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.

September, 2022

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.ai

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 thus required to ensure that the best heuristic is used to train the 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 during training time. 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 out of classic gradient-based heuristics as well as meta-heuristics. The empirical process was executed on fifteen 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. 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, and 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. Bosman, Prof. C.W. Cleghorn and Prof. A.P. Engelbrecht. Thank you for not giving up on me.
- To my creator, God. Thank You . . . for everything!

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

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

> *The Cephalopod retreats and crawls back up against the glass of the fish tank evaluating the situation in front of him. In front of him is a glass jar with prey inside. The lid is shut tightly. Never before has he encountered such an obstacle in nature, prohibiting him from feeding. He curiously swims closer, feeling the shape of the jar with his tentacles. He wraps his tentacles over the jar and with a small turn and pop, he releases the lid. The experiment is repeated and without hesitation, without failure, he opens the jar, faster, more efficiently, every time.*

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 the academia [69, 72, 113, 144], as well as the industry at large [7, 117, 165, 166]. 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 [159]. Furthermore, an improvement on AlphaGo, called AlphaZero, learned to play Go, *tabula rasa* [1] and managed to beat AlphaGo, 100-1 [166].

Historically, ML models were built with a specific purpose in mind and revolved around specialised applications. A notable development was that of Deep Blue by IBM [99], which managed to play chess at a grand master level. Deep Blue was able to finish 2-4 against the greatest chess player at the time, Gary Kasparov. Although Deep Blue lost the tournament, this was a major breakthrough in the field of AI. However, Deep Blue was only able to play chess and could not be utilised for any other purposes [100]. Today, DeepMind's *reinforcement learning* (RL) algorithm is capable of playing all of the original Atari games [126], but is still very limited to a small set of problems that it can solve.

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

[1] Latin word meaning to learn from no prior knowledge.

generally be described as well-organised structures of mathematical computation and are inspired by the biological brain [54]. ANNs are the digital equivalent of how we currently understand the brain to work. ANNs can be trained, which is the equivalent of "learnin" from data. Learning gives rise to the ability to apply some form of decision making. This 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 the training of ANNs.

A popular field of focus for studying ANNs is the process by which these models are trained. FFNNs are specific types of ANNs [147]. The most common way of training FFNNs is supervised learning. 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* [139] is used to assign optimal values to the parameters of the ANN, such that a specified objective function is minimised.

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 of problem class, often does not necessarily work for another. This problem is known as the *no free lunch theorem* (NFL) [194].

Generalisation of ANNs refer to the capabilities of the network to either perform well on unseen data or to be applied to other problem domains. Various attempts have been made to improve the generalisation capabilities of ANNs. Examples of techniques used include weight dropout [172], weight decay [109], meta-learning of learning parameters such as the learning rate and momentum of gradient-based heuristics [37, 119, 199], and removing or postponing the use of outliers in the training data [148]. Other techniques focus on improving generalisation through model design. Design configurations focus on the layout of the neurons and structure of the ANN (architecture) and how neurons are connected to interact (topology). Examples of such techniques include learning of ANN architectures through pruning techniques [28, 55] or constructive techniques [83, 114], adaptive activation functions [56, 64], and quantum methods [150, 191].

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 influence how much a particular learning technique might focus on a particular solution (exploitation), in comparison to seeking out novel solutions (exploration) during training.

An example of dynamically balancing exploitation and exploration during the search process is by dynamically adjusting and learning the heuristic *hyper-parameters* as part of the learning process. This field of study is known as meta-learning [68]. Meta-learning of

heuristic hyper-parameters as applied in the training of ANNs has shown to yield good generalisation results [89, 189].

This dynamic nature of the learning process leads towards the idea that ML models could yield better generalisation capabilities if the learning process and learning mechanism applied are not statically defined, but rather dynamic and under the control of some mechanism. Furthermore, a learning technique that has at its disposal a more diverse set of learning behaviours from which it can dynamically select (when the need arises) could yield better generalisation capabilities [90].

A more 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 differs from problem to problem. 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 [132].

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 [24]. 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-space 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 the strategy of multiple heuristics working together to solve a problem. However, 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.

These selection HHs can be further categorised based on what information they use in the selection strategy. One specific type of selection HH is called a *multi-method population-based meta-heuristic* [187]. 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 [74] mentioned that HHs have been shown to solve a number of problems including bin-packing, examination timetabling, production scheduling, the traveling salesman problem, vehicle routing problem and many more.

This dissertation focuses on the development of a HH to train FFNNs in a supervised learning approach.

1.1 Problem Statement

Many different heuristics have been developed and used to train FFNNs [76, 127, 145]. Each of these heuristics have 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 very problem dependent [4, 48, 140].

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 trail 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 [141]. 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 tedious.

1.2 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.

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 [141]. Therefore, careful consideration is required to include and select from a diverse set of heuristics that is capable of solving the given problem. Importantly, only heuristics that are known to have the potential to solve the problem are applicable and can be included in the set of heuristics to select from.

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. [47], the tabu-search HH of Burke et al. [22], the heterogeneous meta-hyper-heuristic by Grobler [74] and work done by van der Stockt et al. [187] on the analysis of selection hyper-heuristics for population-based metaheuristics in real-valued dynamic optimization. However, research on the application

of HHs in the context of FFNN training is scarce. Nel [131] provides the first research in this field, applying a HH to FFNN training. The shortage of research in this field certainly warrants more exploration.

FFNNs training is seen as a computational search problem. *Hyper-heuristic* as defined by Burke et al. [22] 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. This implies that it should be possible to apply HHs in the context FFNN training.

Grobler et al. [75] 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 [22]. 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 FFNNs training.

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. [138] 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 [201]. 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 high-level heuristic 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 (online learning).

1.3 Objectives

The main objective of this research is focused on developing a novel 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 *artificial neuron* (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 these 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.
- 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 over a number of different datasets.
- 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 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 statistically sound analysis of the results obtained from the empirical study.

1.4 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*.
- 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 (for the particular problem) 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 FFNN in general, with statistical certainty. 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* [2] knowledge in which a predefined selection bias is used for heuristics that are known to be well suited for certain problems.

1.5 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*. The focus is on the training of FFNNs. It is shown how the training of FFNNs is seen as an optimisation problem.
- **Chapter 3** provides details on various types of heuristics and meta-heuristics 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

[2] Latin word, meaning "from what comes before".

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, performance measures used and other smaller details of the implementation of each experiment. 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 the 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. The chapter is concluded with a brief argument in favor of the general applicability of the BHH to more problem domains.

- **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 230 along with the following appendices:

- **Appendix A** provides details on the datasets used for the empirical analysis.

- **Appendix B** provides details on the outcomes of the statistical analyses.

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

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

> *There are an estimated 100 billion neurons in the human brain. That means that there are approximately as many neurons in the human brain as there are stars in our galaxy. Every one of these neurons is highly interconnected with other neurons, yielding a synapse count of 100 trillion. These neural structures are the result of millions of years of evolution. It is with these neural structures that we are able to think, learn and dream.*

The human brain is by far one of the most complicated biological organs in nature. The human brain is what 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 [155]. ANNs are used throughout this dissertation as the model to be trained using HHs. Understanding how ANNs work is an important component of this dissertation.

The purpose of this chapter is to provide the necessary background information needed on ANNs and is thus structured into five main parts:

- **Section 2.1** gives background information on the *biological neuron* (BN).
- **Section 2.2** introduces the AN and the various components that it is made up of. Brief discussions follow on input, weights, net input signal, biases, activation functions, and output.
- Discussions on the BN and the AN set the tone for the ANN which is discussed in detail in **Section 2.3**. Brief discussions follow on ANN architecture, topology, and FFNNs.
- **Section 2.4** gives details on the training process, training sets, supervised learning, error functions, performance measures and stopping conditions.
- Finally, a brief summary of the chapter is given in **Section 2.5**.

2.1 Biological Neuron

This section introduces the BN and provides the necessary background information in order to understand how it has inspired the development of the AN.

The biological neural systems are made up of microscopic nerve cells called neurons [95]. Figure 2.1 illustrates a single BN.

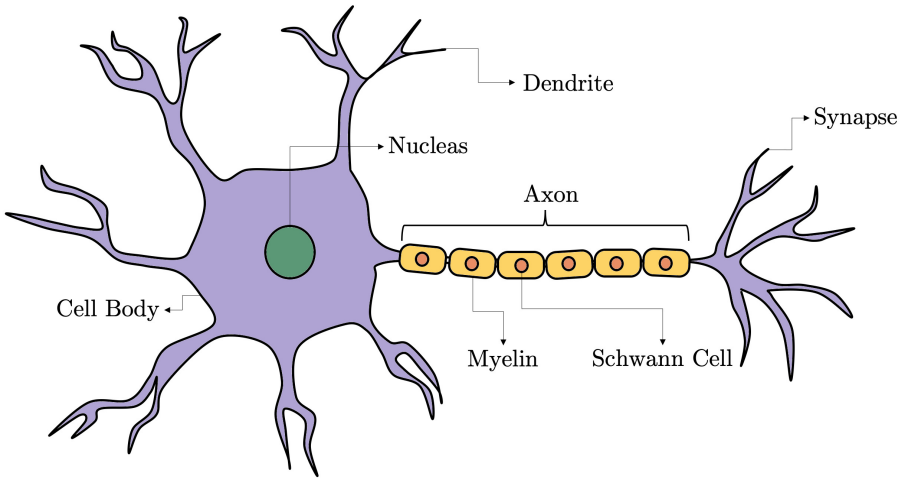


Figure 2.1: An illustration of the Biological Neuron.

The main parts of the BN to focus on include the cell body, the *dendrites* and the *axon* because these components are directly modeled in the AN. *Neural networks* (NNs) are formed by neurons that connect to each other. This is known as *synaptogenesis* [91]. Connections are made between the axons and dendrites of various neurons. Such a connection is referred to as a *synapse*. It is through synapses that neurons communicate with each other. Communication takes place 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 [54]. Neurons like these are massively interconnected. The greater the connection between two neurons, the stronger the communication. Kennedy [103] 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 [105]. The frequent release of these molecules cause the synapse to grow. Connections that grow over time yield stronger signals, while connections that are weak propagate low intensity signals and vanish over time. This is how the human brain learns and forgets. Every time we reinforce a memory in our brains, we "train" the neuron and the connection becomes stronger. This is known as *synaptic plasticity* [91]. this reinforcement of connections plays a pivotal role in the way we algorithmically train ANNs.

2.2 Artificial Neuron

This section introduces the AN. It is shown how the AN is modeled after the BN by providing a brief discussion on the various components that make up the AN. Each component is discussed in more detail, including input modeling, weight modeling, net input signal modeling, biases modeling and output modeling in the AN.

2.2.1 Components

The AN implements a nonlinear mapping from $\mathbb{R}^I$ to $\mathbb{R}^T$, usually in the ranges $[0, 1]$ or $[-1, 1]$, depending on the activation function used [54]. This mapping is given below:

$$f_{AN} : \mathbb{R}^I \rightarrow \mathbb{R}^T \tag{2.1}$$

where f_{AN} is the function mapping 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.

An illustration of the AN is given is Figure 2.2.

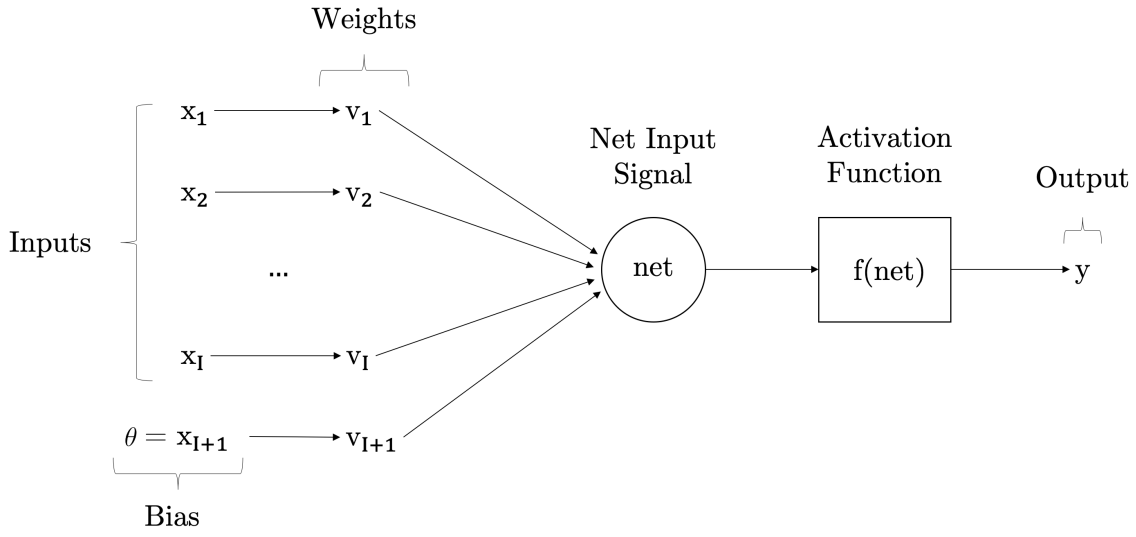


Figure 2.2: An illustration of the Artificial Neuron.

From Figure 2.2, one can see that the AN is made up of the following components:

- **Input:** The input models the activations received from the pre-synaptic neuron or from some environment sensor. Input is represented by the input vector $\vec{x}$.
- **Weights:** The weights model the synapses and connection strengths. Weights are represented by the weight vector, $\vec{v}$.
- **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*.

- **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.
- **Activation function:** Models the action potential of the BN based on the net input signal. The activation function is represented by f.
- **Output:** The output models the activation by the post-synaptic neuron. Output is represented by the output vector $\vec{y}$.

Each of the above mentioned components are discussed in detail in the following sections.

2.2.2 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* [65]. In 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 discussed next.

Input Pre-processing

Input pre-processing can help improve the training process, but also plays an important role in determining the success of a practical application of the ANN [111]. The primary purpose of input pre-processing is to modify the input data so that it can better match predicted output.

There are many techniques to consider when pre-processing input data. From scaling to the correct ranges to dealing with incomplete, invalid or even irrelevant data. For the purposes of this dissertation, only data encoding and normalisation is considered. These topics are now discussed in more detail.

Encoding

For qualitative data, such as class labels, the class labels have to be converted to numerical representations. In general, class labels are encoded as vectors [19, 171]. This dissertation makes use of *one-hot encoding* of class labels which yields a K-dimensional output vector $\vec{y}$ of 0's and a single 1. Each element y_k of $\vec{y}$ uniquely refers to a class. An activation, represented by a value of 1 at a given index thus represents a given class. Harris and Harris [81] 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).

Once data has been encoded into the right form, normalisation is required. This is discussed next.

Normalisation

Input data must be normalised and scaled appropriately. In general, input data is normalised to a range that is appropriate for the activation function being used. This yields input data in a range that is comparable to the output produced. For the purpose of this dissertation, the min-max scaler [3] and standard score scaler [94] are normalisation techniques that are considered. These are discussed next.

Min-Max Scaler

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

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

where $x'_{i,p}$ is the normalised form of $x_{i,p}$, the i-th dimension of the input vector $\vec{x}_p$, $p \in \{1, 2, \ldots, P\}$, 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 $\vec{x}_p, p \in \{1, 2, \ldots, P\}$.

Standard Score Scaler

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

$$x'_{i,p} = \frac{x_{i,p} - \mu_i}{\sigma_i^2} \tag{2.3}$$

where μ_i and σ_i^2 are respectfully the mean and unit variance of the i-th dimension of all input vectors $\vec{x}_p, p \in \{1, 2, \ldots, P\}$.

2.2.3 Weights

It was mentioned in Section 2.1 that synapses can have strong or weak connections. The connection strength is modeled in the AN as weight vectors of numerical values associated with each dimension i of the input data. Weights can either dampen or strengthen the input data by some negative or positive numerical value. By changing the weight associated with a feature, the influence of that particular feature on the predicted output is changed. Finding the correct value for each weight, to yield optimal output, is an optimisation problem [181]. The initial placement of these weight vectors in the search-space is very

important. Research has shown that weight initialisation plays an important role in the efficient training of ANNs [182]. Weight initialisation is discussed next, followed by some common weight initialisation techniques used.

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 [61]. Finding the optimal initialisation values for weights is non-trivial and can be seen as another optimisation problem. Extensive research on this topic has been done in the past [57, 169, 197].

Weight initialisation is dependent on the activation function used. If weights are initialised to values that are too small, the vanishing gradients problem can occur [80]. If weights are initialised to 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 [80, 196].

Research has presented many initialisation techniques [57]. Some techniques have been developed to address the problems that are mentioned above [196]. However, this dissertation will only focus on random uniform sampling, *Glorot uniform (Xavier)* sampling and *Glorot normal* sampling [69]. These techniques are briefly discussed next.

Random Uniform Sampling

Random uniform sampling initialises weights uniformly in the range $[\omega_{min}, \omega_{max}]$, written as $w_i \sim U(\omega_{min}, \omega_{max})$ 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 $(-0.5, 0.5)$ [134].

Glorot Uniform Sampling

Glorot uniform sampling is a specialisation of random uniform sampling whereby $\omega = \sqrt{\frac{6}{fanin+fanout}}$ and $fanin$ 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 [69].

2.2.4 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.

The most common way by which the net input signal is calculated is by means of *summation units* (SUs) [54], which compute the net input signal as the weighted sum of all input signals as shown below:

$$net = \sum_{i=1}^{I} x_i v_i \tag{2.4}$$

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

2.2.5 Biases

A bias/threshold term θ is introduced to help translate the output of the activation function [12]. The value of θ must be learned during the training process along with the weights of the ANN. In order to simplify equations, augmented vectors are used in which input and hidden layers have an additional neuron/unit, called the *bias unit* [54]. Augumented vectors that include the bias unit to the net input signal are presented next.

Augmented Net Input Signal

Input layer x_{I+1} has a net input signal $net^{'} = net - \theta$. The value for θ is learned along with the weights such that $\theta = x_{i+1}v_{i+1}$. If a constant value $x_{i+1} = -1$ is used, the equation for SUs as given in Equation (2.4) is then completed by including θ as shown below:

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

where $net^{'}$ is the augmented net input signal that now includes the bias term θ.

2.2.6 Activation Functions

An activation function is used to model the action potential of the AN [86, 202], and is a function over the augmented net input signal. It is the mechanism that models whether

an AN fires or not. When the output produced by the activation function surpasses some threshold value θ, we consider that neuron to have fired an output signal. Activation functions model *phase shift*. In the context of classification problems, these activation functions form decision boundaries between classes. In the context of regression problems, these 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 below:

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

or

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

Research in the past has presented many activation functions [98] of varying ranges. However, this dissertation focuses on the *rectified linear unit* (ReLU) [97, 129], the *leaky rectified linear unit* (LReLU) [120], the sigmoid [115] and the hyperbolic tangent [118] activation functions. These activation functions are presented next.

Rectified Linear Unit

The ReLU activation function is an activation function defined in the positive part of its argument. It is defined 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 [121, 195] 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 LReLU activation function is a variant of the ReLU activation function [195]. Similar to ReLU, it is not susceptible to the vanishing gradients problem, and it does not suffer from the dying ReLU problem [2], 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 defined as:

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

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 [80].

An illustration of the LReLU with various values for α and $\theta = 0$ is given in Figure 2.3.

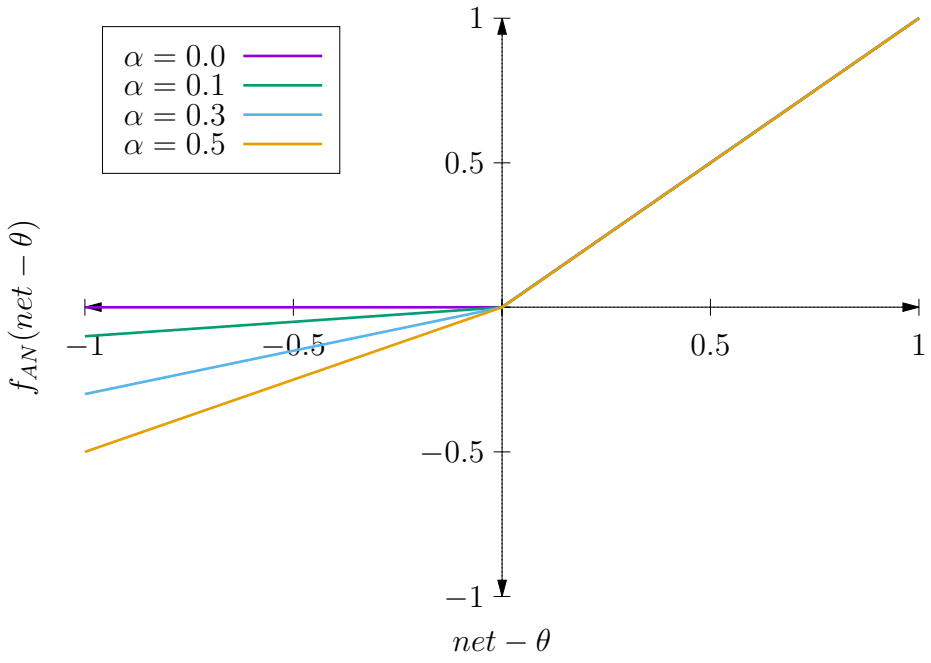


Figure 2.3: An illustration of the LReLU activation function.

Sigmoid

Sigmoid is a very popular activation function, because it is a continuous differentiable approximation of the step function, which was used in the original perceptron model developed by Rosenblatt [154]. The sigmoid activation function is defined as:

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

where λ is a control parameter, which controls the steepness of the function and is usually set to $\lambda = 1$. The sigmoid activation function is a continuous variant of the ramp function and yields $f_{AN} \in (0, 1)$.

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

Hyperbolic Tangent

The hyperbolic tangent activation function is similar to the sigmoid activation function, but yields output in the range $(-1, 1)$. The hyperbolic tangent activation function is defined as:

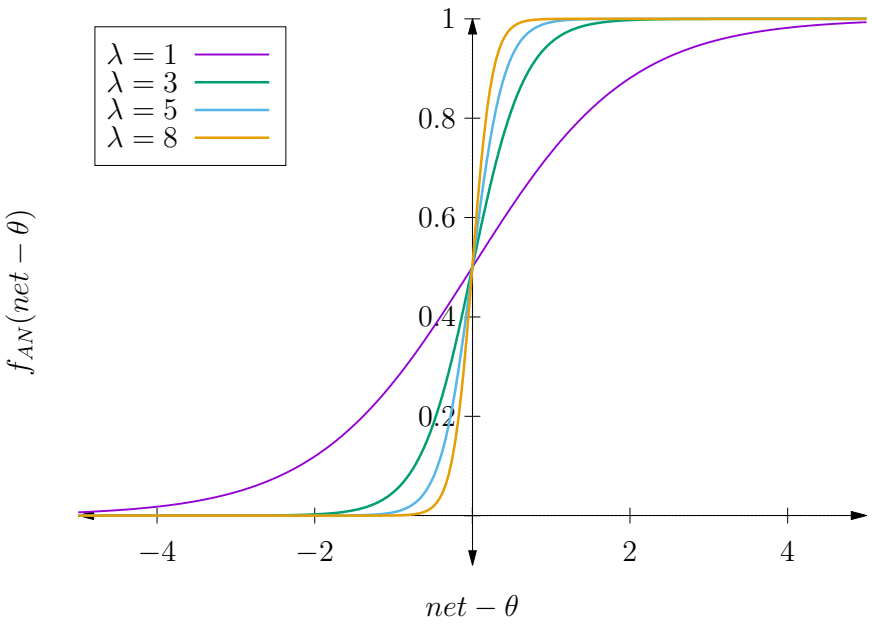


Figure 2.4: An illustration of the sigmoid activation function.

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

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

2.2.7 Output

Output signals are K-dimensional vectors of numerical values that are produced by the activation function. Output signals are often referred to as *predictions*. If target pre-processing was applied, it is necessary to post-process the output data. A brief discussion on post-processing techniques follows.

Output Post-Processing

Output post-processing is necessary in order to convert data back to their original ranges. The post-processing techniques that are applicable depend on the type of problem (regression or classification), input pre-processing techniques used, as well as the activation function used. For the purposes of this dissertation, only data decoding, normalisation and re-scaling techniques are considered.

Decoding

Data decoding is used to convert one-hot encoded vectors as described in Section 2.2.2 back to class labels. This is relevant to classification problems where the predicted output is a one-hot encoded vector representing a class label.

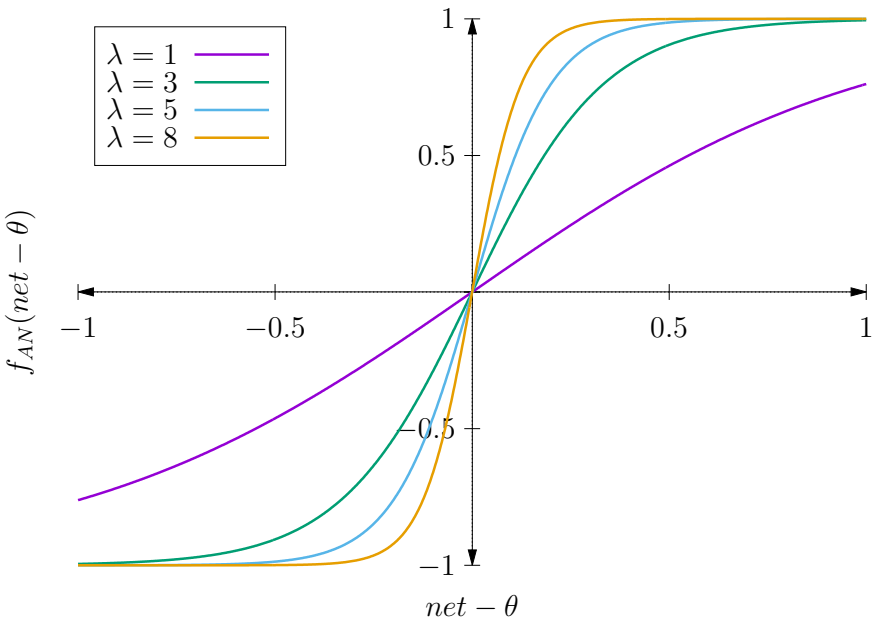


Figure 2.5: An illustration of the hyperbolic tangent activation function.

The high value (represented by 1) at y_k is used to decode the one-hot encoded output vector $\vec{y}$ back to its corresponding class label.

Normalisation of the output data is used to simplify the decoding process. This dissertation considers *softmax* and *argmax* normalisation to assist the decoding process.

Softmax

The softmax function, also known as the *softargmax* [71, p. 184] or *normalised exponential function* [14] is a generalisation of the logistic function that converts the K-dimensional output vector $\vec{y}$ into a K-dimensional output vector $\vec{y'}$ where each element y'_k is in the range $(0, 1)$ and all elements sum up to 1. This is expressed as follows:

$$\vec{y'} : \mathbb{R}^K \rightarrow \left\{ \vec{y'} \in \mathbb{R}^K | y'_k \in (0,1), \sum_{k=1}^{K} y_k = 1 \right\} \tag{2.12}$$

The softmax function is given as:

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

Argmax

The argmax function is similar to the softmax function, with the difference that the element $y_k, k \in \{1, 2, \ldots, K\}$ with the highest output value is set to 1 and the rest are set to 0. This means that 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:

$$y_k^{'} = \begin{cases} 1 & \text{if } y_k = \max(y_1, y_2, \ldots, y_K) \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

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

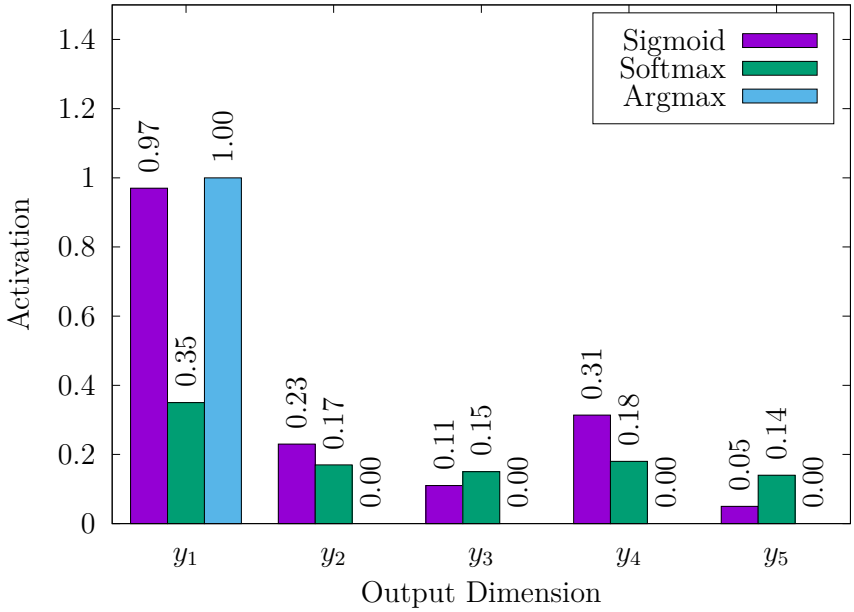


Figure 2.6: An illustration of softmax and argmax applied after the sigmoid activation function.

2.3 Artificial Neural Network

This section presents ANNs. Discussions follow on their applications, architecture, and topologies. Finally, a specific type of ANN called a *feedforward neural network* is discussed in detail.

2.3.1 Applications

ANNs are modelled after our biological brains. The human brain is biologically and chemically incredibly sophisticated. Trying to model an entire brain is at this stage not computationally feasible. Sandberg et al. [158] 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 [54] summarises some common problems that are solved using ANNs. These include:

- **Classification**: Predicting the class of an input vector [104].

- **Pattern Matching**: Producing closely associated patterns based on an input vector [25, 110].
- **Pattern Completion**: Completing the missing parts of an input vector [41].
- **Optimisation**: Producing optimal values of parameters in a optimisation problems [170].
- **Data Mining**: Feature discovery in large datasets [167].

Different ANNs are built for different types of problems, however, most applications of ANNs include some layered architecture. Discussions follow on ANN architecture and topologies.

2.3.2 Architecture

ANNs are collections of ANs that are organised together to form a connected network. The architecture of the ANN refers to the way in which these ANs are organised together as well as the type of activation function used.

ANNs are organised in layers where a single layer can contain multiple ANs. Generally, each layer makes use of the same activation function, but this is not a requirement. 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. The input, hidden and output layers are discussed next.

The input layer contains the input data to the ANN. Since the input layer simply provides the input data, some literature does not consider the input layer as a layer [54].

The hidden layer contains a collection of hidden ANs. These hidden ANs are referred to as *hidden units* or *nodes*. Hidden units are used if the target data is not linearly separable [54]. 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 [88].

The output layer contains the activations or the predictions of the ANN. This layer is used to analyse the accuracy of the prediction and thus the performance of the ANN.

2.3.3 Topology

The topology of the ANN refers to the way ANs are connected to each other. Since ANs are organised in layers, this connectivity is described layer by layer.

There are many different topologies [123], however, this dissertation only focuses on a fully connected topology.

The most common and simplest topology is where each AN in one layer is connected to all ANs in the next. Fully connected topologies have no cycles [200].

2.3.4 Feedforward Neural Networks

There are many different types of ANNs. All ANNs can be described in terms of their architecture and topology. This research will focus on a particular ANN called a *feedforward neural network* (FFNN).

FFNNs were the first and simplest ANNs developed [161]. FFNNs incorporate the basic layers (input, hidden and output layers) by arranging them in sequential order. In FFNNs information moves forward, in one direction, from the input nodes, through the hidden nodes and finally to the output nodes, and, depending on the optimisation algorith used, error correction information can be backpropagated through the network. It can be said that input is *fed* forward, through the ANN. The activation function used is problem dependent. FFNNs incorporate fully connected topologies amongst the layers such that no cycles form [200].

An illustration of a FFNN using a three-layered architecture and a fully connected topology is given in Figure 2.7.

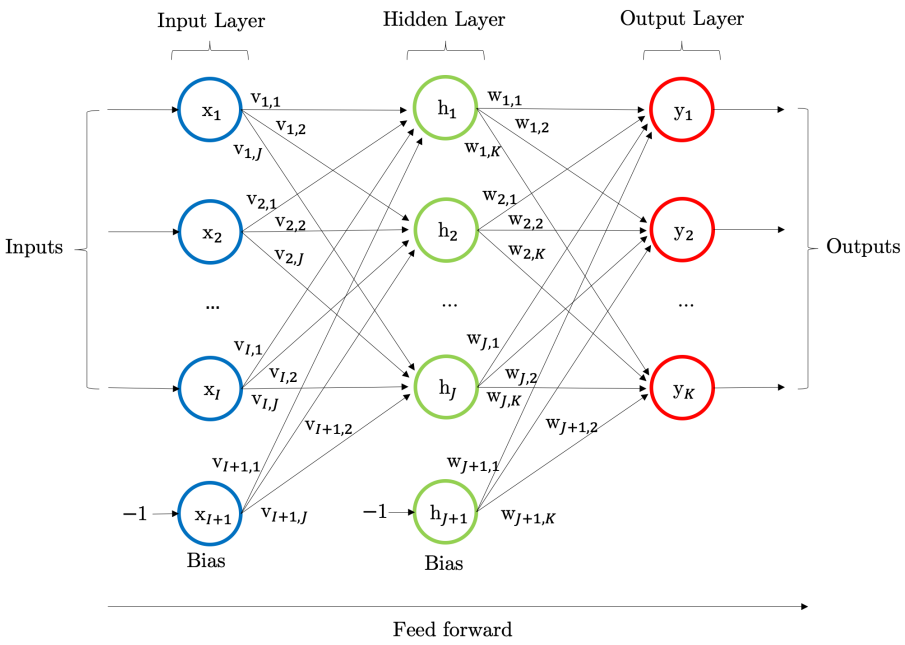


Figure 2.7: An illustration of a feedforward neural network.

In Figure 2.7, x_i refers to the i-th dimension in the input vector $\vec{x}$, h_j refers to the j-th dimension in the hidden layer, y_k refers to the k-th dimension in the output vector $\vec{y}$, $v_{i,j}$ 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.

With the layered architecture and fully connected topology of FFNNs, and assuming the use of SUs, the output for the FFNN at index k, denoted y_k, is calculated using:

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

The remainder of this dissertation makes use of FFNNs as the ANN of interest, sometimes referred to as *the model*.

2.4 Training

Section 2.2.3 mentioned that changing the weight associated with a feature, changes the influence of that particular feature on the predicted output. *Training* is the process whereby the weights of the FFNNs is systematically changed with the aim of improving performance.

Finding the optimal weights that produce the best output is an optimisation problem and therefore, training of FFNNs is done using a search function called a *heuristic*. Heuristics search for possible solutions in the solution-space and make use of information from the search-space to guide to process. Heuristics are discussed in detail in Chapter 3.

Details on the training of FFNNs are presented in this section along with brief discussions on the training process, generalisation capabilities, training sets, supervised learning, error functions, and performance measurement.

2.4.1 Training Process

During the training process, the FFNN is exposed to input data while trying to predict some expected outcome. From this prediction, a *cost* is calculated based on the degree to which the predicted output differs from the expected outcome. This outcome could be predefined and static, or be the result of an action in a dynamic environment. This cost is used by some heuristic as a guide to update the weights slightly such that the performance of the FFNN improves over time.

The training process is executed as follows: training data is retrieved and pre-processed. The pre-processed input data is then split into various datasets. A loss is calculated by exposing all patterns in the training data to the FFNN and evaluating the prediction in comparison to the expected or target output. Exposing the FFNN to all training data once is referred to as an epoch. The FFNN is trained for a maximum number of epochs, or until some stopping condition is reached. After training, the generalisation capabilities of the FFNN is evaluated against never before seen data.

Central to this dissertation is the topic of generalisation. Generalisation refers to the ability of a model to balance under- and overfitting of data. Training a FFNN using a HH aims to do exactly that. Overfitting and underfitting concepts are discussed next.

Overfitting

Overfitting describes a scenario where the trained model performs well on training data, but does not generalise well to never before seen data. [67, 180]. Geron [67] mentions that overfitting occurs when the model is too complex relative to the noisiness of the training data.

Underfitting

Underfitting is the opposite of overfitting. Geron [67] mentions that underfitting occurs when the model is too simple relative to the underlying structure of the training data.

2.4.2 Training Sets

The input data is split proportionally into a training and testing set. Data in the training set is used to train the FFNN [96], while the generalisation capabilities of the FFNN is evaluated over the performance of the trained model when exposed to never before seen data in the testing set. Some research propose a validation set whereby the trained model is evaluated on the validation data while hyper-parameters are tuned [20].

2.4.3 Supervised Learning

Supervised learning is the process of learning where the training data that is presented to the FFNN includes the desired solution [67]. The FFNN learns the mapping function from the input to the target output [18]. These desired solutions are referred to as *labels*. Supervised learning can be used for both classification and regression problems.

There are two types of supervised learning algorithms based on when weights are updated [54]. These include stochastic training and batch training and is discussed next.

Stochastic Training

Stochastic training, also known as *online learning*, is a supervised learning variation whereby weights are adjusted after each training pattern is presented.

If stochastic training is used to train the FFNN, the training data must be shuffled before splitting into sets, in order to avoid overfitting or memorising the order in which patterns are presented in the training set [54]. It has been shown that shuffling the training data can speed up convergence [11].

Batch Training

Batch training, also known as *offline learning* is a supervised training variation whereby the weight changes are accumulated and used to adjust the weights 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 [11]. Mini-batch training is the same as batch training, however, weights are updated after β patterns have been presented, where β is the batch size.

2.4.4 Error Functions

An error function, also known as a *loss/cost function* is used to evaluate the model's capability to approximate output data based on the input patterns presented. It measures the degree to which the model is capable of predicting the correct output data. This error value is used by the heuristic as a guide during the training process.

This dissertation focuses on the following error functions: *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). Each of these are discussed next.

Sum Squared Error

The SSE is computed using:

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

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 $\vec{y}_p$ for pattern p, P is the number of patterns in the training set, and K is the number of dimensions in the output vector $\vec{y}$.

Mean Squared Error

The MSE is computed using:

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

Root Mean Squared Error

The RMSE is computed using:

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

Mean Absolute Error

The MAE is computed using:

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

Binary Cross-Entropy

The BinXE is computed using:

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

BinXE is used in classification problems, where there are only two classes in the target output data.

Categorical Cross-Entropy

The CatEX is computed using:

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

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.

CatEX is used in classification problems where the target output or *label* is a one-hot encoded vector.

Sparse Categorical Cross-Entropy

The SparseCatXE error function is the same as CatEX with the only difference being that the target output or label is a one-hot embedding of a class represented as an integer, $c \in \{1, 2, \ldots, C\}$.

2.4.5 Performance Measures

Performance measures play an important role in understanding how well a FFNN has been trained to predict target output. Performance metrics are used during training on the training set and evaluation on the test set. These performance metrics provide a mechanism by which different implementations of FFNNs and training processes can be compared.

There are many different mechanisms that can be used to measure the performance of a FFNN, however this dissertation only focuses on measures of cost such as loss and speed. Brief discussions follow on loss and speed as performance measures.

Loss/Cost

A measure of cost indicates how well the FFNN is able to predict target output and often include a metric over some distance between the predicted output and target output. This metric is calculated using an error function as described in Section 2.4.4.

In the case of classification problems, the loss/cost can be expressed in terms of *accuracy*. Accuracy refers to the percentage of patterns for which the FFNN was able to successfully predict the target output.

Speed

Speed is a performance measure which calculates how fast a FFNN can reach some target goal. Speed is often expressed relative to another performance measurement metric e.g how many epochs of training is necessary for a FFNN to reach a certain lost/cost threshold.

For this dissertation, speed is a measure of performance relative to training *steps* or *mini-batch* steps.

2.4.6 Stopping Conditions

Stopping conditions are required to stop the training process. The goal of the stopping condition is to try and stop the training process to avoid unnecessary computational cost or to set a comparible timeframe in which FFNNs are trained. Stopping conditions include:

- **Max Epochs**: Stop when a maximum number of epochs has been reached.
- **Performance Measurement Threshold**: Stop the training when the performance measurement metric has reached some acceptable threshold.
- **Stagnation**: Stop the training process if no improvements in the performance of the model is observed over a window of epochs.
- **Overfit**: Stop the training process if the generalisation on the test set indicates that the FFNN is overfitting the data.

2.5 Summary

This chapter presented background information on the BN and it was shown how neurons connect to each other. Emphasis was put on synapses and how communication happens between neurons. The AN was introduced in Section 2.2 and it was shown how the brain inspired the development of ANs. Brief discussions followed on the various components that make up the AN. Details on input, weights, net input signal, biases, activation functions, and output were provided. Section 2.3 introduced ANNs. It was shown how ANs are connected together to form networks. ANN design was described in terms of architecture

and topologies. Special emphasis was put on FFNNs as the model being used throughout the rest of the dissertation. Section 2.4 provided the necessary background information required on the training of FFNNs. It is emphasised that the training of an ANN is an optimisation problem where optimal values for weights and biases are sought such that a given cost function is minimised. The training process was explained and details on training sets was provided. A variant of learning, called supervised learning was presented and led to the discussion on error functions, performance measures, and stopping conditions.

Chapter 3 introduces background information on optimisation and heuristics.

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*

So far the reader has been introduced to the detailed background information on ANNs. The training process was discussed without any reference to specific optimiser algorithms to execute the training process. Specific reference is made to supervised learning in the context of FFNN. This chapter aims to extend to this information by providing the reader with a set of optimisers that have been widely used to train ANNs. In the context of HHs, standardisation of terminology is in order and therefore these optimisers fill the role of low-level heuristics and will be referred to as heuristics from this point on.

Many different techniques have been used to train ANNs [107]. At the time of writing, the majority of work that is published around ANN training involves the use of some gradient-based technique [131]. However, research has shown that these gradient-based techniques do have their disadvantages such as slow-convergence or getting trapped in local optima [125]. Other techniques have also been shown to successfully train FFNNs. These include meta-heuristics such as PSO [145][188], DE [58] and GAs [77]. Various heuristics are considered for training FFNNs as the best heuristic to use is shown to be problem dependent in many cases [106].

Although there are many different types of heuristics, the list of heuristics specified in this Chapter is by no means exhaustive. Broadly speaking this chapter focuses on two different groups of heuristics. These include classical gradient-based approaches and population-based meta-heuristics. Each technique is discussed in detail. Pseudo-code algorithms are provided and where possible and discussions follow on their advantages, disadvantages, capabilities and limitations. This subset of heuristics were selected for their characteristics and were deemed appropriate and sufficient for the problem at hand. The remainder of this chapter is structured as follows.

- **Section 3.1** provides a high-level definition for heuristics.

- **Section 3.2** provides a brief overview of what optimisation is. It is also shown that FFNN training is an optimisation problem.
- **Section 3.3** presents the reader with the concept of gradient-based heuristics. The basics of *back-propagation* (BP) is provided and detailed discussions follow on seven different gradient-based low-level heuristics.
- **Section 3.4** presents the reader with three different meta-heuristics that are show to train FFNNs.
- Finally, a summary of the chapter is provided in **Section 3.5**.

3.1 What is a heuristic?

The term *heuristic* comes from the Latin word *heuristicus* which means "to find out, discover". Romanycia et al. [153] provides a complete study on the history and origins of the term *heuristic*. From their research, a proposal was made to define heuristics in the context of AI, as any device, be it a program, rule, piece of knowledge, which one is not entirely confident will be useful in providing a practical solution, but which one has reason to believe will be useful and which is added to a problem-solving system in expectation that on average the performance will improve.

In the context of this dissertation, the above definition holds, but can be more clearly specified such that a heuristic refers to an algorithmic search technique that serves as a guide to a search process where some good-performing candidate solution to a optimisation problem is being sought out. Different heuristics make use of different information during the search process [106]. Heuristics such as gradient-based techniques make use of the derivatives obtained by evaluating the ANN and thus it can be said that they make use of information directly from the *search space*. On the contrary, heuristics such as meta-heuristics make use of meta-information obtained as a result of evaluating the FFNN [15]. This could be the 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 [13] and are generally less problem-specific than other heuristic techniques [15].

In order to understand the application of heuristics, consider first the concept of optimisation and how that relates to training FFNNs.

3.2 Optimisation

Optimisation is the task of finding a solution to a given problem that is better than alternative solutions. Better stated by Oldewage [135], 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 [54] 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**: These affect 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 x.
- A **set of constraints**: These restrict and limit the values that can be assigned to the unknowns x. Optimisation problems that contain must adhere to a set of constraints are referred to as *Constraint Satisfaction Problems* (CSPs).

The goals of heuristics is to find a set of valid values for x that optimises the objective function while adhering to a set of constraints. Heuristics thus search for solutions, referred to as candidate solutions, in a set of all possible and valid solutions, referred to as the *feasible search space*.

Optimisation problems come in a wide variety of forms. Mostly these functions are 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 has to 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.

From the components of optimisation that was presented above, it can be said that the training process of an FFNN is an optimisation problem. Consider the relationship between optimisation and supervised training of FFNNs as follows. During training, the FFNN is evaluated by means of a loss/cost function as was presented in Chapter 2. In the context of optimisation, this loss/cost function is the object function to be minimised. For supervised learning specifically, it was mentioned that a training dataset is used to train the FFNN. The inputs to the ANN would be these training samples and thus serves the role of independent variables in the context of optimisation. The output of the ANN is directly influenced by the ANN model parameters, called the *weights* of the network and is thus referred to as the dependent variables. Finally, the goal of training ANNs is to find the configuration of *weights* such that the performance of the ANN is optimal.

The following sections present the various types of heuristics that are of concern to this dissertation and the implementation of the BHH.

3.3 Gradient-Based Heuristics

Gradient-based techniques are optimisation technique that makes use of the evaluated gradients of a model in order to optimise the model. Specifically in the context of a minimisation problem, these techniques are called gradient-*descent* heuristics as they minimise the loss/cost function. *Gradient descent* (GD) is generally attributed to Cauchy [116], who first suggested it in 1847. In 1907 Hadamard [79] independently proposed a similar method.

Although GD techniques where not the first techniques used to train FFNN [54], they are certainly the most popular. GD techniques have become increasingly popular partly due to their simplicity, but mostly due to their low computational overhead compared to other techniques such as meta-heuristics and other second-order derivative methods such as Newtons' method. There are many variants of this technique, however, they all fundamentally apply the same generic GD framework called *back-propagation*. The BP framework is provided next and all the other gradient-based techniques that follow are built upon this idea.

3.3.1 Backpropagation

GD techniques require the definition of some error function. In the context of training FFNNs, this error function is referred to as a loss/cost function. Consider the definition of the SSE was given in Chapter9 and is presented in Equation 3.1 below for convenience.

$$\epsilon = \sum_{p=1}^{P_T} (t_p - o_p)^2 \tag{3.1}$$

In Equation 3.1, t_p and o_p denote the target and ground truth output for the p-th training sample/pattern. P_t is the total number of input-target vector pairs in the training set [54].

The aim of GD is then to find some candidate solution for the weights of the FFNN that minimises the error ϵ. Engelbrecht [54] states that this is done by calculating the gradient of ϵ in *weight-space* and to move the weight vector along the negative gradient. An illustration of this is given in Figure 3.1.

