis* 12.1 (2008), pp. 3–23.
- [123] Gisele L Pappa, Gabriela Ochoa, Matthew R Hyde, Alex A Freitas, John Woodward, and Jerry Swan. “Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms”. In: *Genetic Programming and Evolvable Machines* 15.1 (2014), pp. 3–35.
- [124] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984. ISBN: 9780201055948.

- [125] Joe Pharos. *Wikimedia Commons: Stochastic Gradient Descent*. 2006. URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent#/media/File:Stogra.png.
- [126] N. Pillay and R. Qu. *Hyper-Heuristics: Theory and Applications*. Natural Computing Series. Springer International Publishing, 2018. ISBN: 9783319965147.
- [127] Nelishia Pillay. “Intelligent system design using hyper-heuristics”. In: *South African Computer Journal* 56.1 (2015), pp. 107–119.
- [128] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.
- [129] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.
- [130] Lawrence Rabiner and Biinghwang Juang. “An introduction to hidden Markov models”. In: *ieee assp magazine* 3.1 (1986), pp. 4–16.
- [131] Anastassia S Rakitianskaia and Andries P Engelbrecht. “Training feed-forward neural networks with dynamic particle swarm optimisation”. In: *Swarm Intelligence* 6.3 (2012), pp. 233–270.
- [132] Jon Reed, Robert Toombs, and Nils Aall Barricelli. “Simulation of biological evolution and machine learning: I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing”. In: *Journal of theoretical biology* 17.3 (1967), pp. 319–342.
- [133] R. Reed and R.J. MarksII. *Neural Smithing: Supervised Learning in Feed-forward Artificial Neural Networks*. A Bradford Book. MIT Press, 1999. ISBN: 9780262181907.
- [134] CR Reeves. “Modern heuristic techniques for combinatorial problems Blackwell Scientific Press”. In: *Oxford, UK* 1.99 (1993), p. 2.
- [135] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The annals of mathematical statistics* (1951), pp. 400–407.
- [136] Paolo Rocca, Giacomo Oliveri, and Andrea Massa. “Differential evolution as applied to electromagnetics”. In: *IEEE Antennas and Propagation Magazine* 53.1 (2011), pp. 38–49.
- [137] Marc HJ Romanycia and Francis Jeffry Pelletier. “What is a heuristic?”. In: *Computational Intelligence* 1.1 (1985), pp. 47–58.

- [138] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
- [139] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.
- [140] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).
- [141] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.
- [142] A. Sandberg and N. Bostrom. *Whole Brain Emulation: A Roadmap*. Tech. rep. 3. Future of Humanity Institute, Oxford University, 2008.
- [143] Jeffrey Curtis Schlimmer. “Concept acquisition through representational adjustment”. PhD thesis. University of California, Irvine, 1987.
- [144] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.
- [145] Samuel Sanford Shapiro and Martin B Wilk. “An analysis of variance test for normality (complete samples)”. In: *Biometrika* 52.3/4 (1965), pp. 591–611.
- [146] Yuhui Shi and Russell Eberhart. “A modified particle swarm optimizer”. In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE. 1998, pp. 69–73.
- [147] MNH Siddique and MO Tokhi. “Training neural networks: backpropagation vs. genetic algorithms”. In: *IJCNN’01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*. Vol. 4. IEEE. 2001, pp. 2673–2678.
- [148] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–503. URL: <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>.
- [149] David Silver et al. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.
- [150] Yashpal Singh and Alok Singh Chauhan. “NEURAL NETWORKS IN DATA MINING.” In: *Journal of Theoretical & Applied Information Technology* 5.1 (2009).

- [151] Adam Slowik and Michal Bialko. “Training of artificial neural networks using differential evolution algorithm”. In: *2008 conference on human system interactions*. IEEE. 2008, pp. 60–65.
- [152] Celso AR de Sousa. “An overview on weight initialization methods for feedforward neural networks”. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 52–59.
- [153] Donald F Specht. “A general regression neural network”. In: *IEEE transactions on neural networks* 2.6 (1991), pp. 568–576.
- [154] Rainer Storn. “On the usage of differential evolution for function optimization”. In: *Proceedings of North American Fuzzy Information Processing*. IEEE. 1996, pp. 519–523.
- [155] Rainer Storn and Kenneth Price. “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.
- [156] Beata Strack et al. “Impact of HbA1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records”. In: *BioMed research international* 2014 (2014).
- [157] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.
- [158] Richard Sutton. “Two problems with back propagation and other steepest descent learning procedures for networks”. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*. 1986, pp. 823–832.
- [159] Gilbert Syswerda et al. “Uniform crossover in genetic algorithms.” In: *ICGA*. Vol. 3. 1989, pp. 2–9.
- [160] Igor V Tetko, David J Livingstone, and Alexander I Luik. “Neural network studies. 1. Comparison of overfitting and overtraining”. In: *Journal of chemical information and computer sciences* 35.5 (1995), pp. 826–833.
- [161] Dirk Thierens, Johan Suykens, Joos Vandewalle, and Bart De Moor. “Genetic weight optimization of a feedforward neural network controller”. In: *Artificial Neural Nets and Genetic Algorithms*. Springer. 1993, pp. 658–663.

- [162] Georg Thimm and Emile Fiesler. “Neural network initialization”. In: *International Workshop on Artificial Neural Networks*. Springer. 1995, pp. 535–542.
- [163] Ronald A Thisted. “What is a P-value”. In: *Departments of Statistics and Health Studies* (1998).
- [164] Sebastian Thrun and Lorien Pratt. “Learning to learn: Introduction and overview”. In: *Learning to learn*. Springer, 1998, pp. 3–17.
- [165] Ludovic Trottier, Philippe Giguere, and Brahim Chaib-Draa. “Parametric exponential linear unit for deep convolutional neural networks”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2017, pp. 207–214.
- [166] Athanasios Tsanas, Max Little, Patrick McSharry, and Lorraine Ramig. “Accurate telemonitoring of Parkinson’s disease progression by non-invasive speech tests”. In: *Nature Precedings* (2009), pp. 1–1.
- [167] John W Tukey. “Comparing individual means in the analysis of variance”. In: *Biometrics* (1949), pp. 99–114.
- [168] Frans Van Den Bergh et al. “An analysis of particle swarm optimizers”. PhD thesis. University of Pretoria, 2007.
- [169] Frans Van den Bergh and Andries Petrus Engelbrecht. “A study of particle swarm optimization particle trajectories”. In: *Information sciences* 176.8 (2006), pp. 937–971.
- [170] Frans Van den Bergh and Andries Petrus Engelbrecht. “A convergence proof for the particle swarm optimiser”. In: *Fundamenta Informaticae* 105.4 (2010), pp. 341–374.
- [171] Stefan AG Van der Stockt and Andries P Engelbrecht. “Analysis of selection hyper-heuristics for population-based metaheuristics in real-valued dynamic optimization”. In: *Swarm and Evolutionary Computation* (2018).
- [172] Andrich Benjamin Van Wyk. “An Analysis of Overfitting in Particle Swarm Optimised Neural Networks”. MA thesis. University of Pretoria, 2014.
- [173] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [174] Ricardo Vilalta and Youssef Drissi. “A perspective view and survey of meta-learning”. In: *Artificial intelligence review* 18.2 (2002), pp. 77–95.