The update-step for GD can then be formulated as is shown in Equations 3.2-3.4 below.

$$w_i(t) = w_i(t-1) + \Delta w_i(t) \tag{3.2}$$

where

$$\Delta w_i(t) = -\eta \frac{\partial \epsilon}{\partial w_i} \tag{3.3}$$

and

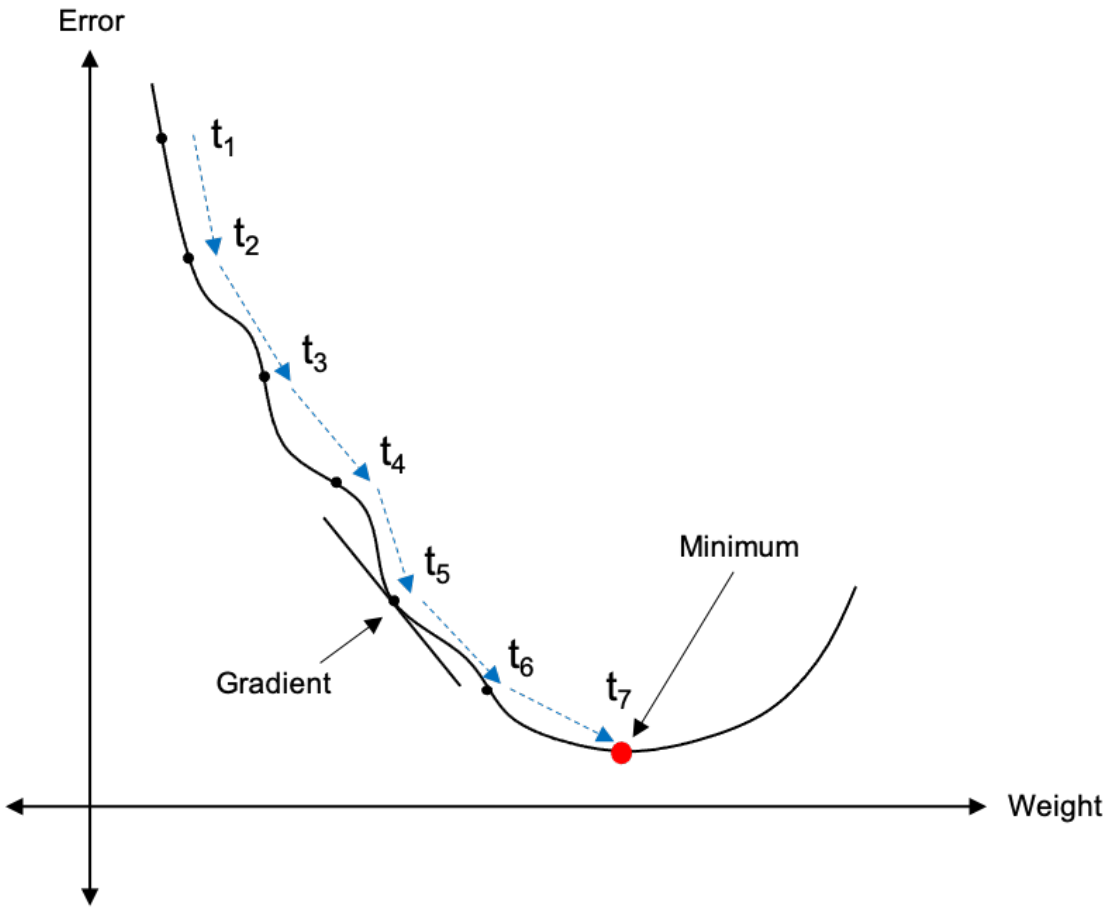


Figure 3.1: An illustration of GD over various timestep showing the minimisation of the error with regard to weight value.

$$\frac{\partial \epsilon}{\partial w_i} = -2(t_p - o_p)\frac{\partial f}{\partial net_p} z_{i,p} \tag{3.4}$$

In Equation 3.3, η is the learning rate. The learning rate is an important hyper-parameter in the GD heuristic as it controls the step-size that is taken in the direction of the negative gradient at each timestep.

In the context of FFNN training, this error signal is propagated backwards in the network. This algorithm is known as BP and was popularised by Werbos [193]. Nel [131] states that BP provides a procedure for updating the network weights by evaluating the derivatives of the error function ϵ with respect to the weights. BP consists out of the two phases [54]. These include:

- **Feedforward Pass**: During this phase the the output values if calculcated for the FFNN for each training pattern.
- **Backward Propagation**: During this phase the error signal that is calculated as is shown above in Equations 3.2-3.4 is propagated backwards from the output layer to the input layer of the FFNN. Weights are then adjusted as functions of the backpropagated error signal.

From the concepts mentioned above, the full BP process for training FFNNs can be formulated. Engelbrecht [54] presents such an example and is given next.

Backpropogation of Feedforward Neural Networks

In order to present the example for BP as applied to FFNNs, a number of assumptions are made. These include:

- The SSE is used as the objective function.
- The sigmoid activation function is used in both the input and output layers.
- Biases are included through augmented vectors as was presented in Chapter 2
- SUs are used in the input and output layers.
- Stochastic learning is used where a small subset of the training patterns, called a *mini-batch*, is presented at a time (SGD is discussed in further detail later in Section 3.3.2).

Then, for each training pattern z_p the output of the objective function is given in Equation 3.5 below.

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

In Equation 3.5, K represents the number of output units and $t_{k,p}$ and $o_{k,p}$ are respectively the target and output values of the k-th output unit.

The rest of the derivations are presented only for a single pattern. The subscript p is thus omitted for convenience. The output of each layer is given as

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}} \tag{3.6}$$

and

$$y_k = f_{y_k}(net_{y_k}) = \frac{1}{1 + e^{-net_{y_k}}} \tag{3.7}$$

The weights of the FFNN are then updated according to Equations 3.8 and 3.9 below.

$$w_{kj}(t) \quad += \quad \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1) \tag{3.8}$$

$$v_{ji}(t) \quad += \quad \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1) \tag{3.9}$$

In Equations 3.8 and 3.9 α represents the *momentum* and is discussed in more detail later in Section 3.3.3. The remainder of the derivations are focused on calculating $\Delta w_{kj}(t)$ and $\Delta v_{ji}(t)$. As such, the reference to time (t) is omitted for convenience.

From Equation 3.8 it follows that

$$\frac{\partial o_k}{\partial net_{o_k}} = \frac{\partial f_{o_k}}{\partial net_{o_k}} = (1 - o_k)o_k = f'_{o_k} \tag{3.10}$$

and

$$\frac{\partial net_{o_k}}{\partial w_{kj}} = \frac{\partial}{\partial w_{jk}} \left(\sum_{j=1}^{J+1} w_{kj} y_j \right) = y_j \tag{3.11}$$

In Equation 3.10 f'_{o_k} represents the derivative of the corresponding activation function. From Equations 3.10 and 3.11, using the chain-rule, it follows that

$$\begin{aligned} \frac{\partial o_k}{\partial w_{kj}} &= \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} \\ &= (1 - o_k)o_k y_j \\ &= f'_{o_k} y_j \end{aligned} \tag{3.12}$$

From Equation 3.5 it follows that

$$\frac{\partial \epsilon}{\partial o_k} = \frac{\partial}{\partial o_k} \left(\frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2 \right) = -(t_k - o_k) \tag{3.13}$$

Define the error signal that needs to be back-propagated as $\delta_{o_k} = \frac{\partial \epsilon}{\partial net_{o_k}}$. Then, from Equations 3.10 and 3.13 it follows that

$$\begin{aligned}\delta_{o_k} &= \frac{\partial \epsilon}{\partial net_{o_k}} \\ &= \frac{\partial \epsilon}{\partial o_k}\frac{\partial o_k}{\partial net_{o_k}} \\ &= -(t_k - o_k)(1 - o_k)o_k \\ &= -(t_k - o_k)f'_{o_k}\end{aligned} \tag{3.14}$$

The changes in the hidden-to-output weights are then computed from Equations 3.12, 3.13 and 3.14 as follows.

$$\begin{aligned}\Delta w_{kj} &= \eta\left(-\frac{\partial \epsilon}{\partial w_{kj}}\right) \\ &= -\eta\frac{\partial \epsilon}{\partial o_k}\frac{\partial o_k}{\partial w_{kj}} \\ &= -\eta\delta_{o_k}y_j\end{aligned} \tag{3.15}$$

Continuing on with the input-to-hidden weights as follows

$$\begin{aligned}\frac{\partial y_j}{\partial net_{y_j}} &= \frac{\partial f_{y_j}}{\partial net_{y_j}} \\ &= (1 - y_j)y_j \\ &= f'_{y_j}\end{aligned} \tag{3.16}$$

and

$$\begin{aligned}\frac{\partial net_{y_j}}{\partial v_{ji}} &= \frac{\partial}{\partial v_{ji}}\left(\sum_{i=1}^{I+1} v_{ji}z_i\right) \\ &= z_i\end{aligned} \tag{3.17}$$

From Equations 3.16 and 3.17 it follows that

$$\begin{aligned}\frac{\partial y_j}{\partial v_{ji}} &= \frac{\partial y_j}{\partial net_{y_j}}\frac{\partial net_{y_j}}{\partial v_{ji}} \\ &= (1 - y_j)y_j z_i \\ &= f'_{y_j}z_i\end{aligned} \tag{3.18}$$

and

$$\begin{aligned}\frac{\partial net_{o_k}}{\partial y_j} &= \frac{\partial}{\partial y_j}\left(\sum_{j=1}^{J+1} w_{kj}y_j\right) \\ &= w_{kj}\end{aligned} \tag{3.19}$$

From Equations 3.15 and 3.19 it follows that

$$\begin{aligned}\frac{\partial \epsilon}{\partial y_j} &= \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2\right) \\ &= \sum_{k=1}^{K}\frac{\partial \epsilon}{\partial o_k}\frac{\partial o_k}{\partial net_{o_k}}\frac{\partial net_{o_k}}{\partial y_j} \\ &= \sum_{k=1}^{K}\frac{\partial \epsilon}{\partial net_{o_k}}\frac{\partial net_{o_k}}{\partial y_j} \\ &= \sum_{k=1}^{K}\delta_{o_k} w_{kj}\end{aligned} \tag{3.20}$$

Define the hidden layer error which needs to be back-propagated from Equations 3.16 and 3.20 as

$$\begin{aligned}\delta_{y_j} &= \frac{\partial \epsilon}{\partial net_{y_j}} \\ &= \frac{\partial \epsilon}{\partial y_j}\frac{\partial y_j}{\partial net_{y_j}} \\ &= \sum_{k=1}^{K}\delta_{o_k} w_{kj} f'_{y_j}\end{aligned} \tag{3.21}$$

Finally, the changes to the input-to-hidden weights can then be calculated from Equations 3.18, 3.20 and 3.21 as follows

$$\begin{aligned}\Delta v_{ji} &= \eta\left(-\frac{\partial \epsilon}{\partial v_{ji}}\right) \\ &= -\eta\frac{\partial \epsilon}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}} \\ &= -\eta\delta_{y_j} z_i\end{aligned} \tag{3.22}$$

If the FFNN include direct weights from the input to output layer, the following additional weight updates are required.

$$\begin{aligned}\Delta u_{ki} &= \eta\left(-\frac{\partial \epsilon}{\partial u_{ki}}\right) \\ &= -\eta\frac{\partial \epsilon}{\partial o_k}\frac{\partial o_k}{\partial u_{ki}} \\ &= -\eta\delta_{o_k} z_i\end{aligned} \tag{3.23}$$

In Equation 3.23 above, u_{ki} is a weight from the i-th input unit to the k-th output unit.

The derivations above assumed stochastic learning, however, if batch learning is applied, weight updates are given in Equations 3.24 and 3.25 below

$$\Delta w_{kj}(t) = \sum_{p=1}^{P_T} \Delta w_{kj,p}(t) \tag{3.24}$$

$$\Delta v_{kj}(t) = \sum_{p=1}^{P_T} \Delta v_{kj,p}(t) \tag{3.25}$$

where $\Delta w_{kj}(t)$ and $\Delta v_{kj}(t)$ are weight updates for individual patterns p and P_T is the total number of patterns in the training set.

Equations 3.5 through 3.22 provided the reader with the detailed derivations of the BP weight-update components as it resolves from the error signal. The following sections provide the reader with variants of the BP algorithm as presented in this section.

3.3.2 Stochastic Gradient Descent

Section 3.3.1 already assumed the use of stochastic learning and thus provided the detailed derivation of a specific variant of BP called *stochastic gradient descent*. Separation of theory and implementation is done such that the detail of the theory need not be repeated for each gradient-based variant. This section shines light on the algorithmic implementation of BP with specific context to SGD. The algorithm is provided followed by a simplification of the update-step that provides the basis for the next gradient-based heuristics that follow.

The pseudo-code implementation for the SGD algorithm is taken from [54] and is presented in Algorithm 1 below.

Algorithm 1 The pseudo code algorithm for the SGD heuristic.

Initialise weights, η, α and the number of epochs $t = 0$;
while stopping condition not met **do** let $\epsilon_T = 0$
  for each training pattern p **do**
    Do the feedforward pass phase to calculate $y_{j,p}$ $(\forall j = 1, \ldots, J)$ and $o_{k,p}$ $(\forall k = 1, \ldots, K)$;
    Compute the output error signals $\delta_{o_{k,p}}$ and the hidden layer error signals $\delta_{y_{j,p}}$;
    Adjust the weights w_{kj} and v_{ji} (backpropagation of errors);
    ϵ_T += $[\epsilon_p = \sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2]$;
  end for
  $t = t + 1$;
end while

In order to provide a simplified frame of reference from which the gradient-based heuristics are compared, consider a simplification of the update-step for the weights of the FFNN as is done using SGD below.

$$w = w - \eta \nabla_w \mathcal{E}(w) \tag{3.26}$$

In Equation 3.26 w refers to the weights of the FFNN, η refers to the learning rate as before and $\nabla_w \mathcal{E}(w)$ refers to the gradient of the loss/cost function w.r.t the output of the FFNN as a result of its weights. Note that all subscripts w.r.t. layers and training patterns are omitted for convenience.

So far detailed mathematical descriptions have been provided to illustrate the concept of GD and how it applies to the SGD heuristic as applied to FFNN training. It was mentioned that SGD was one of the first widely used heuristics to train FFNN, however, it is not without flaws. With stochastic learning, only one training pattern is presented at each iteration. As such, weight updates are done with high variance and noise [156]. An illustration of the the fluctuations caused by SGD during training is given in Figure 3.2.

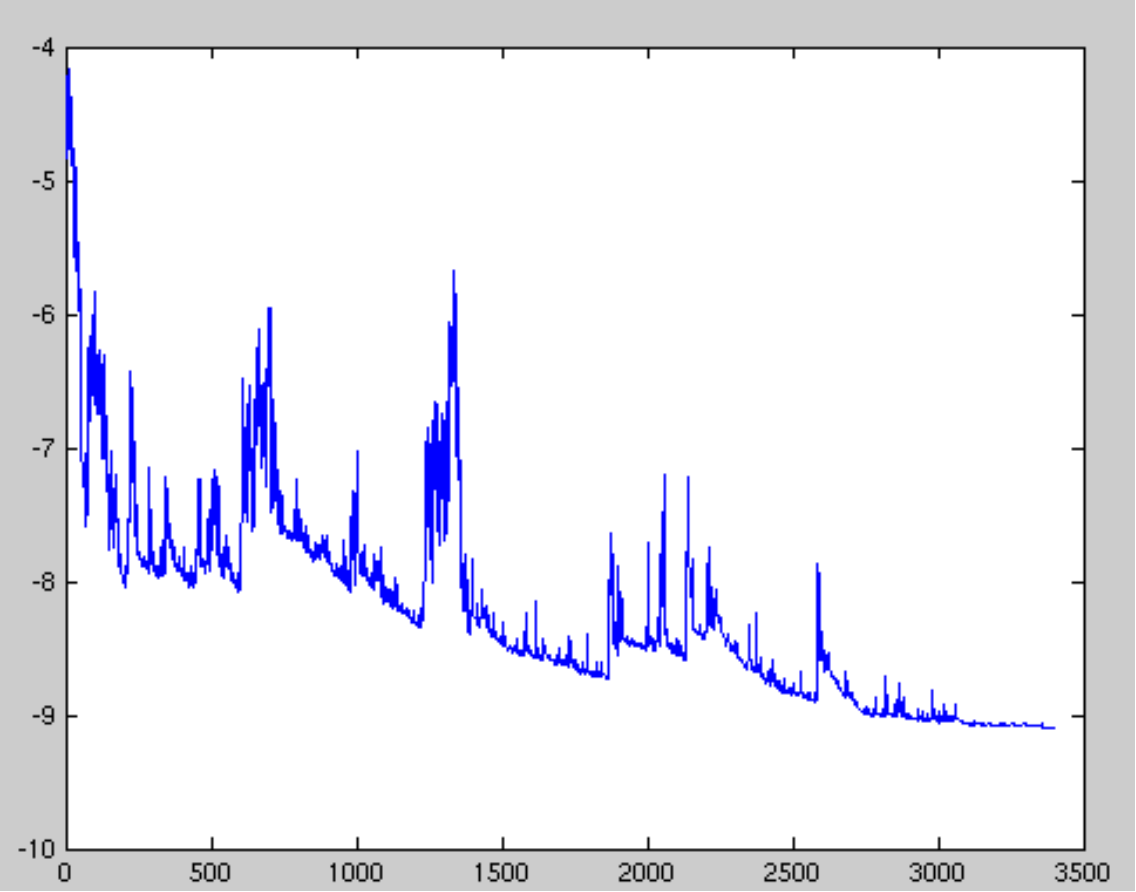


Figure 3.2: An illustration of SGD fluctuations during training as taken from [29].

With batch learning, all the training patterns are presented at once and the FFNN converges to the minimum of the loss/cost function w.r.t w. However, this is computationally expensive and intractable to implement in many situations. While SGD is able to jump out of local minima into new, potentially better local minima [156], it does at first this

seems advantageous. However, this complicates convergence to the exact minima as SGD can potentially keep overshooting better minima. It is shown that a slow decrease in the learning rate η will lead to the same convergence behaviour as with batch learning, almost certainly converging to local or global minima for non-convex and convex optimisation respectively. A compromise to this problem is to include mini-batches of training patterns where a small number of training patterns are presented to the FFNN at once. This means that the input patterns from mini-batches are just approximations of the total population of training patterns. According to Ruder [156] this has two main advantages

- Reduces the high variance of weight updates which leads to better convergence
- Allows for the implementation of SGD using highly optimised matrix operations, common to the state-of-the-art ML libraries used today.

In the context of this dissertation, SGD refers to the mini-batch learning implementation. Although mini-batch SGD does provide a compromise between single-pattern and batch learning there are still a number of challenges posed by this approach. 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 [151] 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 thus problem specific [37].
- The learning rate that has been introduced so far is applied to all weight updates. If the training data is sparse and the features have very different frequencies an equal update to all weights is inefficient. Larger weight updates are required for less frequently occurring features. This problem eludes to the credit assignment problem [157] common to these GD variants.
- It is difficult to avoid getting trapped in local minima, especially for highly non-convex loss/cost functions used for ANN training. [40] argues that this difficulty arises not from local minima, but from sadle points. Sadle points are points at which one dimensions slopes upwards while another dimension slopes downwards. [156] mentions that these sadle points are usually surrounded by plateaus of the same error, leading to gradients that are close to zero in all dimensions.

Alternative methods have been proposed that lead to better control over the convergence characteristics caused by GD variants. The first of these include *momentum* (Momentum) and is presented next.

3.3.3 Momentum

Research shows that SGD has difficulty navigating ravines i.e. areas where the surface curves much more steeply in one dimension than in another [178]. 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. An illustration of this is given in Figure 3.3.

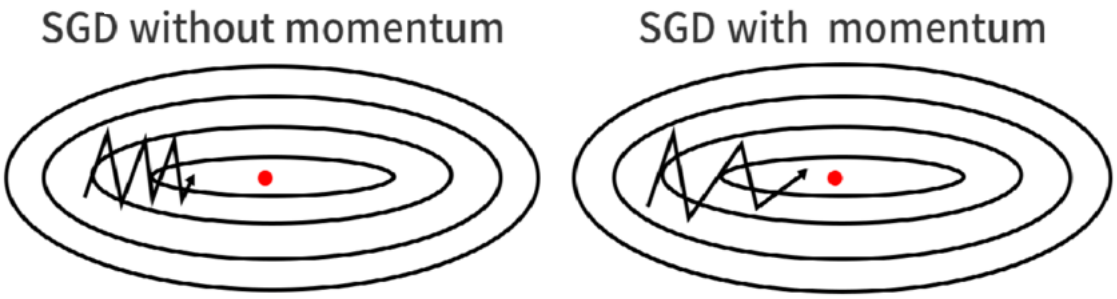


Figure 3.3: An illustration of SGD with and without momentum taken from [49].

Momentum [143] is a variant of the SGD that helps accelerate SGD in the relevant direction, dampening oscillations. [156] mentions that this is done by adding a fraction α of the weight update vector of the previous timestep to the current weight update vector. The accumulation of momentum is presented in Equation 3.27 while the update step is then amended in Equation 3.28 below.

$$v_t = \alpha v_{t-1} + \eta \nabla_w \mathcal{E}(w) \tag{3.27}$$

$$w = w - v_t \tag{3.28}$$

By redefining the SGD update steps as shown above, 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 [54][156].

3.3.4 Nesterov Accelerated Gradients

Nesterov accelerated gradients is a variant of the Momentum heuristics developed by Ilya Sutskever [176] and is inspired from Yurii Nesterov's [133] 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 [156]. An illustration of this is taken from Geoffrey Hinton's lecture on mini-batch GD [85] and is presented in Figure 3.4 below.

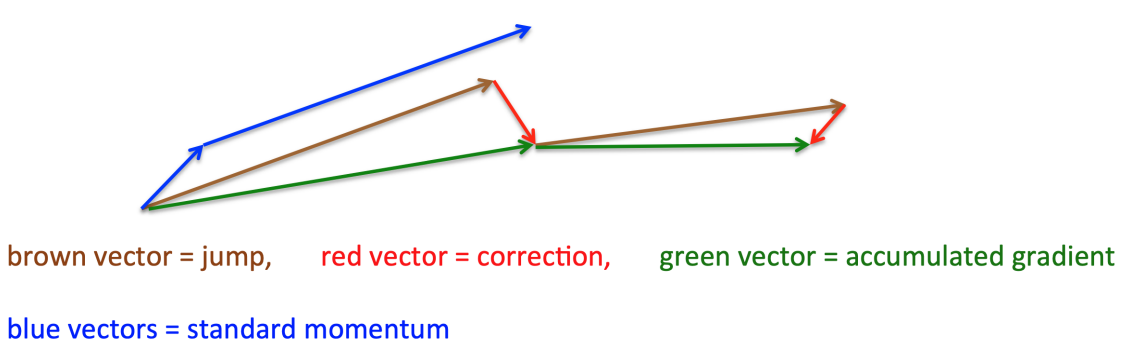


Figure 3.4: An illustration of the weight update vector for NAG taken from [85].

Ruder [156] explains the update step as is done by NAG is follows. Momentum first computes the current gradient. This is seen by the small blue vector presented in Figure 3.4. A large step is taken in the direction of the updated accumulated gradient, presented by the big blue vector. NAG then 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 then makes a correction (red vector), which results in the complete NAG update (green vector). By anticipating approximate future positions of the weights, the weight update step as defined by NAG controls the optimisation process from from going too fast and results in increased responsiveness. The NAG weight vector update steps are given in Equations 3.29 and 3.30 below.

$$v_t = \alpha v_{t-1} + \eta \nabla_w \mathcal{E}(w - \alpha v_{t-1}) \tag{3.29}$$

$$w = w - v_t \tag{3.30}$$

The next logical step in the improvement of the weight vector update step is to tailor the update steps for each weight parameter controlling the magnitude of the update step based on the relevance and importance for that dimension.

3.3.5 Adaptive Gradients

Adaptive gradients is an adaptation of SGD which implements a learning rate for every parameter in the weight vector and is developed by Duchi et al. [50]. Ruder [156] mentions that Adagrad adapts the learning rate to the weights, 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 with very sparse training data.

In the GD variants presented thus far, the weight updates have been applied to all weights at once using the same learning rate η for all dimensions of the weight update vector. Adagrad uses a different learning rate for every weight w_i at every timestep t. The weight update step as implemented by Adagrad is presented in Equation 3.31 below.

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} . \nabla_{w_i} \mathcal{E}(w_{t,i}) \tag{3.31}$$

where w_i is the i-th dimension of the weight vector, $G_{ii} \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element i, i is the sum of the squared of the gradients w.r.t. w_i up to timestep t and ϵ is a smoothing term that avoids division by zero, usually set in the order of 1×10^{-8} [156]. Since G_t contains the sum of the squares of the gradients with respect to all parameters w along its diagonal, the weight update step can be vectorised and updated using the matrix-vector product between G_t and $\nabla_{w_i} \mathcal{E}(w_i)$. This is presented in Equation 3.32 below.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla_{w_i} \mathcal{E}(w_t) \tag{3.32}$$

Adagrad's main benefit is that it does not require the need to manually tune the learning rate. However, a problem with Adagrad is the accumulation of the sums of squared gradients in the denominator. Since every term that is added is positive, this number keeps growing, leading to a situation where the learning rate shrinks to the point that the heuristic is no longer able to learn. *adaptive learning rate* aims to improve on this problem.

3.3.6 Adaptive Learning Rate

In Section 3.3.5 it was mentioned that one of the main problems of Adagrad is that it has an aggressively monotonically decreasing learning rate because of the accumulation of the sum of squared gradients in its denominator. Zeiler [199] presents an improvement on Adagrad that eliminates this problem. Ruder [156] mentions that instead of accumulating all past squared gradients as in the case of Adagrad, Adadelta 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 the sum of gradients over W steps is defined recursively as a decaying average of all past squared gradients. The moving average $E[g^2]_t$ at timestep t then depends only a fraction α of the previous average and current gradient, similar to the update step for Momentum. In order to simplify the notation of update steps, let $g = \nabla_{w_i} \mathcal{E}(w_t)$. The moving average update step for $E[g^2]_t$ is then presented in Equation 3.33 below.

$$E[g^2]_t = \alpha E[g^2]_{t-1} + (1-\alpha) g_t^2 \tag{3.33}$$

The value for α is set to a similar value, 0.9, as was done with Momentum. For sake of clarity, the basic SGD update step is rewritten in terms of the weight update vector Δw_t. This is presented in Equations 3.34 and 3.34 below.

$$\Delta w_t = -\eta g_{t,i} \tag{3.34}$$

$$w_{t+1} = w_t + \Delta w_t \tag{3.35}$$

The weight update vector for Adagrad as defined in Equation 3.32 is then rewritten as

$$\Delta w_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{3.36}$$

The diagonal matrix G_t is then replaced with the decaying average over the past W squared gradients as presented in Equation 3.37 below.

$$\Delta w_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{3.37}$$

Since the denominator is the *root mean squared* (RMS) error criterion of the gradient, the criteria short-hand notation is used as follows.

$$\Delta w_t = -\frac{\eta}{RMS[g]_t} g_t \tag{3.38}$$

Ruder[156] highlights that the original authors [199] noted that the units of the update step as presented in Equation 3.34 do not match i.e. the update vector should have the same hypothetical units as the weight vector. To realize this, another exponentially decaying average is defined in terms of squared weight updates as is presented in Equation 3.39 below.

$$E[\Delta w^2]_t = \alpha E[\Delta w^2]_{t-1} + (1-\alpha) \Delta w^2 \tag{3.39}$$

The RMS error of weight updates is thus

$$RMS[\Delta w]_t = \sqrt{E[\Delta w^2]_t + \epsilon} \tag{3.40}$$

Note that $RMS[\Delta w]_t$ is unknown at timestep t. It is thus approximated with the RMS of weight updates until the previous timestep. The learning rate in Equation 3.38 is then replaced as follows.

$$\Delta w_t = -\frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t \tag{3.41}$$

Finally, the weight update step for Adadelta is concluded as follows.

$$w_{t+1} = w_t + \Delta w_t \tag{3.42}$$

The main advantage of Adadelta is that one does not have to provide a default learning rate a priori as it is no longer included in the weight update step.

3.3.7 Root Mean Squared Error Propagation

Root mean squared error propagation (RMSProp) is a similar heuristic to Adadelta presented by Geoffrey Hinton [85] and was developed independently around the same time. Ruder[156] mentions that RMSProp is in fact the same as the first weight update vector as defined for Adadelta. The update step for RMSProp is then defined in terms of a decaying running average of gradients squared as is presented again for convenience in Equations 3.43 and 3.44 below.

$$E[g^2]_t = \alpha E[g^2]_{t-1} + (1-\alpha) g_t^2 \tag{3.43}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{3.44}$$

Note that RMSProp still includes the learning rate term η meaning that a default learning rate is required a priori. This is one of the disadvantages of RMSProp. Hinton suggests α be set to 0.9 and η be set to 0.001 [85].

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 [107]. However, 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 v_t, similar to Momentum [156]. Heusel et al. [84] uses the analogy that Momentum can be seen as a ball running down a slope, while Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

The decaying averages for past gradients and past squared gradients is given in Equations 3.45 and 3.46 respectively.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_t \tag{3.45}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_t^2 \tag{3.46}$$

In Equations 3.45 and 3.46 above, β_1 and β_2 are decay rates, similar to α for Momentum.

Ruder [156] mentions that v_t and w_t presented above are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. It is from these moment terms that the name *adaptive moment estimation* is derived. Kingma et al. [107] mentions that because v_t and w_t is initialised to be vectors of 0's, they are biased towards 0. This is especially true during the initial timesteps and/or when the decay rates β_1 and β_2 are small (i.e. β_1 and β_2 is close to 1). These biases need to be corrected. The bias-corrected first and second moment estimates are presented in Equations 3.47 and 3.48 below.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.47}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.48}$$

The Adam update rule is then presented in Equation 3.49 below.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.49}$$

Kingma et al. [107] suggest default values $B_1 = 0.9$, $B_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$. This section provided the reader with a collection of gradient-based heuristics. The next section introduces meta-heuristics.

3.4 Meta-Heuristics

Gradient-based methods are sensitive to the type of problem that they it applied to, with hyper-parameter selection often dominating the research focus [10][62]. In the context of HHs, it is important to consider a diverse set of low-level heuristics to select from. Apart from gradient-based methods, alternative heuristics must thus be considered. [15] mentions since the 1980's, a new kind of approximate algorithm has emerged which basically tries

to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods referred to as metaheuristics. This section aims to introduce the reader to the concept of meta-heuristics. Specifically, this section focuses on population-based meta-heuristics. Three well known meta-heuristics are presented. Each of these have been shown to train FFNN. These meta-heuristics include PSO, DE and GAs. [26] compared various PSO variants for training FFNNs. [58] compared DE and PSO when applied to FFNN training and [77] compared BP to a GA for training FFNNs.

A formal definition for meta-heuristics is required. The term *meta-heuristics* was first introduced by Glover[70] in 1986. The world derives from the composition of two Greek words. Heuristic derives from the verb *heuriskein* which means "to find". The suffix, *meta* means "beyond, in an upper level". Before this term was widely adopted, MHs were often called *modern heuristics* [149]. [15] mentions that there is a debate as to what the formal defintion of MHs is and suggests the definition of MHs to be high level strategies for exploring search spaces by using different methods. Blum et al. [15] provides characterists of meta-heuristics. These characterists are given as follows.

- MH are strategies to guide the search process.
- The goal of MHs is to efficiently explore the search space in order to find (near) optimal solutions.
- Techniques that constitute MH algorithms range from simple local search to complex learning processes.
- MH algorithms are approximate and usually non-deterministic.
- MHs may incorporate some mechanism to avoid getting trapped in local minima.
- The basic concept of MHs permit an abstract level description.
- MHs are not problem-specific.
- MHs may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced MHs use search experience (embodied in some form of memory) to guide the search/optimisation process.

Blum et al. [15] proposes the classification of heuristics as follows:

- Nature inspired vs. non-nature inspired.
- Population-based vs. single point search.
- Dynamic vs. statis objective functions.

- One vs. various neighbourhood structures.
- Memory usage vs. memory-less methods.

The first of these MH algorithms that are relevant to this dissertation include PSO. The concept of PSOs is presented and discussed next.

3.4.1 Particle Swarm Optimisation

PSO is a stochastic population-based search algorithm based on the social behaviour of birds in a flock [102]. By definition, the PSO heuristic is nature-inspired. PSOs were first presented by Kennedy et al.[102]. It was also Kennedy and Eberhart that first applied PSOs to train of FFNNs [53][101]. The application of PSOs in the context of training FFNN have been widely studied [145][188] and it has been found to perform well. This section aims to provide the reader with the detail of PSO implementation.

In general, this dissertation uses the term *entity* for candidate solutions and a *population* for a collection of entities. Engelbrecht[54] 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 hyperdimensional 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. It can be said that the position of a particle is thus influenced by the positions of other particles in the swarm resulting in a symbiotic cooperative heuristic. The social behaviour of particles is modeled such that they stochastically return to previously successful regions in the search space.

Van Wyk[188] mentions that the swarm is usually arranged in a predefined structure, called a *neighbourhood topology* that governs the communication between particles. Two different configurations of neighbourhood topologies exists. These 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 [51]. These include:

- Due to the larger particle interconnectivity of *gbest* PSO, the heuristic converges faster than with *lbest* PSO. [54] 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.[163] proposed a modification of the original PSO as was presented by Kennedy et al. Their implementation focuses on the *gbest* PSO with *inertia* weights. This dissertation focuses on their implementation of *global best* PSO and is discussed next.

The particles in this configuration has a number of properties associated with them [188]. 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 of particle i at timestep t be denoted by $x_i(t)$ and let N be the search space dimensionality.
- **Velocity**: Represents a step size for the particle in the search space. The velocity vector of particle i at timestep t is denoted $v_i(t)$.
- **Solution Quality**: Refers to the evaluation of the particle's position with respect to the objective function. Let $f(x_i(t)$ 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 $y_i(t)$.
- **Global Best Position**: Refers to a social memory construct where the 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 personal best position is denoted $\hat{y}_i(t)$.

During initialisation particles are randomly places within the search space by sampling from a uniform distribution such that $x_i \sim U(x_{min}, x_{max})$ and the velocity is set to vector of 0 such that $v_i = 0$. At timestep 0, the particle's initial position is set to be the particle's personal best solution such that $y_i(0) = x_i(0)$. The particle's update step is then broken into two parts. These include a velocity update step presented in Equation 3.50 followed by a position update step as presented in Equation 3.51 below.

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \tag{3.50}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3.51}$$

In Equations 3.50 and 3.51, i refers to the i-th particle in the swarm, j refers to the j-th dimension of vectors with dimensionality N defined by the optimisation problem. A breakdown of the components introduced above is required. The velocity update step consists of:

- **Previous Velocity**: Denoted by the term $wv_{ij}(t)$. This term represents the particle's momentum and is used to formulate an update step for the particle in the search space. Van Wyk[188] 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.[163] as a mechanism to control the exploration and exploitation abilities of the swarm. It was also introduced as a mechanism to eliminate velocity clamping (discussed later in the section). The introduction of inertia weight was successful in addressing the first objective, however did not quite provide a way to totally eliminate velocity clamping [162].

- **Cognitive Component**: Denoted by the term $c_1 r_1(t)[y_{ij} - x_{ij}(t)]$. This component represents the particle's personal experience. It introduces an attractor to the particle's personal best position so far. The cognitive component is stochastically scaled with random numbers $r1 \sim U(0,1)^N$ and the cognitive acceleration coefficient c_1 is used to control the influence of the cognitive attractor.
- **Social Component**: Denoted by the term $c_2 r_2(t)[\hat{y}_{ij} - x_{ij}(t)]$. This component represents the particle's social experience. It introduces an attractor to the swarm's best position so far. The social component is also stochastically scaled with random numbers $r2 \sim U(0,1)^N$, while also introducing the social acceleration coefficient c_2 that is used to control the influence of the social attractor.

A concept known as *velocity clamping* is mentioned above. Van Wyk[188] mentions that when PSOs where 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 issues, the concept of *velocity clamping* was introduced [51]. The idea behind velocity clamping is to restrict particle velocities to some V_{max} threshold, in essence, modeling a form of terminal velocity. Velocity clamping is applied after the velocity update step and is given in Equation 3.52 below.

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } |v'_{ij}(t+1)| < V_{max,j} \\ -V_{maxj} & \text{if } v'_{ij}(t+1) \leq -V_{max,j} \\ V_{maxj} & \text{if } v'_{ij}(t+1) \geq V_{max,j} \end{cases} \tag{3.52}$$

V_{max} then becomes another hyper-parameter that must be defined a priori. Appropriate values V_{max} may prevent swarm explosion, but also has an effect on the exploration and exploitation of the heuristic. If V_{max} is small, particle update steps are small, resulting in exploitation [51]. If V_{max} is big, it allows for larger update steps, promoting more exploration. A balance is needed between exploration and exploitation. Similar to other strategies, such as learning rate schedules for gradient-based heuristics, adaptive strategies such as those proposed in [59] have been developed, but are beyond the scope of this dissertation.

It should be noted that the choice of control parameter play a vital role on the behaviour and characteristics of the PSO. Van den Berg and Engelbrecht [184][185] 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 [52] 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 [54] and is given in Algorithm 2 below.

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

Create and initialise an N_x dimensional swarm;
while stopping condition not met **do**
  for each particle $i = 1, \ldots, N_s$ **do**
    // Set the personal best position;
    if $f(x_i) < f(y_i)$ **then**
      $y_i = x_i$;
    end if
    // Set the global best position;
    if $f(x_i) < f(\hat{y})$ **then**
      $\hat{y} = x_i$;
    end if
  end for
  for each particle $i = 1, \ldots, N_s$ **do**
    // Perform velocity update step;
    $v_{ij} = wv_{ij} + c_1 r_1[y_{ij} - x_{ij}] + c_2 r_2[\hat{y}_{ij} - x_{ij}]$
    // Perform position update step;
    $x_{ij} = x_{ij} + v_{ij}$
  end for
end while

This section provided the reader with the implementation details of the PSO heuristic. It was mentioned that PSO have been used to train FFNNs on many occasions with successful outcomes. The following section presents the reader with the next population-based MH that is considered.

3.4.2 Differential Evolution

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

by Storm and Price [142] in 1995. DE shares a lot of similarities with other evolutionary MH paradigms such as PSOs and GAs. However, DE differs significantly in the sense that it makes use of distance and direction information from the current population to guide the search process [54]. Originally, DE was focussed on multi-dimensional real-values optimisation problems, but unlike gradient-based heuristics, it does not require any gradient information. This means that DE does not require the underlying optimisation problem to be differentiable, meaning that it can be applied to problems that are discrete, noisy and dynamic [152].

Lots of research has been done on using DE to train FFNNs. Some notable work include [92][168][125]. 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 [54] mentions that for other EAs if both these operators are used, crossover is applied first after which the generated offspring is mutated. For other EAs mutation step sizes are sampled from some probability distribution. DE differs from these implementations in that

- mutation is applied first to generate a *trial vector*, which is then used within the crossover operator to produce one offspring and
- 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. This is under the assumption that entities are initially uniformly placed in the search space. DE aims to exploit this concept in order to find optimal solutions. There are three main components to the DE heuristic. These include mutation, crossover and selection operators [142]. Each of these are presented briefly next.

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 [54]. This trial vector is used in the crossover operator (discussed next) to produce offspring. The mutation process then follows as such. For each each parent $x_i(t)$ generate a trial vector $u_i(t)$ as follows.

- Select a target vector, $x_{i_1}(t)$ from the population the is not the same as the parent i.e. $i \neq i_1$.
- This is followed by randomly selecting two other individuals $x_{i_2}(t)$ and $x_{i_3}(t)$. Importantly, all of these entities must be unique such that $i \neq i_1 \neq i_2 \neq i_3$ and $i_2, i_3 \sim U(1, N_s$ where N_s is the size of the swarm.

- These individual entities are then used to calculate the trial vector by perturbing the target vector as presented in Equation 3.53 below.

$$u_i(t) = x_{i_1} + \beta(x_{i_2}(t) - x_{i_3}(t)) \tag{3.53}$$

In Equation 3.53 $\beta \in (0, \infty)$ is the scale factor and controls the amplification of the differential variation [54]. Note that in the above steps, the selection strategy to select the target vector is not specified. Selection is discussed in Section 3.4.2 below.

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, $u_i(t)$ (as was generated in Section 3.4.2 above) and the parent vector $x_i(t)$ to produce new offspring $x'_i(t)$. The crossover operator is given in Equation 3.54 below.

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in \mathcal{J} \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{3.54}$$

In Equation 3.54, $x_{ij}(t)$ refers to the j-th element of the vector $x_i(t)$ 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 [173][174]. These include:

- **Binomial crossover**: A crossover mask is generated by randomly selecting indices from the set of possible crossover points $\{1, 2, \ldots, N_x$ where N_x is the problem dimension. This techique is presented in Algorithm 3 below. 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 (presented in Chapter 5 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[54] states that with exponential crossover a sequence of adjacent crossover points are selected start from some randomly selected crossover index. This means that the set of possible crossover points $\mathcal{J}$ is a circular array in indices. This technique 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 this point, the next index is selected

until $U(0,1) \geq p_r$ or $|\mathcal{J}| = N_x$. p_r is the same crossover probability as mentioned above for binomial crossover. The implementation of exponential crossover is given in Algorithm 4 below.

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

$j^* \sim U(1, N_x)$;
$\mathcal{J} \leftarrow \mathcal{J} \cup \{j^*\}$;
for each $j \in \{1, \ldots, N_x\}$ **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.

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

Selection

Selection refers to the technique that is used to determine which entities are included in the mutation operator to produce a trial vector [54] as was shown in Section 3.4.2 above. Various selection operators have been suggested. With reference to the mutation operator, most DE implementations make use of random selection or the best entity is used as the target vector $x_{i_1}(t)$. 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 that the parent such that $f(x'_i(t)) \leq f(x_i(t))$. Engelbrecht[54] states that this is to ensure the average fitness of the population does not deteriorate.

General Differential Evolution Algorithm

Algorithm 5 is taken from [54] 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, β and p_r
while stopping condition not met do
    for each entity x_i(t) ∈ C(t) do
        Evaluate the fitness, f(x_i(t));
        Create the trial vector, u_i(t) by applying the mutation operator;
        Create an offspring, x'_i(t) by applying the crossover operator;
        if f(x'_i(t)) is better than f(x_i(t)) then
            Add x'_i(t) to C(t + 1);
        else
            Add x_i(t) to 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 [54]. 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 $(x_{i_2}(t) - x_{i_3}(t))$. 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, however this is beyond the scope of this dissertation.
- **Recombination Probability:** The probability of recombination, p_r has a direct influence on the diversity of the DE heuristic [54]. 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 value dynamically during optimisation.

DE/x/y/z Notation

Many variants of DE have been created and researched [122]. A general notation for DE heuristic variants have been developed by Storn et al. [173][174]. 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 is 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

This section provided the reader with the details around the implementation of the DE heuristic. The last population-based MH is presented next.

3.4.3 Genetic Algorithms

EC refers to a collection of nature-inspired optimisation algorithms that lend their foundation to biological evolution. Engelbrecht[54] 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. It was Charles Darwin's theory of *natural selection* that became the foundation of biological evolution [38]. Engelbrecht[54] summarises the Darwinian theory of evolution [39] 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 the reader has been introduced to two population-based MHs. 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 named *genetic algorithms*. The detail of the implementation of GAs is given

in this section. Importantly, the application of GAs is presented in the context of training FFNNs. GAs have been widely used to train FFNNs [127][164][124]. This section shines light on how this is done.

GAs where first proposed by Fraser [66] and later by Bremermann [17] and Reed et al. [146]. However, Holland [87] 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, modeling survival of the fittest and crossover, modeling reproduction. The remainder of this section follows the same approach as was done with DE where the different operators are discussed. However, it is necessary to first provide the reader with the generic EC algorithm. This algorithm is also referred to as the canonical GA (CGA) and was proposed by Holland [87]. The generic EC algorithm is taken from [54] and is is presented in Algorithm 6 below.

Algorithm 6 The pseudo code for the generic EC heuristic.

Let $t = 0$ be the generation counter;
Create an initialise an N_x-dimensional population, $\mathcal{C}(0)$, to consist of N_s individuals;
while stopping condition not met **do**
    Evaluate the fitness, $f(x_i(t))$ of each individual $x_i(t)$;
    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 chromosomes of some entity.
- **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.

Note that the initial implementations of EC heuristics such as GAs did not contain a mutation operator as this was only introduced later. 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 [54] and are based on arity of the operator i.e. the number of parents used for reproduction.

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

Engelbrecht[54] 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 focussed 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. Selection operators are discussed later in Section 3.4.3. 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 [54]. 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 N_x-dimensional crossover mask is generated randomly [179]. Uniform crossover is illustrated in Figure 3.5 below and the algorithm for uniform crossover is given in Algorithm 7.

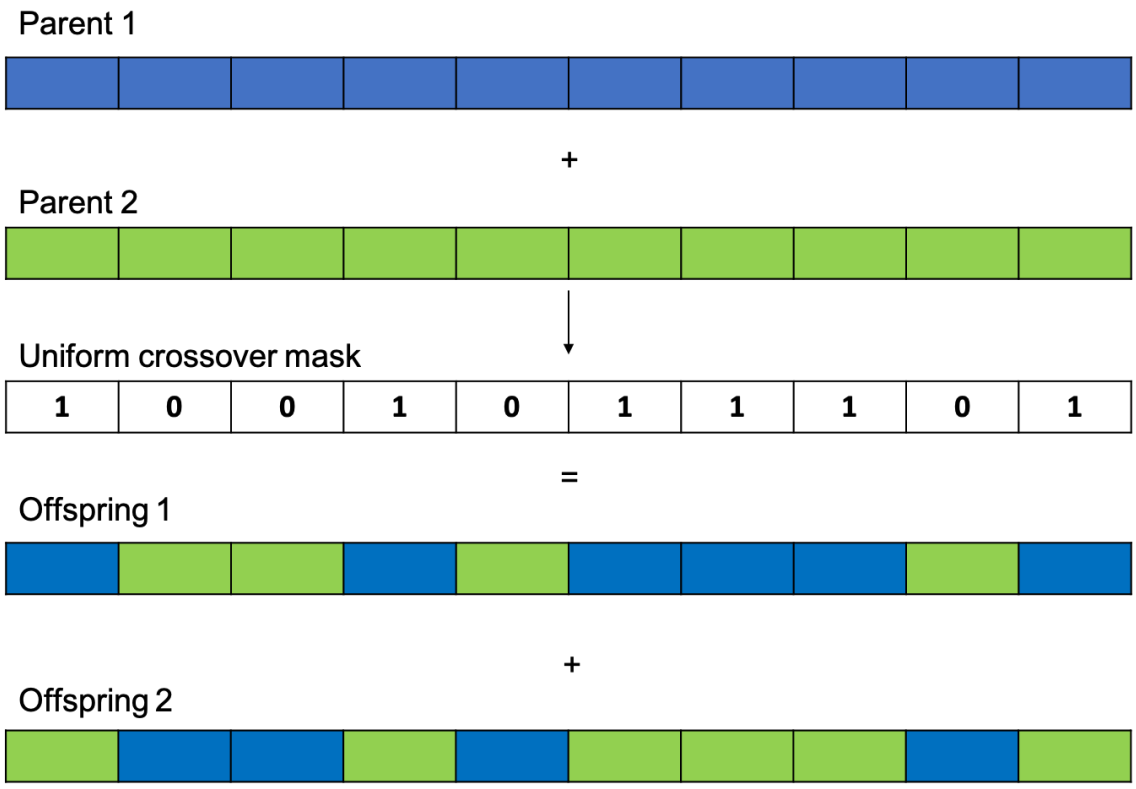


Figure 3.5: An illustration of the uniform crossover operator as it applies to sexual recombination, resulting in two new offspring.

Algorithm 7 The pseudo code for the uniform crossover operator as used by GAs.

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

In Algorithm 7 above, p_x is the bitswapping probability.

Mutation

The mutation operator is applied in order to introduce new genetic material into an existing entity [54]. 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 $x_i(t)$ to produce mutated offspring $x'_i(t)$. Engelbrecht[54] mentions that the mutation probability, also referred to as the mutation rate is 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, $v_{ij}(t) \sim U(-limit, limit)$ and $limit = \sqrt{\frac{6}{fanin+fanout}}$ as presented for Glorot uniform sampling in 2. An adaptation of the uniform mutation operator is then provided in Algorithm 8 below.

Algorithm 8 The pseudo code for the uniform mutation operator as used by GAs.

for $j = 1$ to N_x **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

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

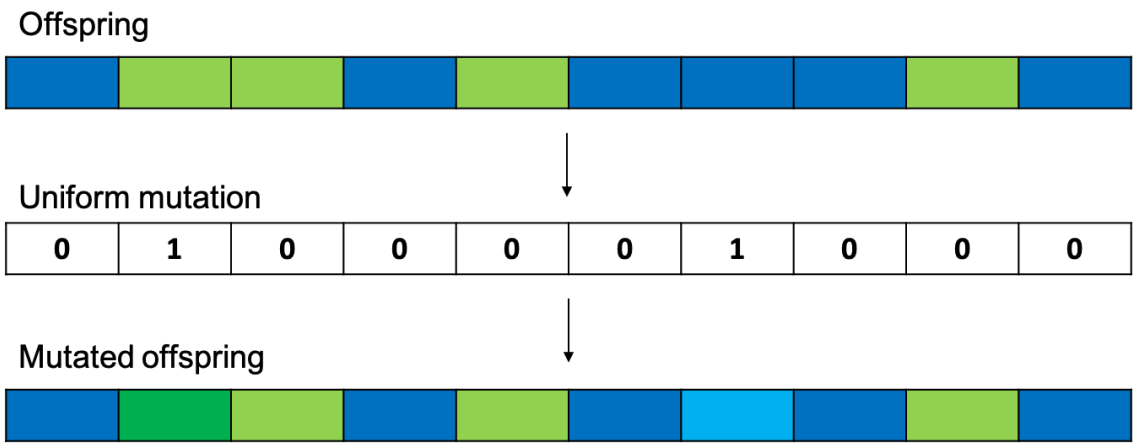


Figure 3.6: An illustration of the adapted uniform mutation operator as it applies to mutated offspring.

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 [54]. This is done in two of the main components of EAs. These include

- **Selection of the new population:** A new population of candidate solutions are is 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[54] 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 [6]. A selection operator with a high selective pressure rapidly decreases the diversity in the population. This could lead to premature convergence. A high selective pressure limits the exploration abilities of the population. Similar to other components and operators presented in this chapter, selection mechanisms 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_{ts} entities randomly from the population such that $N_{ts} < N_s$, where N_s is the population size. The performance of the selected N_{ts} entities are then compared and the best entities from this group is 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 the again for the second parent. Engelbrecht[54] mentions that tournament selection prevents the best entities from dominating provided that N_{ts} is not too large. This results in a lower selective pressure. If N_{ts} 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. It can therefore be said that selection is independent of actual fitness values. Engelbrecht[54] 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 $x_i(t)$ such that $i \sim U(0, U(0, N_s - 1))$. 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 $N_s - 1$.

- **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. It is later shown in 6 that the proposed BHH can incorporate a form of elitism in its credit assignment strategy.

Control Parameters

Throughout this section a number of control parameters have been provided that affect the performance of the GA. These include the population size N, the crossover rate p_c and the mutation rate p_m. Engelbrecht[54] mentions that these values are usually kept static. However, it is widely accepted that these parameters have a significant impact on the performance of the GA. Finding optimal values for p_c and p_m empirically can be a time consuming process. As such, dynamic parameter values can be introduced, however, this is beyond the scope of this dissertation.

3.5 Summary

This chapter provided the reader with detailed background information on different heuristics that have been shown to be able to train FFNNs. Two main groups of heuristics where discussed. These include gradient-based heuristics and *meta-heuristics*. For each of these groups a number of different variants have been discussed. A total of ten different heuristics have been introduced and where possible, their advantages, disadvantages and characteristics have been discussed.

The next chapter aims to build on the foundations provided in this chapter by putting focus on HHs. It is shown that MHs implement a heuristic pool of lower-level heuristics such as those presented in this chapter.

Chapter 4

Hyper-Heuristics

"Live as if you were to die tomorrow. Learn as if you were to live forever."
- Mahatma Gandhi

One of the biggest challenges for any ML researcher is that it is hard to find optimal configurations. These configurations often include the selection of appropriate models, heuristics and hyper-parameters. The optimal configuration is more often than not, problem-specific and it is difficult to find general approaches that can be applied to many situations. 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 illustrates a problem of generalisation of solutions. A modern approach to this problem is to automate this process. Automation of this process can be done by brute force where many different configurations are considered iteratively, sweeping over different configurations one after the other. Only after all configurations have been considered, can an optimal configuration be identified.

A different approach is to find optimal configurations by means of a search process. This means that a higher level optimisation process can be defined where the optimal configurations can be found as part of a search process. It is important to note that this search process is separate from the underlying problem to be solved. A measure of abstraction comes into play. In the context of training FFNNs, this is equivalent to finding the appropriate heuristic and hyper-parameters at various timesteps throughout the training process.

Automation alone is not enough. It is often the case that a single configuration is not sufficient. A modern suggestion is to include a hybridisation approach where multiple configurations are considered together. This is the nature of HHs. This is different from *ensemble* [45] approaches in the sense that HHs provide a single solution, where as *ensembles* make use of the combination of solutions that undergo a net weighted transformation in order to produce a final solution.

So far the reader has been presented with background information on ANNs and heuristics. This chapter aims to provide the reader with the necessary background information

on HHs. It should be noted that this is a short chapter as the foundations of heuristics have already been presented in Chapter 3. 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.
- **Section 4.3** provides a detailed discussion on the classification and types of HHs.
- **Section 4.4** sheds light on the applications of HHs in the context of training NNs.
- Finally, a summary of the chapter is provided in **Section 4.5**.

4.1 Meta-Learning

Meta learning is a branch of metacognition and is concerned with learning about the learning processes. Meta-learning studies how learning systems can increase in efficiency through experience and is broadly defined as the learning process concerned with the concept of *learning to learn* [189]. The goal of meta-learning is to understand if learning itself can become flexible to the problem domain or task under consideration.

Vilalta et al. [189] mentions that meta-learning differs from base-learning in the scope of the level of adaptation. Meta-learning is concerned with learning how to choosing the right configuration and biases dynamically. On the contrary, for *base*-learning, biases and configurations are predefined and fixed a priori.

4.2 What are Hyper-Heuristics?

The term *hyper-heuristic* was first used in 1997 [22] 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 [36] to describe "heuristics to choose heuristics".

Formal definition of HHs is required. Burke et al.[22] defines HHs as search methods or learning mechanism for selecting or generating heuristics to solve computational search problems. Burke goes on to say that 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 [21].

Grobler [74] states that HHs promote the design of more generally applicable search methodologies and tend to performing relatively well on a large set of different problems. This is in contrast to specialised algorithms which typically focus on outperforming the state-of-the-art for a single application.

Importantly, HH are concerned with finding the best heuristic in heuristic-space, while the underlying low-level heuristics find solutions in the feasible search space. This level of abstraction is illustrated in Figure 4.1 below.

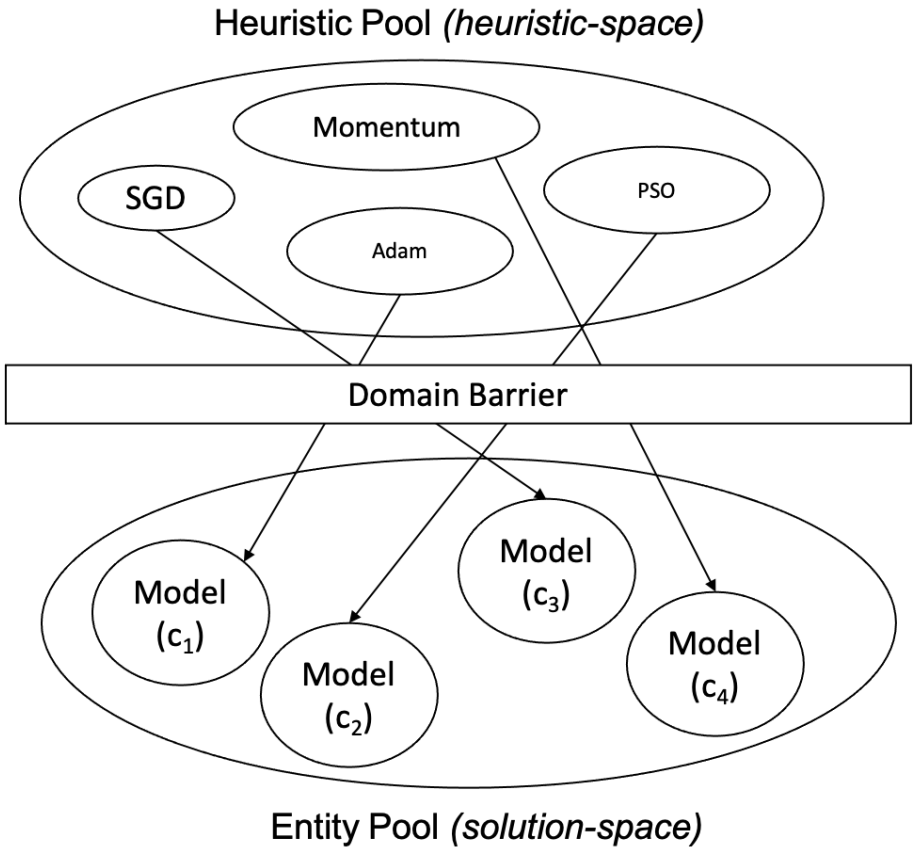


Figure 4.1: An illustration of abstraction introduced by HHs.

One can clearly see from Figure 4.1 the abstraction of the heuristic search process. The two large elliptical figures highlight the entity and heuristic pools, each of these are concerned with their own optimisation process. Note, that at the higher level abstraction, HHs do not make use of domain specific information such as an entity's position or gradient in the search space.

Grobler[74] mentions that two fundamental ideas behind the notion of HH can be identified. 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 [22].

4.3 Classification of Hyper-Heuristics

Burke et al.[22] proposed a modern classification scheme used to classify HHs. According to the proposed classification scheme, HHs are classified in two dimensions. These include the nature of the heuristic search space and the source of feedback during learning. Figure 4.2 was taken from [22] and summarises the classification scheme for HH.

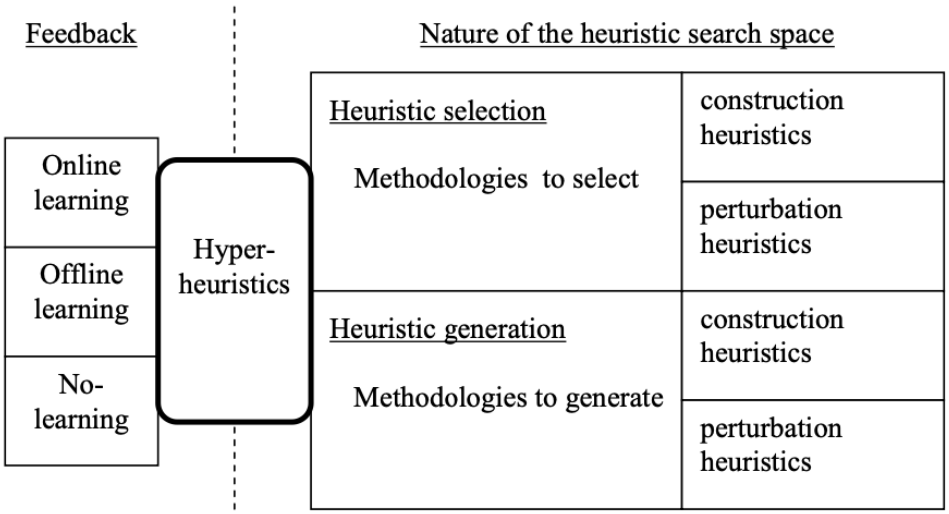


Figure 4.2: A classification of HH approaches, according to two dimensions: (i) the nature of the heuristic search space, and (ii) the source of feedback during learning.

Each of these groups are briefly discussed next.

4.3.1 Selection vs. Generation

According to the classification proposed by [22], the first dimension of classification involves the nature of the search space. Distinction is made between heuristic selection and heuristic generation. For heuristic selection HHs, methodologies are implemented for choosing or selecting existing low-level heuristics. On the contrary, for heuristic generation HHs, methodologies are implemented for generating new heuristics from components of existing heuristics.

Selection mechanisms can include probabilistic methods as is introduced in this dissertation or even some high level MH. For generation mechanisms, EA are often used [22].

4.3.2 Construction vs. Perturbation

The second dimension of classification distinguishes between construction heuristics and perturbation heuristics. Consider first construction heuristics. According to Burke et al.[22],

these 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. These heuristics are generally problem specific and the challenge is to select the heuristic that is somehow the most suitable for the current problem state. This process is repeated until some final state such as a complete solution, is obtained.

Perturbation heuristics 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. The goal here it to iteratively select and apply them to the current complete solution. This process continues until a stopping condition has been met. Notice that these approaches do not have a natural termination condition. This is one of the biggest differences between construction and perturbation heuristics.

- (i) low-level heuristic selection, and
- (ii) move acceptance strategy.

Heuristic selection can be done in a *non-adaptive* or simple way. This is done either randomly or along a cycle and is based on a prefixed heuristic ordering [36]. No learning is involved in these approaches. As an alternative, heuristic selection may incorporate an *adaptive*, *online learning*) mechanism based on the probabilistic weighting of the low-level heuristics [23] or some type of performance statistics [36]. It will later be shown that the proposed BHH implements this adaptive selection, perturbation strategy.

The acceptance strategy is an important component of any local search heuristic. [23] mentions that many acceptance strategies have been explored within HHs. Move acceptance strategies can be divided into two categories. These include *deterministic* and *non-deterministic*. In general, a move is accepted or rejected, based on the quality of the move and the current solution during a single point search. At any point in the search, deterministic move acceptance methods generate the same result for the same candidate solution(s) used for the acceptance test. However, a different outcome is possible if a non-deterministic approach is used. If the move acceptance test involves other parameters, such as the current time, then these strategies are referred to as non-deterministic strategies [23].

4.3.3 Online Learning vs. Offline Learning vs. No Learning

A HH is considered to be a learning algorithm when it uses some feedback from the search process. Therefore, non-learning hyper-heuristics are those that do not use any feedback.

With online learning HHs, learning continues to takes place while the algorithm is solving an instance of the underlying optimisation problem. Task-dependent local properties

can be used by the high-level strategy to determine the appropriate low-level heuristic to apply. Examples of online learning approaches within hyper-heuristics include the use of reinforcement learning for heuristic selection and, generally, the use of MHs as high-level search strategies across a search space of heuristics.

W.r.t offline learning HHs algorithms, the idea is to gather knowledge in the form of rules or programs, from a set of training instances. The hope is that it would generalise to the process of solving unseen instances. Examples of offline learning approaches within hyper-heuristics include *learning classifier systems*, *case-based reasoning* and *genetic programming*.

4.4 Application to Neural Network Training

Training of FFNN is notorious for being sensitive to configurations made beforehand. HHs provide a dynamic framework to train FFNN. However, at the time of writing, only two examples of published work where found where HH are used for training FFNN. These include the works of Nel [131] and Ul et al. [**ref:ul:2018**]

4.5 Summary

This chapter provided the reader with the necessary background information on HHs. A discussion on meta-learning is provided. The definition of a HH is given and modern classifications of HH types was discussed in detail. It was shown that HH can be used to train FFNNs.

This chapter then concludes the background information of heuristics in general. The next chapter aims to provide the reader with the necessary background information on statistics and probability theory. These concepts are required in order to formulate the proposed BHH.

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 17th century. In 1718, De Moivre [42] published the *Doctrine of of Chance*; a book that is widely seen as the first published book on probability theory. In 1763, Thomas Bayes [8] published an article titled *An Essay towards solving a Problem in the Doctrine of Chances* where the first version of Bayes' Theorem was introduced. It is from this theorem that the BHH was named after.

Probability theory and statistics play a vital role in the ML space. A lot of constructs that arise in ML originate from the fields of statistics and probability theory. There are many examples of how these two fields overlap. In 1991, Denker et al., [44] proposed a way to transform ANN outputs to probability distributions. In 1993, Neal [130] developed the first *Markov Chain Monte Carlo* (MCMC) sampling algorithm for Bayesian NNs. These are just a few examples of how the fields of probability theory, statistics and ML overlap.

So far the reader has been provided with the problem statement and goals of this dissertation in Chapter 1 and a detailed literature study and background information on ANNs was provided in Chapter 2. This chapter aims to provide the reader with the necessary background information on probability theory and statistics. These are big fields and focus is put on the elements that are required to formulate the proposed BHH. The remainder of the chapter is structured as follows.

- **Section 5.1** gives a brief overview of what probability is and how it is used.
- **Section 5.3** 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 5.2** introduces conditional probability.

- **Section 5.4** introduces Bayes' Theorem, the fundamental theorem upon which the BHH is built.
- **Section 5.5** presents relevant probability distributions.
- **Section 5.6** presents the conjugate prior probability distributions for the probability distributions provided in Section 5.5.
- **Section 5.7** presents *Bayesian* statistics. Brief discussions follow on the *frequentist* view and *Bayesian* view of probability. It is shown how Bayes' Theorem can be used to apply Bayesian statistics a alternative mechanism to do inference. Detailed discussions follow on Bayesian optimisation methods such as *Bayesian analysis*.
- Finally, a summary of the elements discussed in this chapter is provided in **Section 5.8**.

5.1 Overview

In everyday conversation, the term *probability* is a measure of one's belief in the occurrence of a future event [190]. 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 can not be predicted with absolute certainty [190]. Probability can be 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 $P(A)$. Although these random events cannot be predicted with absolute certainty, the relative frequency with which they occur over many trials if often remarkably stable. Consider flipping an unbiased, fair coin. The coin has two possible outcomes. From this, one can conclude that each side has a $\frac{1}{2}$ or 50% chance of occurring. In statistics, the decimal probability notation is used where $0 <= P(A) <= 1$. Suppose the fair coin is thrown 10 times, one is not guaranteed to observe 0.5 heads and 0.5 tails. There is 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. 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 [190]. 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 [190]. The belief that a random event would occur again sometime in the future, at an expected relative frequency could be useful for decision making. It will later be shown that the proposed BHH implements a probabilistic selection mechanism that continuously updates its beliefs such that it learns what the relative frequency by which a heuristic is selected should be.

Probability can also be expressed over multiple random events. For example, consider flipping a fair, unbiased coin and 6-sided dice together. One could then consider the probability of observing a certain outcome for the coin together with a certain outcome for the dice. Multiple random events can be considered together, dependently or conditionally. The following sections provides insight into conditional and joints probabilities.

5.2 Conditional Probability and Independence

The occurrence of a given random event A can often be dependent 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. This 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 5.2.0.1 below [190].

Definition 5.2.0.1 (**Conditional Probability**). *The conditional probability of an event A, subject to the occurrence of event B is expressed as*

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

where $P(B) > 0$. Notice 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 5.2.0.2 below.

Definition 5.2.0.2 (**Independence of Events**). *Two events, A and B are said to be independent of each other if, and only if the following criteria holds:*

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

otherwise events A and B are said to be dependent random events.

Events can also be considered together. The laws of probability for multiple events are presented next.

5.3 Two Laws of Probability for Multiple Events

Often times probabilities are calculated by considering multiple random events together. Suppose there are two random events A and B, then one can calculate the probability of the union and intersection of these events. From this concept, two laws of probability can be formulated. These are referred to as the *multiplicative* and *additive* laws of probability [190] 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*

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

If A and B are independent, then

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

Proof. Proof is given from Definition 5.2.0.1 above. □

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

$$P(A \cup B) = P(A) + P(B) - P(B \cap A)$$

Proof. The proof of the additive law of probability is supported by the Venn Diagram presented in Figure 5.1 below. Notice 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 $P(A \cup B)$, one can obtain the result $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. □

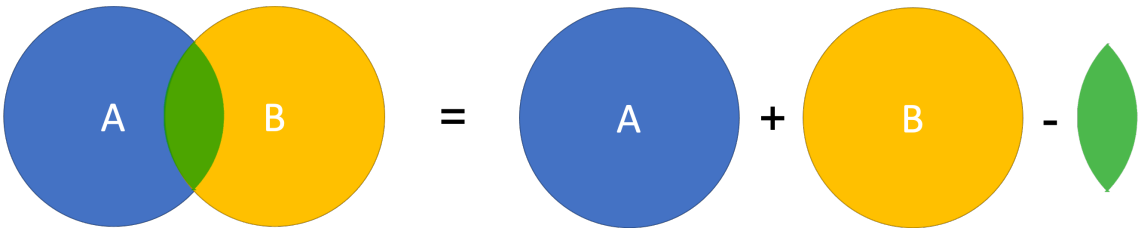


Figure 5.1: A Venn-Diagram showing the proof of the additive law of probability for multiple events.

The next section presents Bayes' Theorem and it is shown how it is related to conditional probability.

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 [198]. On order to derive the formal theorem and proof, first consider Definition 5.4.0.1 below [198].

Definition 5.4.0.1. *For some positive integer, k, let the sets $B_1, B_2, \ldots, B_k$ be such that*

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

Then the collection of sets $[B_1, B_2, \ldots, B_k]$ is said to be a partition of S.

Bayes Theorem is then presented in Theorem 3 below.

Theorem 3 (**Bayes' Theorem**). *Assume that $[B_1, B_2, \ldots, B_k]$ is a partition of S such that $P(B_i) > 0$, for $i = 1, 2, \ldots, k$ then*

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

Proof. The proof follows directly from the definition of of conditional probability as was discussed in 5.2. This is shown below

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

□

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

Bayesian inference is fundamental to Bayesian statistics. Section 5.7 presents the detail around Bayesian statistics. However, a brief detour is required to present to the user, a number of probability distributions. These probability distributions are presented next. Consider now a number of probability distributions as they are implemented.

5.5 Probability Distributions

Probability distributions are mathematical functions that give the probabilities of the occurrences of different possible outcomes in the experiment. They play a crucial role in the architecture and design of the BHH. It will later be shown how the proposed BHH makes extensive use of probability distributions and sampling to formulate its selection mechanism. The theory and equations presented in the following sections where all taken from Wackerly et al. [190].

5.5.1 Beta Distribution

The Beta distribution is a family of univariate continuous probability distributions over some x, with support on the interval $[0, 1]$ [190]. It is parameterised by two shape parameters $\alpha > 0$, $\alpha \in \mathbb{R}$ and $\beta > 0$, $\beta \in \mathbb{R}$. The Beta distribution is denoted as $Beta(\alpha, \beta)$. The *probability density function* (PDF) of the Beta distribution is given below in Equation 5.1.

$$P(x|\alpha, \beta) = f_{Beta}(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{5.1}$$

where the normalising constant $B(\alpha, \beta)$ is defined below in Equation 5.2.

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{5.2}$$

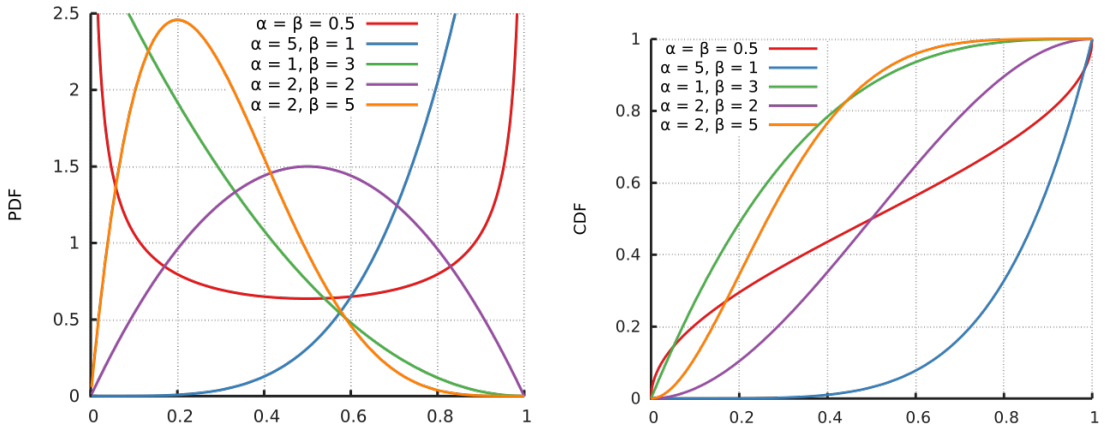
and Γ is the Gamma function as defined below in Equation 5.3.

$$\Gamma(n) = (n-1)! \tag{5.3}$$

it should be noted that the Gamma function can also be written as shown in Equation 5.4 below.

$$\Gamma(n+1) = n! \tag{5.4}$$

It should be noted that 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 distribution. In the case where $\alpha = \beta = 1$ the distribution is equivalent to the uniform distribution over all points in its support. The Beta distribution for various values of α and β, including the symmetric version is presented below in Figure 5.2.



(a) Beta distribution

(b) Cumulative Beta distribution

Figure 5.2: An illustration of the Beta distribution (left) [30] as well as the cumulative Beta distribution (right) [32] for various values of α and β .

The expected value of x is given below in Equation 5.5.

$$E[x] = \frac{\alpha}{\alpha + \beta} \tag{5.5}$$

Similarly, the expected value of the natural logarithm of x can be calculated as shown below in Equation 5.6.

$$E[\ln(x)] = \psi(\alpha) - \psi(\alpha + \beta) \tag{5.6}$$

where ψ is the logarithmic derivative of the Gamma function, called the *Digamma* function. The Digamma function is given below in Equation 5.7.

$$\psi(n) = \frac{d}{dn}\ln(\Gamma(n)) = \frac{\Gamma'(n)}{\Gamma(n)} \tag{5.7}$$

Finally, the mode of the distribution is given in Equation 5.8 below.

$$M[x] = E[x] - 1 = \frac{\alpha - 1}{\alpha + \beta - 1} \tag{5.8}$$

5.5.2 Dirichlet Distribution

The Dirichlet distribution is a family of multivariate continuous probability distributions over some x in K dimensions [190]. The Dirichlet distribution is the multivariate generalization of the Beta distribution and is thus sometimes referred to by its alternative name, the Multivariate Beta Distribution. The Dirichlet distribution is parameterised by some vector $\alpha = (\alpha_1, \ldots, \alpha_K)$ $\forall_{k=1}^{K}\alpha_k > 0$, $\alpha_k \in \mathbb{R}$. α is referred to as the concentration parameter. It is later shown in Chapter 6 that this parameter plays a vital role in the optimisation process of the BHH. The Dirichlet distribution of order $K \geq 2$ with parameters α, denoted $Dir(\alpha)$, has a PDF as given in Equation 5.9 below.

$$P(x|\alpha) = f_{Dir}(x; K, \alpha) = \frac{1}{B(\alpha)}\prod_{k=1}^{K} x_k^{\alpha_k - 1} \tag{5.9}$$

where the normalising constant $B(\alpha)$ is defined in Equation 5.10.

$$B(\alpha) = \frac{\prod_{k=1}^{K}\Gamma(\alpha_k)}{\Gamma(\alpha_0)} \tag{5.10}$$

and α_0 is defined in Equation 5.11 below.

$$\alpha_0 = \sum_{k=1}^{K}\alpha_k \tag{5.11}$$

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 $\forall_{k=1}^{K} x \in [0, 1]$. Under the simplex S, this means that the sum over all values of the vector x must be 1. The simplex can thus be rewritten as $\sum_{k=1}^{K} x_k = 1$.

Similar to the Beta distribution, α 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 distribution of order $K = 3$, for various values of α, including the symmetric version is presented in Figure 5.3 below.

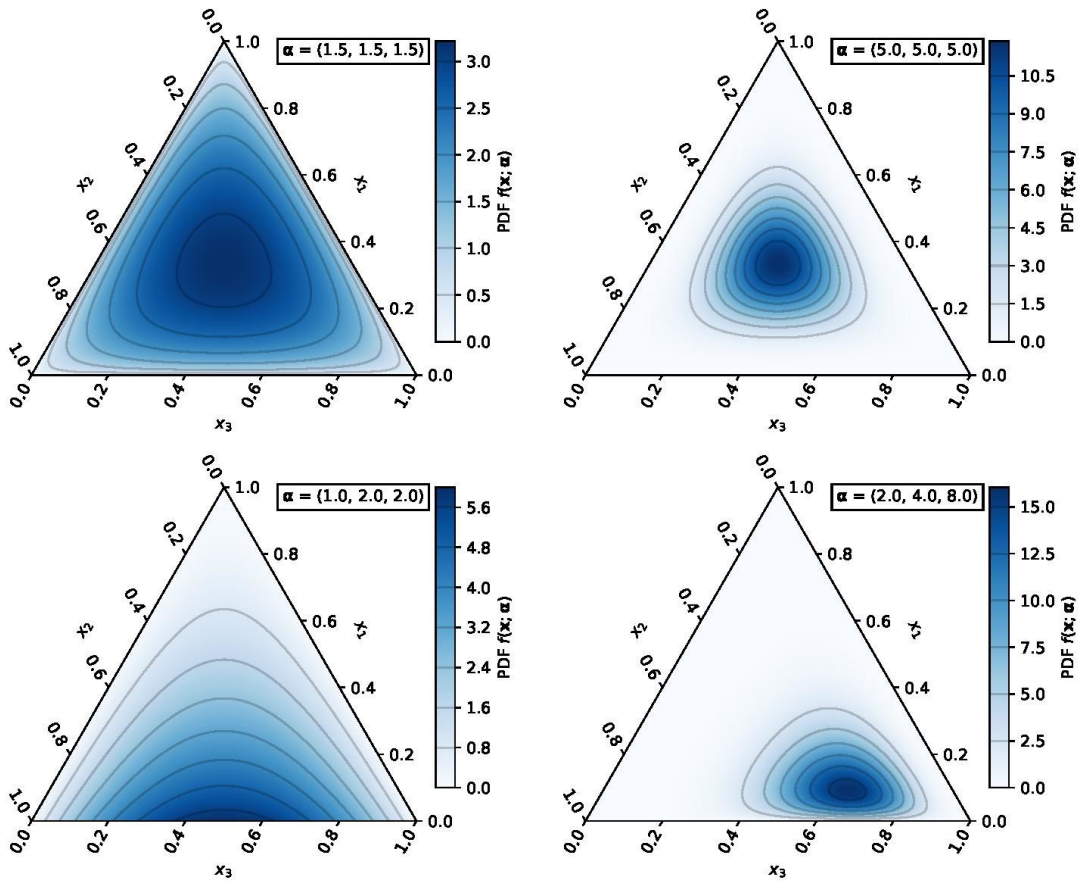


Figure 5.3: The PDFs for the Dirichlet distribution over the 2-simplex. The concentration parameter α is varied. The values of α are set to (1.5, 1.5, 1.5), (5, 5, 5), (1, 2, 2), and (2, 4, 8) respectively, read from top to bottom, left to right. The values of the PDF are shown by the color maps with contour lines at equal values as indicated in the color bars [31].

The expected value of x can be calculated as shown in Equation 5.12 below.

$$E[x_k] = \frac{\alpha_k}{\alpha_0} \tag{5.12}$$

Similarly, the expected value of the natural logarithm of x_k can be calculated as defined in Equation 5.13 below.

$$E[\ln(x_k)] = \psi(\alpha_k) - \psi(\alpha_0) \tag{5.13}$$

where ψ is the Digamma function as defined in Equation 5.7. Finally, the mode of the distribution is given in Equation 5.14 below.

$$\begin{aligned} M[x_k] &= E[x_k] - K^{-1} \\ &= \frac{\alpha_k - 1}{\alpha_0 - K} \end{aligned} \tag{5.14}$$

5.5.3 Bernoulli Distribution

The Bernoulli distribution is a discrete probability distribution over some random variable x that takes the value of 1 with probability θ and 0 with probability $1 - \theta$ [190]. It is denoted as $Ber(\theta)$ with support $x \in \{0, 1\}$. In probability theory, the Bernoulli 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 distribution has a *probability mass function* (PMF) as given below in Equation 5.15.

$$P(x|\theta) = f_{Ber}(x;\theta) = \begin{cases} \theta & \text{if } x = 1 \\ 1 - \theta & \text{if } x = 0 \end{cases} \tag{5.15}$$

The above equation can also be expressed as show in Equation 5.16 below.

$$f_{Ber}(x;\theta) = \theta^x (1 - \theta)^{1-x} \tag{5.16}$$

The mean of the Bernoulli distribution approaches θ over many samples according to the CLT [73]. Figure 5.4 shows a simulation of a fair-coin flipping simulation. One can see how the mean converges to 0.5 for various sample sizes.

The expected value of the distribution is thus given according to Equation 5.17 below.

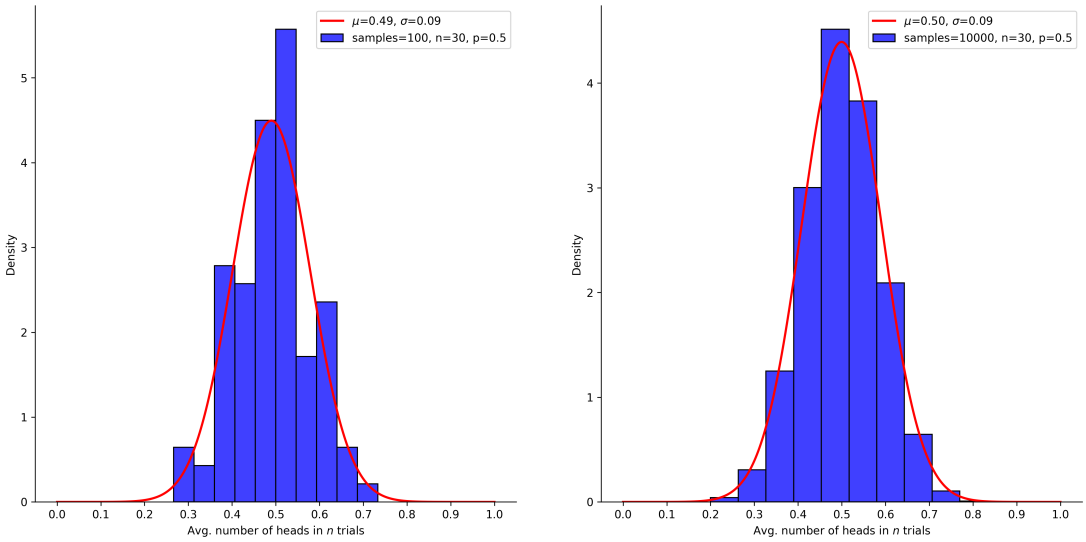
$$E[x] = \theta \tag{5.17}$$

while the mode of the distribution is given in Equation 5.18 below.

$$M[x_k] = \begin{cases} 0 & \text{if } \theta < 0.5 \\ 0, 1 & \text{if } \theta = 0.5 \\ 1 & \text{if } \theta > 0.5 \end{cases} \tag{5.18}$$

5.5.4 Binomial Distribution

The Binomial 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 [190]. The probability of a single independent experiment yielding a success is given as θ and the Binomial distribution is denoted as $Bin(N, \theta)$ with support



(a) 100 Samples (b) 1000 Samples

Figure 5.4: An illustration of the the coin flip situation for different sample sizes that show the convergence of the mean as per the CLT.

$x \in \{0, 1, \ldots, N\}$. It should be noted that the Binomial distribution is the extension of the Bernoulli 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 distribution. When $N > 1$, the sequence of outcomes is referred to as a Bernoulli process. The PMF of the Binomial distribution is given in Equation 5.19 below.

$$P(x|\theta; N) = f_{Bin}(x; N, \theta) = \binom{N}{x} \theta^x (1-\theta)^{1-x} \tag{5.19}$$

Similar to the Bernoulli distribution, the mean of the Binomial distribution is $N\theta$ given the CLT as was shown in Figure 5.4. The expected value of the Binomial distribution is thus given in Equation 5.20 below.

$$E[x] = N\theta \tag{5.20}$$

The mode of the distribution is given in Equation 5.21 below.

$$\begin{aligned} M[x_k] &= E[x] + \theta \\ &= N\theta + \theta \\ &= (N+1)\theta \end{aligned} \tag{5.21}$$

5.5.5 Categorical Distribution

The Categorical distribution is a discrete probability distribution over some random variable x, taking on any one of K possible categories [190]. There is no innate underlying ordering to these categories and therefore, for simplicity, each category is assigned a numerical representative value such that $k = (1, \ldots, K)$. The probabilities for all outcomes is given by the probability vector $\theta = (\theta_1, \ldots, \theta_K)$. This means that the probability $P(x = k) = \theta_k$ has support $x \in \{1, \ldots, K\}$. The Categorical distribution, denoted $Cat(\theta)$ is the generalization of the Bernoulli distribution and is sometimes referred to it by its alternative names, the generalised Bernoulli Distribution or the Multinoulli distribution. In probability theory, the Categorical distribution is often used to explain the outcome of a single experiment with more than two possible outcomes such as rolling a die [190]. The PMF of the Categorical distribution is given in Equation 5.22 below.

$$P(x|\theta; K) = f_{Cat}(x; K, \theta) = \prod_{k=1}^{K} \theta_k^{[x=k]} \tag{5.22}$$

where $[x = k]$ is the Iversion Bracket [93], yielding 1 if $x = k$ and 0 otherwise. From this, one can conclude that for a given class k, the categorical distribution simply yields θ_k as is shown in Equation 5.23 below.

$$f_{Cat}(x = k; K, \theta) = \theta_k \tag{5.23}$$

The random variable x can also be encoded in binary format, yielding a vector $x = (x_1, \ldots, x_K)$ of Bernoulli 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 distribution can then be rewritten as follows in Equation 5.24 below.

$$f_{Cat}(x; K, \theta) = \prod_{k=1}^{K} \theta_k^{\mathbb{1}_1(x_k)} \tag{5.24}$$

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

Since there is no innate order to the underlying categories, the mean of the distribution does not yield any relevant information. The mode of the distribution is given in Equation 5.25 below.

$$M[x] = \arg\max_{k}(\theta_1, \ldots, \theta_K) \tag{5.25}$$

5.5.6 Multinomial Distribution

The Multinomial distribution is a discrete probability distribution over some random variable $x = (x_1, \ldots x_K)$ that takes on the counts for each occurrence of K possible classes in N independent trials [190]. The probabilities for all possible outcomes in a single trial is

given by the probability vector $\theta = (\theta_1, \ldots, \theta_K)$. The Multinomial distribution, denoted $Mul(N, K, \theta)$, is thus the generalization of the Binomial distribution to K dimensions. Consider the following special cases.

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

The support for the Multinomial is $\forall_{i=1}^{K} x_k \in \{1, \ldots, N\}, \sum_{k=1}^{K} x_k = N$ and the PMF for the Multinomial distribution is given below in Equation 5.26.

$$P(x|\theta; N; K) = f_{Mul}(x; N, K, \theta) = \frac{N!}{\prod_{k=1}^{K} x_k!} \prod_{k=1}^{K} \theta_k^{x_k} \tag{5.26}$$

Similar to the Categorical distribution, the random variable x can also be encoded in binary format, yielding an $N \times K$ matrix X of Bernoulli distributions. The support is then given as $X \in \{0, 1\}^{N \times K}, \forall_{i=1}^{N} \sum_{k=1}^{K} x_{i,k} = 1$ so that the standard $K - 1$ probability simplex S still holds for each trial. The PMF of the Multinomial distribution can then be rewritten as is presented in Equation 5.27

$$\begin{aligned} f_{Mul}(x; N, K, \theta) &= \frac{N!}{\prod_{k=1}(\sum_{i=1}^{N} x_{i,k})!} \prod_{i=1}^{N} \prod_{k=1}^{K} \theta_k^{\mathbb{1}_1(x_{i,k})} \\ &= \frac{N!}{\prod_{k=1}(\sum_{i=1}^{N} x_{i,k})!} \prod_{k=1}^{K} \theta_k^{\sum_{i=1}^{N} \mathbb{1}_1(x_{i,k})} \\ &= \frac{N!}{\prod_{k=1}(\sum_{i=1}^{N} x_{i,k})!} \prod_{k=1}^{K} \theta_k^{N_k} \end{aligned} \tag{5.27}$$

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

The reason why the Categorical and Multinomial distributions are presented as binary encoded vectors is to simplify the proof of their conjugate priors as will be shown next. This is further supported by the fact that NNs often use binary encoding of feature vectors. The combination of these probabilistic methods and NNs forms the basic of this dissertation.

5.6 Conjugate Priors

Wackerly et al.[190] states that conjugate priors are prior probability distributions that results 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 for the Binomial likelihood and Categorical/Multinomial likelihood. Each of these are presented next.

5.6.1 Binomial Likelihood

The conjugate prior to a Bernoulli distribution is the Beta distribution [190]. 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 distribution.
- Let θ be the probability of a success. With $\theta|\alpha, \beta \sim Beta(\alpha, \beta)$.
- $P(\theta)$ is the prior probability distribution with the functional form $\mathcal{A}(v)$.
- Let $X = (x_1, \ldots, x_I)$ be the outcomes of independent, identical random events, each with boolean outcome. That is $x_i|\theta \overset{\text{iid}}{\sim} Ber(\theta)$ and thus $\mathcal{L}(x_i|\theta)$ is the Bernoulli likelihood.
- Let $\mathcal{D}$ denote all the prior data 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 then given in Equation 5.28 below.

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= P(\mathcal{D}|\theta) \\ &\propto \theta^{N_1}(1-\theta)^{N_0} \end{aligned} \tag{5.28}$$

By Bayes Theorem, the posterior distribution with given prior data $\mathcal{D}$ is given in Equation 5.29 below.

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})} \tag{5.29}$$

Since the denominator sums to 1, one could get rid of the denominator and constants for the Bernoulli likelihood and the Beta prior, by expressing the posterior as proportional to the likelihood times the prior as is shown in Equation 5.30 below.

$$\begin{aligned} P(\theta|\mathcal{D}) &\propto \left[\theta^{N_1}(1-\theta)^{N_0}\right]\left[\theta^{\alpha-1}(1-\theta)^{\beta-1}\right] \\ &\propto \theta^{(N_1+\alpha)-1}(1-\theta)^{(N_0+\beta)-1} \\ &\propto Beta(N_1+\alpha, N_0+\beta) \end{aligned} \tag{5.30}$$

It is show that the posterior distribution is of 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 distribution is the conjugate prior to the Bernoulli likelihood.

5.6.2 Categorical and Multinomial Likelihood

The conjugate prior to a Categorical and Multinomail distribution is the Dirichlet distribution [190]. Similar to the proof of the conjugate prior for the Bernoulli distribution as shown in Section 5.29 above, this means that the posterior distribution must have the same functional form $\mathcal{A}(v)$ as the prior distribution. This 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 $\alpha = (\alpha_1, \ldots, \alpha_K), \forall_{k=1}^{K} \alpha_k \in \mathbb{R}, \alpha_k > 0$ be the concentration parameters to the Dirichlet distribution.
- Let $\theta = (\theta_1, \ldots, \theta_K), \forall_{k=1}^{K} \theta k \in (0, 1), \sum_{k}^{K} \theta k = 1$ be the probability of each class in K and θ belongs to the standard $K-1$ probability simplex S. With $\theta|\alpha \sim Dir(K, \alpha)$.
- $P(\theta)$ is the prior probability distribution with the functional form $\mathcal{A}(v)$.
- Let $X = (x_1, \ldots, x_I)$ be the outcomes of independent, identical random events, each with K possible outcomes. That is $x_i|\theta \overset{\text{iid}}{\sim} Cat(\theta)$ and thus $\mathcal{L}(x_i|\theta)$ is the Categorical likelihood.
- Let $\mathcal{D}$ denote all the prior data X, parameterised by α.
- Let $N = (N_1, \ldots, N_K), N_k = \sum_{i=1}^{I} \mathbb{1}(x_{i,k} = 1)$, denote the counts for each occurrence of a class k.

The likelihood of the Categorical and Multinomial distributions is then given in Equation 5.31 below.

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= P(\mathcal{D}|\theta) \\ &\propto \prod_{k=1}^{K} \theta_k^{N_k} \end{aligned} \tag{5.31}$$

By Bayes Theorem, the posterior distribution with given prior data $\mathcal{D}$ is given in Equation 5.32 below.

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})} \tag{5.32}$$

Since the denominator sums to 1, one could get rid of the denominator and constants for the Dirichlet prior,by expressing the posterior as proportional to the likelihood times the prior as is shown in Equation 5.33 below.

$$
\begin{aligned}
P(\theta|\mathcal{D}) &\propto \prod_{k=1}^{K} \theta_k^{N_k} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \\
&\propto \prod_{k=1}^{K} \theta_k^{(N_k + \alpha_k) - 1} \\
&\propto Dir(K, N + \alpha)
\end{aligned}
\tag{5.33}
$$

Is it shown that the posterior distribution is of the same form $\mathcal{A}(v)$ as the prior, but with updated prior parameters $\alpha' = N + \alpha$. This shows that the Dirichlet distribution is the conjugate prior to the Categorical and Multinomial likelihood.

The next section aims to provide the reader with the concept of Bayesian statistics and it is shown how Bayesian inference and Bayesian analysis can be used to to train a HH.

5.7 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 happens [78]. To introduce the concept of Bayesian inference and analysis, one first need to understand the difference between these two approaches. Bayesian statistics out-date the frequentist approach, but lacked interest in the early days partly because of the limited applications where the conjugate priors where known [78]. More recent advancements in mathematical methods popularised the Bayesian approach again. A notable contribution to this switch was the invention of MCMC in the 1950's. 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 [78]. Soon after followed one of the earliest papers that use Bayesian statistics in the field of medicine in 1982 [5].

The difference between the frequentist approach and the Bayesian approach can be illustrated using an example. Hackenberger [78] suggests an experiment that investigates whether the sex ration in some hypothetical mice population is $1:1$. Two experiments can be 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 sex. 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 [9]. 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 $\alpha = 0.05$, one would conclude opposite outcomes for these two experiments when it comes to rejecting the null hypothesis, despite using the same data. The reason for this difference is because of the difference in their null distributions. The first approach uses a geometrical approach and the second used a binomial approach as is illustrated on Figure 5.5 below.

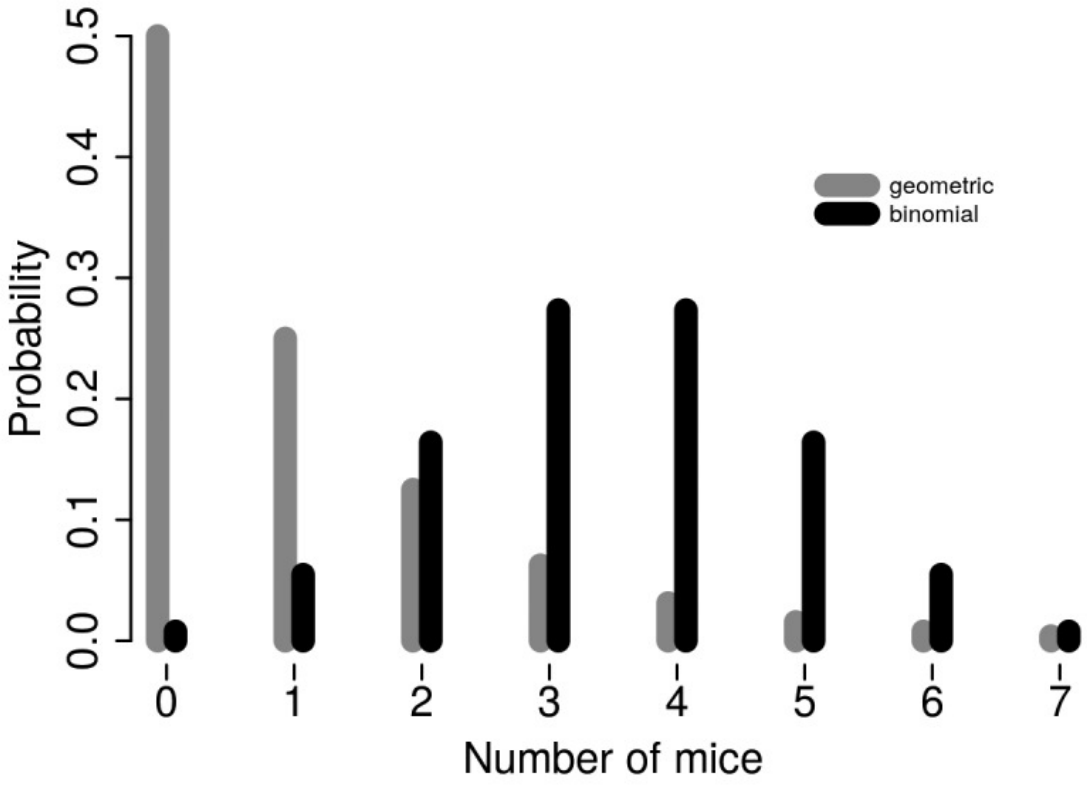


Figure 5.5: The experimental outcomes for the mice-population experiments as was taken from [78] .