- [175] Dennis Wackerly, William Mendenhall, and Richard L Scheaffer. *Mathematical statistics with applications*. Cengage Learning, 2014.
- [176] Samuel George Waugh. “Extending and benchmarking Cascade-Correlation: extensions to the Cascade-Correlation architecture and benchmarking of feed-forward supervised artificial neural networks”. PhD thesis. University of Tasmania, 1995.
- [177] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Vol. 1. John Wiley & Sons, 1994.
- [178] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.
- [179] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. URL: <https://arxiv.org/abs/1505.00853>.
- [180] Jim YF Yam and Tommy WS Chow. “A weight initialization method for improving training speed in feedforward neural network”. In: *Neurocomputing* 30.1-4 (2000), pp. 219–232.
- [181] Edward N Zalta. “Metaphysics Research Lab”. In: *Center for the Study of Language and Information, Stanford University, Stanford, CA* (2015).
- [182] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).
- [183] Andreas Zell. *Simulation neuronaler netze*. Vol. 1. 5.3. Addison-Wesley Bonn, 1994.
- [184] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. “Ensembling neural networks: many could be better than all”. In: *Artificial intelligence* 137.1-2 (2002), pp. 239–263.
- [185] Israel Ziv, Douglas A Baxter, and John H Byrne. “Simulator for neural networks and action potentials: description and application”. In: *Journal of neurophysiology* 71.1 (1994), pp. 294–308.
- [186] Barret Zoph and Quoc Le. “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations*. 2017.

Appendix A

Acronyms

This appendix lists all the acronyms that were used throughout the thesis. Acronyms are typeset in bold, with the corresponding meaning alongside. The list of acronyms is ordered alphabetically.

Adadelta *adaptive learning rate*

Adagrad *adaptive gradients*

Adam *adaptive moment estimation*

AI *artificial intelligence*

AN *artificial neuron*

ANN *artificial neural network*

ANOVA *analysis of variance*

BHH *Bayesian hyper-heuristic*

BinXE *binary cross entropy*

BN *biological neuron*

BP *backpropagation*

CatEX *categorical cross entropy*

CLT *central limit theorem*

CMA-ES *covariance matrix adaptation evolution strategy*

CSP *constraint satisfaction problem*

DE	<i>differential evolution</i>
DNN	<i>deep neural network</i>
DoE	<i>Design of Experiments</i>
EA	<i>evolutionary algorithm</i>
EC	<i>evolutionary computation</i>
FFNN	<i>feedforward neural network</i>
GA	<i>genetic algorithm</i>
GD	<i>gradient descent</i>
HH	<i>hyper-heuristic</i>
HMM	<i>hidden Markov model</i>
LReLU	<i>leaky rectified linear unit</i>
MAE	<i>mean absolute error</i>
MAP	<i>maximum a posteriori estimation</i>
MCMC	<i>Markov Chain Monte Carlo</i>
MH	<i>meta-heuristic</i>
ML	<i>machine learning</i>
MLE	<i>maximum likelihood estimation</i>
Momentum	<i>momentum</i>
MSE	<i>mean squared error</i>
NAG	<i>Nesterov accelerated gradients</i>
NFL	<i>no free lunch theorem</i>
NN	<i>neural network</i>
NN	<i>Bayesian neural network</i>
PDF	<i>probability density function</i>
PMF	<i>probability mass function</i>

PSO *particle swarm optimisation*

ReLU *rectified linear unit*

RL *reinforcement learning*

RMS *root mean squared*

RMSE *root mean squared error*

RMSPProp *root mean squared error propagation*

RNN *recurrent neural network*

SGD *stochastic gradient descent*

SparseCatXE *sparse categorical cross entropy*

SSE *sum squared error*

SU *summation unit*

Appendix B

Symbols

This appendix summarises the important symbols used throughout the dissertation. A short description is provided for each symbol. Symbols are divided according to the chapter in which they were introduced and are ordered alphabetically.

B.1 Chapter 2: Artificial Neural Networks

α	Scaling parameter used by the LReLU activation function.
β	Mini-batch size.
\mathbf{v}	The weight vector.
\mathbf{x}_p	The p -th input vector/pattern.
\mathbf{x}	The input pattern/vector.
\mathbf{y}_p	The output vector for pattern p .
\mathbf{y}	The output vector.
\mathbf{y}'	The softmax form of the output vector.
ϵ	The cost/loss produced by the loss function.
$\hat{y}_{k,p}$	The target output for the ANN at the k -th dimension for the p -th pattern.
λ	Scaling/control parameter used by the sigmoid and tanh activation functions.
\mathbb{R}^I	The real-number space in I dimensions.
\mathbb{R}^K	The real-number space in K dimensions.

\mathbb{R}^T	The real-number space in T dimensions.
\mathbb{R}	The real-number space.
$\mathbb{1}$	The indicator function.
μ_i	The mean of the input at dimension i .
ω_i	The general weight vector used for normalisation at dimension i .
ω_{max}	The upper bound of the uniform distribution used for weights initialisation.
ω_{min}	The lower bound of the uniform distribution used for weights initialisation.
ω	The general weight vector used for normalisation.
σ_i^2	The unit variance of the input at dimension i .
σ	The standard deviation of the truncated normal distribution used for weight initialisation.
τ	Phase shift threshold parameter.
θ	The bias unit.
c	Index used for c -th class.
C	Total number of classes.
f_{AN}	The mapping function realised by the AN .
$f(net)$	The activation function over net input signal.
$f(x)$	The function used to describe the LReLU activation function.
f	A shortened form of the activation function.
f_{anin}	The number of input neurons to the weight vector.
f_{anout}	The number of output neurons from the weight vector.
h_j	The j -th dimension of the hidden layer.
i	Index for the input vector.
I	The input pattern's dimension size.
j	Index used for the hidden layer.

k	Index for the output vector.
K	Number of output units.
$net_{h,y}$	The net input signal at the output layer.
$net_{i,h}$	The net input signal at the hidden layer.
net'	The augmented net input signal that includes the bias term.
net	The net input signal to the AN .
p	Index for the input data patterns.
P	Total number of input data patterns.
T	The target pattern's dimension size.
$v_{i,j}$	The weight associated with input node x_i and the hidden node h_j .
v_i	The i -th dimension of the weight vector.
v_{i+1}	The $i + 1$ -th dimension of the weight vector.
$w_{j,k}$	The weight associated with hidden node h_j and the output node y_k .
$x_{i_{max}}$	The input's maximum value at dimension i .
$x_{i_{min}}$	The input's minimum value at dimension i .
$x'_{i,p}$	The normalised form of the input at dimension i , pattern p .
$x_{i,p}$	The original input at dimension i , pattern p .
x_i	The i -th dimension of the input pattern/vector.
x_{i+1}	The $i + 1$ -th dimension of the input pattern/vector.
x_{max}	The input's maximum value used by the min-max scaler.
x_{min}	The input's minimum value used by the min-max scaler.
x^+	The positive part of the input parameter's domain.
$y_{k,p}$	The output produced by the ANN at the k -th dimension for the p -th pattern.
y_k	The k -th dimension of the output vector.
y'_k	The softmax value of the output at dimension k .

B.2 Chapter 3: Heuristics

α	The momentum hyper-parameter.
β_1	First decay rate parameter for Adam .
β_2	Second decay rate parameter for Adam .
β_1^t	First decay rate parameter for Adam at time step t .
β_2^t	Second decay rate parameter for Adam at time step t .
β	Scale factor as implemented by DE .
$\hat{\mathbf{y}}_i(t)$	The i -th particle's global best position vector as implemented by PSO .
$\hat{\mathbf{y}}$	Short form of the global best position vector.
\mathbf{G}_t	The sum of the squares of the gradients.
\mathbf{g}_t	The gradient vector at time step t .
\mathbf{g}_t^2	The squared gradient vector at time step t .
\mathbf{g}	General symbol used to represent the gradient vector, retrieved from the error function relative to the weight vector.
\mathbf{m}_t	The estimate of the first moment at time step t .
\mathbf{m}_{t+1}	The estimate of the first moment at time step $t + 1$.
\mathbf{r}_1	The vector for the first stochastic scaling parameter as implemented by PSO .
\mathbf{r}_2	The vector for the second stochastic scaling parameter as implemented by PSO .
$\mathbf{u}_i(t)$	The i -th entity's trial vector at time step t implemented by DE .
$\mathbf{v}_i(t)$	The i -th particle's velocity vector at time step t as implemented by PSO .
\mathbf{V}_{max}	The upper bound of the velocity clamping vector.
\mathbf{v}_t	The estimate of the second moment at time step t .
\mathbf{v}_{t+1}	The estimate of the second moment at time step $t + 1$.
\mathbf{v}	The velocity vector resulting from Momentum .
\mathbf{w}_t^2	The squared weight vector at time step t .