If Bayesian statistics is used then the experimental design that was chosen does not matter. In Bayesian statistics it is common to use a Beta 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 $Beta(9, 4)$. The process by which this is done is shown later in this section. The $Beta$ distribution was presented in Section 5.5.1. Hackenberger [78] mentions that the $Beta$ distribution can be seen as a probability distribution of the occurrence of specific parameters. From the information that is now known about the $Beta$ distribution, it is possible to calculate the probability that the sex ratio in this mice population is not 1:1 with the $Beta$ 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 experiment design. This is illustrated in Figure 5.6 below.

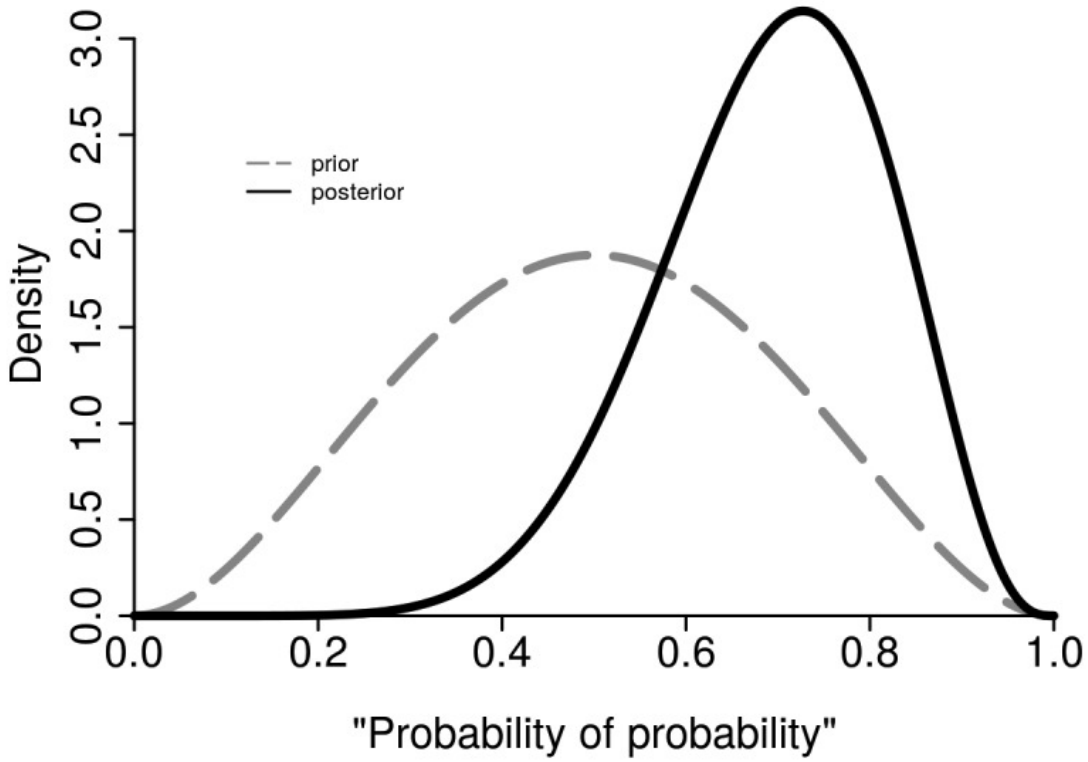


Figure 5.6: An illustration of the prior and posterior probability distributions for the outcomes of the mice-population experiment, using a *Beta* prior, as was taken from [78].

One can see from Figure 5.6 how the posterior distribution differs from the prior distribution. This is the nature of Bayesian analysis and is presented in more detail next.

5.7.1 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 [190]. Let $Y_1, Y_2, \ldots, Y_N$ denote the random variable that is observed over a sample size of N. Then the likelihood of the sample is given as $\mathcal{L}(y_1, y_2, \ldots, y_n|\theta)$. In the discrete case this function is defined to be the joint probability $P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_N = y_n)$ and the continuous case yields the joint density of $Y_1, Y_2, \ldots, Y_N$, evaluated at $y_1, y_2, \ldots, y_n$. The Bayesian view models the parameter θ as a random variable with a probability distribution. This probability distribution is 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, this 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 $Y_1, Y_2, \ldots, Y_N, \theta$ is given in Equation 5.34 below.

$$f(y_1, y_2, \ldots, y_n, \theta) = \mathcal{L}(y_1, y_2, \ldots, y_n|\theta)g(\theta) \tag{5.34}$$

The marginal density or mass function of $Y_1, Y_2, \ldots, Y_N$ is given in Equation 5.35 below.

$$m(y_1, y_2, \ldots, y_n) = \int_{-\infty}^{\infty} \mathcal{L}(y_1, y_2, \ldots, y_n|\theta)g(\theta)d\theta \tag{5.35}$$

Finally, the posterior density of $\theta|y_1, y_2, \ldots, y_n$ according to Bayes' Theorem is given in Equation 5.36 below.

$$g^*(\theta|y_1, y_2, \ldots, y_n) = \frac{\mathcal{L}(y_1, y_2, \ldots, y_n|\theta)g(\theta)}{\int_{-\infty}^{\infty} \mathcal{L}(y_1, y_2, \ldots, y_n|\theta)g(\theta)d\theta} \tag{5.36}$$

Wackerly et al.[190] mentions 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 used to update the priors on θ based on newly observed data. As with the example above, the below is taken from [190].

Let $Y_1, Y_2, \ldots, Y_N$ denote a random sample from a *Bernoulli* 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 formulation of the posterior distribution for p is then shown as follows. Since the *Bernoulli* probability function can be written as

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

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

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

Then

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

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

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

Finally, the posterior density of θ is given as

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

From the above, one can see that the posterior density has the same functional shape $\mathcal{A}(v)$ 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 the reader with all the necessary background on probability theory and statistics that is required to formulate the BHH. Detailed mathematical descriptions have been provided on probability distributions and proofs of conjugate priors are provided with detailed mathematical descriptions. Bayesian statistics have been presented in detailed and finally, it has been shown how Bayesian analysis works. The formulation of the BHH is provided in the next 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: How the Quest for the Ultimate Learning Machine Will Remake Our World.*

The above quote was the inspiration for the development of a novel HH that uses *Bayesian* probability concepts as a selection mechanism to drive the heuristic selection process. Thus far the reader was presented with all of the necessary background information on ANNs in Chapter 2, low-level heuristics/optimisers in Chapter 3, HHs in Chapter 4 and lastly, probability theory in Chapter 5. These elements form the fundamental components that make up the proposed BHH. A detailed specification on the BHH can now be formulated. This chapter provides the detail around the theory, concept and implementation of the BHH and explains how it used to train ANNs. It is shown throughout this chapter that the BHH implements a probabilistic optimisation technique that implements and biases a *Gaussian process* towards good optimisation and performance. The remainder of the chapter is structured as follows:

- **Section 6.1** provides a brief overview of the BHH optimiser.
- **Section 6.2** presents the general architecture, HH framework and the various components that make up the optimiser.
- **Section 6.3** discusses the collection of low-level heuristics, referred to as the *heuristic pool* and puts emphasis on the importance of a diverse set of low-level heuristics.
- **Section 6.4** discusses the detail of the population and its entities, with specific detail provided on entity (local) and population (global) state and memory.
- **Section 6.13** presents the initialisation strategy in detail.
- **Section 6.12** presents the *Bayesian* probabilistic model that is used as a selection mechanism.

- **Section 6.5** discusses heuristic-entity selections and how these two concepts are utilised together.
- **Section6.6** discusses the models that can be optimised using the BHH, specifically referring to its application to training FFNNs.
- **Section 6.7** briefly discussed the role of the train and test datasets during training.
- **Section 6.8** discusses model evaluation and the role of the loss/cost function.
- **Section 6.9** discusses the domain barrier that exists between the high-level BHH and low-level heuristics in the heuristic pool.
- **Section 6.10** presents the performance log and discusses performance measurement in more detail.
- **Section 6.11** discussed credit assignment and move acceptance strategies in detail.
- **Section 6.14** presents the learning mechanisms by which the probabilistic model can be optimised.
- **Section 6.15** summarises and discusses the associated hyper-parameters and default values.
- The pseudo-code algorithm for the BHH is given in **Section 6.16**.
- Finally, a summary of the chapter is provide in **Section 6.17**.

6.1 Overview

This section provides an overview of the workings of the BHH. To start the discussion, consider the following analogy:

Bob is a teacher at a high school. Every year his students perform really well and he wins the prize for best teacher, evaluated by student performance. He is able to do this not just because he is a good teacher, but he understands that each student has a learning technique that works best for him/her. He exploits this concept by tailoring the teaching process to each students such that the result is an optimum mark for that student and by definition, the whole class. Bob knows that it is too timely to get to know each student, so he devises a model that does the allocations for him. This model uses a process of observation. With each evaluation opportunity he experiments by allocating different learning techniques to different students. He evaluates the student for the given opportunity and keeps a log of which learning technique he assigned to the student as well as the outcome. With each evaluation opportunity he updates his model by biasing the assignment process such that students generally get assigned learning techniques that yield the best performance for them.

The key take-out from the analogy above is the concept that Bob can maximise student performance by updating his beliefs, over time, about which learning technique to assign to which (type of) student. He does this by biasing selection/assignment towards the best combinations of student, learning technique and evaluation opportunity. This is the basic nature of the BHH. The students are analogous to a population of entities. The learning techniques are analogous to heuristics/optimisers. Evaluation opportunities are analogous to optimisation problems to be optimised. Student performance is analogous to optimisation capability and performance measurement. The general goal of the BHH can be summarised as aiming to select and managing the appropriate heuristic/optimiser and update step for each entity, at each optimisation step, while optimising some underlying model, which for this dissertation, is to train FFNNs.

Formal classification of the BHH is needed. Burke et al. [22] proposed a classification mechanism that was discussed in detail in Chapter 4. The same classification mechanism is used to classify the BHH. For the first dimension of the classification, the authors propose that the BHH be classified as a *selection* HH. For the second dimension of the classification proposed, the authors propose that the BHH be classified as utilising *perturbation* low-level heuristics. Furthermore, the BHH is also classified as a population-based approach that includes meta-heuristics. Burke et al. [22] further proposed the classification of HHs according to the source of the feedback used during learning. It can thus be said that the BHH employs online learning according to this criteria, classifying it as an *adaptive* HH. A *population-based, selection, meta-hyper-heuristic that utilised perturbation of low-level heuristics and online learning* seems like an mouthful, but technically, that is it's full and formal classification. A breakdown and motivation of these classifications is given below:

- **Population-Based:** The BHH follows a population-based approach where a number of different candidate solutions, referred to as *entities* work together to yield a global best solution. There is also the concept of a shared global state or memory.
- **Selection:** The BHH implements a heuristic selection mechanism that selects from a collection of lower level heuristics' in a so-called *heuristic pool*. There is also a measure of acceptance criteria through credit assignment strategies.
- **Meta:** Incorporates the classical/analytical gradient-based low level heuristics (typically from the ML research space) as well as meta-heuristics such as other population-based meta-heuristics (typically EC research space).
- **Hyper-Heuristic:** There exists a domain barrier where the BHH searches through the *heuristic space* while lower level heuristics search through the *solution space*.
- **Perturbation of Low-Level Heuristics:** Refers to the iterative optimisation approach followed by the BHH. The BHH considers a number of randomly initialised

candidate solutions in the form of entities in a population. It is applied in an iterative manner, during the training process. It generally employs the two main processes of HHs with perturbation of heuristics. These include a *selection step* that selects from low-level heuristics and applies them to entities/candidate solutions. It also includes an *acceptance strategy*. The BHH maintains entity and population state through update step operations that proxy update steps from different heuristics. More details on this concept is given later in Section 6.3.2.

- **Online Learning:** Refers to the source of the feedback used during training. The BHH optimises, at a higher level, the selection and application of the *heuristic-space* while low-level heuristics optimise entities' solution to the underlying model, in this case a FFNN. This happens during training time, yielding the classification of utilising *online* learning.

More detailed consideration of these elements are required. The following sections break down the BHH into its various parts, starting with a discussion on the high-level architecture next.

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. [22] proposed an initial framework for HHs and Grobler [74] further proposed a framework for a heterogeneous meta-HH. These frameworks have been adapted for the implementation of the BHH. An illustration of the high-level architecture of the BHH is given in Figure 6.1 below.

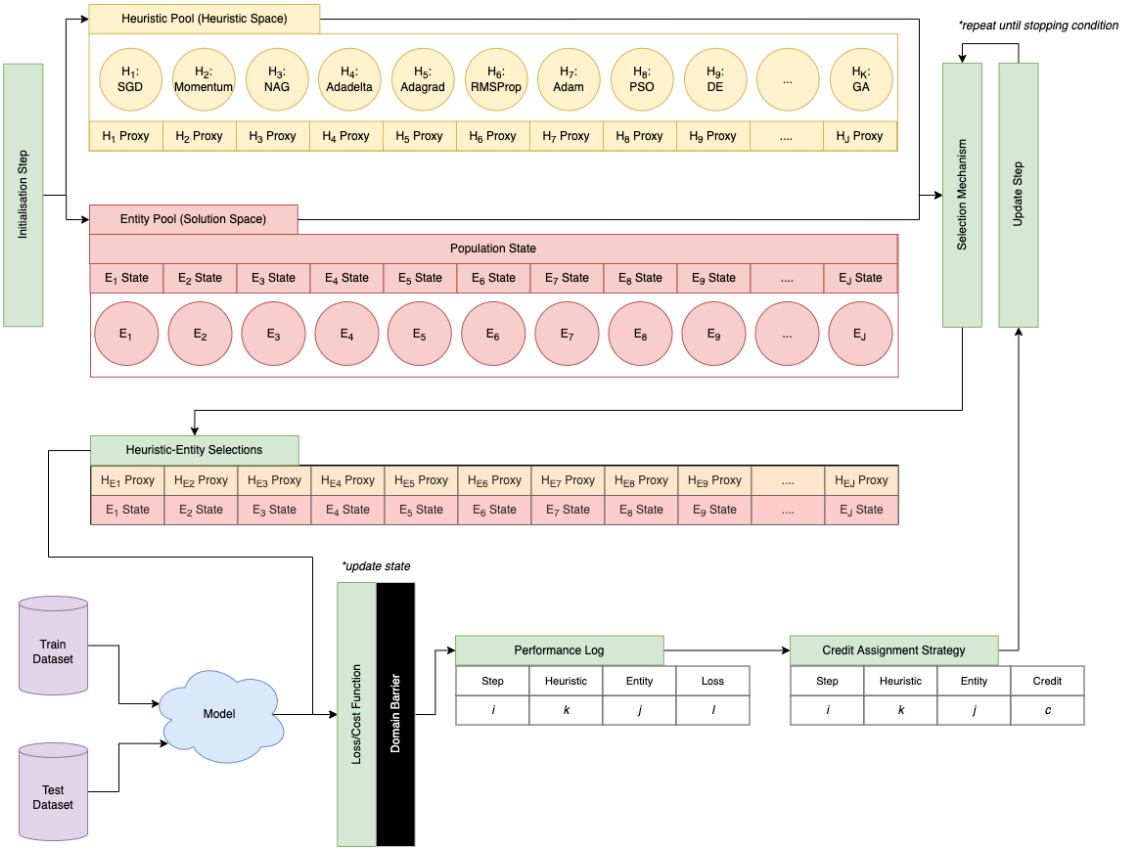


Figure 6.1: An illustration of the architecture and high level components of the Bayesian Hyper-Heuristic.

At a high level, the architecture presented in Figure 6.1 above, presents the following main components:

- **Heuristic Pool**: Contains a collection of low-level heuristics/optimisers. This is the *heuristic* space. Each heuristic has a proxy that is used to relay heuristic state update operations.
- **Entity Pool**: A collection of entities in a population. These entities represent candidate solutions to the model. Each entity has its own state, but there is also a shared global population state.
- **Initialisation Step**: This is the execution of some initialisation strategy that determines how the entities in the entity pool (population) and the heuristics in the heuristic pool are initialised.
- **Selection Mechanism**: This is the implementation of the selection mechanism of selection HHs. In the case of the BHH, the selection mechanism is a Bayesian probabilistic model.
- **Heuristic-Entity Selections**: This is the results of the selection mechanism. Heuristics are sampled/selected for each entity. Each entity's state will thus be

manipulated by that particular heuristic's proxy methods.

- **Model**: The model to be optimised. This forms the problem domain. In the case of this dissertation, the model is a FFNN.Technically the model here simply refers to the architectural structure of the model, while entities represent that candidate solution (weights) to the model.
- **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.
- **Loss/Cost Function**: The loss/cost function is the metric by which the model is evaluated. In the case of supervised learning, the loss/cost function is a measure of distance between predicted output and the ground truth.
- **Domain Barrier**: A constraint on information flow, prohibiting high level components in the *heuristic* space from interacting and relaying information related to the *solution* space.
- **Performance Log**: Contains a record of heuristic-entity performance at each step of the training process.
- **Credit Assignment Strategy**: Implements a specific strategy to determine if the performance of an entity-heuristic combination is better than others or not.
- **Update Step**: The high level heuristic update step. This involves the mechanism by with the BHH is optimised.

The above components are not described in enough detailed to properly illustrate their role in the BHH. To ensure proper clarity is given to each of these elements, a detailed discussion on each of these elements are presented next.

6.3 Heuristic Pool

This section discusses the concept of the heuristic pool, as implemented by the BHH. Generally speaking, they heuristic pool is just a collection of low-level heuristics and make up the set of possible heuristics to select from. This is referred to as heuristic space. Importantly, since HHs optimise in heuristic space, it is from these lower-level heuristics that the BHH must select and apply. The heuristic pool can then be defined in terms of size and/or heuristics that are included in the pool. These two aspects pose very important design choices. Both of these concepts can be empirically tested, however, one could argue the choice of design. Fundamentally, these design choices are about diversity. A brief discussion is given next on the trade-offs between exploration, exploitation and capability.

6.3.1 Diversity: Exploration, Exploitation and Capability

As with all HHs, the heuristics to be included in the heuristic pool is of great importance and should be carefully decided upon. A heuristic's eligibility for inclusion is determined by the following factors:

- **Capability**: Certain heuristics are only capable of solving or optimising for certain problem types and domains. Examples include continuous vs. discrete, differentiable vs. non-differentiable or static vs. dynamic problems.
- **Exploration and Exploitation Trade-off**: Some heuristics explore or exploit more than others. This could be due to the nature of the implementation or due to some hyper-parameter such as a learning rate.

An argument for diversity must thus be made. One of the benefits of HHs in general is that it will learn which heuristics to apply and when. The trade-off between exploration and exploitation should thus be outsourced to the BHH itself. By including a diverse set of heuristics, based on the criteria mentioned above, the BHH will learn to select and apply the appropriate heuristic at the appropriate time throughout the training process, naturally balancing a trade-off between exploration and exploitation. Heuristics that initially explore a lot should be selected when exploration is needed and heuristics that exploit a lot should be selected accordingly.

Notice however that diversity can not be achieved by simply including many different heuristics. The BHH is still constraint by heuristic pool size. Consider that for every heuristic that is included, the demand for samples size required to learn with statistical certainty exponentially increases. Not only does a large heuristic pool drastically complicate the learning that is required by the BHH, but it also drastically complicates the process of maintaining state. The next section provides the concept of heuristic update step proxies, an important component of the BHH.

6.3.2 Proxies

The concept of proxies arise from the sparsity of state as maintained by different heuristics. Since heuristics maintain 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. The other is a meta-heuristic that does not require a gradient at all such as PSO. 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 all of entities. A solution to this is to proxy heuristic state update operations. This allows us to maintain state in two parts:

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

Effectively this means that both primary and proxied state elements must be maintained together. Since entities represent candidate solutions, which are a form of state, entities are ideal components to store these state parameters. Thus, 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 model. At each heuristic-entity application step, all state elements per entity are updated either by primary method or by proxied method. The BHH thus incorporates a state update operation proxy mapping as given in the example in Table 6.1 below.

Table 6.1: State update operation proxy mapping example.

		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.

Notice however that this 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.

These state parameters and update operations should not be considered in isolation. An example of this is position and velocity. Consider for example heuristics such as *differential*

evolution and *genetic algorithms*. These heuristics recombine entities. In this 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 pure displacement through recombination. In this particular case, a solution to this is to apply the recombination operation to all secondary state parameters as well or to nullify secondary state parameters. Unfortunately there is no general solution and each heuristic must be carefully considered. The BHH incorporates both of these approaches. The next section provides the reader with details around the entity pool.

6.4 Entity Pool

This section presents the details around the *entity pool*. The entity pool refers to a collection of so-called individual *entities*. A common naming convention for such a collection is a *population* of entities. Similar to the heuristic pool that was described in Section 6.3 above, the entity pool size is an important design choice. The correct population size is thus a hyper-parameter that can be empirically evaluated.

The entity pool maintains two different types of state. These include entity (local) and population (global) state. Each of these are 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 just discussed above in Section 6.3.2. 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 physical environment. This means that these entities model concepts from physics. For example, the candidate solution is represented as the entity's *position* and an entity's velocity and acceleration, is analogous to the gradient and momentum of an entity respectfully. Examples of entity state parameters as derived from various low-level heuristics is given as follows:

- **position**: 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 can be derived proportionally from the gradient for gradient-based heuristics such as Momentum.
- **gradient**: The last know gradient as derived from gradient-based heuristics.
- **position delta**: The last computed position delta between the current timestep and the previous timestep.
- **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**: Parameter that tracks that best known position by the entity thus far. As required and maintained by heuristics such as PSO.
- **personal best loss**: Parameter that tracks that best known loss by the entity thus far. As required and maintained by heuristics such as PSO.
- **loss**: General parameter that tracks the loss as achieved by the the entity throughout training.

From the list above it should become clear that entity state becomes increasingly complicated as the number of distinct heuristics included in the heuristic pool increases. Consider next the global state that is maintained.

6.4.2 Population State

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

- ***entities***: Naturally, the population state contains the list of entities in the population. List size determined by population size. Naturally this list contains multiple entities as with population based approaches such as PSOs and GAs.
- ***ibest*** and ***ibest loss***: Refers to the best position and loss achieved by the population for the current iteration/step. This parameter is introduced by the BHH itself through the requirements of the *ibest* credit assignment strategy.
- ***rbest*** and ***rbest loss***: Refers to the best position and loss achieved by the population for the current replay buffer/window size. This parameter is introduced by the BHH itself through the requirements of the *rbest* credit assignment strategy.
- ***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 and is also introduced by the BHH itself through the requirements of the *gbest* credit assignment strategy.

The reader was now introduced to both the heuristic pool and entity pool. The next section considers how the BHH is initialised and makes specific reference to how these two pools of elements are initialised.

6.5 Heuristic-Entity Selections

The BHH selects from the heuristic pool a low-level heuristic to be applied to an entity. The outcome of this selection process is a table that tracks which heuristic has been selected for which entity. The selection process is executed by the selection mechanism of the BHH and is discussed later in Section 6.12. These heuristic-entity combinations are applied to an underlying model. The next section provides clarity on the model component from the architecture of the BHH.

6.6 Model

The model refers to the target model to be optimised and is usually expressed as a complex mathematical function. For this dissertation, the authors focused specifically on shallow FFNNs (only one hidden layer). Figure 6.2 below presents such a shallow FFNNs.

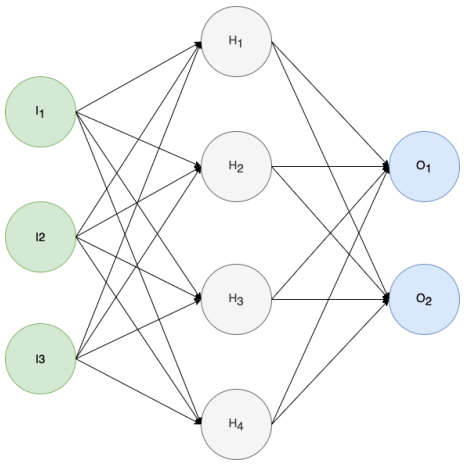


Figure 6.2: An illustration of a shallow Feedforward Neural Network.

6.7 Train and Test Datasets

The training of FFNN is a supervised learning problem as was highlighted in Chapter 2. Therefore, the BHH trains the model on training data from the training set, while is reserves an unseen set of test data in the test dataset for evaluation. It should be mentioned

that the goal of the BHH is to generalise well to the test set across a number of problem domains. As was previously mentioned, the test set should be sufficiently large to truly evaluate for generalisation capabilities. For the BHH, it is proposed to have a 80-20% split between the training and test set.

6.8 Loss/Cost Function

Chapter 2 presented the concept of a loss/cost function that is used to evaluate the performance of ANNs. The BHH uses this loss/cost function to evaluate the performance of a heuristic-entity combination. For classification problems, generally some entropy loss is used. For classification problems with 2 classes, BinXE is used and for classification problems with more than 2 classes, the SparseCatXE is used. For regression problems it is recommended to use some mean error measurement such as MSE or RMSE.

The loss/cost function is very important as this metric is used to evaluate model performance as a result of the application of a heuristic to an entity's candidate solution. The BHH used this performance metric to bias towards good heuristic-entity combinations. The continuous valued cost/loss function is translated into a discrete valued representation through a credit assignment strategy that is implemented in the BHH. The credit assignment strategies is discussed in Section 6.11.

6.9 Domain Barrier

The domain barrier presents the boundary of information flow between the heuristic space and the solution space. It is mentioned in Section 6.8 that the heuristic-entity loss is used by the BHH. Notice that the BHH uses this performance information and no information from the solution space itself. Furthermore, the BHH maps this loss to a credit assignment problem as is mentioned by Burke et al. [22]. The BHH at a high level, is not aware of the actual candidate solution that is represented by the entity, but rather just that there is a candidate solution with a unique identifier that, together with a heuristic, produced a certain level of performance. The domain barrier is a feature that occurs in al HHs. Burke et al. [22] states that the HH collects domain-independent information from the domain barrier. A number of examples of domain independent information is given, these include the number of heuristics, the changes in evaluation function, whether a new solution was found or generated, the distance between two solutions, whether it has become stuck.

6.10 Performance Log

Recall that the BHH incorporates a Bayesian probabilistic implementation. Evidence is required to train the BHH. The performance log is used to keep memory of a slice of

the performance history. The performance log is simply a tabular implementation that tracks random events. The BHH suggests the following metrics to keep track of in the performance log:

- **step**: The current mini-batch step.
- **heuristic**: The selected heuristics' index.
- **entity**: The applied entity's index.
- **loss**: The heuristic-entity performance metric.
- **ibest loss**: Keeps track of the *iteration* best loss. Thus, the best loss value achieved by all 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 all entity-heuristic combinations over all mini-batch steps currently in the performance log.
- **gbest loss**: Keeps track of the *global* best loss. Thus, the best loss value achieved by all all entity-heuristic combinations over all mini-batch steps thus far.

The performance log as implemented by the BHH is given in Table 6.2 below.

Table 6.2: Performance log example showing the first 5 entity's selected heuristics for step 1 and their resulting performance measurements.

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. Since the log contains a slice of memory, there should exists a trade-off between the following parameters:

- **performance log size**: The larger the performance log, the longer memory is retained.The performance log size controls the recency of evidence from which new beliefs must be derived. This parameter is called the *replay* window size, a term borrowed from RL and should be empirically tested. If the replay window size is too small, it does not accumulate enough samples/evidence to really statistically make accurate classifications. If the replay window size is too back, the BHH might remember too much of past performance and it could be that past performance is not

indicative of future performance during the training process. Notice then that the performance log must be pruned to maintain a fixed replay window size. Dynamic replay window sizes is left as a research component for the future.

- **performance log reanalysis**: Consider the temporal nature of the performance log. Since it tracks all random events across all steps in the training process, it is needed to reanalyse the performance log appropriately in order to update the past information in the performance log with newly added data. An example of this is when a new rbest value has been found. Then the entire performance log's rbest loss must be updated.

The performance log should be considered along with the selection of credit assignment strategy used. These are discussed next.

6.11 Credit Assignment Strategy

The credit assignment strategy is an implementation that maps a continuous performance metric into a discrete credit metric. This component implements the "move acceptance" process as proposed by Özcan et al. [136, 137]. It determines if a given performance is deemed good or not. This simple discrete transformation is subject to the credit assignment problem as discussed by Burke et al. [22]. A good credit assignment strategy will correctly allocate credit to the appropriate heuristic-entity combination. THe magnitude of this credit is to be empirically tested. It is later shown how these credit metrics are used to accumulate "pseudo counts" which concentrates/biases heuristic selection towards good performing heuristics. The BHH propose the following initial credit assignment strategies.

- **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 it's 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**: A credit assignment strategy that gives credit to all entity-heuristic combinations regardless of performance. This credit assignment strategy suggests that "symmetric" credit assignment be given. As a result, any bias that arise from uncertainty or as a result of an untrained BHH model, in the beginning, is maintained and further strengthened throughout the training process . Notice that this credit assignment strategy does not randomly assign credit, it simply assigns credit to the all events that are a result of random selection, which could be possibly be biased.

The implementation of a credit assignment strategy is a pure, stateless function that translates input P from the performance log into output C as is given in Table 6.3 below.

Table 6.3: Credit assignment strategy output table showing *ibest* credit assignment for the first 5 entity's 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 assignment is a key component of the BHH is it will later be shown in Section 6.14 how these credit assignments are used to optimise the BHH. First, it is necessary to present the selection mechanism.

6.12 Selection Mechanism

This section aims to provide the reader with the detail around the selection mechanism as it is implemented by the BHH. It is mentioned throughout this document that the BHH implements a predictive model based on Bayesian probabilistic concepts. In probabilistic modeling it is necessary to identify all the random events that can occur.

6.12.1 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*.

It should be noted that this information is observed direclty from the performance log presented in Table 6.3 above. From the above list, the event C is dependent on the occurrence of H and E. However, for simplification, an argument for independence can be made.

6.12.2 Independence

The dependence between random events can have a drastic 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. Notice that the BHH is fundamentally a classifier, classifying which heuristic to select given an entity and performance criteria. It is later shown in Section 6.12.5 that the BHH implements a Naïve Bayes classifier. Domingos et al. [46] mentions that although Bayesian classifiers' probability estimates are only optimal under quadratic loss if the independence assumption holds, the classifier itself can 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 and not to get real probabilities. Evaluating for proportionality and not precision allows for the assumption of independence.

6.12.3 Bayes' Theorem

Section 6.12.1 presented the reader with the random events that are observed by the BHH and Section 6.12.2 provided an argument for independence between events. Chapter 5 presented Bayes Theorem, but for convenience, it is given again in Equation 6.1 below.

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

As was indicated in Section 6.12.2, Equation 6.1 can be evaluated for proportionality since the BHH is not concerned with actual accurate selection probabilities, but rather implements a Bayesian classifier. The resulting proportionality is expressed in Equation 6.2 below

$$P(A|B) \propto P(B|A)P(A) \tag{6.2}$$

Notice how the Equation 6.2 implements a Bayesian conditionally. This conditionality can be extended to include multiple criteria. The next section shows how the Bayesian classifier is implemented by providing the detail around the implemented predictive model.

6.12.4 Predictive Model

Finally the reader is presented with the underlying predictive model as implemented by the BHH. This component is arguably the most important component of the entire BHH

as it fundamentally drives the output of the BHH. Bayes Theorem as was shown again in Section 6.12.3 above can be used to implement a predictive model to predict which heuristic to select, given the conditionality of a particular entity and a particular credit assignment criteria is met. Consider the following 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, $L = 2$.
- $\alpha = (\alpha_1, \ldots, \alpha_K)$: Denote the concentration parameters for the heuristic probability distribution.
- $\beta = (\beta_1, \ldots, \beta_K)^J$: Denote the concentration parameters for the entity-heuristic probability distributions.
- $\gamma = (\gamma_1, \ldots, \gamma_K)^L$: Denote the concentration parameters for the credit-heuristic probability distribution.
- $\theta|\alpha \sim Dir(\alpha; K)$: Denote the heuristic probability distribution, implementing a Dirchlet probability distribution parameterised by α and K. Heuristic selection probabilities are sampled from this distribution.
- $\phi|\beta \sim Dir(\beta; K)^J$: Denote the entity-heuristic probability distribution, implementing a Dirchlet 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.
- $\psi|\gamma_1, \gamma_0 \sim Beta(\gamma_1, \gamma_0)$: 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.
- $H|\theta \sim Mult(\theta; I, K)$: Denote the distribution of heuristics (event H), implementing a Multinomial distribution, parameterised by the sampled heuristic selection probability θ, the heuristic pool size K and maximum number of instances I.
- $E|\phi \sim Mult(\phi; I, K)^J$: Denote the distribution of entity-heuristic combinations (event E), implementing a Multinomial distribution, parameterised by the sampled entity-heuristic selection probability ϕ, the heuristic pool size K and maximum number of instances I, for each entity in J.

- $C|\psi \sim Bin(\psi, I)$: Denote the distribution of credit-heuristic combinations (event C), implementing a Binomial distribution, parameterised by the sampled credit-heuristic selection probability ψ and the maximum number of instances I.

The parameterised predictive model, as derived from Bayes theorem presented in Equation 6.1 is then given in Equation 6.3 below.

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

Notice the joint probability over events E and C given the selection of a heuristic H denoted by the sums of the products of the separate parts, for E and C, in the denominator. The calculation in the denominator can be intractable as has been previously indicated. However, in order to derive the posterior distribution as given in Equation 6.1), one only has to evaluate to proportionality as is given in Equation 6.4 below.

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

The predictive model thus models the *proportional* probability of the event (selection of) heuristic H, given allocation to entity E and credit C, parameterised by sampled θ, ϕ and ψ. These parameters are in turn parameterised by concentrations α, β, γ_1 and γ_0 which denote the prior beliefs of the BHH.

6.12.5 Naïve Bayes

This section aims to dissect the probabilistic model that is presented in Equestion 6.4. It has been established in Section 6.12.2 that for naïve Bayes classifiers, independence between events can be assumed. The following derived PMFs are provided as fundamental building blocks that are later shown to be used in the optimisation process of the BHH as is shown in Section 6.14.

The independence between events for the class label H simply yields the PMF of the Multinomial distribution as presented in Equation 6.5 below.

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

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

The independence between events E given class label H is denoted by the likelihood of E conditional to the occurrence of heuristic k and model parameter ϕ as presented in Equation 6.6 below.

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

Where $N_{j,k}$ is a summary variable such that $N_{j,k} = \sum_{i=i}^{I} \mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})$, 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 entity e and heuristic k occurring together in I independent, identical runs.

Finally, the independence between events for the performance criteria feature C given class label H is denoted by the likelihood of C conditional to the occurrence of heuristic k and model parameter ψ as given in Equation 6.7 below.

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

Where N_k is the summary variable as described for Equation 6.7 above. $N_{1,k}$ 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 ($c_i = 1$) 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 $N_{0,k} = N_k - N_{1,k}$ would denote the count of occurrences of the events c_i taking on a failure ($c_i = 0$) and h_i taking on class k.

Given the assumptions of the naïve Bayes classifier, one could supplement Equations 6.5, 6.6 and 6.7 above into the proportional evaluation of the predictive model as given in Equation 6.4. This is presented in Equation 6.8 below.

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

Computationally the equation presented in Equation 6.8 will underflow if the resulting probabilities are very small.

6.12.6 Numerical Stability

When evaluating for Equation 6.8, the numerical stability is shown to underflow if the resulting probabilities from its parts are very small. Multiplying 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 fulfills a credit assignment, yielding an extremely small probability for ψ. A solution to this problem is to apply the *log-sum-exp trick*, parameterising distributions no longer using logits rather than probabilities. The transformation of Equation 6.8 using the log-sum-exp trick is given below in Equation 6.9.

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

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* and is discussed next.

6.12.7 Mode Collapse

Mode collapse is the situation that occurs where the selective pressure towards a heuristic is close to zero, as a results of random sampling, low initial bias or bad performance. This leads to a situation where to probabilistic model continually decrease the selective pressure until the selection probability of that heuristic is 0, yielding no further observations of that particular random event. This could be problematic and should be addressed.

The BHH addresses this issue by:

- Using *symmetric* initialisation of the concentration parameters α, β, γ_1 and γ_0.
- Setting the lower-bound of the concentration parameters α, β, γ_1 and γ_0 to 1, so that a selection probability of 0 is highly improbable.
- 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 model as 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 this outcome.

This section briefly touched on parameter initialisation. The next section discusses the initialisation process of the BHH in detail.

6.13 Initialisation Step

This section sheds light into the initialisation of the BHH and how it applies to the heuristic and entity pools. Since the heuristic pool is a collection of stateless heuristic proxy operations, there is no explicit initialisation that is applied to the heuristic pool itself. On the contrary, the entity pool is stateful and maintain candidate solutions. Chapter 2 presented various initialisation strategies to randomly place candidate solutions in the solution space. The BHH can make use of any of these initialisation strategies and as such,

by default makes use of Glorot uniform sampling. All other entity state parameters are initialised to be zero.

The selection mechanism also requires initialisation. Section 6.12.4 introduces the concentration parameters α, β, γ_1 and γ_0. These parameters *symmetrically* initialised, meaning that they are all initialised to the value of one in all dimensions. Section 6.14.2 shows that these concentration parameters can be used to introduce *a priori* biases based on expert knowledge. The next section provides the detail around the selection mechanism that is implemented by the BHH.

6.14 Optimisation Step

Thus far the reader has been presented with all the fundamental components of the BHH. The final step that is missing is to present the optimisation step by which the optimisation process takes place. This section aims to provide a solid mathematical explanation to show exactly how the BHH is able to learn.

The BHH is an adaptive HH, meaning that it implements online learning. Online learning refers to the continuous learning process that is executed by the BHH while training is taking place. 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 the careful updating of these concentration parameters.

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 6.14.3 and 6.14.4 respectively. It should be mentioned that the BHH makes use of MAP to optimise the underlying model, however, the authors deem it necessary to provide the details of MLE as it contains fundamental mathematical building blocks that are required to understand MAP.

It has been established that the BHH is a selective HH that biases heuristic selection towards a certain performance bias. The concept of performance biases is discussed next.

6.14.1 Performance Bias

The intent of the BHH is to gather evidence that is can use to update its 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. Sections 6.14.3 and 6.14.4 show how the concentration parameters are updated throughout the training process, yielding the mechanism by which optimisation takes place.

Specifically, it can be said, that the optimisation process employed by the BHH updates

pseudo counts of events that it observes in its 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 that was discussed in Section 6.11, these pseudo counts are biased towards entity-heuristic combinations that meet credit requirements and yield credit allocations i.e. credit pseudo counts. It is through this concept that the BHH is able to bias selection towards performance.

The above mentioned concepts discuss heuristic selection bias by means of online learning. Another possibility is to introduce *a priori* information in the form of expert knowledge. This concept is discussed next.

6.14.2 A Priori Bias

Section 6.14.1 above discusses the role of concentration parameters on the BHH. It is further mentioned in Section 6.13 that these concentration parameters are symmetrically initialised. This simply means that they are uniformly initialised to all have the value of one in all dimensions.

Consider that the intent of the BHH is to learn which heuristics to select during training. An expert can inject expert knowledge into the BHH by means of these concentration parameters. An example of this is given as follows. It might be well known that a particular heuristic will perform well on a given problem domain. This could be through empirical analysis, part experiences, logical reasoning and deduction or simply by nature of the heuristic itself. A slight bias can be introduced to this heuristic before training starts but slightly scaling the initialised concentration parameter for that particular heuristic so that is more than the other concentration parameters. The BHH interprets this encoded concentration parameter as a prior belief and aims to update this parameter as new evidence is collected. The next section introduces the mathematics behind MLE.

6.14.3 Maximum Likelihood Estimation

In probability theory and statistics, MLE is a method that estimates the parameters of some assumed prior probability distribution given newly observed data and evidence. This section shows that the values for θ, ϕ and ψ can be estimated by MLE to yield equations 6.10, 6.11 and 6.12 below.

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

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

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

The calculations of the MLE for $\hat{\theta}_k$, $\hat{\phi}_{j,k}$ and $\hat{\psi}_k$ are now presented. The log likelihood of $\hat{\theta}$, $\hat{\phi}$ and $\hat{\psi}$ as derived from the Equation 6.4 is given in Equation 6.13 below.

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

Equation 6.13 can be broken down into each of its factors. Consider the log likelihood of ψ as denoted by Equation 6.14 below.

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

The MLE for $\hat{\psi}_k$ can be calculated by taking the derivative of Equation 6.14 with respect to ψ_k and equating to zero as presented in Equation 6.15 as follows.

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

The value for $\hat{\theta}_k$ can be calculated similarly. However, one has to compensate for the $K-1$ simplex S by adding error factor ϵ to correct values for θ where $\sum_k^K \theta_k \neq 1$. The new log likelihood function is then given in Equation 6.16 below.

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

Solving first for ϵ yields Equation 6.17 below.

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

Then solving for θ_k yields Equation 6.18 as presented below.

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

Substituting back into the Equation yields Equation 6.19 below.

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

The value for $\hat{\phi}_{j,k}$ is calculated very similarly. To compensate for the $K-1$ simplex S an error factor $\lambda = (\lambda_1,\ldots,\lambda_J)$ is added. This follows in Equation 6.20 as is presented below.

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

Solving first for λ_j yields Equation 6.21 as is given below.

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

Then solving for $\phi_{j,k}$ yields Equation 6.22 below.

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

Substituting back into the Equation yields equation 6.23 below.

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

This section provided the mathematical derivations of the MLE equations for θ, ϕ and ψ. The next section provides the detail around MAP, the optimisation process utilised by the BHH.

6.14.4 Maximum a Posteriori Estimation

Another approach to optimising 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 the Bayesian analysis and MAP as it is implemented by the BHH.

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

$$POSTERIOR \propto LIKELIHOOD \times PRIOR$$

The likelihood is presented in Equation 6.14.4 above. The event H is a multinomial distribution with parameter θ. It is known that the conjugate prior to a multinomial distribution is a Dirichlet distribution (see Chapter 5. The *prior* probability distribution for θ_k is thus presented in Equation 6.24 below.

$$P(\theta|\alpha) \propto \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \tag{6.24}$$

Furthermore, the event E is also a multinomial distribution with parameter ϕ. The *prior* probability distribution for $\phi_{j,k}$ is thus presented in Equation 6.25 below.

$$P(\phi|\beta) \propto \prod_{j=1}^{J} \prod_{k=1}^{K} \phi_{j,k}^{\beta_{j,k} - 1} \tag{6.25}$$

Finally, the event C is a binomial distribution with parameter ψ. It is known that the conjugate prior to a binomial distribution is a Beta distribution (see Chapter 5). The prior probability distribution for ψ_k is thus presented in Equation 6.26 below.

$$P(\psi|\gamma_1, \gamma_0) \propto \prod_{k=1}^{K} \psi_k^{\gamma_{1,k}} (1 - \psi_k)^{\gamma_{2,k}} \tag{6.26}$$

Putting the likelihood and priors together yields the posterior distribution as given in Equation 6.26 below.

$$\begin{aligned}
&P(\theta, \phi, \psi|H, E, C; \alpha, \beta, \gamma_1, \gamma_0) \\
&\propto P(H|E, C; \theta, \phi, \psi) P(\theta, \phi, \psi|\alpha, \beta, \gamma_1, \gamma_0) \\
&\propto P(E|H; \phi) P(C|H; \psi) P(H|\theta) P(\phi|\beta) P(\psi|\gamma_1, \gamma_0) P(\theta|\alpha) \\
&\propto \left[\prod_{j=1}^{J} \prod_{k=1}^{K} \phi_{j,k}^{N_{j,k}}\right] \left[\prod_{k=1}^{K} \psi_k^{N_{1,k}} (1 - \psi_k)^{(N_k - N_{1,k})}\right] \left[\prod_{k=1}^{K} \theta_k^{N_k}\right] \\
&\times \left[\prod_{j=1}^{J} \prod_{k=1}^{K} \phi_{j,k}^{\beta_{j,k} - 1}\right] \left[\prod_{k=1}^{K} \psi_k^{\gamma_{1,k} - 1} (1 - \psi_k)^{\gamma_{2,k} - 1}\right] \left[\prod_{k=1}^{K} \theta_k^{\alpha_k - 1}\right] \\
&\propto \left[\prod_{j=1}^{J} \prod_{k=1}^{K} \phi_{j,k}^{(N_{j,k} + \beta_{j,k}) - 1}\right] \\
&\times \left[\prod_{k=1}^{K} \psi_k^{(N_{1,k} + \gamma_{1,k}) - 1} (1 - \psi_k)^{(N_{0,k} + \gamma_{2,k}) - 1}\right] \\
&\times \left[\prod_{k=1}^{K} \theta_k^{(N_k + \alpha_k) - 1}\right]
\end{aligned} \tag{6.27}$$

From Equation 6.27 above, it can be seen that the posterior distribution has the same form $\mathcal{A}(v)$ as the prior distribution. The concentration update operations can then be given in equations 6.28, 6.29, 6.30 and 6.31 below.

$$\alpha_k(t+1) = N_k + \alpha_k(t) \tag{6.28}$$

$$\beta_{j,k}(t+1) = N_{j,k} + \beta_{j,k}(t) \tag{6.29}$$

$$\gamma_{1,k}(t+1) = N_{1,k} + \gamma_{1,k}(t) \tag{6.30}$$

$$\gamma_{2,k}(t+1) = N_{0,k} + \gamma_{2,k}(t) \tag{6.31}$$

It can be seen that the prior and posterior probability distributions are of the same shape $\mathcal{A}(v)$. It can therefore be said that the BHH implements a Gaussian process. Since the reselection of heuristics happen at regular intervals, the outcome of a selection in one iteration may influence the outcome of another in the next iteration. It can thus be said that the BHH implements a *Hidden Markov Model* (HMM).

The section concludes the optimisation process that is implemented by the BHH. Throughout this chapter a number of hyper-parameters have been identified. These are summarised and discussed in the next section.

6.15 Hyper-Parameters

This section provides the reader with a breakdown of all the hyper-parameters that were identified and briefly mentioned in this chapter. 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. Were possible, a range of possible values is provided. Each of the hyper-parameters are discussed in their own sections next.

6.15.1 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. This configuration 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 and meta-heuristics.

- The low-level heuristic's hyper-parameters. At a lower-level, each of these heuristics have their own set of hyper-parameters.
- The heuristic pool size. This refers to the number of heuristics in the heuristic pool.

The granularity of design choices at a high level is not as strict for the BHH as it is for low-level heuristics. If one is unsure as to which low-level hyper-parameters to use, one could simply include multiple configurations of that low-level heuristic, each just with its own unique hyper-parameter configurations. Take note that it has not been established yet if this is to the benefit or detriment of BHH. These configurations increase the heuristic pool size and could also counteract eachother. It is thus still important to include relevant heuristics that is appropriate to the underlying problem being solved. These heuristics should contain a balance between exploration and exploitation, either through the nature of their implementation or through careful hyper-parameter selection. Often times hyper-parameter schedules are used. This is also possible with the BHH.

6.15.2 Population Size

Population size refers to the number of entities included in the entity pool/population. The exact population size to use is to be determined empirically. Naturely, an assumption is made that a larger population could lead to better results, assuming that entities are initialised to be spread uniformly across the solution search space. Oldewage [135] mentions that a large number of particles (entities) may be able to traverse a greater portion of the search space as 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 authors propose the following considerations 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 one includes DE into the heuristic pool, one needs a population size of at least 4. For PSOs, the minimum population size is 3.
- When considering the upper bound of possible population sizes, take into account that the bigger the population size, the more computationally expensive it is to execute the training process. Furthermore, there could exists a point of *diminishing returns* where an increase in population size does not yield any better outcome. Once again, this is to be empirically tested. To extend on this, consider that the BHH is a HH that utilises a Bayesian probabilistic approach in it's selection mechanism. This means that a larger observed sample size (of entity and heuristic selection events) is needed to statistically and reliably update the selection mechanism's trainable parameters.