\mathbf{w}_{t+1}	The weight vector at time step $t + 1$.
\mathbf{w}	General symbol used to represent the weight vector.
\mathbf{w}	The weight vector.
$\mathbf{x}_{i_1}(t)$	The i -th entity's target vector at time step t implemented by DE.
$\mathbf{x}_{i_2}(t)$	The i -th entity's first selected individual's position vector at time step t implemented by DE.
$\mathbf{x}_{i_3}(t)$	The i -th entity's second selected individual's position vector at time step t implemented by DE.
$\mathbf{x}_i(0)$	The i -th particle's position vector at time step 0 as implemented by PSO.
$\mathbf{x}_i(t)$	The i -th particle's position vector at time step t as implemented by PSO. Also represents the i -th parent's position vector at time step t as implemented by DE.
\mathbf{x}_i	The i -th particle's position vector as implemented by PSO. Also represents the i -th parent's position vector as implemented by DE.
\mathbf{x}_i	The i -th particle's position vector as implemented by PSO.
\mathbf{x}_{max}	The upper bound constraint vector of the position vector's values.
\mathbf{x}_{min}	The lower bound constraint vector of the position vector's values.
$\mathbf{x}'_i(t)$	The i -th offspring's position vector at time step t as implemented by DE.
$\mathbf{y}_i(0)$	The i -th particle's personal best position vector at time step 0 as implemented by PSO.
$\mathbf{y}_i(t)$	The i -th particle's personal best position vector at time step t as implemented by PSO.
$\Delta \mathbf{w}_t^2$	The delta of the squared weight vector at time step t .
$\Delta w_i(t)$	The change in the i -th dimension of the weight vector at time step t .
\emptyset	The empty set.
ϵ_{T_p}	The error produced for pattern p at time step T .
ϵ_T	The error at time step T .
ϵ	The error between the output of the FFNN and the target pattern, produced by the loss function. Also used as a small error value used by heuristics to avoid division by 0.

η	Learning rate.
$\hat{\mathbf{m}}_t$	The bias-corrected first moment at time step t .
$\hat{\mathbf{v}}_t$	The bias-corrected second moment at time step t .
$\hat{y}_{ij}(t)$	The i -th particle's j -th dimension of the global best position vector at time step t .
\hat{y}_{ij}	The i -th particle's j -th dimension of the global best position vector.
$\mathcal{C}(0)$	The set of entities in the population at time step 0.
$\mathcal{C}(t)$	The set of entities in the population at time step t .
$\mathcal{C}(t + 1)$	The set of entities in the population at time step $t + 1$.
\mathcal{C}	The set of entities in the population.
\mathcal{E}	The error function.
\mathcal{J}	The set of crossover points.
\odot	The matrix-vector product.
c_1	The cognitive control parameter as implemented by PSO .
c_2	The social control parameter as implemented by PSO .
$E[\mathbf{g}^2]_{t-1}$	The expected value of the squared gradients at time step $t - 1$.
$E[\mathbf{g}^2]_t$	The expected value of the squared gradients at time step t .
$E[\Delta \mathbf{w}^2]_{t-1}$	The expected value of the delta of the squared weight vector at time step $t - 1$.
$E[\Delta \mathbf{w}^2]_t$	The expected value of the delta of the squared weight vector at time step t .
$f(\hat{\mathbf{y}})$	The evaluation of the global best position.
$f(\mathbf{x}_i(t))$	The evaluation of the i -th entity's candidate solution at time step t .
$f(\mathbf{x}'_i(t))$	The evaluation of the i -th offspring's candidate solution at time step t .
$f(\mathbf{y}_i)$	The evaluation of the i -th particle's personal best position.
$f(\hat{\mathbf{y}})$	The evaluation of the global best position.
$f(x_i)$	The evaluation of the i -th particle's solution with respect to the objective function.

$f(x)$	The objective function over x .
f	A shortened form of the objective function being optimised.
$g_{t,i}$	The i -th dimension of the gradient vector at time step t .
$G_{t,i}$	The sum of the squared gradients with regards to the i -th dimension of the weight vector.
$G_{t,ii}$	Diagonal matrix containing the sum of the squared gradients with regards to the i -th dimension of the weight vector.
i_1	Index for the target vector in DE .
i_2	Index for the first selected individual in DE .
i_3	Index for the second selected individual in DE .
i	Index used for the weight vector. Also used as the index for the i -th particle in the population, implemented by PSO .
I	The total number of particles in the swarm (particle swarm size).
j^*	Randomly selected crossover point.
j	Index used for the dimensions of candidate solutions.
J	The total number of dimensions in the position vector for particles in PSO .
k	Index used for the output layer.
K	Total number of output units.
$m_j(t)$	The j -th dimension of the crossover mask, m , at time step t .
N_t	Tournament selection size.
N	An alternative to the particle swarm size.
net_p	The net input signal for pattern p .
$o_{i,p}$	The i -th dimension of the output for pattern p .
$o_{k,p}$	The k -th dimension of the output for pattern p .
p_c	Probability of producing offspring.
p_m	The mutation probability.
p_r	The crossover probability i.e. recombination probability.

p_x	Bitswapping probability.
p	Index used for input data patterns.
$r_{1j}(t)$	The j -th dimension of the first stochastic scaling parameter at time step t as implemented by PSO .
$r_{2j}(t)$	The j -th dimension of the second stochastic scaling parameter at time step t as implemented by PSO .
$RMS[\mathbf{g}]_t$	The root mean squared gradient vector at time step t .
RMS	The root mean squared error criterion.
$t_{i,p}$	The i -th dimension of the target for pattern p .
$t_{k,p}$	The k -th dimension of the target for pattern p .
t	Time step.
T	Time step.
$u_{ij}(t)$	The j -th dimension of the i -th entity's trial vector at time step t implemented by DE .
U	The uniform distribution.
$v_{ij}(t)$	The i -th particle's j -th dimension of the velocity vector at time step t . Also used in the mutation operator for GAs to represent some sampled mutation update value.
$v_{ij}(t+1)$	The i -th particle's j -th dimension of the velocity vector at time step $t+1$.
$V_{max,j}$	The j -th dimension of the upper bound of the velocity clamping vector.
$v'_{ij}(t+1)$	The i -th particle's j -th dimension of the clamped velocity vector at time step $t+1$.
$w_i(t-1)$	The i -th dimension of the weight vector at time step $t-1$.
$w_i(t)$	The i -th dimension of the weight vector at time step t .
w_i	The i -th dimension of the weight vector.
$w_{t,i}$	The i -th dimension of the weight vector at time step t .
$w_{t+1,i}$	The i -th dimension of the weight vector at time step $t+1$.
w	The inertia weight hyper-parameter as implemented by PSO .
W	Window size of previous squared gradients.