6.15.3 Credit

The credit hyper-parameter refers to the credit assignment strategy that is used. This dissertation proposed 5 different credit assignment strategies that were presented in Section 6.11. The choice of credit assignment is assumed to be problem-dependent, however, this is to be empirically tested. Apart from the credit assignment strategy itself, one has to consider the following components related to credit assignment:

- When to apply credit?
- What should the credit assignment score be (default = 1)
- To what degree should credit assignment be allocated to past results in the performance log

It should be noted that this dissertation proposed an initial set of credit assignment strategies that are discrete in outcome. Perhaps an alternative is to include some measure of success, not just an indicator of success. This allows for the implementation of continuous valued outcomes from the credit assignment strategies. Take into account that this could also drastically increase the complexity of the underlying predictive model implemented by the BHH.

A number of other hyper-parameters are specifically included to address the above challenges. These include *reselection*, *replay* and *reanalysis window sizes* along with *burn in*, *discounted rewards* and *normalisation*. These are discussed next.

6.15.4 Reselection

The reselection window size 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 as was discussed in Section 6.12.4. The choice of reselection should be influenced by the following considerations. If reselection happens too frequently, it does not allow heuristics sufficient time to smooth out update steps which could be detrimental to the outcomes of the BHH. If reselection happens too infrequently, it does not allow for sufficient room to collect samples/evidence from which learning is done. Naturally, the correct reselection window size to use is to be determined empirically.

6.15.5 Replay

The replay window size is a hyper-parameter that controls how much historical performance information is kept in memory for the BHH to learn from. This is a parameter that is borrowed from RL and aims to control the effect that past performances has on the BHH. An assumption is made that the correct replay window size to use could be problem dependent. However, logical arguments can be deduced. A replay window size that is too

small includes little to no memory. This results in a situation where the BHH simply looks at the most recent evidence that is collected and nothing more. This could be a beneficial or detrimental to the BHH. On the contrary, if the replay window size is big, then the BHH tracks a longer history of past performances. Once again, the this could be beneficial or detrimental to the BHH.

6.15.6 Reanalysis

The reanalysis window size is a hyper-parameter that controls how often the BHH updates its past beliefs (priors). Consider that this parameter goes hand in hand with the reselection and replay window sizes. If the reanalysis window 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 window size is to be determined empirically. However, some logical exclusions can be made. A reanalysis window size that is smaller than the reselection window size does not make sense, since the effects of reanalysis is only realised during reselection. A reanalysis window size that is too big could lead to a situation where the delay in updating priors could be detrimental to the outcomes of the BHH.

6.15.7 Burn In

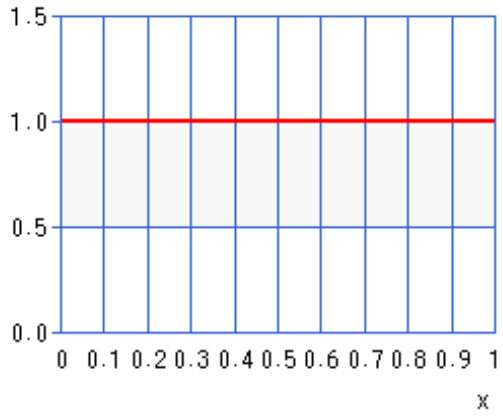
Burn in is a hyper-parameter that is borrowed from MCMC and aims to delay the learning process of the BHH. The intent of this hyper-parameter is to determine if heuristics need time to execute before reselection starts. Notice that a delay in the learning process does not mean that reselection doesn't occur or performance information is not collected in the replay window. Therefore, it should be noted that the burn in window size, reselection window size and replay window size should be considered together. If the replay size is smaller than the burn in window size, no performance information is accumulated and the BHH essentially starts with a prior that is no better than the initial symmetric initialisation that was done.

6.15.8 Discounted Rewards

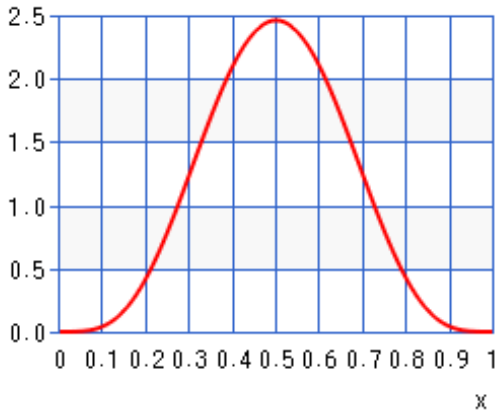
Discounted rewards is another hyper-parameter that is borrowed from RL. This hyper-parameter is a flag that determines if discounted rewards is enabled or not. Discounted rewards refers to an operation where the credit assignments for past performance is exponentially decreased further into the past. An exponential decay factor or 0.5 is proposed. The intent of the discounted rewards flag is to scale the effect of past performances that are further back into the performance log such that they have a smaller impact than more recent evidence on the performance achieved by the BHH.

6.15.9 Normalisation

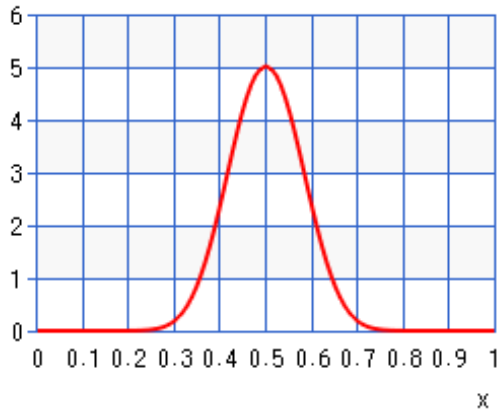
Normalisation is a hyper-parameter that aims to provide a mechanism of control for the degree to which exploration is allowed. Figure 6.3 shows the effect of different concentrations (α and β) on the Beta distribution. By normalising the concentration parameters for the BHH (α, β, γ_1 and γ_0), the variance of the selection probability distributions increase, allowing for sampled probabilities further from the mean and thus allowing for more exploration.



(a) $\alpha = 1$, $\beta = 1$



(b) $\alpha = 5$, $\beta = 5$



(c) $\alpha = 20$, $\beta = 20$

Figure 6.3: Beta distribution with varying α and β values.

This concludes the hyper-parameters that are used by the BHH. The following section discusses the default value for proxied update step operations.

6.15.10 Defaults

Section 6.3.2 provided the reader with the concept of proxied heuristic update step operations. As was mentioned in Section 6.15.1, low-level heuristics each have their own

set of hyper-parameters as well. This means that 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 (should) differ from each other. In the case of Adam, there are 4 hyper-parameters that include learning rate, $\beta 1$, $\beta 2$ and ϵ. If another heuristic needs to proxy Adam's "expected gradient mean operation", a default $\beta 1$ parameter must be supplied. Notice that this is only the case where 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. The next section provides the reader with the pseudo-code algorithm for the BHH.

6.16 Algorithm

This section provides the reader with a high level pseudo-code implementation of the BHH. This is given in Algorithm 9 below.

Algorithm 9 The pseudo code for the Bayesian Hyper-Heuristic optimiser

```
step ← 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 to performance log
        if step < burn-in window size then
            select heuristic
        else
            if step % reanalysis window size = 0 then
                apply Bayesian analysis
            end if
            if step % reselection window size = 0 then
                select heuristic
            end if
            if step > replay window size then
                prune performance log
            end if
        end if
    end for
    step ← step + 1
end while
```

6.17 Summary

This chapter provided the reader with 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 optimisation process, by means of Bayesian

analysis has been presented and a pseudo code implementation of the BHH was presented.

This concludes the background information and literature studies that is required to propose a methodology for an empirical process. This methodology is provided in the next chapter.

Chapter 7

Methodology

Observation, reason and experimentation make up what we call the scientific method. ∼ Richard P. Feynman

Chapter 1 set out the objectives of the dissertation. These objectives can broadly be split between background information and an empirical component. So far, chapters 1-6 provided the necessary background information. This chapter now provides the detailed specification of the methodology that is followed for the empirical process and the analysis of the results. The remainder of the chapter is structured as follows:

- **Section 7.1** provides a brief overview of the entire empirical process, leaving the detail to subsequent sections.
- **Section 7.2** presents the detail around the datasets that are used.
- **Section 7.3** provides the details of the models (FFNNs) that are being 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 is executed.
- **Section 7.10** sheds some light as to the implementation and execution of the empirical process.
- **Section 7.9** presents the procedures that are followed in during the statistical analysis of the results.
- Finally, a summary of the chapter is provided in **Section 7.11**

7.1 Overview of Empirical Process

The purpose of the empirical process is to present and execute a carefully crafted experiment/test that produces data that can be used to reject of verify some hypothesis about the element under investigation. This dissertation has outlined a set of 3 objectives that require such empirical processes to evaluate the BHH. This section outlines what these empirical processes entail.

Each empirical test sets out some configuration of elements to be evaluated. These 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 (NN) is trained across a number of datasets. Each dataset is split into a training and test set. The training set is used to train the model, while the test set is used to evaluate the model. Training epochs are split into mini-batch iterations, with each dataset containing a specified mini-batch size. Evaluation takes place at every mini-batch step. These experiments contain a number of components, each with a set of paramaters. Due to the stochastic nature of the experiments, each experiment is repeated over a number of runs to provide sufficient sample for statistical certainty. Each run contains a different random seed. Each evaluation's results are analysed for statistical significance. It is from these statistical analyses that findings and conclusions are then made.

Detail around each of these elements are now provided, starting with the datasets that are used.

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 [1]. Datasets are grouped by problem type and include 8 classification and 7 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	[63]
car	multivariate	categorical	6	4	1728	128	14	[16]
abalone	multivariate	categorical, integer, real	8	28	4177	256	17	[192]
mushroom	multivariate	categorical	22	2	8214	512	17	[160]
wine_quality	multivariate	real	12	11	4898	256	20	[33]
bank	multivariate	real	17	2	45211	512	89	[128]
diabetic	multivariate	integer	55	3	100000	1024	98	[175]
adult	multivariate	categorical, integer	14	2	48842	256	191	[108]

Table 7.2: Regression datasets

dataset	output	types	attributes	instances	batch	steps	citation
fish_toxicity	multivariate	real	7	908	64	15	[27]
housing	univariate	real	13	506	32	16	[82]
forest_fires	multivariate	real	13	517	32	17	[34]
student_performance	multivariate	integer	33	649	32	21	[35]
parkinsons	multivariate	integer, real	26	5875	256	23	[183]
air_quality	multivariate, timeseries	real	15	9358	256	37	[43]
bike	univariate	integer, real	16	17389	256	68	[60]

Details on how the datasets where preprocessed and prepared is given in Appendix A. The concept of class balancing is briefly discussed next.

7.2.1 Class Balancing

A number of classification datasets as given in Table 7.1 above contain unbalanced classes. A decision was made to not make use of class balancing. This was simply to eliminate as many variables adn factors in the empirical process as possible, as it is already a complicated process. It is therefore suggested that the BHH first be studied under the assumption of balanced classes and then only apply class balancing. In future research opportunities, class balancing should be utilised.

The next section presents the models and the configurations that are used.

7.3 Models

As the title of the dissertation suggests, these empirical process are focused on the training of FFNNs. In theory, one should be able to swap these FFNNs out for any other mathematical model that can be optimised. All models that are trained in this dissertation are shallow NNs, meaning they only have one hidden layer. The architecture of a model is dependent on the dataset being trained on, the type of problem it is (classification or regression), the number of input dimensions and the number of output dimensions. The models and their configuration, as it is used for each dataset, is given below in Table 7.3

It should be noted that for the classification problems, the softmax activation is not actually part of the model. It is just added for predictive output and is not needed during evaluation. The loss functions (SparseCatXE and BinXE) that are used contains a softmax function. Models' initial weights are initialised by means of Glorot uniform sampling.

7.4 Heuristics/Optimisers

This section provides the details of the standalone heuristics/optimisers that are used in the empirical process.

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

Table 7.4 contain a list of all the standalone heuristics that are used as well as their hyper-parameter configuration.

Take note that for Table 7.4, values that contains an asterisk ($*$) are configured on a schedule with the initial value depicted first and the decay rate is depicted last. The number of steps is the total number of steps for that particular dataset it is being applied to.

Also take note that a learning rate was added to the PSO as an attempt to avoid overshooting solutions later in the training process. This parameter does not traditionally form part of the PSO, but it was found to work sufficiently here.

The next section provides the details around the BHH baseline.

TODO: Proxy Mapping Table

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 reasonable performance. It was decided that this configuration will be the cornerstone configuration from which all other heuristics and their configurations are evaluated. The BHH baseline represents the main component that is evaluated in this dissertation. The configuration for the BHH baseline is given in Table 7.5 below:

Table 7.5 refers to the heuristic pool that is used as *all*. This refers to a configuration where the heuristic pool contains all the low-level heuristics as presented in Section 7.4. This list of low-level heuristics naturally also then form the list of standalone heuristics/optimisers to which the BHH is compared and analysed as is described later in Section 7.8.2. Specifically the finer details of the implementation of the BHH, the low-level heuristic pool and how it

Table 7.4: Heuristics/optimisers and their hyper-parameter configuration of their

heuristic	configuration	value	citation
sgd	learning_rate	0.1**0.01	[177]
momentum	learning_rate	0.1**0.01	[177]
	momentum	0.9	
nag	learning_rate	0.1**0.01	[177]
	momentum	0.9	
adagrad	learning_rate	0.1**0.01	[50]
	epsilon	1E-07	
rmsprop	learning_rate	0.1**0.01	[85]
	rho	0.95	
	epsilon	1E-07	
adadelta	learning_rate	1.0**0.95	[199]
	rho	0.95	
	epsilon	0.0000001	
adam	learning_rate	0.1**0.01	[107]
	beta1	0.9	
	beta2	0.95	
	epsilon	1E-07	
pso	population_size	10	[186]
	learning_rate	0.1**0.01	
	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	[122]
	selection_strategy	best	
	xo_strategy	exp	
	recombination probability	0.9**0.1	
	beta	2.0**0.1	
ga	population_size	10	[112]
	selection_strategy	rand	
	xo_strategy	bin	
	mutation_rate	0.2**0.05	

Table 7.5: BHH baseline configurations

hyper-heuristic	variant	configuration	value
bhh	baseline	heuristic_pool	all
		population	5
		credit	ibest
		reselection	10
		replay	10
		reanalysis	10
		normalise	false
		discounted_rewards	false

is used is given in Chapter 6.

7.6 Performance Measures

This section sheds light into the performance measures that are used to evaluate the different experimental runs.

Chapter 2 provided the reader with a number of techniques that are used to evaluate FFNN performance during training. Broadly these techniques measure the output of some loss/cost function and accuracy for classification problems. For the classifications problems, the loss metrics used included SparseCatXE and BinXE. Their accuracy equivalent metrics are then also used. For regression problems, RMSE is used as a loss metric.

As mentioned in the preceding sections, datasets are split into a training set and a test dataset. The training set is used to train the model and the test set is used to test the model. Evaluation takes place at the end of each mini-batch iteration. Loss and accuracy is measured for both training and test datasets and is captured at each mini-batch iteration.

Post processing of the results yield a average rank (by test loss metric) between configurations at each mini-batch step across all runs. This test loss metric and rank then form experimental results that are then statistically analysed.

7.7 Stopping Conditions

This section sheds light on the stopping conditions that are used.

Early stopping is a technique where training is prematurely stopped due to training saturation. A popular technique for early stopping includes a check to see if the validation/test loss has improved in the last k steps. If the loss has not improved, the training is halted and the last best model weights are used.

It was decided not to use early stopping in this empirical process. Since the BHH is a novel HH, there are lots of uncertainty around its performance. It is unsure as to how the BHH will behave, should overfitting take place. By eliminating early stop, the BHH is evaluated until a maximum number of epochs (30) is reached. It is recommended that future research make use of early stopping.

The different experimental groups are given next.

7.8 Experiments

This section presents the experimental groups that form part of the empirical process. There are broadly 3 main groups. These include a behavioural case study, a critical evaluation of the BHH baseline's performance compared to other standalone low-level heuristics and finally an analysis of different BHH variants is done by analysing the effects of different parameter values on the outcomes of the BHH. Each of these sections are discussed in more detail below.

7.8.1 Behavioural Case Study

This experimental group is concerned with objectively studying the behaviour of the BHH baseline across a number of runs. This is meant to be an introductory analysis of the BHH and includes analysis of the selection mechanism as well as the perturbative nature of the BHH. It is important to mention that this experimental group is not meant to be statistically analysed, but to rather provide a detailed visual analysis of a number of hand-picked example runs, to determine if the BHH is behaving as expected. It also provides a gentle introduction to the BHH's inner workings. From these observations, it is determined if the BHH is learning and that selection is indeed biasing towards better performance. It also provides an opportunity to determine the outcome of the perturbative component of the BHH which includes proxied heuristic update step operations.

Two different configurations of the BHH baseline is trained using the iris dataset [63]. These configurations include a replay window size of 10 (short memory) and a high replay window size of 250 (long memory). Each configuration is repeat 3 times. Finally these configurations are compared to a run that is purely random.

The analysis is primarily done by dissecting the output of various parameters and studying the plots of their values. From these plots, conclusions about the BHH's behaviour is made.

7.8.2 Standalone Optimisers

THis section presents the details with the experimental group that focuses on evaluating the performance of the BHH with that of standalone /optimisers.

For this experimental group, the heuristics and optimisers as presented in Section 7.4 are used, along with their specified parameters. Each of these are then compared to that of the BHH baseline configuration which is presented in Section 7.5.

The next section provides the details around different BHH variants that are evaluated.

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 and their possible configurations is given in Table 7.6 below.

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

Take note that the heuristic pool options *gd* and *mh* refers to the heuristic pool configurations where the heuristic pools contain only gradient-descent heuristics or only {indexmeta-heuristicsmeta-heuristics respectively.

The next section explains in detail how the results are statistically analysed.

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 the details of the configuration is given in the preceding sections. Each of these experimental groups are statistically analysed on their own. To ensure there is statistically sufficient sample, each experimental group and configuration is trained for 30 epochs, repeated over 30 runs. As it is mentioned above, the results contain performance evaluation data in the form of training and testing, loss and accuracy measurements. Each experimental run is then ranked by these metrics. Each run is

executed using a unique seeding value such such that each run is identical in its setup and configuration (apart from seeding values) and independent of the other runs.

Each experimental group goes through a set of steps that are followed during the statistical analysis process. First, the descriptive analysis is done to ensure that each experiment has the required data it needs. This includes 30 runs per configuration, across 15 datasets, across the applicable number of experimental configurations set up by that particular experimental group. The spread of the data is analysed to evaluate for overfit and outliers are identified. The skewness of the results is evaluated per dataset and the Shapiro-Wilk test for normality ($\alpha = 0.001$) is used to determine in the results are normally distributed. The full statistical analysis reports are presented in Appendix B and provides descriptive plots of the data's distribution. Furthermore, the Levene's test for equality of variance ($\alpha = 0.001$) is used.

Dependent on the outcomes of the above statistical tests, the appropriate statistical significance test is then executed. For configurations of independent variables where there are only 2 classes, the Mann-Whitney U independent samples t-Test ($\alpha = 0.001$) is executed and for configurations of 3 or more configuration classes, the *analysis of variance* (ANOVA) statistical test ($\alpha = 0.001$) is used. The Kruskal-Wallis ranked non-parametric test for statistical significance ($\alpha = 0.001$) is used for cases where data is not normally distributed.

As mentioned earlier in the chapter, results are analysed for statistical significance over all the steps during the training process across all runs. This helps determine if there is statistical difference in the execution of various experimental configurations, throughout the entire training process, but also to cater for overfitting, which is to be studied as well.

Regardless of the statistical test that is used, a post-hoc Tukey honest significant difference test ($\alpha = 0.001$) is used from which significant ranking is retrieved. Descriptive and critical difference plots are then retrieved from these results to provide visual aid, but all conclusions are made from statistically analysed data as mentioned above.

7.10 Implementation and Execution

This section sheds some light as to the details of the implementation and execution of the empircal process.

All implementation is done from first principles in Python 3.9 using Tensorflow 2.7 and Tensorflow Probability 0.15.0. Most underlying function are reused from the Tensorflow library, however, all heuristics/optimisers are implemented from first principles to fit the HH framework that was developed. All source code and data is provided is resources in this dissertation.

It should be noted that this implementation makes heavy use of CPU processing, due to the nature of modeling flattening for the heuristics. For this reason, execution is much more timely and costly than with GPU training. The authors hope that as the BHH

evolves, that better GPU implementations may speed up execution times.

With regards to the computational power that was used the execute this empirical process. All experiments were run on the CHPC's cluster. This included 14 servers each running 24-56 cores and 256GB memory each. The entire empirical process took 6 days.

7.11 Summary

This chapter provided the detail around the methodology that is used to execute the empirical process. The datasets, models and heuristics that are used during the empirical process have been presented in detail. A baseline BHH has been formulated. The empirical process was defined in terms of a number of different experiments 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 and objective discussion of empirical findings with good statistical backing. From the problem statements and motivations made in Chapter 1, goals have been defined that broadly include literature studies and a sound empirical process. Thus far, the reader has been provided with a vast scope of background information relevant to the development of the BHH. Chapters 1 - 5 provided the background information, literature studies and existing landscape of research for ANNs, heuristics/optimisers, HHs and probability theory. These elements all share common elements that are required to understand the proposed BHHs which was presented in Chapter 6. The details around the empirical process was formally proposed in the methodology as presented in Chapter 7. Finally, the results for the empirical process can be presented and this chapter aims to provide these results.

As a reminder, the empirical process is split into two main experimental groups. These include a behavioural case study and set of comparative studies. This chapter is organised into the same logical pattern. Each of these experimental groups is provided as its own section for detailed analysis and discussion of findings. The remainder of the chapter is structured as follows:

- **Section 8.1** provides a brief overview of the general empirical process, high level discussion points and a brief discussion on results interpretation.
- **Section 8.2** provides a detailed analysis of the behaviour of the BHH over a few example runs during training and testing and is meant to introduce the analysis of the BHH from a behavioural characteristic point of view. In this section, focus is put on the learning process itself and to determine what happens during training. It aims to provide insight into "how" and "when" the BHH learning happens.
- **Section 8.3** provides the first of a series of comparative studies. This section presents the comparative analysis that studies the performance capabilities of the

baseline configuration of the BHH to that of standalone/individual low level heuristics/optimisers. The sections that follow focus on the effect of various parameters on the BHH.

- **Sections 8.4** - 8.12 capture the results of these empirical processes and focus on the configuration of the *heuristic pool*, the *population size*, *credit assignment strategies*, *reselection* window size, *replay* window size, *reanalysis* window size, *burn in*, *normalisation* and *discounted rewards*, respectfully.
- **Section 8.14** contains a brief discussion on the computational requirements.
- **Section 8.13** provides a brief discussion on overfitting.
- Finally, a summary of the findings is given in **Section 8.15**.

8.1 Overview

This section aims to provide a general high level discussion on the outcomes of the empirical process as an introduction to the more detailed discussions to follow. Consider again the goals that have been defined in 1. The empirical component addresses three different angles of analysis:

1. Analysing the behaviour of the BHH during training and testing to determine if the the concept works and that the BHH is indeed learning. This component looks at the BHH in isolation and simply evaluates the outcomes of its behaviour over a few example runs and repeats this process for two groups of BHH variants based on *replay* window size.
2. Comparing the performance capabilities of the BHH to that of well-known low-level heuristics that are well researched and understood. This component looks at how the BHH performs given a frame of reference to compare to. Careful consideration of results interpretation must be given here and is discussed in more detail below.
3. Analysing the effects of various configurations of hyper-parameters for the BHH. This component looks at which parameters have a statistically significant impact on the outcomes of the BHH.

It is important to briefly discuss how results should be interpreted. Traditionally, for typical low-level heuristic training, evaluation of performance is done at a chosen point in time and based on a particular dependent variable. This could be a fixed number of training steps/epochs, early stopped conditions or any other stopping condition. The metric of interest could be a loss value, an accuracy or a rank. Algorithms/heuristics in question are then logically ranked/organised according to their final solution and conclusions are made

based on these findings. However, the following arguments suggest an alternative approach for this empirical study:

- Since there does not exist any research on this specific implementation of HHs and its application to FFNN training, there is no clear point at which it is known that the learning of the BHH becomes detrimental to the training process. For example, there exists a currently unknown point during training where the performance bias towards the "best" heuristic no longer yield improved training or test generalisation outcomes. To cater for this, a fixed maximum number of training epochs have been chosen that empirically have been shown to be sufficient to notice diminishing returns in the outcome. Early stopping was considered, but purposefully not implemented. This is to allow the study of the behaviour and capabilities of the BHH up to and beyond the point of diminishing returns.

- The training process is to some degree analogous and comparable to a dynamic optimisation problem as previously discussed. This means that the performance of heuristics, and thus the BHH, vary at different times during the training process. This hints to the hypothesis that was set in Chapter 1 and suggests that there is no single heuristic that is always the best heuristic to use during all of training. Rather, there exists an applicable, relevant heuristic (or combination of heuristics) to use at a given point in time during the training process and can change as the process continues. This means the selected heuristic at time step t is not necessarily the best heuristic to select at timestamp $t + 1$. This argument is further supported by the fact that these experiments are trained on mini-batches, resulting in a lot of noise during training. This noise can cause the BHH to want to "correct" a certain learnt belief. This introduces the same challenges as with RL: Consider delayed rewards, size of rewards, unlearning behaviour that is no longer required, all of which affect the training process throughout and not just at the point of stopping.

Based on the points made above, each experimental group is to be evaluated in isolation and has a specific area of focus. For all experimental groups, heuristics are evaluated during **all** the steps of the training and testing process. For this reason, ranking heuristic should not necessarily lead to the conclusion that one heuristic is *better* than another, but rather, that there is a statistically significant difference between the outcomes of heuristics, relative to the entire training and testing process for a given experimental group. The authors suggest further research to include early stopping conditions for the ranking to make entirely sense. However, that does not mean the ranking approach is useless. It still provides a measure of comparison and is indeed indicative of statistically significant outcomes.

The points that have been discussed above have been observed during the empirical process. It is shown in the succeeding sections that the BHH is capable of training FFNNs

and that it does so relatively well. It is shown that the BHH is generally capable of providing good test generalisations when compared to existing low-level heuristics. However, it is shown that the majority of the influence of the BHH occurs during the first few epochs of the training process. Initially uniformly randomised candidate solutions has more room to learn in the initial phases of training in comparison to later phases where learning has become stagnated. It is also shown that this pivotal point of diminishing return is problem specific. Further to this pivotal point, it is shown that the BHH is subject to over-fitting, quite drastically. There are various conclusions that can be made from this point, but the detail is left to the relevant sections to follow.

The following section provides the results and findings for a behavioural case study of the BHHs on the iris dataset.

8.2 Case Study

This section provides a study on the behaviour of the BHH during training and testing. It is important to mention that this experimental group does not necessarily focus on the exact metrics, but rather on the general behaviour of the BHH. This means that statistical analysis and statistical certainty as is not required. Instead, a number of example runs have been executed purely for the intent of analysing behaviour. Furthermore, this behavioural case study is executed on a very small, but well-known toy dataset called *iris*. Iris is not usually considered for full empirical processes due to its size and simplicity. Although there are fourteen other datasets to choose from, iris provides a very simple, relevant and relate-able case study that illustrate the necessary points sufficiently.

Figure 8.1 illustrate these runs during training and testing. Take note of the legend as provided in Figure 8.1e. There are two variants of the BHH included here, each with three repeated runs, seeded according to run number:

- BHHs with a very **short memory**, i.e. replay windows size = 10
- BHHs with a very **long memory**, i.e. replay windows size = 250

These two groups have been chosen as their parameters have been shown to provide drastically different outcomes from each other during training and forms a good basis for behavioural analysis. For more details on replay window size, see Section 8.8.

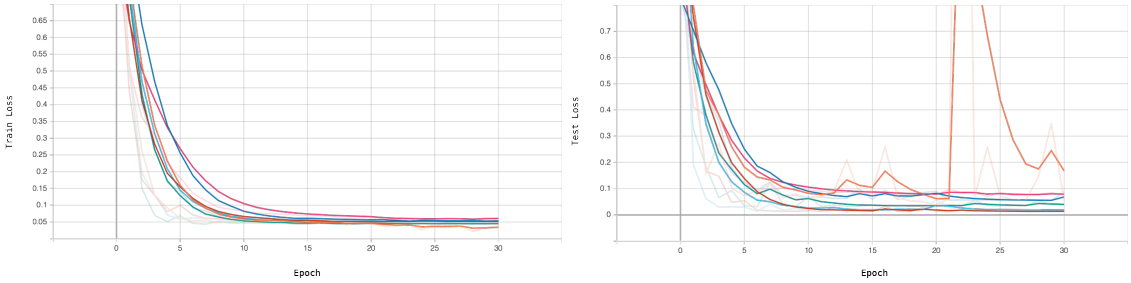
A breakdown of the run notation as provided in Figure 8.1e is then given as follows:

- **iris** - The dataset in question
- **bhh** - The heuristic in question
- **hp:all** - The heuristic pool (hp) in question. “All” refers to all the implemented lower-level heuristics including both classical gradient-based heuristics/optimisers as well as meta-heuristics.

- **ps:5** - The population size (ps) in question. In this case, population size = 5.
- **bi:0** - The burn in window size (bi) in question. In this case, burn in window size = 0.
- **rp:10** - The replay window size (rp) in question. In this case, replay window size = 10.
- **rs:10** - The reselection window size (rp) in question. In this case, reselection window size = 10.
- **ra:10** - The reanalysis window size (rp) in question. In this case, reanalysis window size = 10.
- **nm:False** - The normalisation flag (nm) in question. In this case, normalisation is disabled.
- **ct:ibest** - The credit assignment strategy (ct) in question. In this case, the credit assignment strategy used is *ibest*.
- **dr:False** - The discounted rewards flag (dr) in question. In this case, discounted rewards is disabled.

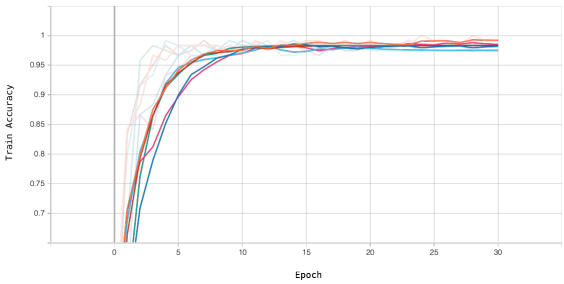
From figure 8.1 it is shown that the BHH exhibit similar behaviours to that of lower-level heuristics. Figures 8.1a and 8.1c illustrates a smooth training process, converging to a well-known minimum loss for the iris dataset. In Figures 8.1b and 8.1d it is shown that the BHH generalised well on the test set. Generalisation improved up to about epoch 10 before overfitting starts occurring. From this, it can be concluded that most of the learning, for this particular dataset, takes place in the first 10 epochs of the training process. From this point onwards, the BHH tries to undo "bad" selections made earlier. It is interesting to notice that despite overfitting, the BHH was able to "correct" itself to some degree (up to epoch 20), but never really converged back on the minimum it reached up to epoch 10. Early stopping should prohibit this from happening, but this is left for further research.

A high level view of the train and test metric plots is good for drawing high level conclusions, but more detail is required. Figure 8.2 aims to provide more detail as to what is happening during training and plots the value of α during training. Each sub-figure represents the value of α for a particular index, representing a particular heuristic. From sub-figures 8.2a, 8.2c and 8.2g, the effect of a larger replay value is clear. Although a more detailed discussed on replay window sizes is left for Section 8.8, one can clearly see the accumulation of "knowledge"/"evidence" in the concentration parameter α. It should be noticeable then from these sub-figures how α changes over time for runs with lower replay window sizes and is indicative of the learning capabilities of the BHH. From these changes, one can formulate a prediction of which heuristics are then likely to be selected in the next steps.

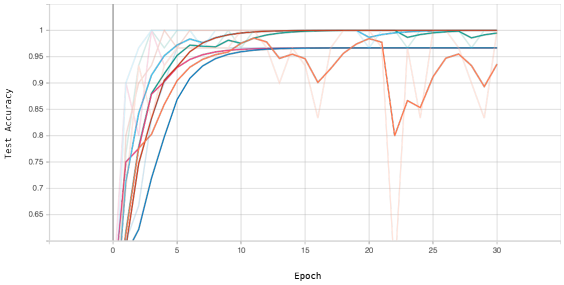


(a) Train loss

(b) Test loss



(c) Train accuracy



(d) Test accuracy

(e) Legend

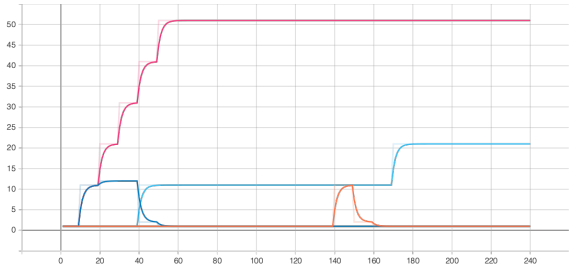
Figure 8.1: Train and test behaviour of the BHH baseline for different configurations of replay window size

Figures 8.2f and 8.2j contain a run in orange that shows how the concentration for a particular heuristic (Adadelta and DE) and can change over time. This suggests that the BHH has the ability to dynamically switch between heuristics as the need requires. This concept bears similarities to dynamic optimisation algorithms. It can thus be said that an observation is made that the BHH is able to learn (to some degree) to balance exploration and exploitation by dynamically changing the heuristic of interest at that point in time. Notice that it has not yet been established if the attempt to balance leads to good or bad outcomes. However, this is a still a useful feature of the BHH and warrants further discussion and reasoning. These statements further support the hypothesis that there might not be a no single specific heuristic that is the best heuristic to train the FFNN during the entire training process. In the succeeding sections, it is shown how this statement is further supported as the behaviour of the BHH is shown to be problem-dependent.

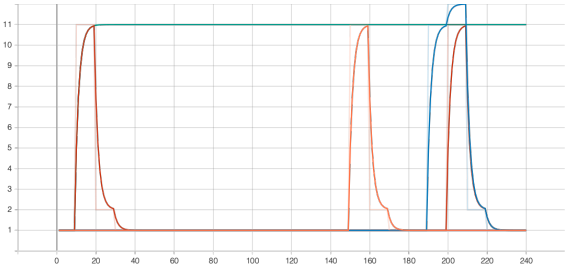
A clear distinction between Figures 8.1 and 8.2 should be pointed out. Figures 8.1 plot metrics for every epoch, while Figure 8.2 plot metrics for every mini-batch iteration. This is a subtle difference in visualisation, but nonetheless important. A more fine-grained visualisation illustrates the learning opportunities better since it is clear that a single step of learning happens on a single mini-batch iteration. This is important to keep in mind for various different sizes of datasets.

Due to the Bayesian nature of the BHH, one can expect the plots from figure 8.2 to drive the plots presented in 8.3 as these two parameters are related. The parameter α is the concentration parameter for the Dirichlet probability distribution represented by θ. The reader is now urged to consider Figures 8.2g and 8.3g together. These two figures illustrate the above mentioned Bayesian concepts well. It should be noted that as α changes, θ will change. More specifically, the longer the "memory" of the BHH the greater the magnitude of α resulting in a sampled selection probability θ that converges to the "learnt" expected selection probability ($E[\theta]$). The impact of this concept (retained memory) is addressed in Sections 8.8.

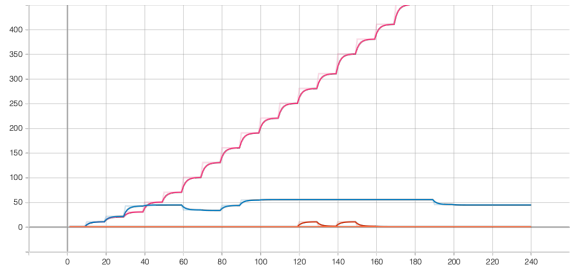
Another observation that can be made from Figures 8.2 and 8.3 is that none of the runs are the same. This is of course due to the stochastic nature of the BHH and the noise caused by using mini-batches. However, it highlights the fact that the learning process for the BHH has to be adaptive from run to run. This is evident in the plots presented in Figure 8.4 as one can see how the heuristic selection probability changes over time. This is proof that the BHH is learning something, but it is yet to be established what exactly it is learning. It is said that the BHH updates its *beliefs* (prior knowledge) about which heuristic to select by biasing selection towards heuristic performance. It can therefore be concluded that any learning that is done, is done as a result of learning from performance. Since the performance criteria mechanism for the BHH, called the *credit assignment strategy* marks a successful event as one meeting the performance criteria. Although this event is very much a local event, the continuous application of it over all runs means that learning does account for positive learning towards the overall performance goals.



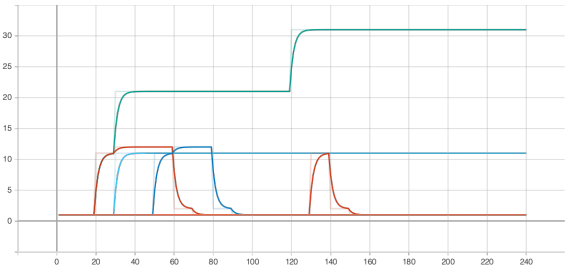
(a) α_0 - SGD



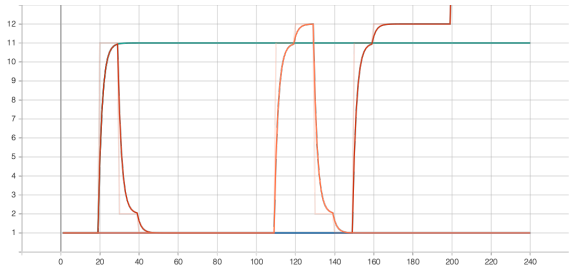
(b) α_1 - Momentum



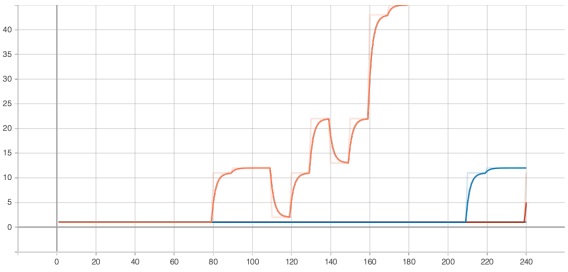
(c) α_2 - NAG



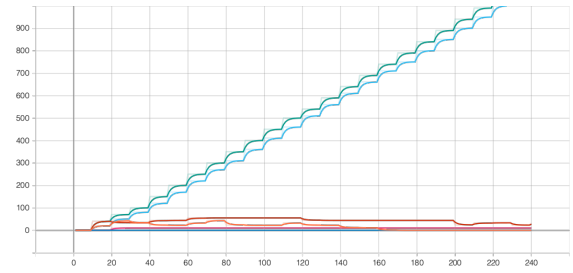
(d) α_3 - Adagrad



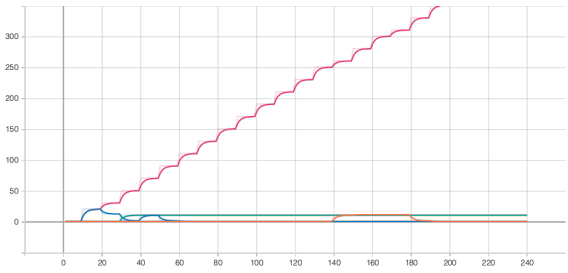
(e) α_4 - RMSProp



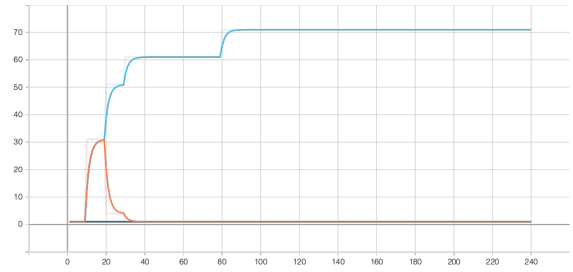
(f) α_5 - Adadelta



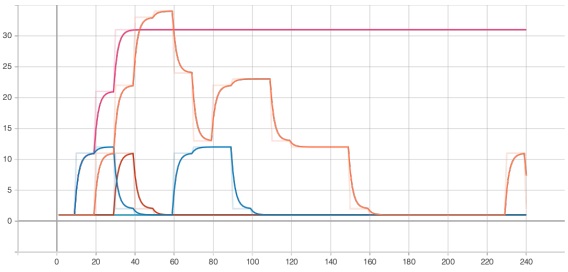
(g) α_6 - Adam



(h) α_7 - PSO

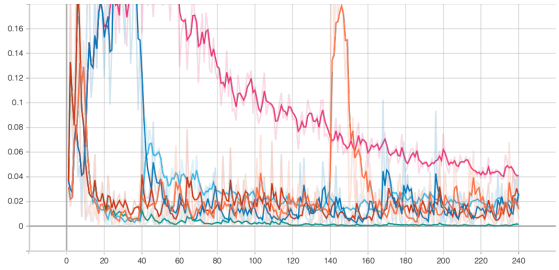


(i) α_8 - GA

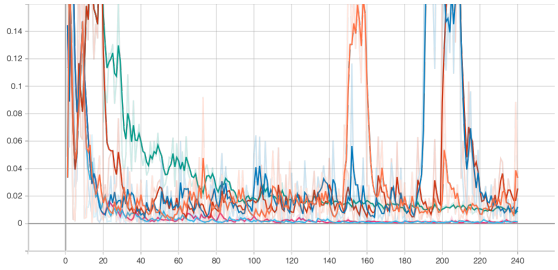


(j) α_9 - DE

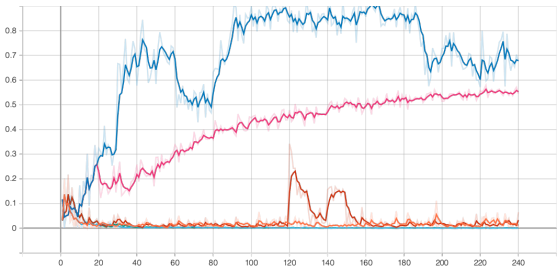
Figure 8.2: Change in the concentration of α during training for various heuristics



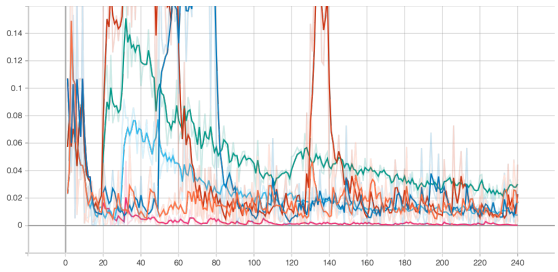
(a) $P(\theta_0|\alpha)$ - SGD



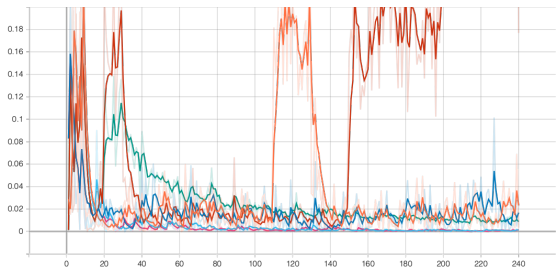
(b) $P(\theta_1|\alpha)$ - Momentum



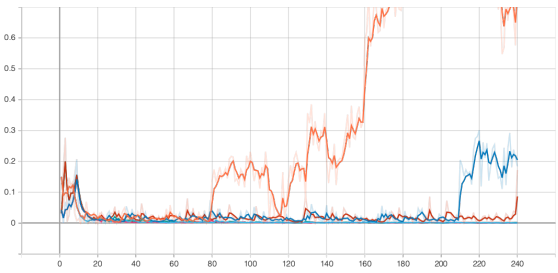
(c) $P(\theta_2|\alpha)$ - NAG



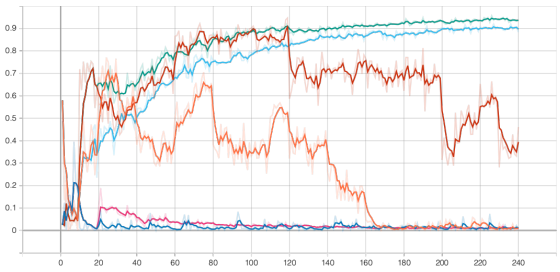
(d) $P(\theta_3|\alpha)$ - Adagrad



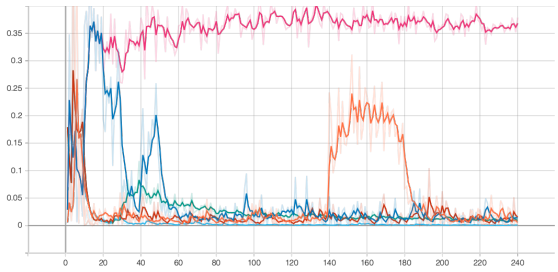
(e) $P(\theta_4|\alpha)$ - RMSProp



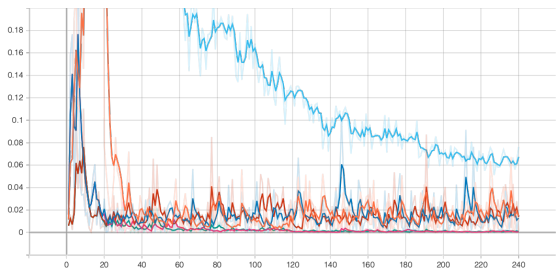
(f) $P(\theta_5|\alpha)$ - Adadelta



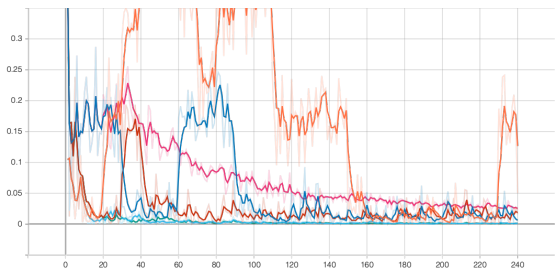
(g) $P(\theta_6|\alpha)$ - Adam



(h) $P(\theta_7|\alpha)$ - PSO

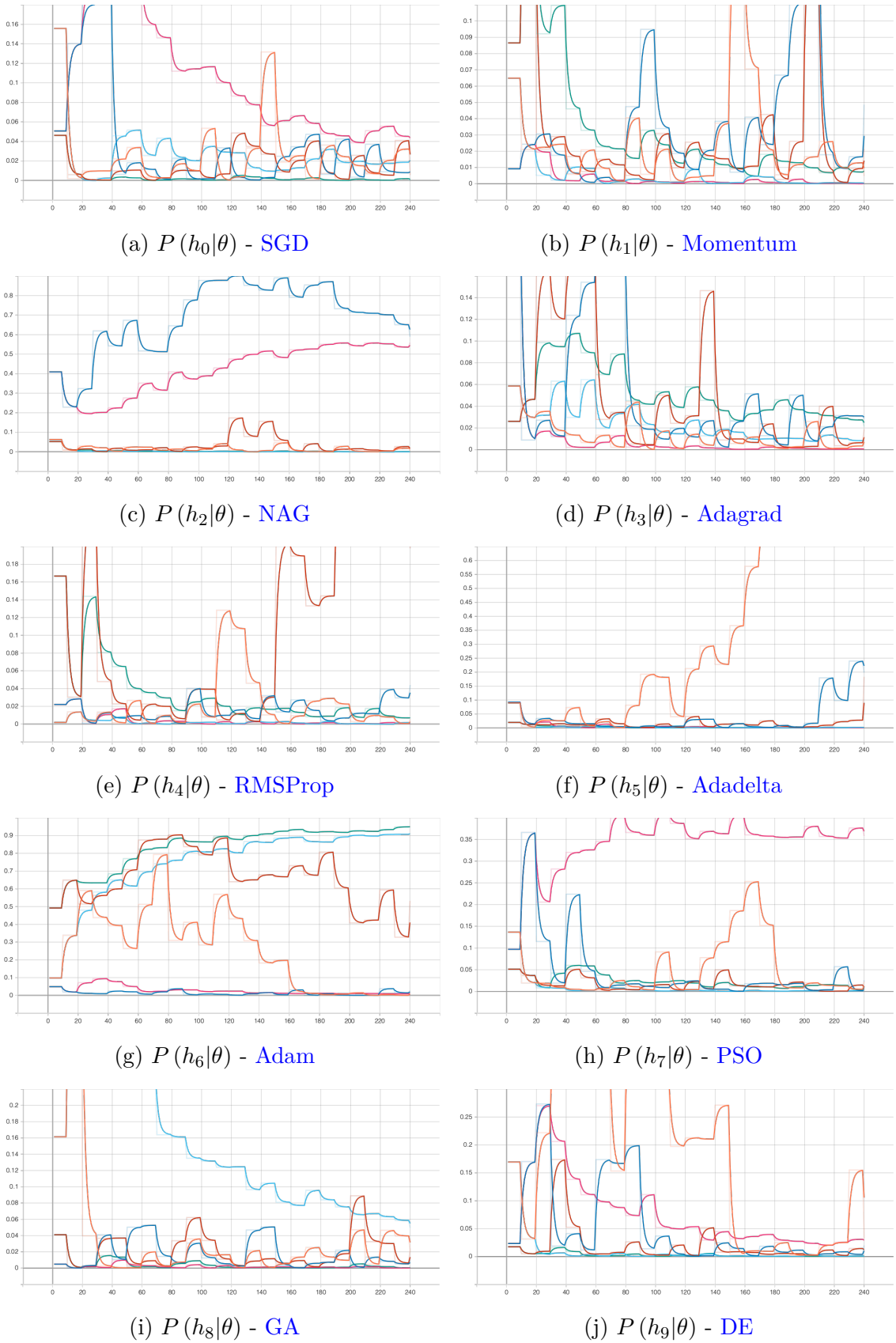


(i) $P(\theta_8|\alpha)$ - GA



(j) $P(\theta_9|\alpha)$ - DE

Figure 8.3: Change in the sampled selection probability of θ during training for various heuristics



(a) $P(h_0|\theta)$ - SGD

(b) $P(h_1|\theta)$ - Momentum

(c) $P(h_2|\theta)$ - NAG

(d) $P(h_3|\theta)$ - Adagrad

(e) $P(h_4|\theta)$ - RMSProp

(f) $P(h_5|\theta)$ - Adadelta

(g) $P(h_6|\theta)$ - Adam

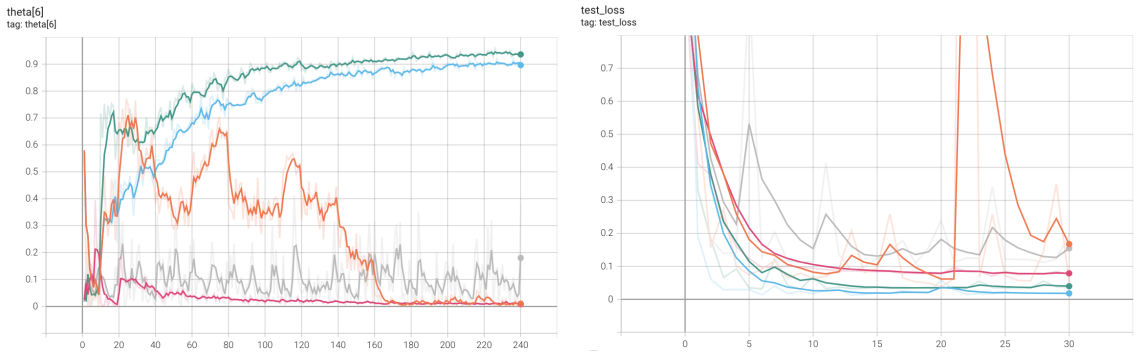
(h) $P(h_7|\theta)$ - PSO

(i) $P(h_8|\theta)$ - GA

(j) $P(h_9|\theta)$ - DE

Figure 8.4: Change in the prior heuristic selection probability $P(H|\theta)$ as the BHH learns during training

Consider now Figure 8.3g again, but this time include a configuration of the BHH where no Bayesian Analysis takes place, resulting in a purely random heuristic selection probability throughout. Such an configuration is given in Figure 8.5 below for the sampled heuristic selection probability for Adam (gray).