$x_{ij}(t)$	The i -th particle's/entity's j -th dimension of the position vector at time step t .
$x_{ij}(t + 1)$	The i -th particle's j -th dimension of the position vector at time step $t + 1$.
x_{ij}	The i -th particle's/entity's j -th dimension of the position vector.
$x_{max,j}$	The j -th dimension of the upper bound constraint for an entity's position.
$x_{min,j}$	The j -th dimension of the lower bound constraint for an entity's position.
$x'_{ij}(t)$	The i -th offspring's j -th dimension of the position vector at time step t .
x	The independent variables to the objective function. Also used in DE notation to denote the selection mechanism for the trial vector.
$y_{ij}(t)$	The i -th particle's j -th dimension of the personal best position vector at time step t .
y_{ij}	The i -th particle's j -th dimension of the personal best position vector.
y	Used in DE notation to denote the number of difference vectors to include.
$z_{i,p}$	The input value at index i for pattern p .
z	Used in DE notation to denote the type of crossover operator to use.

B.3 Chapter 4: Hyper-Heuristics

K	The number of folds to execute for K -fold crossover validation.
-----	--

B.4 Chapter 5: Probability

α_0	The sum of all the values in each dimension of the concentration parameter α .
α_1	The first dimension of the concentration parameter α .
α_k	The k -th dimension of the concentration parameter α .
α_K	The last dimension of the concentration parameter α .
α'	The update form of the prior parameter, α , for a Beta probability distribution.

α	Shape parameter for the Beta probability distribution. Also used for confidence levels for the statistical tests.
\bar{A}	Not observing random event A .
β'	The update form of the prior parameter, β , for a Beta probability distribution.
β	Shape parameter for the Beta probability distribution.
α'	Update form of the prior parameter, α , for a Dirichlet probability distribution.
α	The concentration parameter vector that parameterises the Dirichlet probability distribution.
\mathcal{D}	All the prior data of \mathbf{X}
θ	The probability vector in multiple dimensions used by multiple probability distributions.
N	The counts for each occurrence of a category k over I independent, identical (iid) events.
\mathbf{x}_1	The outcomes of the first random event in I independent, identical (iid) events for k categories.
\mathbf{x}_i	The outcomes of the i -th random event in I independent, identical (iid) events for k categories.
\mathbf{x}_I	The outcomes of the last random event in I independent, identical (iid) events for k categories.
\mathbf{X}	Matrix of Bernoulli probability distributions. Also used to represent the outcomes of I independent, identical (iid) random events.
\mathbf{x}	The random variable for various probability distributions in multiple dimensions.
\mathbf{Y}	A vector of random events.
\emptyset	The empty set.
$\Gamma'(n)$	The first derivative of the Gamma function over input n .
$\Gamma(n)$	The Gamma function over input n .
Γ	The Gamma function.
\mathbb{R}	Real-number space.
\mathbb{R}	The real-number space.

$\mathbb{1}$	Indicator function.
$\mathcal{A}(v)$	Functional form of priors.
\mathcal{L}	The likelihood function.
$\stackrel{\text{iid}}{\sim}$	Independent, identical (iid) random events.
ψ	The logarithmic derivative of the Gamma function, called the Digamma Function.
θ_1	The first dimension of the probability vector $\boldsymbol{\theta}$.
θ_k	The k -th dimension of the probability vector $\boldsymbol{\theta}$.
θ_K	The last dimension of the probability vector $\boldsymbol{\theta}$.
θ	Success probability for the Bernoulli probability distribution.
A	Random event.
B_1	The first set in the partition S .
B_2	The second set in the partition S .
B_i	The i -th set in the partition S .
B_j	The j -th set in the partition S .
B_K	The last set in the partition S .
$B(\alpha, \beta)$	The normalising constant used in the PDF of the Beta probability distribution.
$B(\boldsymbol{\alpha})$	The normalising constant used in the PDF of the Dirichlet probability distribution.
B	Random event.
Ber	The Bernoulli probability distribution.
$Beta$	The Beta probability distribution.
Bin	The Binomial probability distribution.
c	Symmetric distribution constant.
Cat	The Categorical probability distribution.
Dir	The Dirichlet probability distribution.

$E[\ln(x_k)]$	The expected value of the natural logarithm of the k -th dimension of the random variable.
$E[\ln(x)]$	The expected value of the natural logarithm of x .
$E[x_k]$	The expected value of the k -th dimension of the random variable.
$E[x]$	The expected value of the distribution of x .
f_{Ber}	The PMF for the Bernoulli probability distribution.
f_{Beta}	The PDF of the Beta probability distribution.
f_{Bin}	The PMF for the Binomial probability distribution.
f_{Cat}	The PMF for the Categorical probability distribution.
f_{Dir}	The PDF of the Dirichlet probability distribution.
f_{Mul}	The PMF for the Multinomial probability distribution.
f	General symbol used for functions. Used for likelihood functions, PMFs and PDFs .
F	The symbol used for a female outcome in the mice experiment.
g^*	Posterior density.
g	Prior density.
i	Index to track sets in partition S . Also used as a general index in random variables in I dimensions.
I	The total number of independent, identical (iid) random events.
j	Index to track sets in partition S . Also used as a general index in random variables in J dimensions.
k	Index used to denote class k in K classes. Also used as a general index in random variables in K dimensions.
K	The total number of sets in the partition S . Also used to denote the number of classes in the Dirichlet probability distribution.
$M[x_k]$	The mode of the k -th dimension of the random variable.
$M[x]$	The mode of the distribution of x .
m	The marginal density function.
M	The symbol used for a male outcome in the mice experiment.
Mul	The Multinomial probability distribution.

N_0	Summary variable tracking the number of unsuccessful Boolean outcomes.
N_1	Summary variable tracking the number of successful Boolean outcomes.
N_k	Summary variable, denoting the number of times a category k occurs over all trials in N .
N_K	Summary variable denoting the number of times category k occurs.
n	Input parameter to the Gamma function.
N	The total number of events observed.
P	The probability of some event.
p	The statistical p -value.
S	The union of mutually exclusive subsets. Also referred to as the probability simplex.
x_1	The first dimension of the random variable. Also used as the first random event.
$x_{i,k}$	The k -th dimension of the i -th random event vector \mathbf{x} .
x_i	The i -th dimension of the random variable. Also used as the i -th random event.
x_I	The last dimension of the random variable with I dimensions. Also used as the last random event in I random events.
x_j	The j -th dimension of the random variable. Also used as the j -th random event.
x_k	The k -th dimension of the random variable. Also used as the k -th random event.
x_K	The last dimension of the random variable with K dimensions. Also used as the last random event in K random events.
x	Random variable used by various probability distributions. Also referred to as a realisation of an event that occurred as a results of a random process X .
Y_1	The first random event in \mathbf{Y} .
y_1	The realisation of the first random event in \mathbf{Y} .
y_2	The realisation of the second random event in \mathbf{Y} .
Y_2	The second random event in \mathbf{Y} .
Y_i	The i -th random event in \mathbf{Y} .

y_i	The realisation of the i -th random event in \mathbf{Y} .
Y_N	The last random event in \mathbf{Y} with N events.
y_n	The realisation of the n -th random event in \mathbf{Y} with N events.

B.5 Chapter 6: Bayesian Hyper-Heuristic

α_1	The first dimension of the concentration parameter, α .
α_K	The k -th dimension of the concentration parameter, α .
α_k	The dimension of α associated with class k .
α	Concentration parameter for the Beta probability distribution.
β_1	The first dimension of the concentration parameter, β .
β_2	The second dimension of the concentration parameter, β .
$\beta_{j,k}$	The j -th entity's concentration parameter associated with heuristic k .
β	Concentration parameter for the Beta probability distribution.
α	The concentration parameters for the heuristic probability distribution.
β	The concentration parameters for the entity-heuristic probability distributions.
γ	The concentration parameters for the credit-heuristic probability distribution.
$\hat{\phi}$	The expected value of the entity-heuristic probabilities.
$\hat{\psi}$	The expected value of the heuristic-credit-assignment probabilities.
$\hat{\theta}$	The expected value of the heuristic selection probabilities.
λ	Error vector.
ϕ	The entity-heuristic probability distribution.
ψ	The credit-heuristic probability distribution.
θ	The heuristic probability distribution.
C	The event of observing credit assignments upon meeting credit assignment criteria.
E	The event of observing entities.