(a) $P(\theta_6|\alpha)$ (b) Test loss

Figure 8.5: Comparing baseline BHHs to that of a BHH with pure random selection.

One can see that the random configuration (gray) illustrated almost a fixed pattern behaviour, from which we can draw the following conclusions:

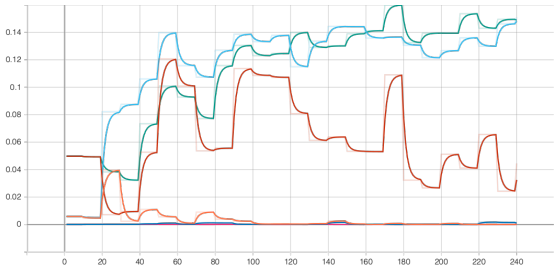
- Despite a completely random heuristic selection probability, the underlying BHH was still able to learn and perform relatively well. If the ability to learn is then purely based on the *perturbative* components of the BHH it can be concluded that the proxied heuristic step operations are successful.
- This shows that the BHH selection mechanism is indeed facilitating a learning process and since Adam is known to perform well on the iris dataset, it can be concluded that the learning process is successful and beneficial to the outcome of the BHH.

The statements made above conclude the successful working of the BHH. Both the perturbative element (through proxied heuristic update steps) and the selection mechanism (through Bayesian analysis) is shown to work successfully. Figure 8.6 shows the resulting output of the predictive model for each of the five entities in the heuristic pool. This predicted heuristic selection probability is used to parameterise a new categorical distribution from which the heuristics are finally sampled and (re)selected.

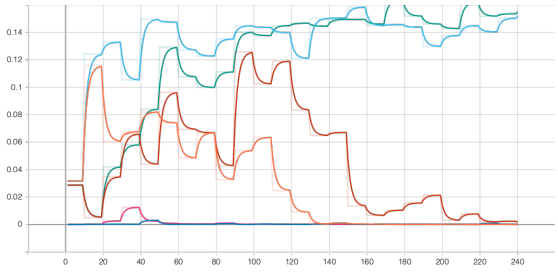
Figure 8.7 show the resulting selected heuristic, per entity, over time. From these plots, one can see how convergence on Adam occurs at least at some point during training for the majority of the entities.

Finally, Figure 8.8 below shows all of the probabilistic components involved in the selection mechanism of the BHH.

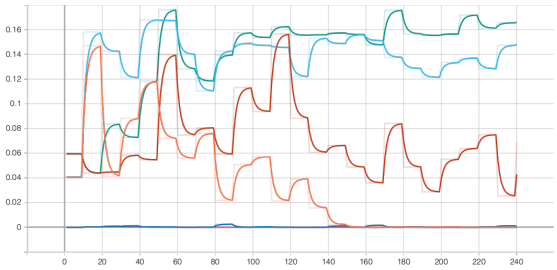
The behaviour of the BHH has been studied and it is found to be successful in its intent to learn which heuristics to use throughout the training process. It has also been



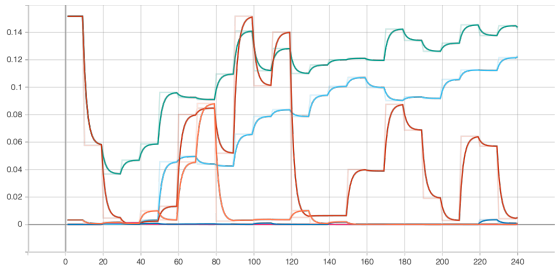
(a) $P\left(h_6|e_0, c_1\right)$ - entity 0 + Adam



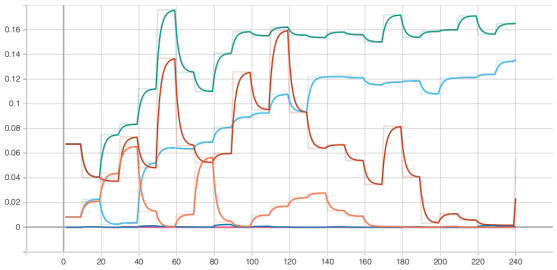
(b) $P\left(h_6|e_1, c_1\right)$ - entity 1 + Adam



(c) $P\left(h_6|e_2, c_1\right)$ - entity 2 + Adam

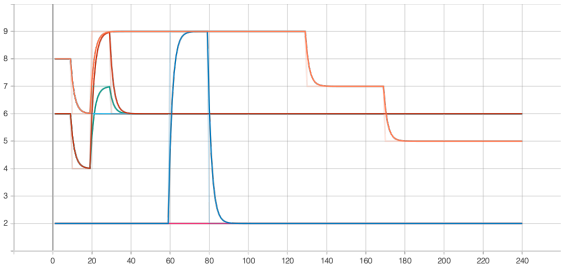


(d) $P\left(h_6|e_3, c_1\right)$ - entity 3 + Adam

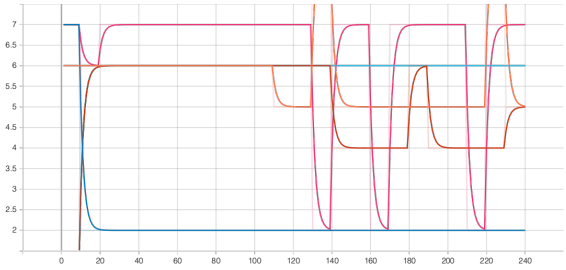


(e) $P\left(h_6|e_4, c_1\right)$ - entity 4 + Adam

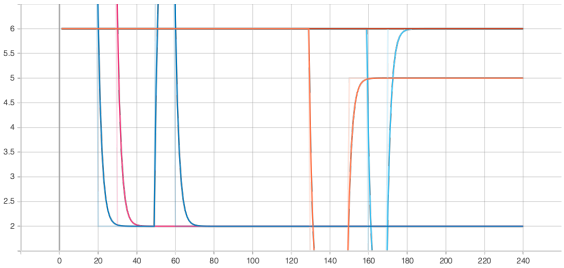
Figure 8.6: Change in the realised output of the predictive model $P\left(H|E, C; \theta, \phi, \psi\right)$ showcasing the selection of the Adam optimiser/low-level heuristic as an example



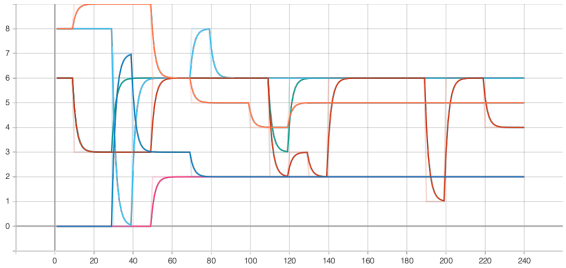
(a) HgEC[0]- entity 0



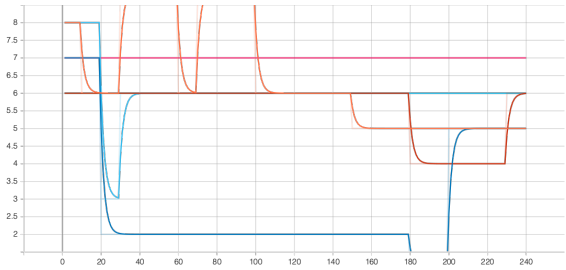
(b) HgEC[1]- entity 1



(c) HgEC[2]- entity 2

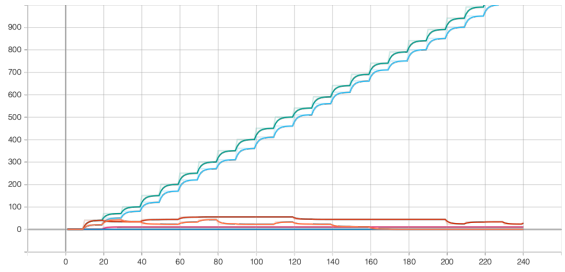


(d) HgEC[3]- entity 3

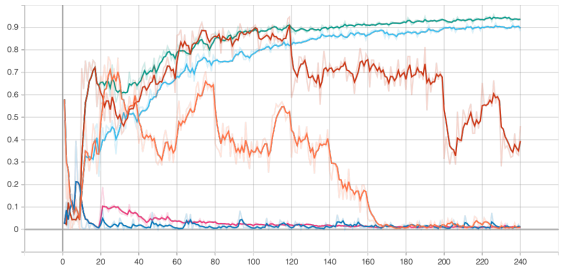


(e) HgEC[4]- entity 4

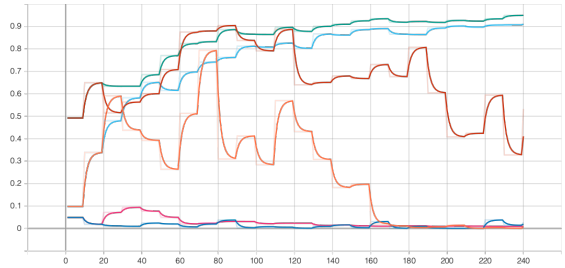
Figure 8.7: Change in selected heuristics (HgEC) during training for various entities



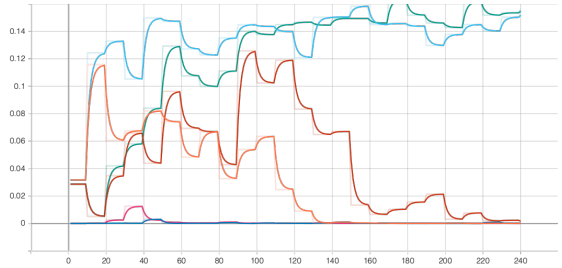
(a) α_6 - Adam



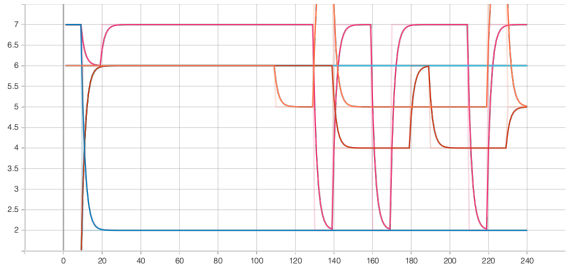
(b) $P(\theta_6|\alpha)$ - Adam



(c) $P\left(h_6|\theta\right)$ - Adam



(d) $P\left(h_6|e_1, c_1\right)$ - entity 1 + Adam



(e) HgEC[1]- entity 1

Figure 8.8: An illustration that shows the effects of learning as for Adam

shown that the training of the FFNNs with BHHs is done by means of a heuristic selection mechanism and proxied heuristic update steps, both which have been shown to attribute to the successful training of the FFNNs.

Although these observations are good, these are still high level and not based on any statistical data. It is purely based on observation over a few runs. The following section aims to provide the first set of conclusions based on a statistically certain comparison of the BHH's performance to that of standalone low-level heuristics.

8.3 Standalone vs BHH Baseline

This section provides the empirical results of the experiment group that compare the performance of the BHH to that of well-known, standalone, low-level heuristics. Table 8.1 below shows the count, average and standard deviation of the ranks achieved by die various heuristics for all timestamps, across all datasets. Furthermore, the table orders the heuristics by a normalised overall rank.

From Table 8.1 it can be seen that heuristic performance is problem/dataset specific. Despite this fact, it can be seen that the BHH managed to perform fairly well across almost all datasets (apart from the adult dataset for which discussions follow later) and achieved a normalised overall rank of 5 out of 12. As mentioned before, the rank should be taken with a pinch of salt, however, it does show the general capabilities of the BHH. It is expected that the performance of the BHH should be close to that of the best heuristic for that particular dataset. Both the rank and test loss metrics are indicative of this. Table 8.2 shows the results of the ANOVA statistical test and it shows that the results are statistically significant between the heuristics.

Table 8.3 presents the results of the Post Hoc Tukey test and Figure 8.9 provides a descriptive plot that provides an alternative visualisation to see how the performance of the BHH compares across datasets and how it compares to other low-level heuristics.

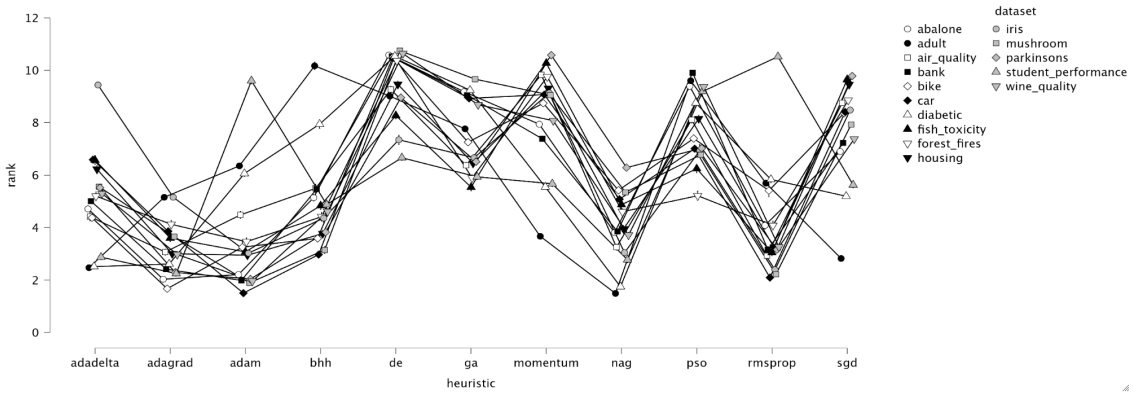


Figure 8.9: Descriptive plots between the average ranks of standalone heuristics compared to the baseline BHH per dataset, across all runs and steps.

Table 8.1: Empirical results showcasing rank statistics for different standalone heuristics compared to the baseline BHH across multiple datasets

step	(All)											
		heuristic										
dataset	statistic	adagrad	adam	nag	rmsprop	bhh	adadelta	ga	sgd	pso	momentum	de
abalone	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.0323	2.2086	3.8108	4.0849	5.1430	4.7151	9.1839	6.9043	9.3914	7.9441	10.5817
	std	1.3436	1.5671	1.2760	2.2093	1.9327	1.1671	0.9030	0.7740	1.5268	0.9717	1.1291
adult	count	930	930	930	930	930	930	930	930	930	930	930
	avg	5.1570	6.3559	1.4914	5.6968	10.1710	2.4720	7.7688	2.8237	9.6043	3.6688	9.0140
	std	1.2838	1.4823	0.6870	1.4006	2.2688	1.2974	1.3461	1.1988	1.6718	1.3834	1.5614
air_quality	count	930	930	930	930	930	930	930	930	930	930	930
	avg	3.0688	4.4849	3.2527	2.9269	5.4989	4.4204	6.3785	8.7548	8.1151	9.8301	9.2688
	std	1.8148	2.0087	1.7266	2.0466	2.5018	2.5575	1.7707	1.4074	1.9775	1.3046	1.9578
bank	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.4161	1.9882	3.8516	3.1591	5.4688	5.0108	9.0505	7.2301	9.9043	7.3892	10.5312
	std	1.3873	1.3985	1.4056	1.8653	1.7489	0.9864	0.9955	0.9144	1.2374	0.8907	1.0445
bike	count	930	930	930	930	930	930	930	930	930	930	930
	avg	1.6710	3.2559	5.4247	5.4140	3.5903	4.3774	7.2591	8.3656	7.4000	8.7441	10.4978
	std	1.1854	3.2580	0.9310	3.7231	1.0534	1.0684	1.0992	1.3163	1.6037	1.2745	1.2717
car	count	930	930	930	930	930	930	930	930	930	930	930
	avg	3.8591	1.5022	5.0839	2.0946	2.9720	6.5903	8.9301	8.4108	7.0118	9.0731	10.4720
	std	0.8054	1.1602	0.6925	1.1095	1.0151	1.4439	1.2877	1.2333	1.2970	1.1743	1.2916
diabetic	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.6129	6.0634	1.7473	5.8387	7.9452	2.5129	9.2409	5.1968	8.7527	5.5516	10.5376
	std	1.3914	1.6944	1.2492	2.0128	2.4141	1.3843	1.2206	1.2711	0.9424	1.3025	1.0607
fish_toxicity	count	930	930	930	930	930	930	930	930	930	930	930
	avg	3.6032	3.0742	4.8742	3.0462	4.8387	6.5785	5.5419	9.6462	6.2484	10.2731	8.2753
	std	2.0999	2.0463	2.2127	1.8158	2.2145	2.7393	2.2031	1.2726	2.4357	1.1959	2.0085
forest_fires	count	930	930	930	930	930	930	930	930	930	930	930
	avg	4.1269	3.4634	4.6075	4.0860	4.4172	5.2011	5.8978	8.8591	5.2172	9.7538	10.3699
	std	2.4170	2.2577	1.7329	2.4576	2.5304	2.5924	1.9010	1.0073	2.6924	1.0954	1.7882
housing	count	930	930	930	930	930	930	930	930	930	930	930
	avg	3.0043	2.9484	3.9495	3.2505	3.7720	6.2290	6.4538	9.4570	8.1570	9.3140	9.4645
	std	1.6877	1.5719	2.0695	1.7785	1.9772	2.1729	1.6569	1.3649	1.9825	1.3537	1.8465
iris	count	930	930	930	930	930	930	930	930	930	930	930
	avg	5.1613	3.0269	3.0462	2.3925	4.3624	9.4419	6.6667	8.4839	6.9333	9.1344	7.3505
	std	1.2630	1.9064	1.6829	1.5569	2.4400	1.6211	1.4704	1.2011	3.6375	1.2817	2.7492
mushroom	count	930	930	930	930	930	930	930	930	930	930	930
	avg	3.6548	1.8935	5.3484	2.2183	3.1484	5.5613	9.6634	7.9290	6.7774	9.0559	10.7495
	std	0.8854	1.5571	0.8150	1.1594	2.0245	0.9914	0.9675	0.8373	0.7826	0.9205	1.2513
parkinsons	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.2925	2.0376	6.2882	3.1634	3.8269	5.5226	6.5172	9.7839	7.0301	10.5785	8.9591
	std	1.2860	1.4103	1.1341	1.9798	1.4677	1.8893	1.3465	0.9741	1.4772	1.1491	1.1386
student_performance	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.2495	9.6022	2.7731	10.5204	4.8806	2.8742	5.9376	5.6280	9.1957	5.6731	6.6656
	std	1.5171	1.8140	1.7252	1.1290	2.5544	1.5612	1.5940	1.7346	0.8806	1.6999	1.4815
wine_quality	count	930	930	930	930	930	930	930	930	930	930	930
	avg	2.9774	1.9634	3.7194	3.2624	4.6140	5.2989	8.6882	7.3806	9.3785	8.0796	10.6376
	std	1.7194	1.4491	1.6072	1.4218	1.5987	1.9742	1.2730	0.9310	1.4409	1.0672	1.1360
overall	**avg**	**3.1925**	**3.5913**	**3.9513**	**4.0770**	**4.9766**	**5.1204**	**7.5452**	**7.6569**	**7.9411**	**8.2709**	**9.5584**
normalised	**rank**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

Table 8.2: ANOVA - Rank - Standalone vs BHH Baseline

Cases	Sum of Squares	df	Mean Square	F	p
dataset	248.990	14	17.785	6.515	< .001
heuristic	699163.514	10	69916.351	25612.484	< .001
dataset * heuristic	421571.950	140	3011.228	1103.104	< .001
Residuals	418433.761	153285	2.730		

Figure 8.10 provides the critical difference plots for the average ranks of the heuristics across all datasets. From this one can see five statistically significant groups of heuristics, with the BHH falling in the middle in group three.

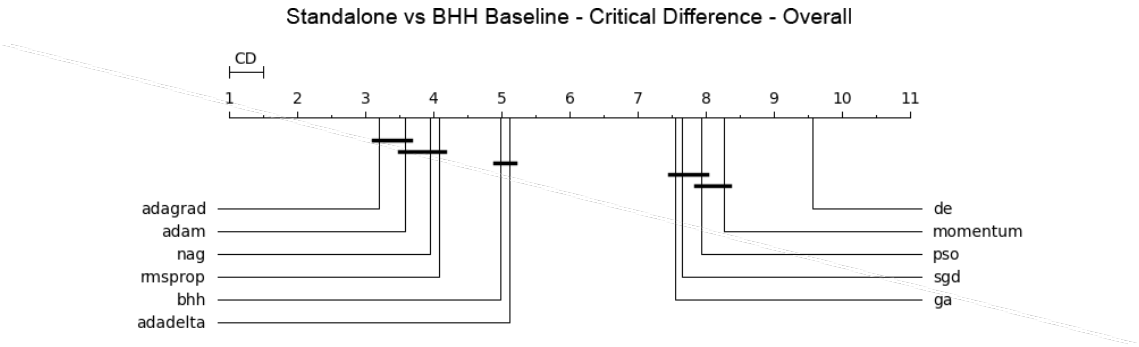


Figure 8.10: Critical difference plots between the average ranks of standalone heuristics compared to the baseline BHH, across all datasets, runs and steps.

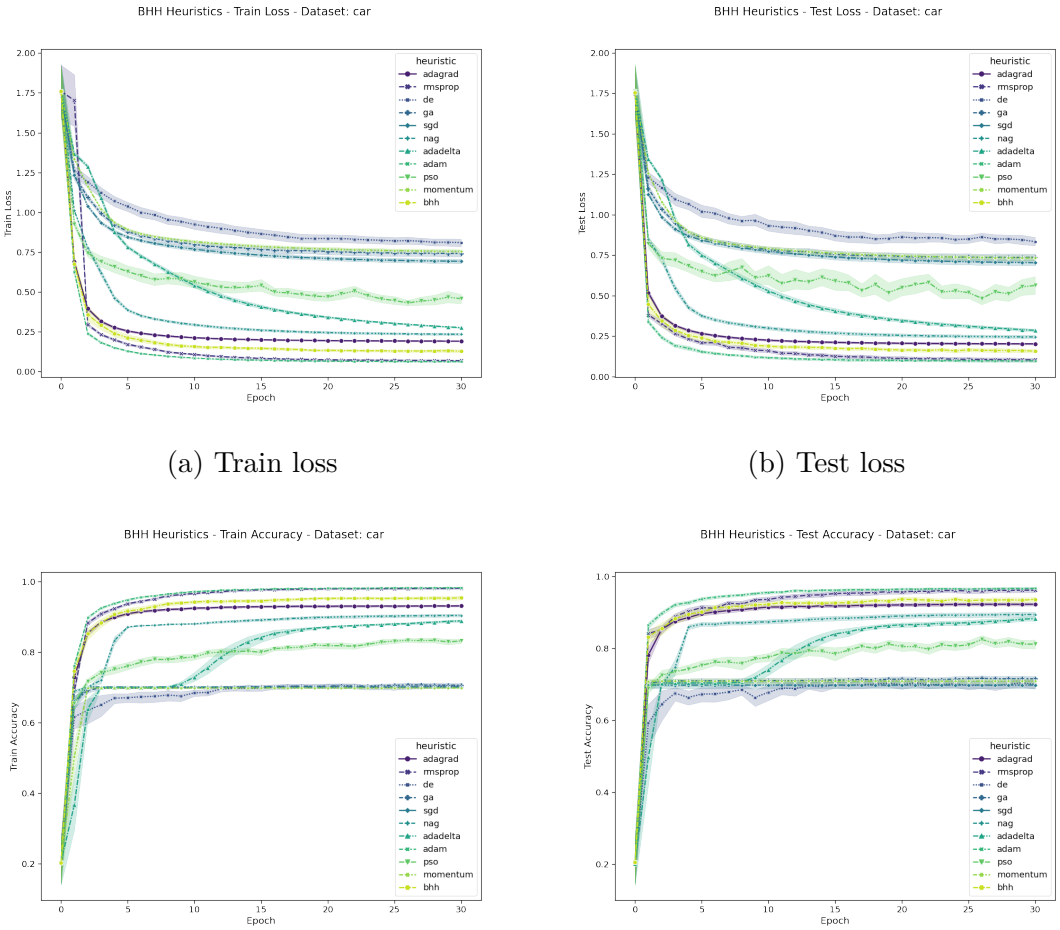
Figure 8.11 provides example training and test loss and accuracy plots for all heuristics for the car dataset and Figure 8.12 provides the same plots, but for the iris dataset. These plots are provided to visualise the training and test outcomes over time. Figure 8.12b was included to show how drastic the overfitting can become if early stopping is not employed. One can see that most of the gradient-based heuristics performed as expected, by the population-based approaches really struggle after learning has become stagnated. This is an important concept, because one has to remember that the BHH might "try-out" a certain heuristic and in a single step could cause the heuristic to generalise badly on the test set. Once again, the noise of mini-batches plays a large role here.

Finally, a collection of test loss plots for various datasets is given in Figures 8.13 and 8.14.

From this experimental group it was shown that the BHH was able to perform relatively well, but was not able to outperform the best heuristics. This is understandable as there is a lot of noise that makes the learning process imperfect. Although learning was shown to be successful, there are many suggestions that could lead to better outcomes. One such a suggestion is to add a validation step, similar to that of MCMC (ref???), before concentration parameters are accepted and updated. Another is to provide the BHH

Table 8.3: Post Hoc Comparisons - Standalone vs BHH Baseline

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
adadelta	adagrad	1.928	1.864	1.992	0.020	97.455	< .001
	adam	1.529	1.466	1.593	0.020	77.298	< .001
	bhh	0.144	0.080	0.207	0.020	7.269	< .001
	de	−4.438	−4.502	−4.374	0.020	−224.330	< .001
	ga	−2.425	−2.488	−2.361	0.020	−122.570	< .001
	momentum	−3.150	−3.214	−3.087	0.020	−159.251	< .001
	nag	1.169	1.106	1.233	0.020	59.100	< .001
	pso	−2.821	−2.884	−2.757	0.020	−142.583	< .001
	rmsprop	1.043	0.980	1.107	0.020	52.744	< .001
	sgd	−2.536	−2.600	−2.473	0.020	−128.216	< .001
adagrad	adam	−0.399	−0.462	−0.335	0.020	−20.158	< .001
	bhh	−1.784	−1.848	−1.720	0.020	−90.187	< .001
	de	−6.366	−6.430	−6.302	0.020	−321.786	< .001
	ga	−4.353	−4.416	−4.289	0.020	−220.026	< .001
	momentum	−5.078	−5.142	−5.015	0.020	−256.707	< .001
	nag	−0.759	−0.822	−0.695	0.020	−38.355	< .001
	pso	−4.749	−4.812	−4.685	0.020	−240.039	< .001
	rmsprop	−0.885	−0.948	−0.821	0.020	−44.711	< .001
	sgd	−4.464	−4.528	−4.401	0.020	−225.671	< .001
adam	bhh	−1.385	−1.449	−1.322	0.020	−70.029	< .001
	de	−5.967	−6.031	−5.903	0.020	−301.628	< .001
	ga	−3.954	−4.018	−3.890	0.020	−199.868	< .001
	momentum	−4.680	−4.743	−4.616	0.020	−236.549	< .001
	nag	−0.360	−0.424	−0.296	0.020	−18.197	< .001
	pso	−4.350	−4.414	−4.286	0.020	−219.881	< .001
	rmsprop	−0.486	−0.549	−0.422	0.020	−24.553	< .001
	sgd	−4.066	−4.129	−4.002	0.020	−205.513	< .001
bhh	de	−4.582	−4.645	−4.518	0.020	−231.599	< .001
	ga	−2.569	−2.632	−2.505	0.020	−129.839	< .001
	momentum	−3.294	−3.358	−3.231	0.020	−166.520	< .001
	nag	1.025	0.962	1.089	0.020	51.831	< .001
	pso	−2.965	−3.028	−2.901	0.020	−149.852	< .001
	rmsprop	0.900	0.836	0.963	0.020	45.476	< .001
	sgd	−2.680	−2.744	−2.617	0.020	−135.485	< .001
de	ga	2.013	1.949	2.077	0.020	101.760	< .001
	momentum	1.287	1.224	1.351	0.020	65.079	< .001
	nag	5.607	5.543	5.671	0.020	283.431	< .001
	pso	1.617	1.554	1.681	0.020	81.747	< .001
	rmsprop	5.481	5.418	5.545	0.020	277.075	< .001
	sgd	1.901	1.838	1.965	0.020	96.115	< .001
ga	momentum	−0.726	−0.789	−0.662	0.020	−36.681	< .001
	nag	3.594	3.530	3.658	0.020	181.670	< .001
	pso	−0.396	−0.460	−0.332	0.020	−20.013	< .001
	rmsprop	3.468	3.405	3.532	0.020	175.315	< .001
	sgd	−0.112	−0.175	−0.048	0.020	−5.645	< .001
momentum	nag	4.320	4.256	4.383	0.020	218.352	< .001
	pso	0.330	0.266	0.393	0.020	16.668	< .001
	rmsprop	4.194	4.130	4.258	0.020	211.996	< .001
	sgd	0.614	0.550	0.678	0.020	31.036	< .001
nag	pso	−3.990	−4.054	−3.926	0.020	−201.683	< .001
	rmsprop	−0.126	−0.189	−0.062	0.020	−6.356	< .001
	sgd	−3.706	−3.769	−3.642	0.020	−187.316	< .001
pso	rmsprop	3.864	3.800	3.928	0.020	195.328	< .001
	sgd	0.284	0.221	0.348	0.020	14.367	< .001
rmsprop	sgd	−3.580	−3.644	−3.516	0.020	−180.960	< .001



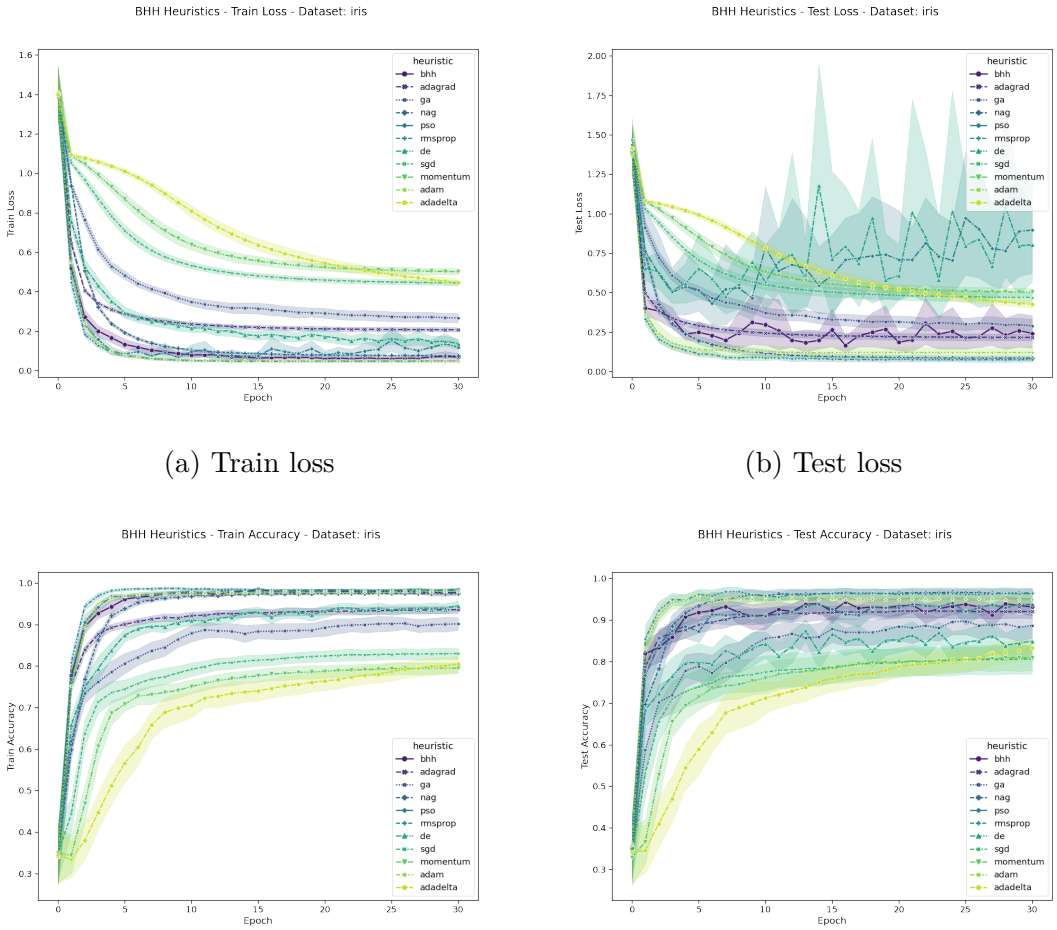
(a) Train loss

(b) Test loss

(c) Train accuracy

(d) Test accuracy

Figure 8.11: Standalone vs. BHH baseline - train, test loss and accuracy - dataset: car



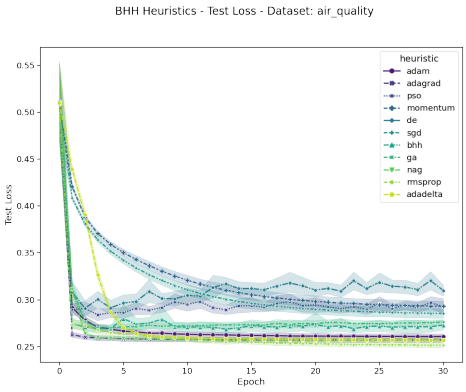
(a) Train loss

(b) Test loss

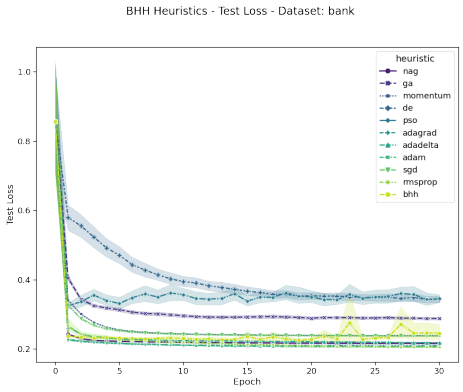
(c) Train accuracy

(d) Test accuracy

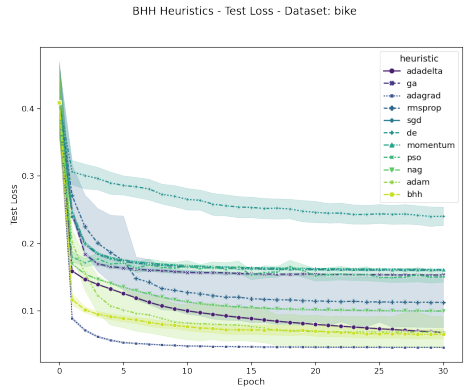
Figure 8.12: Standalone vs. BHH baseline - train, test loss and accuracy - dataset: iris



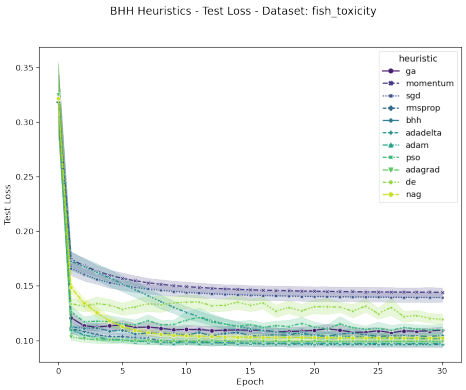
(a) air_quality



(b) bank

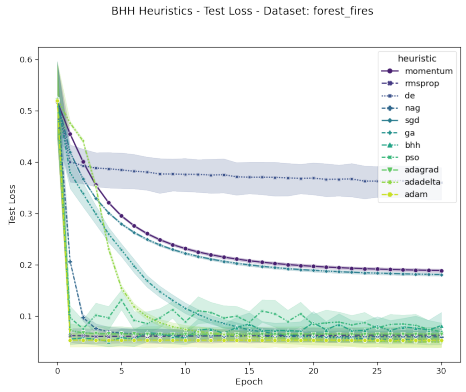


(c) bike

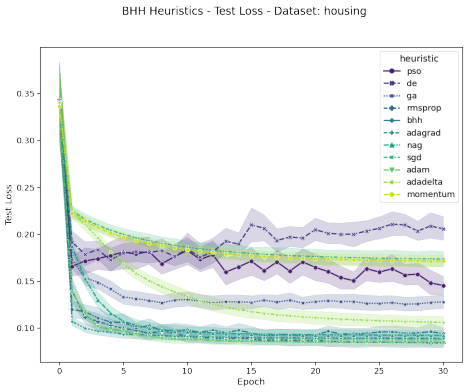


(d) fish_toxicity

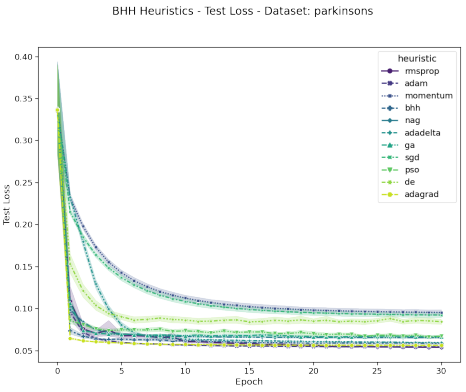
Figure 8.13: Standalone vs. BHH baseline - test loss across different datasets part 1



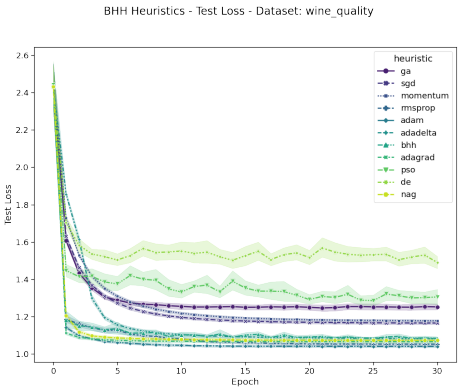
(a) forest_fires



(b) housing



(c) parkinsons



(d) wine_quality

Figure 8.14: Standalone vs. BHH baseline - test loss across different datasets part 2

with expert knowledge in its priors. For example: if it is known that a certain heuristic works well for a certain dataset, one could bias the priors by biasing the value of α, the concentration parameter for the prior probability distribution, to have an higher initial selection probability (compared to uniform/symmetric selection) for Adam. This is a major advantage that the BHH has above other heuristics, the ability to provide expert prior knowledge if needed.

The experiment is deemed successful and the results show promise for the field of HHs in general, but specifically in training FFNNs. The following sections delve into the hyper-parameters of the BHH and how it impacts its performance.

8.4 Heuristic Pool

This section provides the empirical results for an experimental group where the heuristic pool configuration is studied in comparison to the baseline BHH. As a reminder to the reader, the heuristic pool refers to the collection of low-level heuristics from which the BHH can select. The baseline BHH has a heuristic pool configuration that includes all ten of the low-level heuristics including gradient-based heuristics and meta-heuristics. Two alternative heuristic pools are then considered for investigation. These includes a configuration where the heuristic pool only consist out of gradient-based low-level heuristics and the other configuration includes only meta-heuristics. Table 8.4 presents the results of the average ranks, per heuristic, per dataset, across all runs, similar to what has been done for other experiments in this chapter.

As one would expect, the heuristic pool for gradient-based-only heuristics consistently do significantly better than the baseline and the meta-heuristic configurations. This is to be expected for the following reasons:

- Gradient-based approaches are currently the state-of-the art in FFNN training and is the standard approach for doing so. (Ref???)
- Switching between gradient-based approaches are easier as they require very little proxied state operations. Where they do require proxied state operations, almost all of them borrow from Adam, one of the best gradient-based heuristics at the time of writing.
- All the meta-heuristics included are population-based. Since an entire population can be updated at once, the impact of a wrong selection is much larger.

Although it is quite intuitive that the gradient-based heuristics would perform well, it does place a constraint on the BHH that the correct heuristics have to be included in the heuristic pool for the BHH to perform similar to the best heuristics. However, if a general pool of heuristics with a wide set of capabilities and characteristics is included, the

Table 8.4: Empirical results showcasing rank statistics for different heuristic pool configurations used by the BHH across multiple datasets

step	(All)

	pool	metrics							
	all			gd			mh		
dataset	count	avg	std	count	avg	std	count	avg	std
abalone	930	1.7946	0.6411	930	1.3828	0.5604	930	2.8226	0.4198
adult	930	2.8484	0.5058	930	1.1559	0.4077	930	1.8989	0.4104
air_quality	930	2.1871	0.8153	930	1.6828	0.7506	930	2.1301	0.7883
bank	930	1.7731	0.5617	930	1.3215	0.4920	930	2.9054	0.3341
bike	930	1.6247	0.5529	930	1.4387	0.5321	930	2.9366	0.2808
car	930	1.5903	0.5113	930	1.4409	0.5180	930	2.9688	0.2275
diabetic	930	2.4516	0.7353	930	1.2806	0.5756	930	2.2677	0.5798
fish_toxicity	930	1.9903	0.8485	930	1.8344	0.7684	930	2.1753	0.7959
forest_fires	930	2.0667	0.8338	930	1.8269	0.7813	930	2.1065	0.8066
housing	930	1.6043	0.6869	930	1.6602	0.6648	930	2.7355	0.5238
iris	930	1.8581	0.8005	930	1.7806	0.7272	930	2.3613	0.7960
mushroom	930	1.5204	0.5815	930	1.5484	0.5273	930	2.9312	0.2890
parkinsons	930	1.5570	0.6238	930	1.6011	0.5915	930	2.8419	0.4448
student_performance	930	1.9828	0.8220	930	1.7903	0.8480	930	2.2269	0.7152
wine_quality	930	1.5785	0.5498	930	1.4935	0.5630	930	2.9280	0.2938
overall	**13950**	**1.8952**	**0.7730**	**13950**	**1.5492**	**0.6651**	**13950**	**2.5491**	**0.6684**

BHH should perform reasonably well and generalise well on the test set. From the results presented throughout this section, it is shown that the performance of the BHH is itself problem-dependent. Therefore, one can not expect the BHH to perform equally well on all datasets. Earlier in Figure 8.1 it was shown that the BHH performed well on all datasets except for adult dataset. This is due to overfitting chaos early-on as will be shown later. However, if the gradient-only heuristic pool configuration was used, the BHH would've performed much better. As a reminder to the reader, the goal of this dissertation is not to develop a universal best heuristic, but rather one that could automate the tedious process of heuristic and parameter search, saving a lot of time.

Table 8.5 provides the results of the ANOVA statistical test and one can see the statistical significance of the results.

Table 8.5: ANOVA - Rank - BHH Variant: Heuristic Pool

Cases	Sum of Squares	df	Mean Square	F	p
dataset	2.710	14	0.194	0.495	0.938
heuristic_pool	7193.497	2	3596.749	9192.243	< .001
dataset * heuristic_pool	4376.103	28	156.289	399.430	< .001
Residuals	16357.497	41805	0.391		

Similar to the analysis for other experimental groups, the post hoc tukey test results are given in Table 8.6 below. From this table one can see how statistically different the gradient-based heuristic pool performs from the other configurations.

Table 8.6: Post Hoc Comparisons - BHH Variant: Heuristic Pool

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
all	gd	0.346	0.328	0.364	0.007	46.189	< .001
	mh	−0.654	−0.671	−0.636	0.007	−87.306	< .001
gd	mh	−1.000	−1.017	−0.982	0.007	−133.495	< .001

Figure 8.15 provides the descriptive plots for this experimental group. It shows the average rank achieved by heuristics across different datasets. Once again, we can see how the performance of the heuristics are problem-dependent.

As with the other sections, figure 8.16 provides the critical difference plots and finally, example test loss plots are given in Figure 8.17.

In this section it was established that the choice of heuristic pool has a significant impact on the outcome of the BHH. It was shown that the gradient-only pool significantly

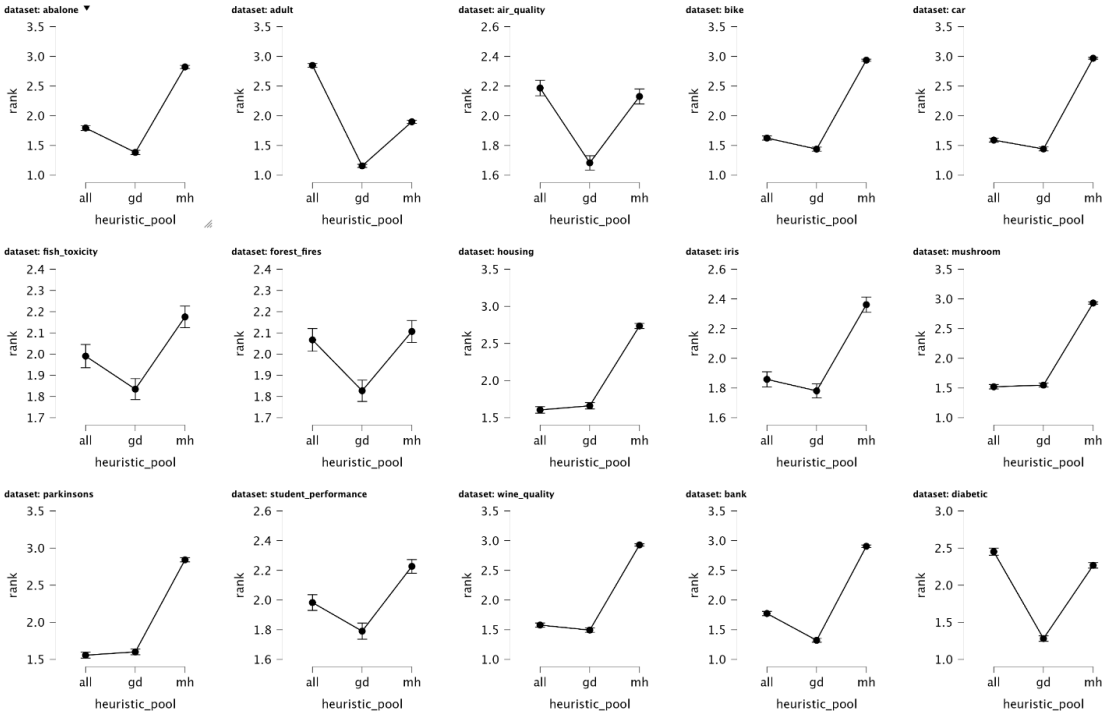


Figure 8.15: Descriptive plots between the average ranks of BHHs with varying heuristic pools per dataset, across all runs and steps.

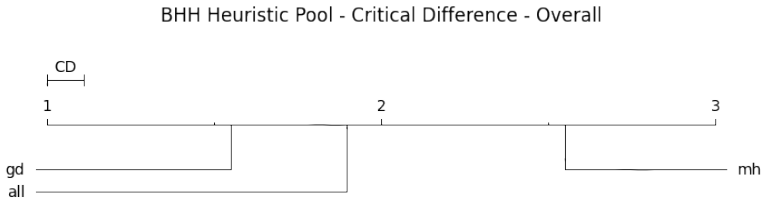
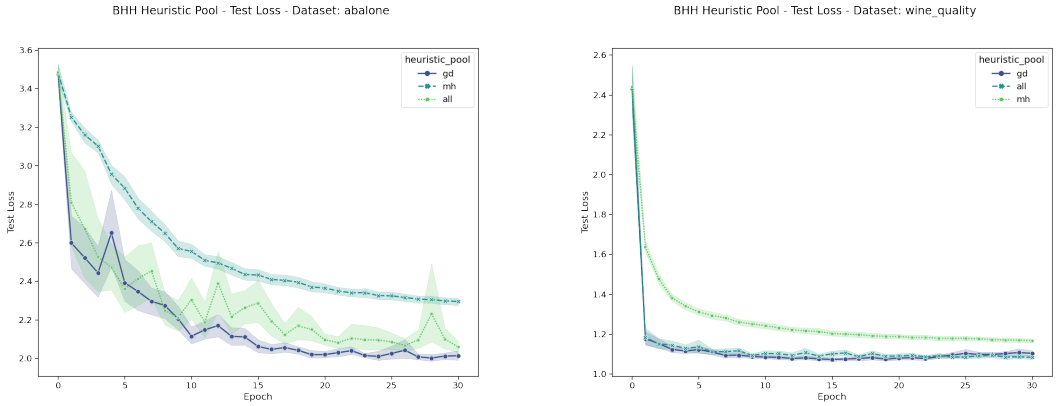


Figure 8.16: Critical Difference plots between the average ranks of BHHs with varying heuristic pools across all datasets, runs and steps.



(a) abalone (b) wine_quality

Figure 8.17: BHH Heuristic Pool - example test loss plots for varying heuristic pools

performs better than the baseline configuration that includes all the heuristics and the meta-heuristic-only pool.

The following section presents the empirical results for the experimental group that focuses on studying the effects of population size on the BHH.

8.5 Population

This section investigates the effects of population size on the BHH. Initially the expectation was that a larger population size would perform best as this is the general consensus for population-based heuristics (ref ????). However, it has been found that population size is largely problem-dependent with some datasets preferring a small population size and others a large population size. Table 8.7 presents these empirical results.

Table 8.7: Empirical results showcasing rank statistics for different population size values used by the BHH across multiple datasets

step	(All)														
	population	metrics													
	5			10			15			20			25		
dataset	count	avg	std	count	avg	std	count	avg	std	count	avg	std	count	avg	std
abalone	930	2.4387	1.3396	930	2.7452	1.3922	930	3.0129	1.3865	930	3.4065	1.3513	930	3.3968	1.3514
adult	930	3.5720	1.4693	930	3.2828	1.3751	930	2.8108	1.3625	930	2.6484	1.3222	930	2.3634	1.3063
air_quality	930	2.7527	1.4283	930	2.9258	1.3904	930	2.9656	1.3791	930	3.3140	1.3789	930	3.0419	1.4374
bank	930	2.8129	1.4102	930	3.1591	1.4348	930	2.9989	1.4199	930	3.0505	1.4438	930	2.9785	1.3414
bike	930	3.8957	1.3355	930	2.8688	1.2850	930	2.9645	1.3447	930	2.7430	1.3762	930	2.5280	1.3278
car	930	3.1892	1.3915	930	3.0409	1.3945	930	3.0398	1.4523	930	2.8559	1.3938	930	2.8742	1.4151
diabetic	930	2.7978	1.5078	930	2.9387	1.4133	930	3.2129	1.4525	930	2.9613	1.3036	930	3.0892	1.3534
fish_toxicity	930	2.8151	1.4249	930	3.0409	1.4288	930	3.0925	1.3230	930	3.1280	1.4073	930	2.9237	1.4634
forest_fires	930	3.0172	1.4026	930	2.9215	1.3800	930	2.9806	1.3833	930	3.0968	1.4124	930	2.9839	1.4879
housing	930	2.8366	1.4040	930	2.8258	1.3802	930	2.9753	1.4393	930	2.9849	1.4111	930	3.3774	1.3680
iris	930	2.8301	1.3038	930	2.8731	1.3970	930	3.0505	1.4129	930	3.1860	1.4251	930	3.0602	1.4987
mushroom	930	2.7989	1.2052	930	2.9441	1.3250	930	3.0462	1.4521	930	3.1366	1.4940	930	3.0538	1.5664
parkinsons	930	2.6645	1.4411	930	3.1065	1.3272	930	3.0247	1.4265	930	3.0495	1.4420	930	3.1548	1.3810
student_performance	930	2.6376	1.4357	930	2.8645	1.4108	930	3.1548	1.4042	930	3.2387	1.3970	930	3.1043	1.3395
wine_quality	930	2.5505	1.3165	930	2.7817	1.4152	930	3.1581	1.3841	930	3.1806	1.3941	930	3.3290	1.4140
overall	**13950**	**2.9073**	**1.4377**	**13950**	**2.9546**	**1.3904**	**13950**	**3.0325**	**1.4045**	**13950**	**3.0654**	**1.4108**	**13950**	**3.0173**	**1.4306**

Tables 8.8 presents the ANOVA statistical test results and Table 8.9 presents the post hoc Tukey test results. Both these tables present the results as statistically significant.

Table 8.8: ANOVA - Rank - BHH Variant: Population

Cases	Sum of Squares	df	Mean Square	F	p
dataset	17.974	14	1.284	0.659	0.816
population	225.672	4	56.418	28.961	< .001
dataset * population	3879.662	56	69.280	35.564	< .001
Residuals	135730.233	69675	1.948		

Table 8.9: Post Hoc Comparisons - BHH Variant: Population

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
5	10	−0.047	−0.093	−0.002	0.017	−2.831	0.037
	15	−0.125	−0.171	−0.080	0.017	−7.494	< .001
	20	−0.158	−0.204	−0.112	0.017	−9.458	< .001
	25	−0.110	−0.156	−0.064	0.017	−6.580	< .001
10	15	−0.078	−0.124	−0.032	0.017	−4.663	< .001
	20	−0.111	−0.156	−0.065	0.017	−6.627	< .001
	25	−0.063	−0.108	−0.017	0.017	−3.749	0.002
15	20	−0.033	−0.078	0.013	0.017	−1.965	0.283
	25	0.015	−0.030	0.061	0.017	0.914	0.892
20	25	0.048	0.003	0.094	0.017	2.878	0.033

From the descriptive plots given in Figure 8.18 one can see how radically different the outcomes are for each dataset. This clearly shows that population size, in most cases prefer a small population size, but there are multiple cases where a larger population size is preferred. Consider the plots for the abalone and adult datasets in this figure. They exhibit opposite outcomes for population size. Abalone prefers a small population size, while adult prefers larger population sizes. This makes the population size relatively problem dependent with a general tendency to prefer smaller population sizes.

Another interesting observation to make is that for many of the descriptive plots such as for student_performance, fish_toxicity, iris, mushroom and bank, the average rank and population size is not linearly correlated which suggests an interesting opportunity for further investigation.

Postulate for a moment why in some cases a smaller population size could be preferred. A larger population size means more combinations of heuristics and entities. Statistically these require many more samples to be able to achieve the same level of learning as with smaller populations. Since the nature of the BHH is in statistical and probabilistic components it makes sense that a larger sample leads to a harder learning and training process. As a reminder to the reader, the BHH selection mechanism is a posterior probability distribution that is dependent on an entity and performance criteria. The larger the number of entities, the bigger the number of combinations in which this posterior distribution can be realises. Furthermore, with larger population sizes, the difficulty of managing and maintaining consistent entity state from iteration to iteration, between the switching of heuristics, increases drastically. This means that invalid selections and bad state update operations are more likely to propagate throughout the population. This argument is yet

to be determined empirically, but it could be that the outcome of population size could be determined by the difference in importance between the selection mechanism's influence on the outcome or the perturbative mechanism's influence.

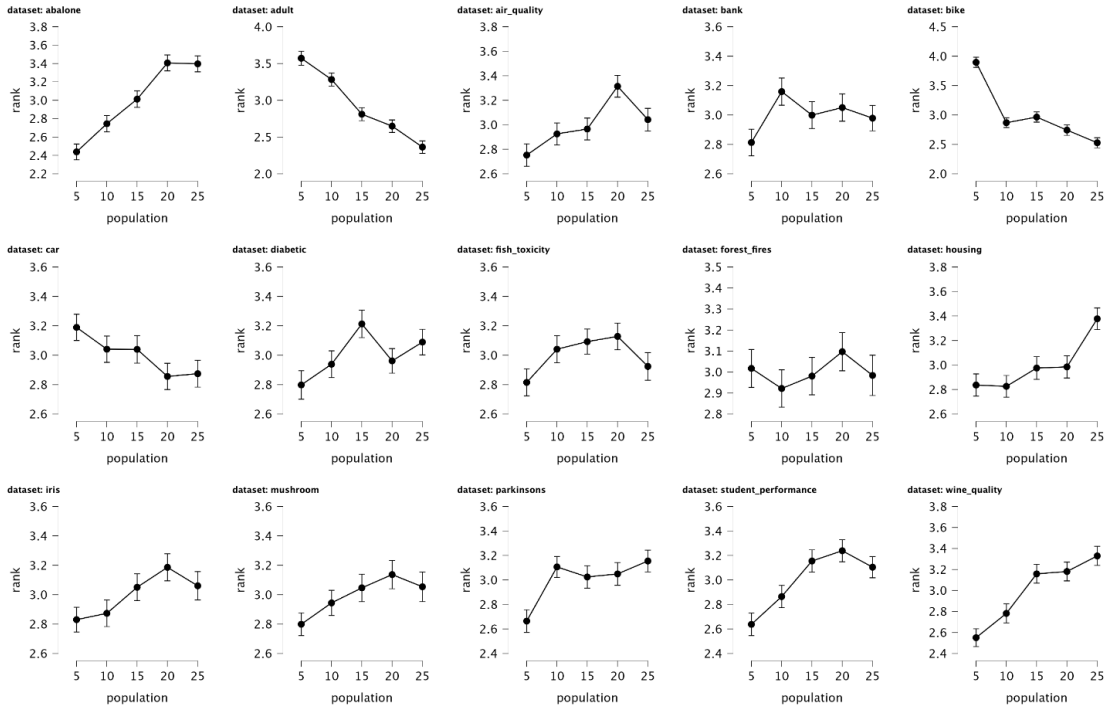


Figure 8.18: Descriptive plots between the average ranks of BHHs with varying population sizes per dataset, across all runs and steps.

As with the other experimental groups, the critical difference plot for the various population sizes over all datasets is given in Figure 8.19. The plot here shows just exactly how problem-depenent the population size is. This is evident from the insignificance between population sizes is you consider their performance over all datasets, yet the descriptive plots from Figure 8.18 clearly show statistically significant results for each configuration of population size for each dataset.

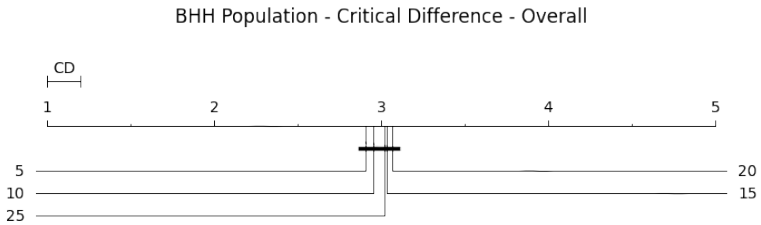
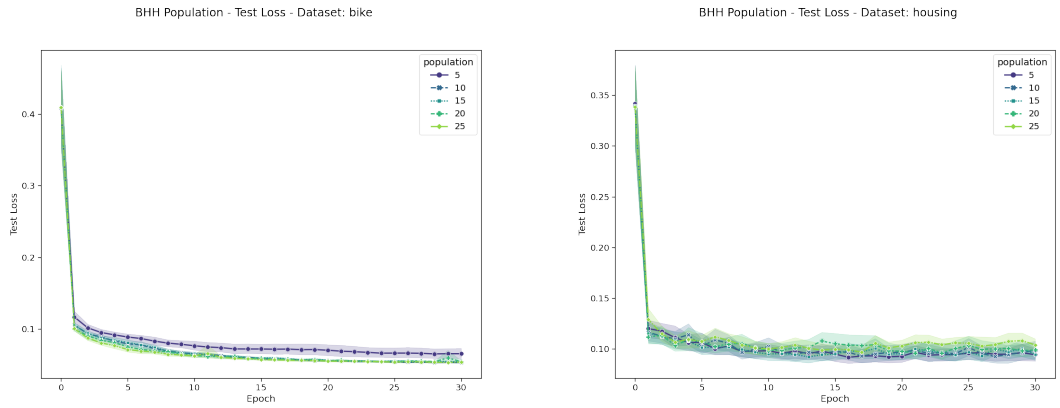


Figure 8.19: Critical Difference plots between the average ranks of BHHs with varying population sizes across all datasets, runs and steps.

Typically, as with the other sections, Figure 8.20 presents some example test loss plots for the bike and housing datasets. Here one can clearly see the differences in the runs for

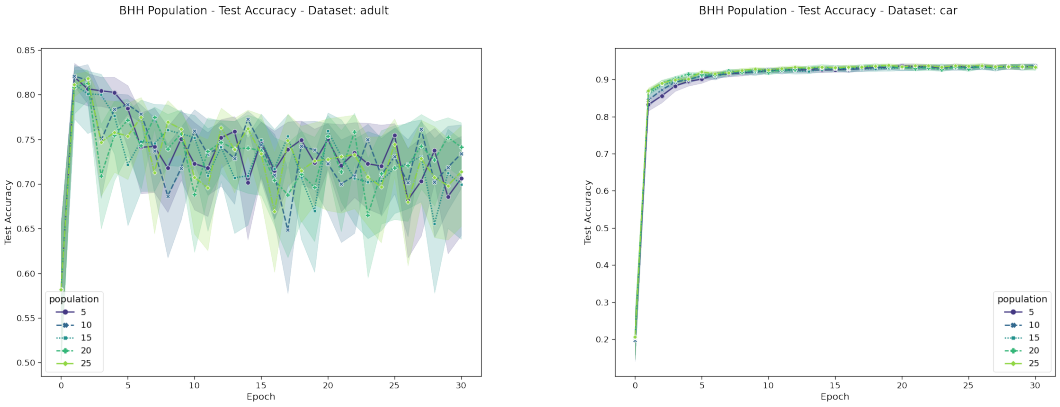
the various population size configurations.



(a) bike (b) housing

Figure 8.20: BHH Population - example test loss plots for BHHs with varying population sizes

Alternative to the test loss plots is Figure 8.20 which was carefully selected. This figure plots the test accuracy for adult, the dataset for which the BHH performed worst compared to standalone heuristics as was shown in Section 8.3. One can see that early stopping measures would've stopped the training process early (< 5 epochs), but since there is no early stopping, overfitting takes place in drastic effect. Notice how all population size configurations suffer from this problem.



(a) adult (b) car

Figure 8.21: BHH Population - example test accuracy plots for BHHs with varying population sizes

Another suggestion for investigation could be the relationship between entity population size and heuristic pool size. As a reminder, the baseline BHH has a default population size of 5. From the results, we can see that this was also the population size that performed well. In fact, it is the only case where population size $<$ heuristic pool size (10). There could

thus exists some relationship between these parameters, perhaps such that population size < heuristic pool size. Thus certainly warrants for further investigation in the future.

In this section it was found that population size does have a significant impact on the outcomes of the BHH and surprisingly, it was found that a smaller population size is preferred instead of large ones. The following section provides a detailed analysis of the credit assignment strategies.

8.6 Credit

This section provides the empirical results for the experimental group that focuses on studying the effects of different credit assignment strategies on the outcomes of the BHH. As a reminder to the reader, the BHH baseline makes use of the *ibest* credit assignment strategy. The empirical results are presented in Table 8.10 below. At first glance, it should be noticed that the selection of the correct credit assignment strategy to use is very problem-dependent.

Table 8.10: Empirical results showcasing rank statistics for different credit assignment strategies used by the BHH across multiple datasets

step	(All)														
	credit	metrics													
	ibest			pbest			rbest			gbest			symmetric		
dataset	count	avg	std	count	avg	std	count	avg	std	count	avg	std	count	avg	std
abalone	930	3.1677	1.3188	930	2.8613	1.4588	930	2.9968	1.3993	930	2.9667	1.4547	930	3.0075	1.4214
adult	930	2.4355	1.1177	930	2.6763	1.5019	930	2.8065	1.5003	930	2.4355	1.1177	930	3.3538	1.5682
air_quality	930	3.1312	1.3472	930	3.0376	1.3566	930	2.5914	1.4646	930	3.1280	1.3358	930	3.1118	1.4871
bank	930	2.8591	1.3442	930	2.9903	1.4320	930	2.8882	1.4098	930	3.0441	1.4154	930	3.2183	1.4424
bike	930	3.0527	1.3401	930	3.0086	1.3935	930	3.0667	1.4825	930	2.8742	1.3393	930	2.9978	1.5028
car	930	3.1151	1.3562	930	3.0312	1.4826	930	3.2516	1.4024	930	2.6892	1.3540	930	2.9129	1.4111
diabetic	930	2.8914	1.3488	930	2.6269	1.4171	930	3.4151	1.3653	930	2.8925	1.3620	930	3.1742	1.4487
fish_toxicity	930	3.1516	1.4360	930	2.9903	1.4521	930	2.7581	1.4748	930	3.2043	1.2988	930	2.8957	1.3579
forest_fires	930	3.0968	1.2771	930	3.1806	1.3040	930	2.8559	1.3562	930	3.1215	1.5018	930	2.7452	1.5627
housing	930	2.9011	1.4938	930	2.8527	1.3168	930	2.9022	1.3974	930	3.3108	1.3243	930	3.0333	1.4833
iris	930	3.1892	1.4281	930	3.0839	1.4412	930	3.0591	1.3755	930	2.8237	1.4085	930	2.8441	1.3844
mushroom	930	2.8075	1.4594	930	3.0183	1.4107	930	2.8957	1.3985	930	3.0839	1.4178	930	3.1720	1.3813
parkinsons	930	2.5645	1.4835	930	2.8925	1.3429	930	3.5065	1.2190	930	3.0796	1.3920	930	2.9570	1.4548
student_performance	930	2.6624	1.3124	930	3.0290	1.4067	930	3.1892	1.3821	930	2.7978	1.4702	930	3.3215	1.3938
wine_quality	930	3.1871	1.3080	930	2.6366	1.4706	930	3.0140	1.3712	930	2.9419	1.4115	930	3.2204	1.4300
overall	**13950**	**2.9475**	**1.3805**	**13950**	**2.9277**	**1.4222**	**13950**	**3.0131**	**1.4209**	**13950**	**2.9596**	**1.3921**	**13950**	**3.0644**	**1.4594**

Table 8.11 provides the ANOVA statistical test results and it can be seen that these credit assignment strategies are indeed statistically signficant from each other. Table 8.12 provides the post hoc Tukey test results.

The descriptive plots as presented in Figure 8.22 shows how different credit assignment strategies are prefered for different datasets. Interestingly, for the forest fires dataset, the preferred credit assignment strategy is statistically the *symmetric* credit assignment strategy. As a reminder to the reader, the symmetric credit assignment strategy marks all performance outcomes as successful and thus as a result, no performance bias occurs, rendering the selection mechanism for the BHH useless since priors will never be updated.

Table 8.11: ANOVA - Rank - BHH Variant: Credit

Cases	Sum of Squares	df	Mean Square	F	p
dataset	289.361	14	20.669	10.532	< .001
credit	172.812	4	43.203	22.015	< .001
dataset * credit	2680.102	56	47.859	24.387	< .001
Residuals	136733.281	69675	1.962		

Table 8.12: Post Hoc Comparisons - BHH Variant: Credit

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
gbest	ibest	0.012	−0.034	0.058	0.017	0.718	0.952
	pbest	0.032	−0.014	0.078	0.017	1.898	0.319
	rbest	−0.054	−0.099	−0.008	0.017	−3.192	0.012
	symmetric	−0.105	−0.151	−0.059	0.017	−6.248	< .001
ibest	pbest	0.020	−0.026	0.066	0.017	1.180	0.763
	rbest	−0.066	−0.111	−0.020	0.017	−3.910	< .001
	symmetric	−0.117	−0.163	−0.071	0.017	−6.966	< .001
pbest	rbest	−0.085	−0.131	−0.040	0.017	−5.090	< .001
	symmetric	−0.137	−0.182	−0.091	0.017	−8.146	< .001
rbest	symmetric	−0.051	−0.097	−0.005	0.017	−3.056	0.019

Notice that this does not necessarily make the symmetric credit assignment strategy completely random. With the random strategy, a single heuristic is selected and gets a successful count, where with symmetric credit assignment strategy gives a successful count for all selected heuristics.

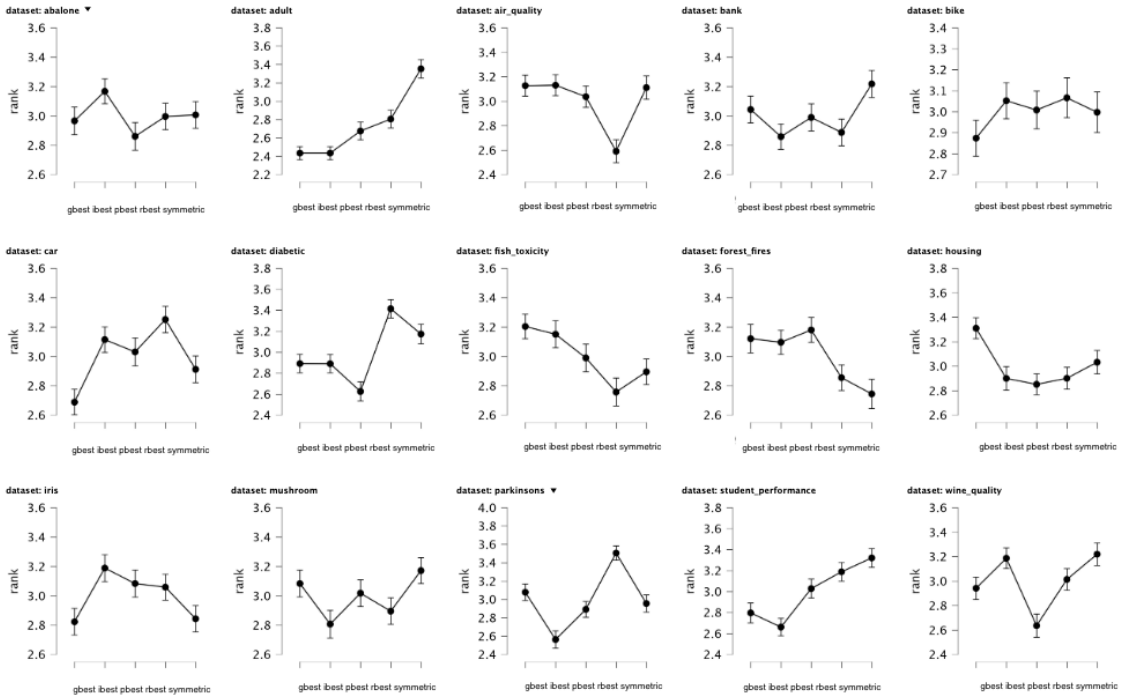


Figure 8.22: Descriptive plots between the average ranks of BHHs with varying credit assignment strategies per dataset, across all runs and steps.

The critical difference provided in Figure 8.23 supports the statement that the selection of the credit assignment strategy to use is problem-dependent.

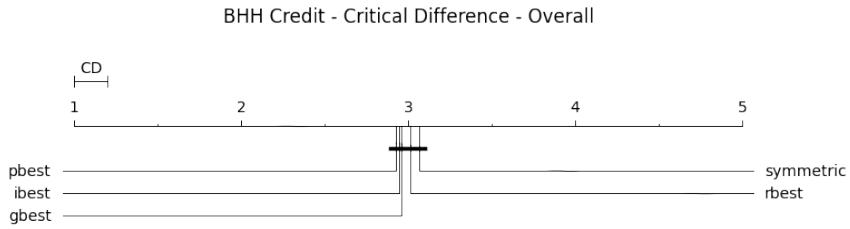
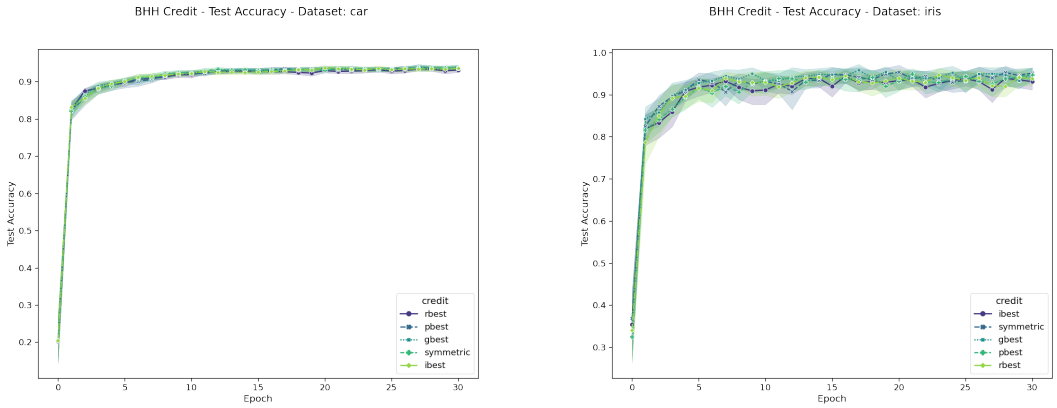


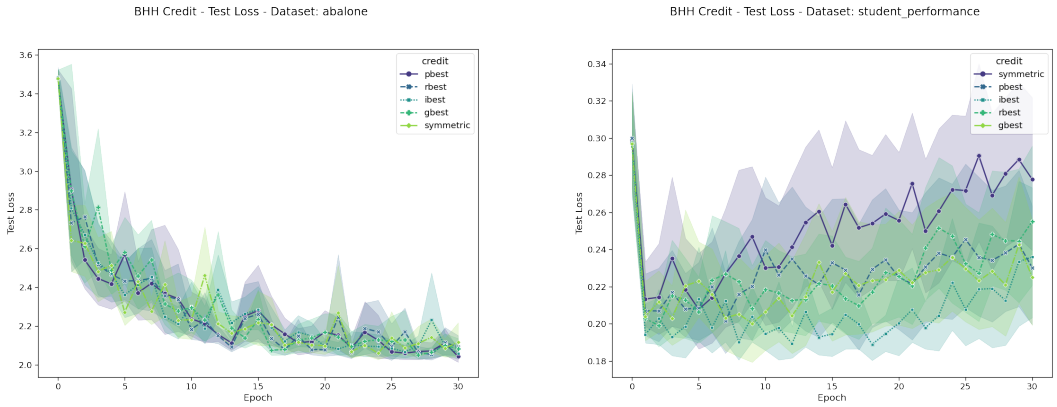
Figure 8.23: Critical Difference plots between the average ranks of BHHs with varying credit assignment strategies across all datasets, runs and steps.

Figure 8.24 provides good example test accuracy plots as for the car and iris datasets and Figure 8.25 provides example test loss plots as for the abalone and student_performance datasets. Figure 8.25b have been included to show the effects of the credit assignment stategies after overfitting has started.



(a) car (b) iris

Figure 8.24: BHH Credit - example test accuracy plots for varying credit assignment strategies



(a) abalone (b) student_performance

Figure 8.25: BHH Credit - example test loss plots for varying credit assignment strategies

This section has shown that there is a statistical difference between the outcomes of different credit assignment strategies per dataset. However, the choice of credit assignment strategy to use is shown to be problem-dependent.

The following section presents the findings for an experimental group which includes different reselection configurations.

8.7 Reselection

This section provides the results for the next experimental group, which focuses on different reselection window sizes. As a reminder to the reader, the reselection window is the parameter that controls how often new heuristics are selected. The baseline BHH makes use of a reselection window size of 10.

Table 8.13 provides the empirical results for this experimental group. From these results it can be seen that a larger reselection window size is generally preferred. From a statistical point of view this makes sense. By lowering the frequency by which reselections happen, the BHH accumulates more information per entity, heuristic and credit combination. Since the BHH makes use of a probabilistic model, the bigger the sample size the better. However, this parameter can not be considered in isolation. Consider the replay window size and the reanalysis window size. These two parameters put constraints on the window of information that is considered for learning. If reselection is large, it does not make sense for the replay size to be low since the BHH will just forget information before reselection has occured. Furthermore, as mentioned before, there is inherently a trade-off between exploration and exploitation that is required, even at a HH level. Admittidly, it is difficult to balance the trade-off between exploration and exploitation. Further research could investigate dynamic techniques such as parameter schedules.

Another possibility to consider is that larger reselection window sizes could be required because of noise. Perhaps there are too many noisy elements between iterations such that the BHH then requires a more stable configuration to learn sufficiently. Much more investigations is required here to confirm if this is the case, however, this was out of the scope for this dissertation. However, from a argumentative side it does make sense. Every time reselection occurs, state update operations also change due to the perturbative nature of the BHH. This could lead to many invalid selections, which in turn leads to bad state updates and ultimately affect the learning process negatively. Allowing selected low-level heuristic more time to run and learn allows for more consistent state update operations between iterations, catering best for noise introduced through mini-batch iterations and invalid selections. Such investigation should pair well with early stopping and the suggestion that was made in 8.3 to add a validation step before a state update operations occur. This should insure that the update step operations only occur when needed and when valid. Nonetheless, an interesting result. The remainder of this section simply provides the results

in similar fashion to the sections before.

Table 8.13: Empirical results showcasing rank statistics for different reselection values used by the BHH across multiple datasets

step	(All)														
reselection	metrics														
	1			5			10			15			20		
dataset	count	avg	std	count	avg	std	count	avg	std	count	avg	std	count	avg	std
abalone	930	4.3548	0.9760	930	2.8473	1.2885	930	2.7505	1.2622	930	2.5602	1.2988	930	2.4871	1.3182
adult	930	1.7097	1.2671	930	4.4559	1.0052	930	3.4398	1.0453	930	2.7656	0.9837	930	2.3065	1.0670
air_quality	930	3.7355	1.4046	930	3.4075	1.3790	930	2.9688	1.2783	930	2.4237	1.2281	930	2.4645	1.2906
bank	930	4.5677	0.7039	930	3.6323	1.1271	930	2.4237	1.1333	930	2.2462	1.1139	930	2.1301	1.0957
bike	930	4.7935	0.6124	930	3.9204	0.6480	930	2.5161	0.8632	930	2.0763	0.9007	930	1.6935	0.8911
car	930	4.6409	0.7847	930	2.9527	1.1581	930	2.6548	1.1748	930	2.5312	1.2440	930	2.2204	1.2169
diabetic	930	3.0129	1.0478	930	4.4624	0.8688	930	2.7957	1.1753	930	2.6065	1.4391	930	2.1226	1.2639
fish_toxicity	930	3.7484	1.2377	930	3.1376	1.3174	930	3.0344	1.4062	930	2.4656	1.3039	930	2.6140	1.4318
forest_fires	930	3.8194	1.4330	930	3.2720	1.2477	930	2.8656	1.3464	930	2.8194	1.3048	930	2.2237	1.2186
housing	930	4.1892	1.0993	930	3.0172	1.2882	930	2.5871	1.3541	930	2.5312	1.2654	930	2.6753	1.3399
iris	930	2.9839	1.4244	930	2.9710	1.3165	930	3.3462	1.4182	930	3.0140	1.4054	930	2.6849	1.4289
mushroom	930	4.2065	1.2356	930	2.9688	1.2992	930	2.5860	1.2336	930	2.7882	1.3397	930	2.4505	1.2259
parkinsons	930	4.7645	0.7563	930	3.1882	1.0208	930	2.2957	1.2089	930	2.4968	1.1282	930	2.2548	1.0973
student_performance	930	3.9710	1.0942	930	4.0226	1.1196	930	2.4774	1.2206	930	2.2495	1.2002	930	2.2796	1.1324
wine_quality	930	4.0688	1.1220	930	2.9204	1.2934	930	2.9559	1.3831	930	2.4032	1.3779	930	2.6516	1.2796
overall	**13950**	**3.9044**	**1.3641**	**13950**	**3.4118**	**1.2914**	**13950**	**2.7799**	**1.2811**	**13950**	**2.5318**	**1.2660**	**13950**	**2.3506**	**1.2541**

Table 8.14 provides the ANOVA statistical test results where one can see the statistical significance of the results. As with the other sections, Table 8.15 provides the post hoc Tukey test results. From this table, one can see the statistical significance that exists between all the different reselection window sizes.

Table 8.14: ANOVA - Rank - BHH Variant: Reselection

Cases	Sum of Squares	df	Mean Square	F	p
dataset	18.065	14	1.290	0.895	0.564
reselection	23391.735	4	5847.934	4055.181	< .001
dataset * reselection	15911.336	56	284.131	197.027	< .001
Residuals	100477.574	69675	1.442		

The outcome of this experimental group is best illustrated in Figure 8.26 which provides the reader with descriptive plots of the reselection window size. For this experimental group, even though the outcomes are slightly different per dataset, it can be seen that there is a general pattern here that suggests larger selection window sizes across all datasets. The only real exception to this is the adult dataset, for which it has already been establish that the BHH overfits drastically. There is a slight difference in outcome in the diabetic dataset and that could also be for the same reason.

The critical difference plot as presented in 8.27 visually illustrates the statistical significance between average ranks achieved by the baseline BHH with various reselection windows size values.

Table 8.15: Post Hoc Comparisons - BHH Variant: Reselection

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
1	5	0.493	0.453	0.532	0.014	34.265	< .001
	10	1.125	1.085	1.164	0.014	78.211	< .001
	15	1.373	1.333	1.412	0.014	95.461	< .001
	20	1.554	1.515	1.593	0.014	108.064	< .001
5	10	0.632	0.593	0.671	0.014	43.946	< .001
	15	0.880	0.841	0.919	0.014	61.196	< .001
	20	1.061	1.022	1.100	0.014	73.799	< .001
10	15	0.248	0.209	0.287	0.014	17.250	< .001
	20	0.429	0.390	0.468	0.014	29.853	< .001
15	20	0.181	0.142	0.220	0.014	12.603	< .001

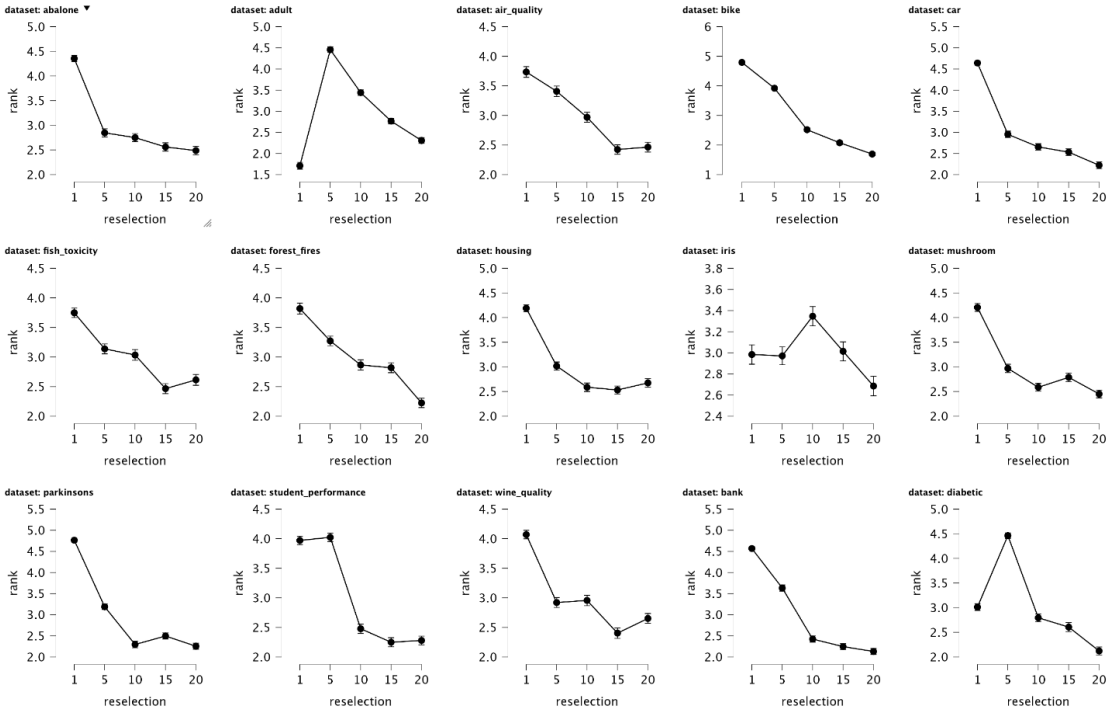


Figure 8.26: Descriptive plots between the average ranks of BHHs with varying reselection values per dataset, across all runs and steps.

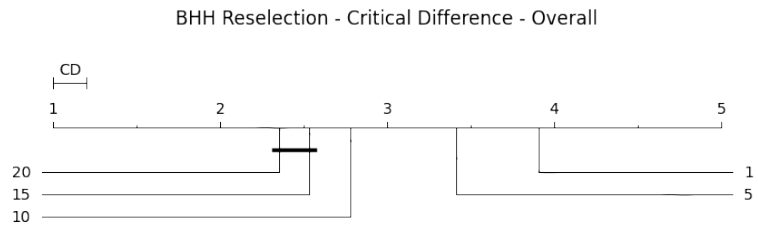
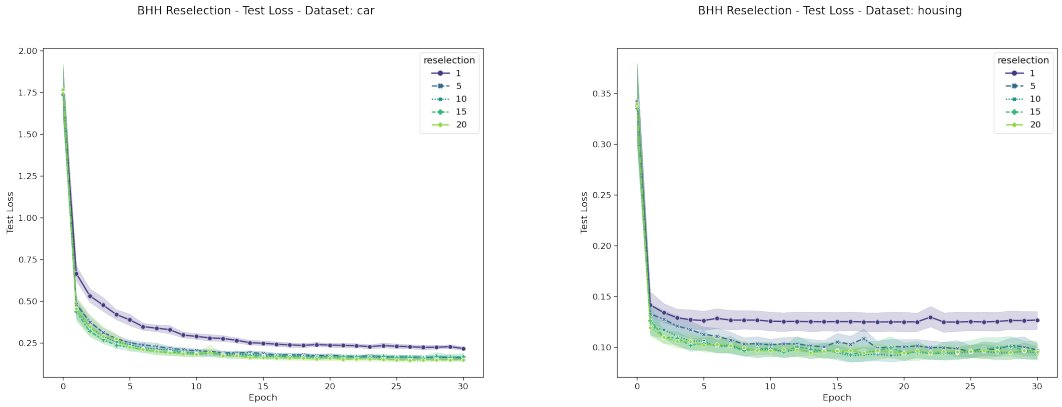


Figure 8.27: Critical Difference plots between the average ranks of BHHs with varying reselection values across all datasets, runs and steps.

Finally, Figure 8.28 provide example test loss plots for car and housing datasets where one can clearly see the difference in test loss outcomes with the larger reselection windows sizes is seen to perform best.



(a) car (b) housing

Figure 8.28: BHH Selection - example test loss plots for BHHs with varying reselection window sizes

The section has shown that a longer reselection window size is generally preferred. However, it has been identified that this component requires more investigation as to why this is the case. The next sections are closely related to reselection window sizes as they also influence the intervals between learning/selection. The next sections provides the results for the experimental group that focuses on the effects of different replay windows sizes and reanalysis window sizes.

8.8 Replay

In the previous section it was mentioned how the reselection windows size parameter is closely related to the replay window size and these parameters do have an impact on each other. This section provides the empirical results for the experimental group that focuses on the effects of different replay window sizes on the BHH. As a reminder to the reader,

the replay window is the size of the BHH's memory buffer i.e. how much information is kept in the performance log. If this number is low the BHH quickly forgets and if the number is too high it remembers information that is no longer applicable. Similar to some of the other parameters discussed in this chapter, it is then clear that this parameter is also subject to a balance between exploration and exploitation. The same recommendations that was made for reselection window sizes also account for replay window size.

Table 8.16 present the empirical results for this experimental group. It can be seen that the best value for the replay window size overall is 5, which is slightly smaller than the BHH baseline, but this value is clearly problem-dependent. This is further supported by the results presented in the ANOVA statistical test in Table 8.17 as well as the post hoc Tykey test presented in Table 8.18.

Table 8.16: Empirical results showcasing rank statistics for different replay window sizes used by the BHH across multiple datasets

step	(All)														
	replay	metrics													
	1			5			10			15			20		
dataset	count	avg	std	count	avg	std	count	avg	std	count	avg	std	count	avg	std
abalone	930	3.0258	1.3816	930	2.6785	1.4551	930	3.1968	1.3460	930	3.0903	1.3851	930	3.0086	1.4503
adult	930	2.8785	1.3824	930	2.8269	1.3595	930	2.9430	1.4804	930	3.0538	1.3714	930	2.9645	1.5686
air_quality	930	2.8742	1.3851	930	3.0151	1.4970	930	2.9978	1.4294	930	2.9505	1.2966	930	3.1624	1.4430
bank	930	2.9484	1.4597	930	3.0570	1.4731	930	2.8602	1.3439	930	3.0172	1.4202	930	3.1172	1.3592
bike	930	2.9602	1.2739	930	2.7366	1.3401	930	3.1452	1.4478	930	2.9226	1.4174	930	3.2355	1.5274
car	930	2.7054	1.3572	930	2.8376	1.3898	930	3.0763	1.3723	930	3.3269	1.4631	930	3.0538	1.4086
diabetic	930	3.2473	1.3690	930	3.3247	1.4058	930	2.5935	1.3719	930	2.8075	1.3844	930	3.0269	1.4113
fish_toxicity	930	2.7269	1.4190	930	3.0785	1.4896	930	3.1774	1.3980	930	2.9892	1.3865	930	3.0280	1.3373
forest_fires	930	2.8054	1.4926	930	2.9151	1.3812	930	2.9914	1.4081	930	3.0591	1.4111	930	3.2290	1.3416
housing	930	3.1548	1.3007	930	2.9774	1.4068	930	2.8548	1.4501	930	3.0355	1.4898	930	2.9774	1.4037
iris	930	3.0720	1.3757	930	3.1581	1.4187	930	3.2204	1.3919	930	2.6183	1.3964	930	2.9312	1.4102
mushroom	930	3.0538	1.3243	930	3.1430	1.4369	930	3.0419	1.4140	930	2.8613	1.3823	930	2.8957	1.4966
parkinsons	930	3.1720	1.3719	930	2.8430	1.2437	930	2.6731	1.4535	930	3.1172	1.4189	930	3.1946	1.4977
student_performance	930	3.0398	1.3684	930	2.9226	1.4106	930	2.6548	1.4028	930	3.0935	1.3225	930	3.2892	1.4874
wine_quality	930	3.1742	1.4300	930	2.7323	1.4556	930	3.1581	1.4179	930	3.0419	1.3565	930	2.8935	1.3624
overall	**13950**	**2.9892**	**1.3892**	**13950**	**2.9497**	**1.4224**	**13950**	**2.9723**	**1.4224**	**13950**	**2.9990**	**1.4021**	**13950**	**3.0672**	**1.4400**

Table 8.17: ANOVA - Rank - BHH Variant: Replay

Cases	Sum of Squares	df	Mean Square	F	p
dataset	19.257	14	1.375	0.695	0.781
replay	109.057	4	27.264	13.784	< .001
dataset * replay	1876.507	56	33.509	16.941	< .001
Residuals	137815.766	69675	1.978		

The descriptive plots presented in Figure 8.30 show the effect of different replay window sizes per dataset. The reader is reminded that the baseline BHH has a replay window size of 10. This value seems to be the best replay window size for the parkinsons and student

Table 8.18: Post Hoc Comparisons - BHH Variant: Replay

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
1	5	0.039	−0.006	0.085	0.017	2.346	0.131
	10	0.017	−0.029	0.063	0.017	1.005	0.853
	15	−0.010	−0.056	0.036	0.017	−0.579	0.978
	20	−0.078	−0.124	−0.032	0.017	−4.627	< .001
5	10	−0.023	−0.069	0.023	0.017	−1.341	0.666
	15	−0.049	−0.095	−0.003	0.017	−2.924	0.028
	20	−0.117	−0.163	−0.071	0.017	−6.973	< .001
10	15	−0.027	−0.073	0.019	0.017	−1.584	0.508
	20	−0.095	−0.141	−0.049	0.017	−5.632	< .001
15	20	−0.068	−0.114	−0.022	0.017	−4.048	< .001

performance datasets. There is no clear patterns between datasets. This makes sense, since each dataset has different training requirements, error landscapes and search spaces. In some situations a short memory is required to explore more and in other situations a longer memory is needed. An interesting opportunity for further research is to investigate how this value changes at different steps in the training process.

The critical difference plots show insignificant differences between overall average rank across all datasets. This is visual confirmation how the best value for the replay window size is problem dependent.

As with the other experimental groups, Figure 8.31 provide example test loss plots for the car and student performance datasets. These two datasets have been carefully selected to showcase a successful case where no overfitting took place(car dataset), while also showing the results of a case where overfitting is clear (student performance dataset). This highlight the extent of overfitting that can occur, supporting the idea to use early stop as suggested before. Furthermore, the overfit plot is show to illustrate the effects of invalid selections and the effect is has along with noise. Consider the large variance that occurs between runs at the point of overfitting in Figure 8.31 as for the student performance dataset.

In this section it was shown that replay window size configurations are problem-dependent.The replay window size was shown to be related to the reselection window size. The next section provides the empirical results for the the next related hyper-parameter, the reanalysis window size.