\mathbf{H}	The event of observing heuristics.
ϵ	Error factor.
η	Learning rate.
γ_0	The zeroth dimension of the concentration parameter, γ .
$\gamma_{1,k}$	The first dimension of the concentration parameter, γ associated with heuristic k .
γ_1	The first dimension of the concentration parameter, γ .
$\gamma_{2,k}$	The second dimension of the concentration parameter, γ associated with heuristic k .
γ_K	The k -th dimension of the concentration parameter, γ .
$\hat{\phi}_{j,k}$	The expected value of the probability of selecting heuristic k for entity j .
$\hat{\psi}_k$	The expected value of the probability of selecting heuristic k and observing credit assignments.
$\hat{\theta}_k$	The expected value of the probability of selecting heuristic k .
λ_j	The j -th dimension of the error vector, λ .
λ_J	The last dimension of the error vector, λ in J dimensions.
$\mathbb{1}_0$	The indicator function yielding a failure/non-occurrence.
$\mathbb{1}_1$	The indicator function yielding a success/occurrence.
$\mathcal{A}(v)$	Functional form of priors.
\mathcal{L}	The likelihood function.
$\phi_{j,k}$	The entity-heuristic selection probability for entity j and heuristic k .
ψ_k	The successful credit assignment probability for heuristic k .
θ_k	The heuristic selection probability for heuristic k .
A	A random event.
B	A random event.
$Beta$	The Beta probability distribution.
Bin	The Binomial probability distribution.

c_1	Successful credit allocation.
$c_{i,k}$	The realisation of the i -th event of observing a credit assignment i to heuristic k .
c_i	The i -th event of observing credit assignments.
Dir	The Dirichlet probability distribution.
$e_{i,j,k}$	The realisation of the i -th event of observing heuristic k for selected for entity j .
e_i	The i -th event of observing entities.
e_j	The j -th entity.
$E[\phi_{j,k}]$	The expected value of the probability of selecting heuristic k for entity j .
$E[\psi_k]$	The expected value of the probability of selecting heuristic k and observing credit assignments.
$E[\theta_k]$	The expected value of the probability of selecting heuristic k .
$h_{i,k}$	The realisation of the i -th event of observing heuristic k .
h_i	The i -th event of observing heuristics.
h_k	The k -th heuristic.
i	Index generally associated with event/run i .
I	The maximum number of instances in the replay window.
j	Index generally associated to entity j .
J	The entity pool (population) size.
k	Index generally associated to heuristic k .
K	The heuristic pool size.
L	The number of credit assignment output classes.
LSE	The log-sum-exp trick/function.
$Mult$	The Multinomial probability distribution.
$N_{0,k}$	The count of occurrences of the events c_i taking on a failure.
$N_{1,k}$	The count of occurrences of the events c_i taking on a success.

$N_{j,k}$	A summary variable denoting the count of occurrences of the events e_i taking on class j and h_i taking on class k in I independent, identical runs.
N_j	The count of the occurrences of observing entity j .
N_k	A summary variable denoting the count of occurrences of the event h_i taking on class k in I independent, identical runs.
N	The maximum number of random events observed.
S	The probability simplex.
t	Time step.
λ_1	The first dimension of the error vector, λ .

B.6 Chapter 7: Methodology

α	Used as a scaling parameter for the LReLU activation function. Also used as a threshold parameter for the statistical significance tests.
η	Learning rate.
\mathbb{R}	Real-number space.
v_{ij}	The j -th dimension of the velocity vector of the i -th particle in the population implemented by PSO .
x_{ij}	The j -th dimension of the position vector of the i -th particle in the population implemented by PSO .

B.7 Chapter 8: Results

α_0	The zeroth dimension of the concentration parameter α and is associated to heuristic 0 (SGD).
α_6	The sixth dimension of the concentration parameter α and is associated to heuristic 6 (Adam).
α_7	The seventh dimension of the concentration parameter α and is associated to heuristic 7 (PSO).
α_8	The eighth dimension of the concentration parameter α and is associated to heuristic 8 (GA).
α_i	The i -th dimension of the concentration parameter α and is associated to heuristic i .
α	The concentration parameters for the heuristic probability distribution.

ϕ	The entity-heuristic probability distribution.
ψ	The credit-heuristic probability distribution.
θ	The heuristic probability distribution.
C	The event of observing credit assignments upon meeting credit assignment criteria.
E	The event of observing entities.
H	The event of observing heuristics.
θ_0	The zeroth dimension of the heuristic selection probabilities θ and is associated to heuristic 0 (SGD).
θ_6	The sixth dimension of the heuristic selection probabilities θ and is associated to heuristic 6 (Adam).
θ_7	The seventh dimension of the heuristic selection probabilities θ and is associated to heuristic 7 (PSO).
θ_8	The eight dimension of the heuristic selection probabilities θ and is associated to heuristic 8 (GA).
c_1	Successful credit allocation.
Cat	The Categorical probability distribution.
Dir	The Dirichlet probability distribution.
e_0	Entity 0.
h_0	Heuristic 0 (SGD).
h_6	Heuristic 6 (Adam).
h_7	Heuristic 7 (PSO).
h_8	Heuristic 8 (GA).
i	General index associated with event/run number.
K	Heuristic pool size.

Appendix C

Datasets

This appendix provides insight into the preparation and usage of datasets that were used as part of the empirical process. The data processing tasks included:

- Exploratory analysis and visualisation of data distribution.
- Dropping of rows that contain missing data in and standardisation of data types.
- Normalising all numerical features using the standard score scaler and one-hot encoding of all categorical features.
- Splitting the datasets into a training set (80%) and a testing set (20%) and then shuffling and batching of data.

Appendix D

Statistical Analysis

This appendix provides the output of the various statistical tests that were executed for the empirical process. The appendix is structured as follows:

- **Section D.1** provides the output for the statistical analysis of the experimental group concerned with comparing the performance of [BHH](#) with individual, standalone, low-level heuristics.
- **Section D.2** provides the output for the statistical analysis of the experimental group that analyses the effects of the *heuristic pool* hyper-parameter on the outcomes of the [BHH](#).
- **Section D.3** provides the output for the statistical analysis of the experimental group that analyses the effects of the *population size* hyper-parameter on the outcomes of the [BHH](#).
- **Section D.4** provides the output for the statistical analysis of the experimental group that analyses the effects of the *credit assignment strategy* hyper-parameter on the outcomes of the [BHH](#).
- **Section D.5** provides the output for the statistical analysis of the experimental group that analyses the effects of the *reselection interval* hyper-parameter on the outcomes of the [BHH](#).
- **Section D.6** provides the output for the statistical analysis of the experimental group that analyses the effects of the *replay window size* hyper-parameter on the outcomes of the [BHH](#).
- **Section D.7** provides the output for the statistical analysis of the experimental group that analyses the effects of the *reanalysis interval* hyper-parameter on the outcomes of the [BHH](#).

- **Section D.8** provides the output for the statistical analysis of the experimental group that analyses the effects of the *burn in window size* hyper-parameter on the outcomes of the BHH.
- **Section D.9** provides the output for the statistical analysis of the experimental group that analyses the effects of the *normalisation* hyper-parameter on the outcomes of the BHH.
- **Section D.10** provides the output for the statistical analysis of the experimental group that analyses the effects of the *discounted rewards* hyper-parameter on the outcomes of the BHH.

D.1 BHH vs. Low-Level Heuristics

Table D.1: ANOVA - Rank - BHH vs. Low-Level Heuristics

Cases	Sum of Squares	df	Mean Square	F	p
dataset	7.680×10^{-5}	13	5.908×10^{-6}	1.422×10^{-6}	1.000
heuristic	$1.163 \times 10^{+6}$	12	96928.657	23323.376	< .001
dataset * heuristic	503842.812	156	3229.762	777.159	< .001
Residuals	702664.298	169078	4.156		

Table D.2: Kruskal-Wallis - BHH vs. Low-Level Heuristics

Factor	Statistic	df	p
heuristic	83080.880	12	< .001

Table D.3: Post Hoc Comparisons - BHH vs. Low-Level Heuristics - Part A

			95% CI for Mean Difference					
Mean Difference			Lower	Upper	SE	t	<i>P</i> _{tukey}	
adadelata	adagrad	2.722	2.638	2.805	0.025	107.716	< .001	
	adam	2.341	2.258	2.425	0.025	92.666	< .001	
	de	−5.181	−5.265	−5.097	0.025	−205.061	< .001	
	ga	−2.879	−2.963	−2.796	0.025	−113.966	< .001	
	momentum	−4.015	−4.099	−3.931	0.025	−158.907	< .001	
	nag	1.433	1.350	1.517	0.025	56.729	< .001	
	pso	−3.172	−3.256	−3.089	0.025	−125.557	< .001	
	rmsprop	1.704	1.620	1.787	0.025	67.426	< .001	
	sgd	−3.372	−3.455	−3.288	0.025	−133.448	< .001	
	bhh_all	0.840	0.756	0.924	0.025	33.247	< .001	
	bhh_gd	1.638	1.554	1.722	0.025	64.837	< .001	
	bhh_mh	−1.513	−1.596	−1.429	0.025	−59.872	< .001	
adagrad	adam	−0.380	−0.464	−0.297	0.025	−15.050	< .001	
	de	−7.903	−7.986	−7.819	0.025	−312.777	< .001	
	ga	−5.601	−5.685	−5.517	0.025	−221.683	< .001	
	momentum	−6.737	−6.820	−6.653	0.025	−266.623	< .001	
	nag	−1.288	−1.372	−1.205	0.025	−50.987	< .001	
	pso	−5.894	−5.978	−5.810	0.025	−233.273	< .001	
	rmsprop	−1.018	−1.102	−0.934	0.025	−40.290	< .001	
	sgd	−6.093	−6.177	−6.010	0.025	−241.165	< .001	
	bhh_all	−1.882	−1.965	−1.798	0.025	−74.470	< .001	
	bhh_gd	−1.083	−1.167	−1.000	0.025	−42.880	< .001	
	bhh_mh	−4.234	−4.318	−4.151	0.025	−167.589	< .001	
	adam	de	−7.522	−7.606	−7.439	0.025	−297.727	< .001
ga		−5.221	−5.305	−5.137	0.025	−206.632	< .001	
momentum		−6.356	−6.440	−6.273	0.025	−251.573	< .001	
nag		−0.908	−0.992	−0.824	0.025	−35.937	< .001	
pso		−5.514	−5.597	−5.430	0.025	−218.223	< .001	
rmsprop		−0.638	−0.721	−0.554	0.025	−25.240	< .001	
sgd		−5.713	−5.797	−5.629	0.025	−226.115	< .001	
bhh_all		−1.501	−1.585	−1.418	0.025	−59.420	< .001	
bhh_gd		−0.703	−0.787	−0.619	0.025	−27.830	< .001	
bhh_mh		−3.854	−3.938	−3.770	0.025	−152.539	< .001	
de		ga	2.302	2.218	2.385	0.025	91.095	< .001
		momentum	1.166	1.082	1.250	0.025	46.154	< .001
	nag	6.614	6.531	6.698	0.025	261.790	< .001	
	pso	2.009	1.925	2.092	0.025	79.504	< .001	
	rmsprop	6.885	6.801	6.968	0.025	272.487	< .001	
	sgd	1.809	1.726	1.893	0.025	71.612	< .001	
	bhh_all	6.021	5.937	6.105	0.025	238.307	< .001	
	bhh_gd	6.819	6.736	6.903	0.025	269.897	< .001	
	bhh_mh	3.668	3.585	3.752	0.025	145.188	< .001	

Table D.4: Post Hoc Comparisons - BHH vs. Low-Level Heuristics - Part B

		95% CI for Mean Difference					
		Mean Difference	Lower	Upper	SE	t	P _{Tukey}
ga	momentum	-1.135	-1.219	-1.052	0.025	-44.941	< .001
	nag	4.313	4.229	4.397	0.025	170.696	< .001
	pso	-0.293	-0.377	-0.209	0.025	-11.591	< .001
	rmsprop	4.583	4.499	4.667	0.025	181.393	< .001
	sgd	-0.492	-0.576	-0.409	0.025	-19.482	< .001
	bhh_all	3.720	3.636	3.803	0.025	147.213	< .001
	bhh_gd	4.518	4.434	4.601	0.025	178.803	< .001
	bhh_mh	1.367	1.283	1.450	0.025	54.094	< .001
momentum	nag	5.448	5.365	5.532	0.025	215.636	< .001
	pso	0.843	0.759	0.926	0.025	33.350	< .001
	rmsprop	5.719	5.635	5.802	0.025	226.333	< .001
	sgd	0.643	0.560	0.727	0.025	25.459	< .001
	bhh_all	4.855	4.771	4.939	0.025	192.154	< .001
	bhh_gd	5.653	5.569	5.737	0.025	223.744	< .001
	bhh_mh	2.502	2.419	2.586	0.025	99.035	< .001
nag	pso	-4.606	-4.689	-4.522	0.025	-182.286	< .001
	rmsprop	0.270	0.187	0.354	0.025	10.697	< .001
	sgd	-4.805	-4.889	-4.721	0.025	-190.178	< .001
	bhh_all	-0.593	-0.677	-0.510	0.025	-23.483	< .001
	bhh_gd	0.205	0.121	0.289	0.025	8.107	< .001
	bhh_mh	-2.946	-3.030	-2.862	0.025	-116.602	< .001
pso	rmsprop	4.876	4.792	4.960	0.025	192.984	< .001
	sgd	-0.199	-0.283	-0.116	0.025	-7.891	< .001
	bhh_all	4.012	3.929	4.096	0.025	158.804	< .001
	bhh_gd	4.811	4.727	4.894	0.025	190.394	< .001
	bhh_mh	1.660	1.576	1.743	0.025	65.685	< .001
rmsprop	sgd	-5.075	-5.159	-4.992	0.025	-200.875	< .001
	bhh_all	-0.864	-0.947	-0.780	0.025	-34.180	< .001
	bhh_gd	-0.065	-0.149	0.018	0.025	-2.590	0.316
	bhh_mh	-3.216	-3.300	-3.133	0.025	-127.299	< .001
sgd	bhh_all	4.212	4.128	4.295	0.025	166.695	< .001
	bhh_gd	5.010	4.926	5.094	0.025	198.285	< .001
	bhh_mh	1.859	1.775	1.943	0.025	73.576	< .001
bhh_all	bhh_gd	0.798	0.714	0.882	0.025	31.590	< .001
	bhh_mh	-2.353	-2.436	-2.269	0.025	-93.119	< .001
bhh_gd	bhh_mh	-3.151	-3.235	-3.067	0.025	-124.709	< .001

D.2 Heuristic Pool

Table D.5: ANOVA - Rank - BHH Variant: Heuristic Pool

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.000	13	0.000	0.000	1.000
heuristic_pool	7332.913	2	3666.456	9049.146	< .001
dataset * heuristic_pool	2898.104	26	111.466	275.107	< .001
Residuals	15808.983	39018	0.405		

Table D.6: Post Hoc Comparisons - BHH Variant: Heuristic Pool

		95% CI for Mean Difference					
		Mean Difference	Lower	Upper	SE	t	<i>P</i> _{tukey}
all	gd	0.250	0.231	0.268	0.008	31.660	< .001
	mh	−0.768	−0.787	−0.750	0.008	−97.404	< .001
gd	mh	−1.018	−1.037	−1.000	0.008	−129.064	< .001

Table D.7: Kruskal-Wallis - BHH Variant: Heuristic Pool

Factor	Statistic	df	p
heuristic_pool	10999.088	2	< .001

D.3 Population Size

Table D.8: ANOVA - Rank - BHH Variant: Population Size

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.072	13	0.006	0.003	1.000
population	519.146	4	129.786	66.439	< .001
dataset * population	2701.713	52	51.956	26.597	< .001
Residuals	127034.063	65030	1.953		

Table D.9: Post Hoc Comparisons - BHH Variant: Population Size

		95% CI for Mean Difference		SE	t	<i>P</i> _{tukey}
		Mean Difference	Lower	Upper		
5	10	−0.071	−0.119	−0.024	0.017	−4.119 < .001
	15	−0.189	−0.236	−0.141	0.017	−10.885 < .001
	20	−0.235	−0.283	−0.188	0.017	−13.585 < .001
	25	−0.204	−0.251	−0.157	0.017	−11.785 < .001
10	15	−0.117	−0.164	−0.070	0.017	−6.766 < .001
	20	−0.164	−0.211	−0.117	0.017	−9.466 < .001
	25	−0.133	−0.180	−0.086	0.017	−7.666 < .001
15	20	−0.047	−0.094	4.779×10^{-4}	0.017	−2.700 0.054
	25	−0.016	−0.063	0.032	0.017	−0.900 0.897
20	25	0.031	−0.016	0.078	0.017	1.800 0.373

Table D.10: Kruskal-Wallis - BHH Variant: Population Size

Factor	Statistic	df	p
population	259.454	4	< .001

D.4 Credit Assignment Strategy

Table D.11: ANOVA - Rank - BHH Variant: Credit Assignment Strategy

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.088	13	0.007	0.003	1.000
credit	76.779	4	19.195	9.756	< .001
dataset * credit	2245.456	52	43.182	21.949	< .001
Residuals	127940.670	65030	1.967		

Table D.12: Post Hoc Comparisons - BHH Variant: Credit Assignment Strategy

			95% CI for Mean Difference				
Mean Difference			Lower	Upper	SE	t	p_{tukey}
gbest	ibest	0.013	−0.035	0.060	0.017	0.742	0.947
	pbest	0.051	0.004	0.099	0.017	2.951	0.026
	rbest	−0.031	−0.078	0.017	0.017	−1.776	0.388
	symmetric	−0.047	−0.094	7.230×10^{-4}	0.017	−2.686	0.056
ibest	pbest	0.038	−0.009	0.086	0.017	2.209	0.176
	rbest	−0.044	−0.091	0.004	0.017	−2.518	0.087
	symmetric	−0.060	−0.107	−0.012	0.017	−3.428	0.005
pbest	rbest	−0.082	−0.130	−0.035	0.017	−4.727	< .001
	symmetric	−0.098	−0.145	−0.051	0.017	−5.637	< .001
rbest	symmetric	−0.016	−0.063	0.032	0.017	−0.910	0.893

Table D.13: Kruskal-Wallis - BHH Variant: Credit Assignment Strategy

Factor	Statistic	df	p
credit	38.373	4	< .001

D.5 Reselection Interval

Table D.14: ANOVA - Rank - BHH Variant: Reselection Interval

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.000	13	0.000	0.000	1.000
reselection	25571.533	4	6392.883	4372.618	< .001
dataset * reselection	9552.887	52	183.709	125.654	< .001
Residuals	95075.581	65030	1.462		

Table D.15: Post Hoc Comparisons - BHH Variant: Reselection Interval

		95% CI for Mean Difference		SE	t	<i>p</i> _{Tukey}
		Mean Difference	Lower	Upper		
1	5	0.724	0.683	0.765	0.015	48.314 < .001
	10	1.328	1.288	1.369	0.015	88.649 < .001
	15	1.546	1.505	1.587	0.015	103.168 < .001
	20	1.707	1.667	1.748	0.015	113.936 < .001
5	10	0.604	0.564	0.645	0.015	40.334 < .001
	15	0.822	0.781	0.863	0.015	54.854 < .001
	20	0.983	0.943	1.024	0.015	65.622 < .001
10	15	0.218	0.177	0.258	0.015	14.519 < .001
	20	0.379	0.338	0.420	0.015	25.287 < .001
15	20	0.161	0.120	0.202	0.015	10.768 < .001

Table D.16: Kruskal-Wallis - BHH Variant: Reselection Interval

Factor	Statistic	df	p
reselection	12785.570	4	< .001

D.6 Replay Window Size

Table D.17: ANOVA - Rank - BHH Variant: Replay Window Size

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.003	13	2.458×10^{-4}	1.246×10^{-4}	1.000
replay	103.598	4	25.900	13.132	< .001
dataset * replay	1854.151	52	35.657	18.079	< .001
Residuals	128254.247	65030	1.972		

Table D.18: Post Hoc Comparisons - BHH Variant: Replay Window Size

		95% CI for Mean Difference			SE	t	<i>P</i> _{tukey}
Mean Difference		Lower	Upper				
1	5	0.039	−0.009	0.086	0.017	2.220	0.172
	10	0.023	−0.025	0.070	0.017	1.306	0.687
	15	0.002	−0.045	0.050	0.017	0.119	1.000
	20	−0.077	−0.125	−0.030	0.017	−4.444	< .001
5	10	−0.016	−0.063	0.032	0.017	−0.913	0.892
	15	−0.037	−0.084	0.011	0.017	−2.100	0.220
	20	−0.116	−0.163	−0.068	0.017	−6.663	< .001
10	15	−0.021	−0.068	0.027	0.017	−1.187	0.759
	20	−0.100	−0.148	−0.053	0.017	−5.750	< .001
15	20	−0.079	−0.127	−0.032	0.017	−4.563	< .001

Table D.19: Kruskal-Wallis - BHH Variant: Replay Window Size

Factor	Statistic	df	p
replay	51.795	4	< .001

D.7 Reanalysis Internal

Table D.20: ANOVA - Rank - BHH Variant: Reanalysis Interval

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.320	13	0.025	0.012	1.000
reanalysis	47.470	4	11.868	6.017	< .001
dataset * reanalysis	2012.676	52	38.705	19.625	< .001
Residuals	128257.510	65030	1.972		

Table D.21: Post Hoc Comparisons - BHH Variant: Reanalysis Interval

		95% CI for Mean Difference		SE	t	P _{tukey}
	Mean Difference	Lower	Upper			
1 5	0.046	−0.001	0.094	0.017	2.661	0.060
	10	0.082	0.035	0.130	4.735	< .001
	15	0.060	0.012	0.107	3.420	0.006
	20	0.041	−0.006	0.089	2.378	0.121
5 10	0.036	−0.011	0.084	0.017	2.074	0.231
	15	0.013	−0.034	0.061	0.759	0.942
	20	−0.005	−0.052	0.043	−0.282	0.999
10 15	−0.023	−0.070	0.025	0.017	−1.315	0.682
	20	−0.041	−0.088	0.006	−2.356	0.128
15 20	−0.018	−0.066	0.029	0.017	−1.041	0.836

Table D.22: Kruskal-Wallis - BHH Variant: Reanalysis Interval

Factor	Statistic	df	p
reanalysis	23.713	4	< .001

D.8 Burn In

Table D.23: ANOVA - Rank - BHH Variant: Burn In

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.135	13	0.010	0.006	1.000
burn_in	6237.667	4	1559.417	850.761	< .001
dataset * burn_in	4842.323	52	93.122	50.804	< .001
Residuals	119197.865	65030	1.833		

Table D.24: Post Hoc Comparisons - BHH Variant: Burn In

		95% CI for Mean Difference		SE	t	p _{tukey}
		Mean Difference	Lower	Upper		
0	5	-0.171	-0.217	-0.126	0.017	-10.212 < .001
	10	-0.367	-0.413	-0.321	0.017	-21.884 < .001
	15	-0.595	-0.641	-0.549	0.017	-35.451 < .001
	20	-0.877	-0.923	-0.831	0.017	-52.281 < .001
5	10	-0.196	-0.242	-0.150	0.017	-11.672 < .001
	15	-0.424	-0.469	-0.378	0.017	-25.239 < .001
	20	-0.706	-0.752	-0.660	0.017	-42.069 < .001
10	15	-0.228	-0.273	-0.182	0.017	-13.567 < .001
	20	-0.510	-0.556	-0.464	0.017	-30.397 < .001
15	20	-0.282	-0.328	-0.237	0.017	-16.830 < .001

Table D.25: Kruskal-Wallis - BHH Variant: Burn In

Factor	Statistic	df	p
burn_in	3117.030	4	< .001

D.9 Normalisation

Table D.26: ANOVA - Rank - BHH Variant: Normalisation

Cases	Sum of Squares	df	Mean Square	F	p
dataset	4.992×10^{-4}	13	3.840×10^{-5}	1.555×10^{-4}	1.000
normalisation	5.751	1	5.751	23.282	< .001
dataset * normalisation	78.260	13	6.020	24.369	< .001
Residuals	6425.988	26012	0.247		

Table D.27: Post Hoc Comparisons - BHH Variant: Normalisation

		95% CI for Mean Difference		SE	t	p_{tukey}
	Mean Difference	Lower	Upper			
False True	-0.030	-0.042	-0.018	0.006	-4.825	< .001

Table D.28: Kruskal-Wallis - BHH Variant: Normalisation

Factor	Statistic	df	p
normalisation	23.005	1	< .001

D.10 Discounted Rewards

Table D.29: ANOVA - Rank - BHH Variant: Discounted Rewards

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.264	13	0.020	0.082	1.000
discounted_rewards	18.019	1	18.019	73.148	< .001
dataset * discounted_rewards	83.831	13	6.449	26.177	< .001
Residuals	6407.866	26012	0.246		

Table D.30: Post Hoc Comparisons - BHH Variant: Discounted Rewards

		95% CI for Mean Difference		SE	t	p _{tukey}
	Mean Difference	Lower	Upper			
False True	-0.053	-0.065	-0.041	0.006	-8.553	< .001

Table D.31: Kruskal-Wallis - BHH Variant: Discounted Rewards

Factor	Statistic	df	p
discounted_rewards	72.075	1	< .001

Appendix E

Derived Publications

The following publications were derived from this dissertation.

- A.N. Schreuder, A.P. Engelbrecht, A.S. Bosman, C.W. Cleghorn. “Bayesian Hyper-Heuristics”. *Submitted to Information Sciences journal*.

Index

- activation function, [10](#), [13–23](#), [28](#), [34](#)
- argmax, [21](#), [22](#)
- artificial neuron, [11](#), [12](#)
- axon, [11](#)

- batch training, [26](#)
- Bayes’ theorem, [66](#), [67](#), [69](#), [70](#), [79–81](#),
[84](#), [98–100](#)
- Bayesian analysis, [6](#), [8](#), [67](#), [108](#)
- Bernoulli probability distribution,
[74–79](#), [84](#)
- Beta probability distribution, [71–73](#),
[78](#), [79](#), [82](#), [114](#)
- biases, [10](#)
- Binomial probability distribution,
[75–77](#), [100](#), [109](#)
- biological neuron, [10](#), [11](#)

- Categorical probability distribution,
[76–79](#), [140](#)
- chromosomes, [52](#)
- crossover, [47–49](#), [51–57](#)

- dendrites, [11](#)
- Dirichlet probability distribution,
[72–74](#), [79–81](#), [100](#), [108](#), [134](#)
- discounted rewards, [113](#)
- dying ReLU problem, [18](#)

- elitism, [57](#)
- encoding, [13](#)

- ensemble networks, [5](#)
- entity pool, [88](#), [93](#), [99](#), [100](#), [111](#)
- error function, [10](#)

- features, [13](#)
- feedforward neural network, [24](#)
- fully connected topology, [23](#)

- Glorot normal sampling, [15](#), [16](#)
- Glorot uniform sampling, [15](#), [55](#), [120](#)

- heterogeneous meta-hyper-heuristic, [5](#),
[62](#)
- heuristic, [2](#), [25](#), [29–32](#), [42](#), [57–61](#), [63](#),
[64](#), [86–88](#), [90–97](#), [110](#), [111](#),
[117](#), [118](#), [120–122](#), [124–134](#),
[136–151](#), [153](#), [154](#), [157](#), [158](#),
[161](#), [164–167](#), [170](#), [171](#), [174](#),
[176](#), [177](#), [179](#), [182](#), [183](#), [186](#),
[187](#), [190](#), [192–204](#), [244](#)
- heuristic pool, [86–88](#), [90–92](#), [94](#), [95](#),
[99](#), [100](#), [110](#), [111](#), [114](#), [121](#),
[126](#), [127](#), [130](#), [132](#), [134](#), [136](#),
[142–146](#), [151–155](#), [157](#), [161](#),
[179](#), [195–197](#), [199](#), [200](#),
[202–204](#), [244](#)
- heuristic pool , [165](#)
- hyper-parameters, [3](#), [59](#), [87](#)
- hyperbolic tangent, [14](#), [17](#), [19](#), [20](#)

- independent variables, [13](#)

- Kruskal-Wallis, [127](#)
- Levene, [127](#)
- loss function, [32](#)
- Mann-Whitney U, [127](#)
- maximum a posteriori estimation, [108](#)
- maximum likelihood estimation, [105](#)
- meta-heuristic, [4](#), [31](#), [42](#), [91](#), [124](#), [142](#),
[146](#), [151](#), [153](#), [157](#)
- meta-learning, [3](#), [5](#), [6](#), [8](#), [59–61](#), [64](#)
- min-max scaler, [14](#)
- multi-method, [4](#)
- multi-method populated-based
 meta-heuristic, [4](#)
- Multinomial probability distribution,
 [77–80](#), [100](#), [101](#)
- mutation, [47](#), [49](#), [53–57](#)
- Naïve Bayes classifier, [8](#)
- Naïve Bayes, [101](#)
- net input signal, [10](#), [12](#), [16–18](#), [28](#), [34](#)
- normalisation, [14](#)
- normalised exponential function, [21](#)
- offline learning, [26](#), [58](#), [63](#)
- one-hot encoding, [13](#)
- online learning, [26](#), [63](#), [64](#), [131](#)
- optimisation algorithm, [31](#)
- overfitting, [26](#)
- query by committees, [5](#)
- random uniform sampling, [15](#)
- Shapiro-Wilk, [127](#)
- sigmoid, [14](#), [17](#), [19](#), [21](#), [22](#), [34](#)
- simulated annealing, [5](#), [62](#)
- softargmax, [21](#)
- softmax, [21](#), [22](#), [120](#)
- standard score scaler, [14](#), [243](#)
- stochastic training, [26](#)
- supervised learning, [1](#), [10](#), [25](#), [26](#), [28](#),
 [32](#)
- synapse, [11](#), [12](#)
- synaptic plasticity, [11](#)
- synaptogenesis, [11](#)
- tabu-search, [5](#), [62](#)
- unity-based normalisation, [14](#)
- vanishing gradients problem, [17](#), [18](#)
- weights, [10](#)
- z-score scaler, [14](#)