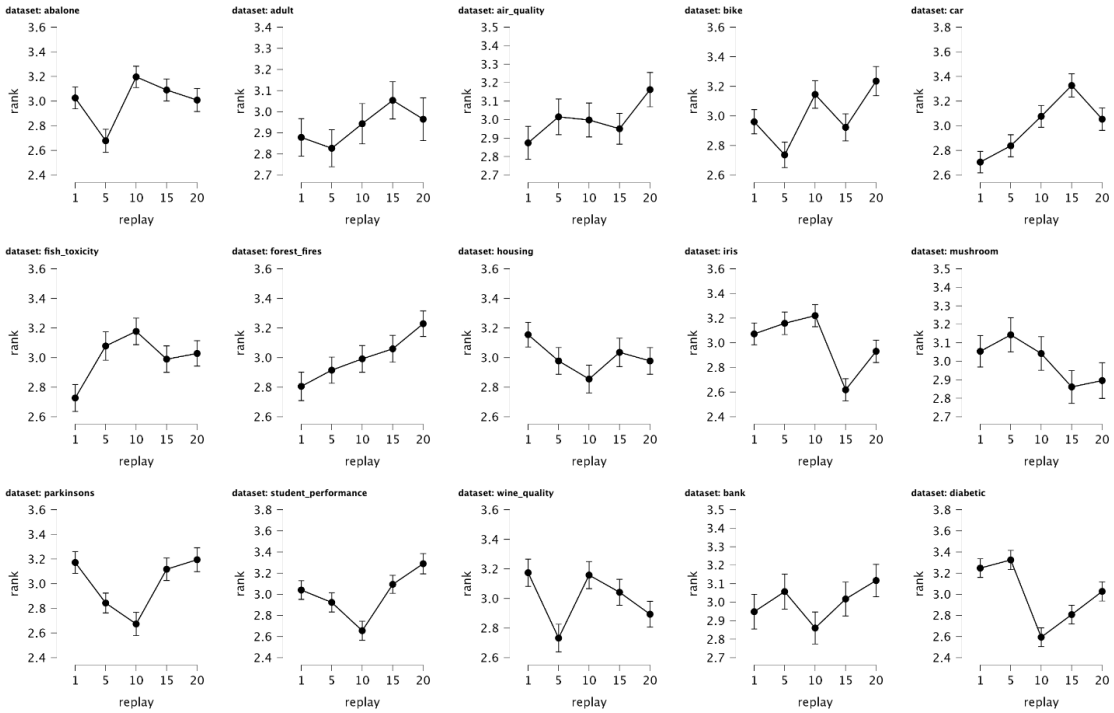


Figure 8.29: Descriptive plots between the average ranks of BHHs with varying replay window sizes per dataset, across all runs and steps.

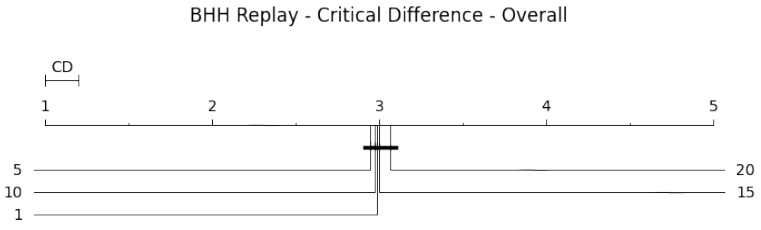
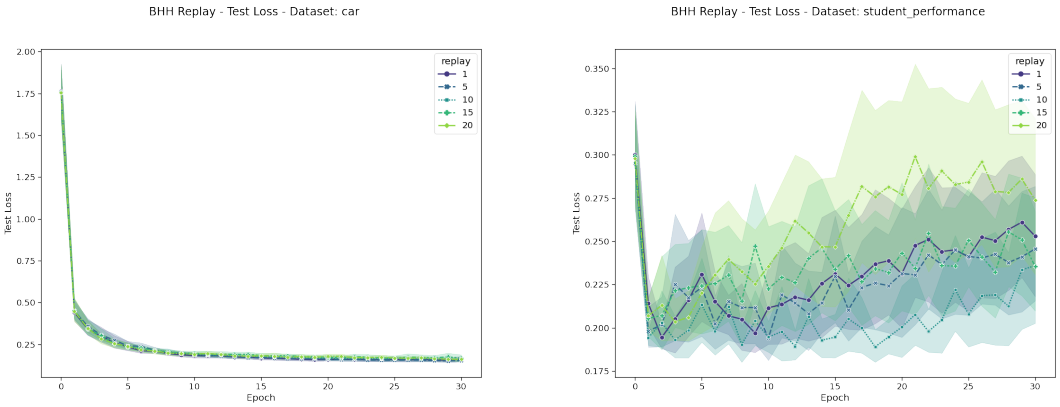


Figure 8.30: Critical Difference plots between the average ranks of BHHs with varying replay window sizes across all datasets, runs and steps.



(a) car (b) student_performance

Figure 8.31: BHH Replay - example test loss plots for BHHs with varying replay window sizes

8.9 Reanalysis

This section provides the empirical results for the experimental group that focuses on the reanalysis window size and the effect it has on the outcomes of the BHH. As a reminder to the reader, the reanalsysis window size is another hyper-parameter which aims to provide a balance between exploration and exploitation by delaying or accelerating the update of beliefs/priors in the BHH. This parameter is related to the reselection window size as well as the replay window size. The configuration of these three parameters determine how often reselection occurs and if reselection occurs, what data should be considered. The reanalysis window size delays or acceleration the addition or removal or information from the performance log, while the replay window size controls how much data is kept in the performance log. The baseline BHH uses a reanalysis value that is equal to the reselection value, which is 10.

Table 8.19 as before provides the empirical results for this experimental group. The results show that in general, the baseline reanalysis window size of 10 yielded the best outcomes overall. However, similar to the outcomes of the replay window size, one can clearly see that the best value for the reanalysis window size is also largely problem dependent.

Tables 8.20 and 8.21 provide the ANOVA and post hoc Tukey statistical test results respectively. From these results one can see the statistical certainty of the results, with the post hoc Tukey test indicating statistical significant differences between individual reanalysis window size configurations.

It can be seen from Figure 8.32 for all datasets that a pivot point exists in the results at reanalysis window size 10. More specifically, it should be stated that a pivot point exists at the configuration points where the reanalsysis window is equal to the reselection window. Since the baseline BHH has reselection and reanalys window sizes of 10, this pivot point

Table 8.19: Empirical results showcasing rank statistics for different reanalysis values used by the BHH across multiple datasets

step	(All)														
	reanalysis	metrics													
	1			5			10			15			20		
dataset	count	avg	std	count	avg	std	count	avg	std	count	avg	std	count	avg	std
abalone	930	2.9849	1.3965	930	3.0699	1.4423	930	3.1710	1.3205	930	2.9441	1.4447	930	2.8301	1.4433
adult	930	1.9860	0.8523	930	1.9860	0.8523	930	1.9860	0.8523	930	2.7914	1.7091	930	3.0151	1.7236
air_quality	930	2.9613	1.3442	930	2.8387	1.4383	930	3.2559	1.4024	930	3.0538	1.4200	930	2.8903	1.4297
bank	930	3.1161	1.4322	930	3.0452	1.3827	930	2.7355	1.3985	930	2.9280	1.4028	930	3.1753	1.4151
bike	930	3.1409	1.2581	930	2.7581	1.3914	930	3.1946	1.3653	930	3.0978	1.4978	930	2.8086	1.4905
car	930	3.1237	1.4229	930	2.7237	1.3575	930	2.8763	1.2987	930	3.1538	1.3846	930	3.1226	1.5473
diabetic	930	3.2452	1.3050	930	3.0720	1.3913	930	2.7785	1.4067	930	3.0290	1.4249	930	2.8753	1.4940
fish_toxicity	930	3.1452	1.3620	930	3.0032	1.4455	930	3.2172	1.3639	930	3.1000	1.4088	930	2.5344	1.3879
forest_fires	930	2.8398	1.3690	930	3.1215	1.4707	930	3.0043	1.3181	930	2.8860	1.3535	930	3.1484	1.5261
housing	930	3.3129	1.3107	930	2.9892	1.3927	930	2.6871	1.4833	930	2.6968	1.4965	930	3.3140	1.2356
iris	930	3.1204	1.3010	930	3.0968	1.3909	930	3.1269	1.4320	930	2.6366	1.4492	930	3.0194	1.4352
mushroom	930	2.9312	1.3794	930	3.1538	1.4031	930	2.8473	1.3836	930	3.1215	1.4530	930	2.9032	1.4714
parkinsons	930	2.9688	1.3850	930	3.0086	1.3559	930	2.6129	1.4814	930	3.1032	1.3804	930	3.3065	1.3778
student_performance	930	2.9409	1.4118	930	3.1849	1.3804	930	2.7710	1.4097	930	3.1258	1.3655	930	2.9774	1.4675
wine_quality	930	2.8032	1.4440	930	2.9204	1.3042	930	3.2022	1.4104	930	2.9247	1.3491	930	3.1495	1.5171
overall	**13950**	**2.9747**	**1.3710**	**13950**	**2.9315**	**1.3960**	**13950**	**2.8978**	**1.3999**	**13950**	**2.9728**	**1.4464**	**13950**	**3.0047**	**1.4805**

Table 8.20: ANOVA - Rank - BHH Variant: Reanalysis

Cases	Sum of Squares	df	Mean Square	F	p
dataset	1814.177	14	129.584	66.508	< .001
reanalysis	97.535	4	24.384	12.515	< .001
dataset * reanalysis	2924.722	56	52.227	26.805	< .001
Residuals	135755.285	69675	1.948		

Table 8.21: Post Hoc Comparisons - BHH Variant: Reanalysis

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
1	5	0.043	−0.002	0.089	0.017	2.586	0.073
	10	0.077	0.031	0.123	0.017	4.602	< .001
	15	0.002	−0.044	0.047	0.017	0.112	1.000
	20	−0.030	−0.076	0.016	0.017	−1.793	0.378
5	10	0.034	−0.012	0.079	0.017	2.016	0.258
	15	−0.041	−0.087	0.004	0.017	−2.475	0.096
	20	−0.073	−0.119	−0.028	0.017	−4.379	< .001
10	15	−0.075	−0.121	−0.029	0.017	−4.491	< .001
	20	−0.107	−0.152	−0.061	0.017	−6.395	< .001
15	20	−0.032	−0.077	0.014	0.017	−1.904	0.315

can be seen at reanalysis window size 10 for all datasets. At this point, the performance either reach a minimum or a maximum for almost all the different configurations. From this, it can be concluded that the reanalysis window size is also problem dependent and that the relationship between the reselection, replay and reanalysis window sizes is clear.

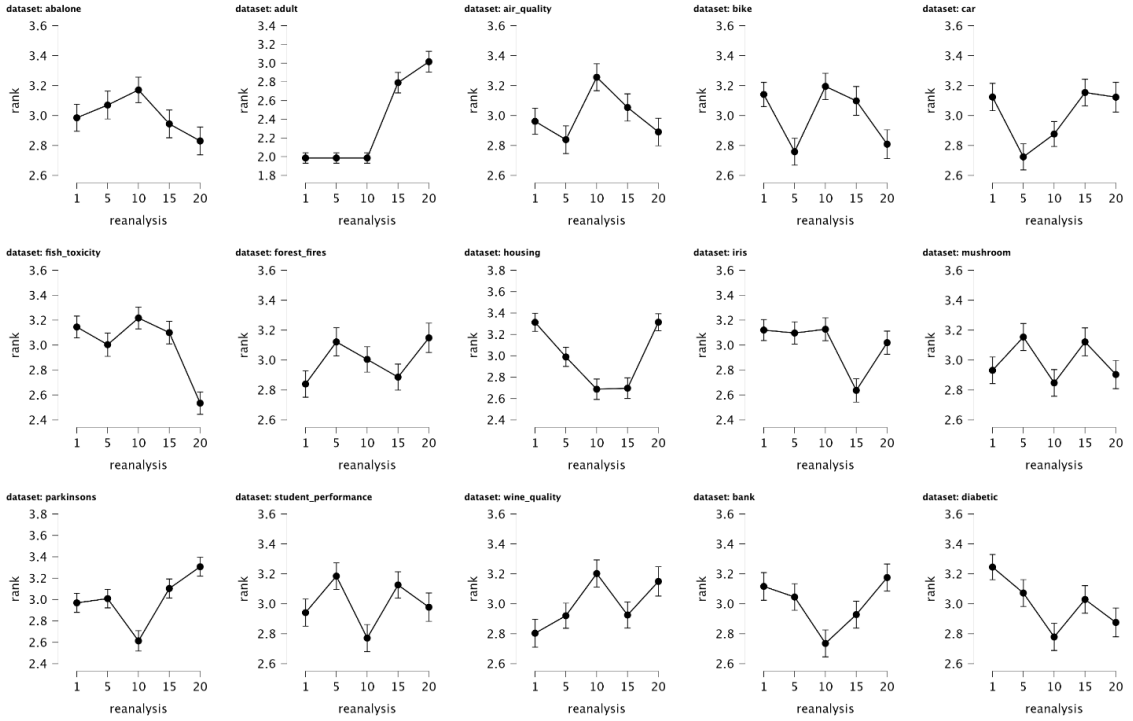


Figure 8.32: Descriptive plots between the average ranks of BHHs with varying reanalysis window sizes per dataset, across all runs and steps.

The critical difference plots beyween the outcomes of different reanalysis window states on the BHH is given in Figure 8.33 from which we can see there is no statistically significant diference between the outcomes overall across all datasets. This further supports the statement that the reanalysis window size is also problem dependent.

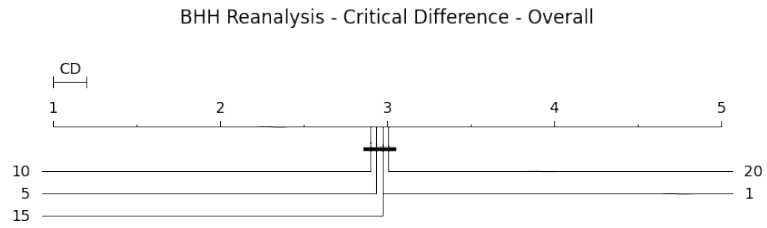
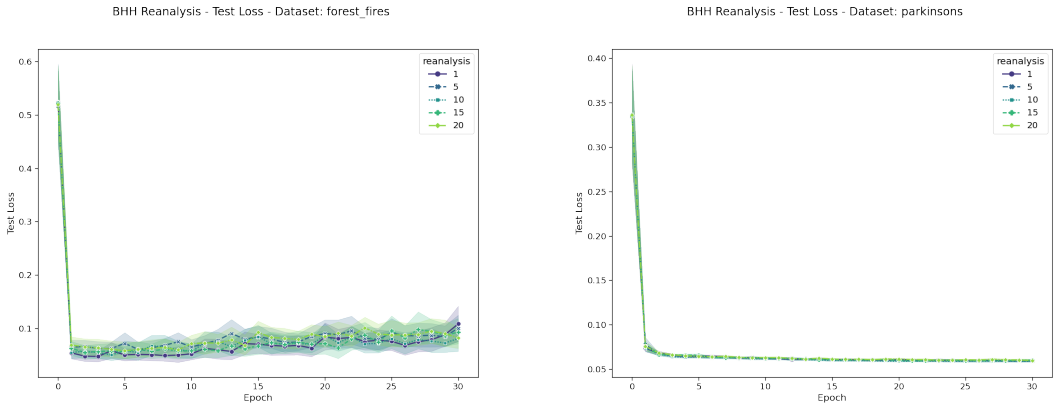


Figure 8.33: Critical Difference plots between the average ranks of BHHs with varying reanalysis window sizes across all datasets, runs and steps.

Finally, Figure 8.34 provides some example test loss plots for the forest fires and parkinsons datasets. These figures showcase how the majority of progress in training and learning is made within the first five epochs.



(a) forest_fires (b) parkinsons

Figure 8.34: BHH Reanalysis - example test loss plots for BHHs with varying reanalysis window sizes

This section has shown that the reanalysis window size is largely problem dependent. A relationship between the reselection, replay and reanalysis window sizes has been identified and it has been concluded that in general a good configuration of reanalysis window size is one where the reanalysis size is equal to the reselection window size, similar to the baseline BHH. Given the relationship between the reselection, replay and reanalysis window sizes, a suggestion can be made to eliminate a hyper-parameter by removing the notion of reanalysis and simply making use of the reselection window size, since these two parameters work best when they are equal. The next section provides the reader with the empirical results for burn in window size.

8.10 Burn In

So far a handful of hyper-parameters have been investigated. It should be noted that a large overarching goal of all these parameters is to get a better understanding of their effects on exploration and exploitation and how that impacts the BHH. So far the results have all shown to yield statistically significant results, with a lot of parameters being shown to be problem-dependent. This section aims to further to the investigation by considering the experimental group that focuses on different burn in window sizes and the effects it has on the BHH. As a reminder to the reader, the burn in window size is a hyper-parameter that defines a delay in the BHH before any learning starts happening. This means that selection, replay and reanalysis window sizes still affect when and how often learning occurs, but burn in controls when learning starts.

As with the other experimental groups, Table 8.22 provide the empirical results. The empirical results indicate a relationship between the burn in window size and the average rank outcome. It can be seen that a lower burn in rate leads to better results. Infact, the

results show on most occasions that no burn in is the best configuration. This outcome makes sense since it has already been established that most of the progress and learning is happening in the initial steps of training. If a burn in value is big, it causes the BHH to miss this opportunity to learn. Take note, even without learning anything before the burn in, the underlying heuristics are still all good and would still lead to decent results. Afterall, this is proof that the proxied state update operations work well

Table 8.22: Empirical results showcasing rank statistics for different burn in values used by the BHH across multiple datasets

step (All)

	burn_in	**results**													
	0			**5**			**10**			**15**			**20**		
dataset	**count**	**avg**	**std**	**count**	**avg**	**std**	**count**	**avg**	**std**	**count**	**avg**	**std**	**count**	**avg**	**std**
abalone	930	2.9151	1.3952	930	2.6935	1.3934	930	2.9172	1.3837	930	2.8398	1.3524	930	3.6344	1.3574
adult	930	2.7366	1.4475	930	2.6430	1.3817	930	2.9892	1.4111	930	3.1731	1.4477	930	3.1355	1.4130
air_quality	930	2.9097	1.4408	930	2.6817	1.4270	930	2.7699	1.3443	930	3.3720	1.3137	930	3.2667	1.4119
bike	930	1.9237	0.9808	930	2.1806	1.0238	930	2.9237	1.2739	930	3.5054	1.1859	930	4.4667	0.8514
car	930	2.3624	1.2315	930	2.5882	1.4246	930	3.2978	1.3922	930	3.3796	1.2711	930	3.3720	1.4009
fish_toxicity	930	3.0634	1.4589	930	2.7935	1.4021	930	2.8204	1.5131	930	3.1022	1.3461	930	3.2204	1.2949
forest_fires	930	2.8161	1.3301	930	2.8495	1.3830	930	2.8645	1.4092	930	3.3774	1.3976	930	3.0925	1.4709
housing	930	2.6903	1.3837	930	2.9516	1.3176	930	2.9968	1.4425	930	3.0581	1.3435	930	3.3032	1.5101
iris	930	2.9473	1.4608	930	2.9720	1.3675	930	3.1903	1.4493	930	2.7710	1.3694	930	3.1194	1.3872
mushroom	930	2.2065	1.2486	930	2.9376	1.2977	930	3.1613	1.2966	930	3.1441	1.4015	930	3.5226	1.5006
parkinsons	930	2.2796	1.2822	930	2.8892	1.2474	930	2.7065	1.3143	930	3.3968	1.4175	930	3.7280	1.3295
student_performance	930	2.1452	1.2052	930	2.6946	1.4032	930	3.0968	1.3525	930	3.4581	1.3539	930	3.6054	1.2312
wine_quality	930	2.9699	1.4373	930	3.0108	1.3041	930	2.6634	1.4186	930	3.1247	1.3980	930	3.2312	1.4471
bank	930	2.7301	1.3210	930	2.8194	1.3350	930	2.8570	1.4848	930	2.9312	1.3841	930	3.6624	1.3391
overall	**13020**	**2.6211**	**1.3814**	**13020**	**2.7647**	**1.3553**	**13020**	**2.9468**	**1.4045**	**13020**	**3.1881**	**1.3754**	**13020**	**3.4543**	**1.4061**

As before, Tables 8.23 and 8.24 provide the ANOVA and post hoc Tukey test results respectively. These tables show the statistical significance of the results.

Table 8.23: ANOVA - Rank - BHH Variant: Burn In

Cases	Sum of Squares	df	Mean Square	F	p
dataset	17.977	14	1.284	0.696	0.781
burn_in	6332.995	4	1583.249	858.000	< .001
dataset * burn_in	4955.754	56	88.496	47.958	< .001
Residuals	128569.751	69675	1.845		

The best illustration of the relationship between the burn in value and the average rank is best shown in Figure 8.35 which provides a descriptive plot showing the difference in outcome by burn in window size and dataset. From these plots one can see that there is some problem dependence, but in general, there is a pattern that snows that lower burn in window sizes are best. There are cases where absolutely no burn in is also not the best, but rather a small, non-zero burn in window size is preferable. See the descriptive plots for fish toxicity, abalone, adult and air quality datasets in Figure 8.35.

Table 8.24: Post Hoc Comparisons - BHH Variant: Burn In

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
0	5	−0.154	−0.198	−0.109	0.016	−9.449	< .001
	10	−0.360	−0.404	−0.315	0.016	−22.107	< .001
	15	−0.584	−0.629	−0.540	0.016	−35.923	< .001
	20	−0.845	−0.890	−0.801	0.016	−51.975	< .001
5	10	−0.206	−0.250	−0.162	0.016	−12.658	< .001
	15	−0.431	−0.475	−0.386	0.016	−26.474	< .001
	20	−0.692	−0.736	−0.647	0.016	−42.525	< .001
10	15	−0.225	−0.269	−0.180	0.016	−13.817	< .001
	20	−0.486	−0.530	−0.441	0.016	−29.868	< .001
15	20	−0.261	−0.305	−0.217	0.016	−16.051	< .001

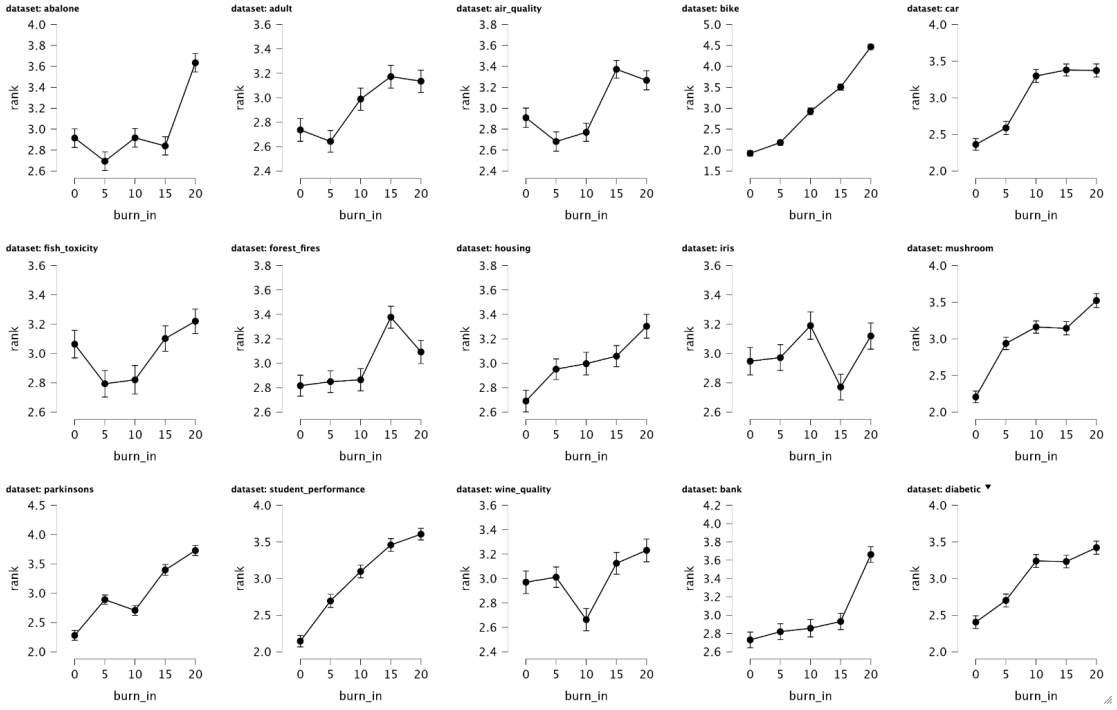


Figure 8.35: Descriptive plots between the average ranks of BHHs with varying burn in values per dataset, across all runs and steps.

The critical difference plot as presented in Figure 8.36 shows the statistical significance between burn in window sizes overall, across all datasets. One can clearly see from this plot that little to no burn in is preferred in most of the cases.

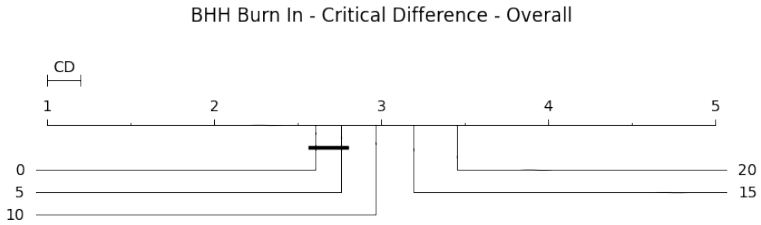
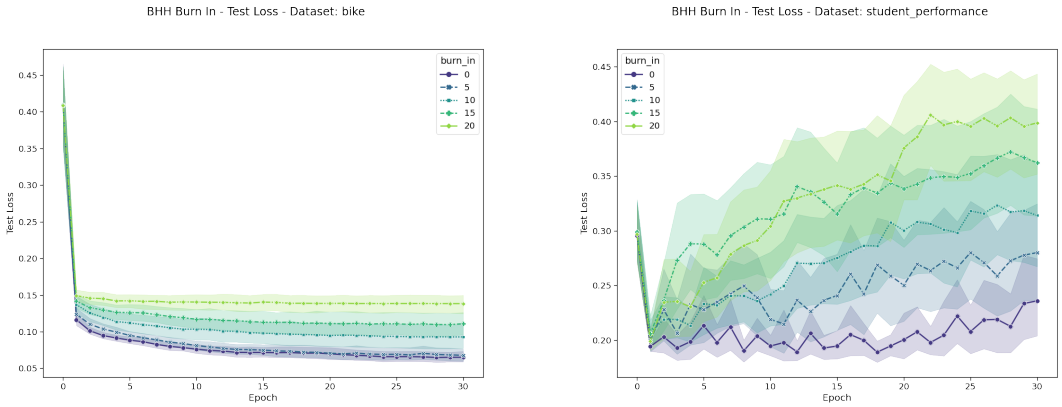


Figure 8.36: Critical Difference plots between the average ranks of BHHs with varying burn in values across all datasets, runs and steps.

Figure 8.37 provides the test loss plots for different burn in window sizes for the bike and student performance datasets. Once again, an example is given of no overfitting (bike dataset) and another of drastic overfitting (student performance dataset). It should be clear from Figure 8.37b how a larger burn in window size has a greater effect on the overfitting. This makes sense, since the BHH only starts to try to learn when most of the progress in FFNN training has already been achieved at that point. At this point, almost no heuristic, no matter how good, is yielding any positive outcomes, resulting in the BHH learning to make bad selections. After this point it has been shown to be difficult for the BHH to recover.



(a) bike (b) student_performance

Figure 8.37: BHH Burn In - example test loss plots for varying burn in values

This section provided the empirical results for the experimental group that focuses on burn in window sizes. It has been show that, in general, a lower burn in window size is preferred. The next sections provide investigation into the normalisation and discounted rewards hyper-parameter flags.

8.11 Normalise

This section presents the empirical results for the experimental group that focuses on normalisation of pseudo counts and the effects it has on the outcomes of the BHH. As a reminder to the reader this normalisation effect aims to promote exploration since it rescales the metric by which priors are updated. It could happen that the BHH gets stuck in some local optima and simply picks the same heursistic over and over again, resulting in other heuristics not being selected at all. This is referred to as *mode-collapse* and is discussed in Chapter 7. Normalisation is an attempt to then reset beliefs proportionally back to the initial symmetric starting concentration of 1.

Table 8.25 present the empirical results for this experimental group. From the table one can see that in general, no normalisation yields the best results, however, it should be pointed out that this parameter is problem-dependent, yielding better results for the bank, fish toxicity, forest fires, iris and wine quality datasets when normalisation is enabled.

Table 8.25: Empirical results showcasing rank statistics for normalisation toggled by the BHH across multiple datasets

step	**(All)**					
	normalisation	**metric**				
	FALSE			**TRUE**		
dataset	**count**	**avg**	**std**	**count**	**avg**	**std**
abalone	930	1.4935	0.5002	930	1.5065	0.5002
adult	930	1.4710	0.4994	930	1.4968	0.5003
air_quality	930	1.4892	0.5002	930	1.5108	0.5002
bank	930	1.5484	0.4979	930	1.4516	0.4979
bike	930	1.4978	0.5003	930	1.5022	0.5003
car	930	1.4258	0.4947	930	1.5742	0.4947
diabetic	930	1.4753	0.4997	930	1.5247	0.4997
fish_toxicity	930	1.5505	0.4977	930	1.4495	0.4977
forest_fires	930	1.5054	0.5002	930	1.4946	0.5002
housing	930	1.4720	0.4995	930	1.5280	0.4995
iris	930	1.5645	0.4961	930	1.4355	0.4961
mushroom	930	1.4323	0.4957	930	1.5667	0.4958
parkinsons	930	1.4000	0.4902	930	1.6000	0.4902
student_performance	930	1.3892	0.4878	930	1.6108	0.4878
wine_quality	930	1.5473	0.4980	930	1.4527	0.4980
overall	**13950**	**1.4842**	**0.4998**	**13950**	**1.5136**	**0.4998**

Similar to the other experimental groups, Tables 8.26 and 8.27 provide the ANOVA and post hoc Tukey statistical test results. These tables show that the results are indeed statistical significance.

Table 8.26: ANOVA - Rank - BHH Variant: Normalise

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.450	14	0.032	0.130	1.000
normalisation	6.055	1	6.055	24.490	< .001
dataset * normalisation	78.266	14	5.590	22.613	< .001
Residuals	6890.195	27870	0.247		

Table 8.27: Post Hoc Comparisons - BHH Variant: Normalise

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
False	True	−0.029	−0.041	−0.018	0.006	−4.949	< .001

From the descriptive plots provided in Figure 8.38, one can see how normalisation is problem dependent and in many cases, normalisation provides not statistically significant difference. This is further evident in Figure 8.39 which shows no statistically significant difference overall across all datasets.

As with the other experimental groups, Figure 8.40 provides example test loss plots with normalisation enabled and disabled for the bank and housing datasets. For these particular plots, there is no real clear distinction between the outcomes of normalisation. Since the effect of this parameter is neglibly small, the authors do not deem this parameter as important as the others although it has been shown to provide some statistically significant differences for some datasets.

This section provided the empirical results for the experimental group that focuses on the effects of normalisation of pseudo counts on the BHH. Although it has been shown to yield interesting results with normalisation not being required in general, while normalisation is preferred for other datasets, the authors conclude that the importance of this parameter is of such nature that it does not have such a significant impact on the BHH as with the other parameters.

The next section considers the empirical results for an experimental group that focus on an alternative strategy to the normalisation approach by providing a discounted rewards flag on the pseudo counts.

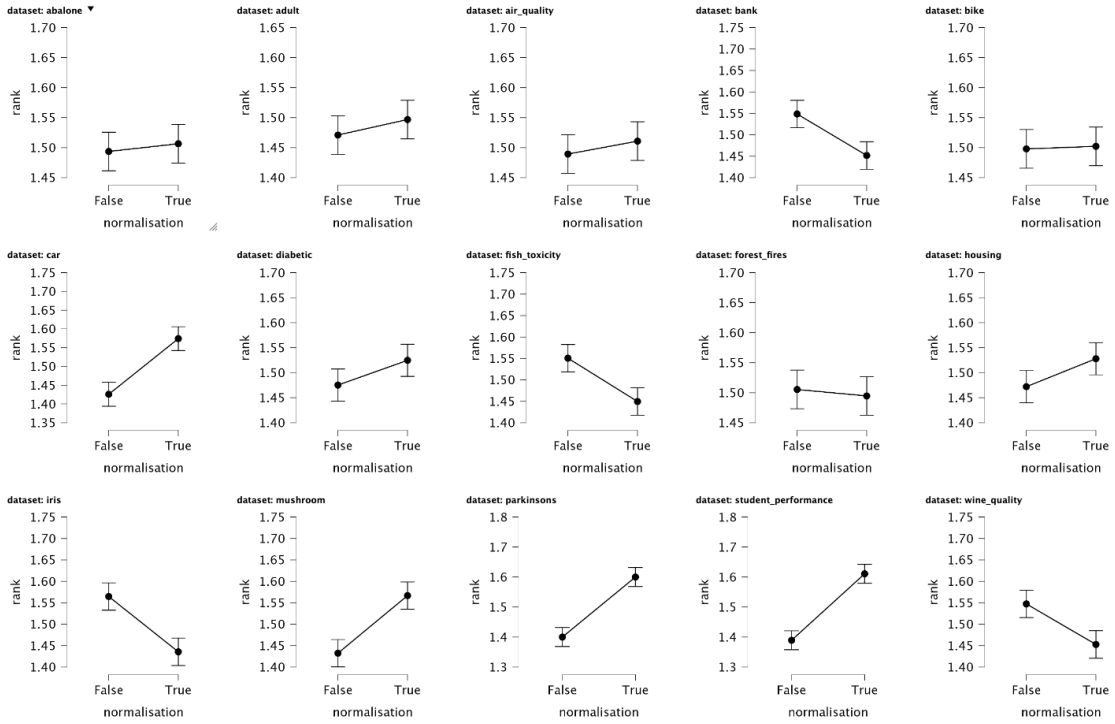


Figure 8.38: Descriptive plots between the average ranks of BHHs with normalisation toggled per dataset, across all runs and steps.

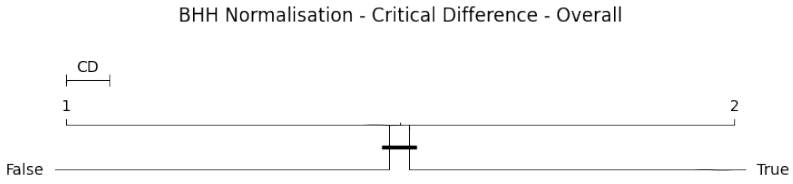
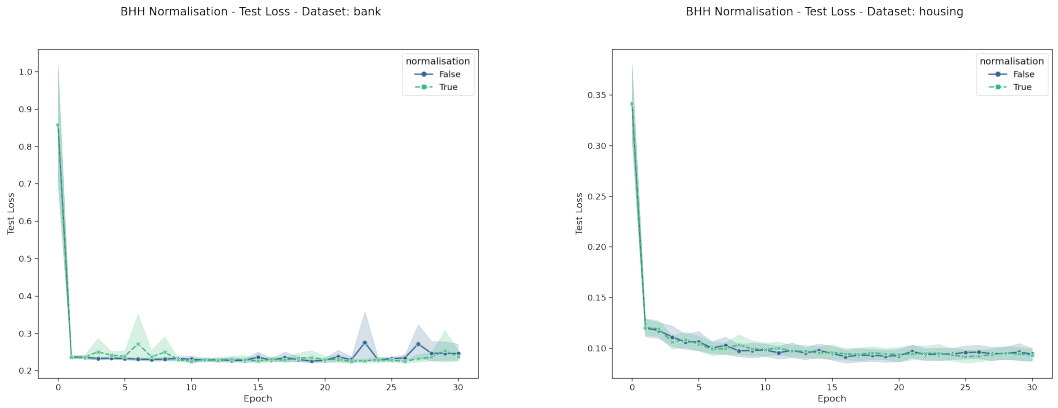


Figure 8.39: Critical Difference plots between the average ranks of BHHs with normalisation toggled across all datasets, runs and steps.



(a) bank (b) housing

Figure 8.40: BHH Normalise - example test loss plots for BHH with normalisation toggled

8.12 Discounted Rewards

This section presents the empirical results for an experimental group that focuses on the effects of discounted rewards on the BHH. As a reminder to the reader, discounted rewards is an strategy that attempts to exponentially decrease the impact of evidence in the performance log for evidence that is further back in history. This concept is borrowed from RL (ref???). The intent of the discounted rewards approach is to determine if a longer memory can be used where importance is placed on more recent evidence. Similar to other parameters discussed so far, this parameter also attempts to provide a mechanism to balance exploration and exploitation.

Table 8.28 provides the empirical results for this experimental group. One can see from the results that the outcomes bear resemblance to the outcomes of the normalisation experimental group. It is found that discounted rewards are generally not required while there are some cases that do benefit from it. This means that the effect of discounted rewards is also problem-dependent.

Tables 8.29 and 8.30 provide the ANOVA and post hoc Tukey statistical test results from which it can be seen that the results are statistically significant.

The descriptive plots as presented in Figure 8.41 shows how some datasets benefit from discounted rewards while others do not. This makes sense, since some datasets might have different training requirements that require long memory of state, the error landscapes could vastly differ or simply contain more noise due to mini-batch training.

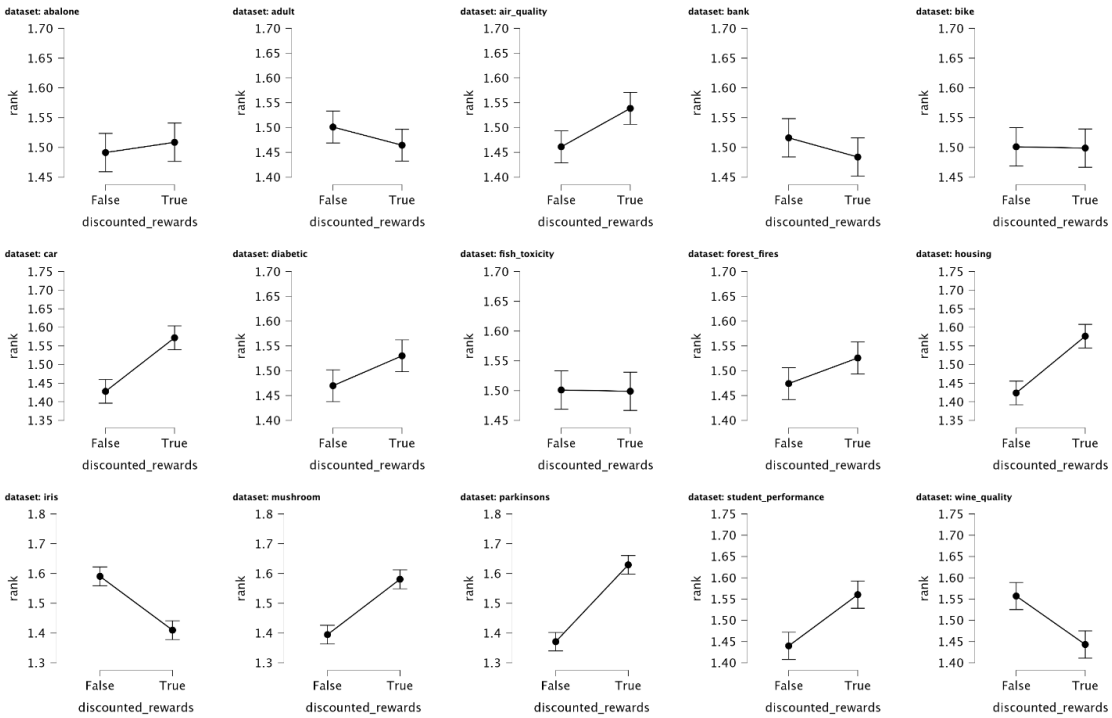


Figure 8.41: Descriptive plots between the average ranks of BHHs with discounted rewards toggled per datasets, across all runs and steps.

Table 8.28: Empirical results showcasing rank statistics for discounted rewards toggled by the BHH across multiple datasets

step (All)

	dr	metrics				
	FALSE			TRUE		
dataset	count	avg	std	count	avg	std
abalone	930	1.4914	0.5002	930	1.5086	0.5002
adult	930	1.5011	0.5003	930	1.4645	0.4990
air_quality	930	1.4613	0.4988	930	1.5387	0.4988
bank	930	1.5161	0.5000	930	1.4839	0.5000
bike	930	1.5011	0.5003	930	1.4989	0.5003
car	930	1.4280	0.4950	930	1.5720	0.4950
diabetic	930	1.4699	0.4994	930	1.5301	0.4994
fish_toxicity	930	1.5011	0.5003	930	1.4989	0.5003
forest_fires	930	1.4742	0.4996	930	1.5258	0.4996
housing	930	1.4237	0.4944	930	1.5763	0.4944
iris	930	1.5903	0.4920	930	1.4097	0.4920
mushroom	930	1.3946	0.4890	930	1.5806	0.4937
parkinsons	930	1.3710	0.4833	930	1.6290	0.4833
student_performance	930	1.4398	0.4966	930	1.5602	0.4966
wine_quality	930	1.5570	0.4970	930	1.4430	0.4970
overall	**13950**	**1.4747**	**0.4994**	**13950**	**1.5214**	**0.4996**

Table 8.29: ANOVA - Rank - BHH Variant: Discounted Rewards

Cases	Sum of Squares	df	Mean Square	F	p
dataset	0.727	14	0.052	0.210	0.999
discounted_rewards	15.190	1	15.190	61.607	< .001
dataset * discounted_rewards	87.282	14	6.234	25.285	< .001
Residuals	6871.694	27870	0.247		

Table 8.30: Post Hoc Comparisons - BHH Variant: Discounted Rewards

			95% CI for Mean Difference				
		Mean Difference	Lower	Upper	SE	t	p_{tukey}
False	True	−0.047	−0.058	−0.035	0.006	−7.849	$< .001$

The critical difference plot as presented in Figure 8.42 shows that there is not statistically significant difference between enabling and disabling discounted rewards overall, across all datasets, once again suggesting problem-dependence.

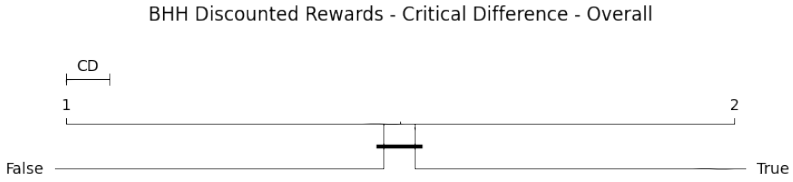
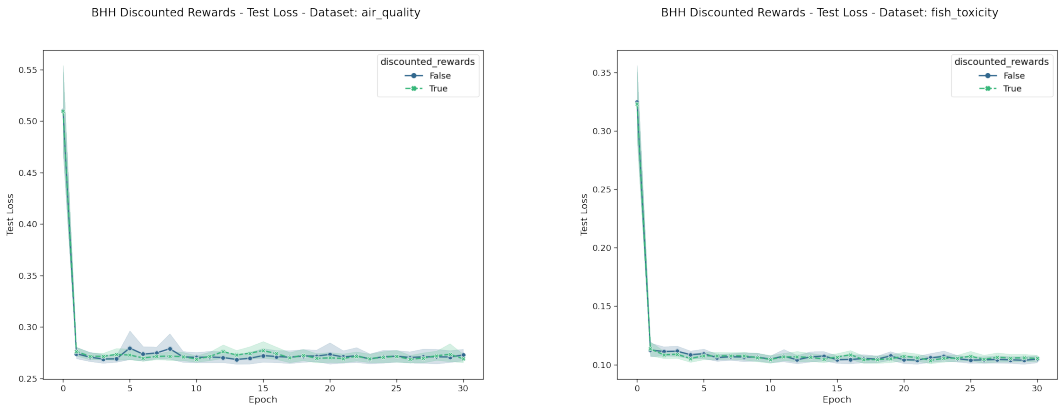


Figure 8.42: Critical Difference plots between the average ranks of BHHs with discounted rewards toggled across all datasets, runs and steps.

Figure 8.43 provides example plots of test loss when discounted rewards is enabled and disabled for the air quality and fish toxity datasets.



(a) air_quality (b) fish_toxicity

Figure 8.43: BHH Discounted Rewards - example test loss plots for discounted rewards toggled on and off

This section presented the finding and results for the experimental group that focuses on discounted rewards of evidence in the performance log. It has been shown that the outcome of this parameter on the BHH is problem dependent. However, the authors deem this parameter to be less important than the other hyper-parameters since their influence is much bigger on the outcome of the BHH than that of discounted rewards.

This concludes the formal experimental groups and empirical aspects to be evaluated. The next section aims to provide more insight into overfitting that occurs with the BHH.

8.13 Overfitting

Throughout this chapter the topic of overfitting was raised on multiple occasions. The authors deem it necessary to dedicate a section to specifically discuss and summarise the effects of overfitting on the BHH. Take note of the fact that overfitting could be the results of many aspects. Overfitting is often the result of noise caused by mini-batch iterations. Overfitting could be the result of training and test sets that do not have the same approximate distributions. Investigation into the data is required. Consider again the experimental group that focused on heuristic pool configurations. Three different configurations were considered of which the gradient-only heuristics pool performed best. It should be observed that overfitting occurs much less frequently when the heuristics pool contains only the gradient-based low-level heuristics. Figure 8.44 below contains the only case where overfitting occurs when the gradient-only heuristic pool is used.

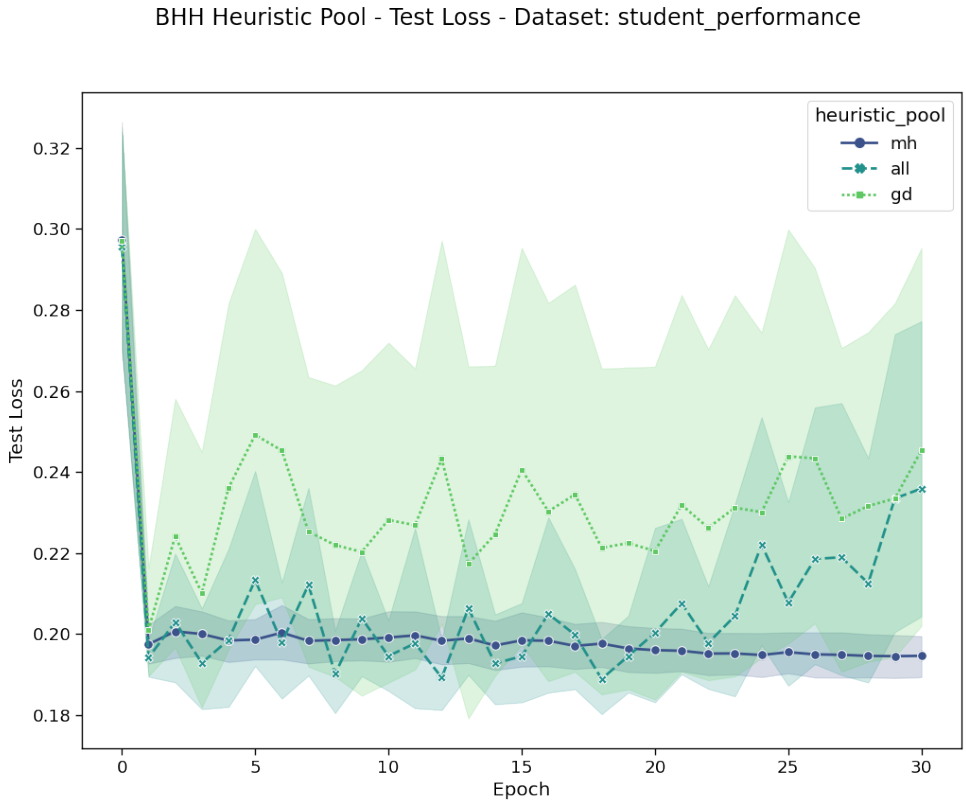


Figure 8.44: Example test loss plot where the BHH-gd is shown to overfit

One can then conclude that most of the overfitting occurs as a result of the inclusion of the meta-heuristics. To some extent this does make sense, since the proxied update

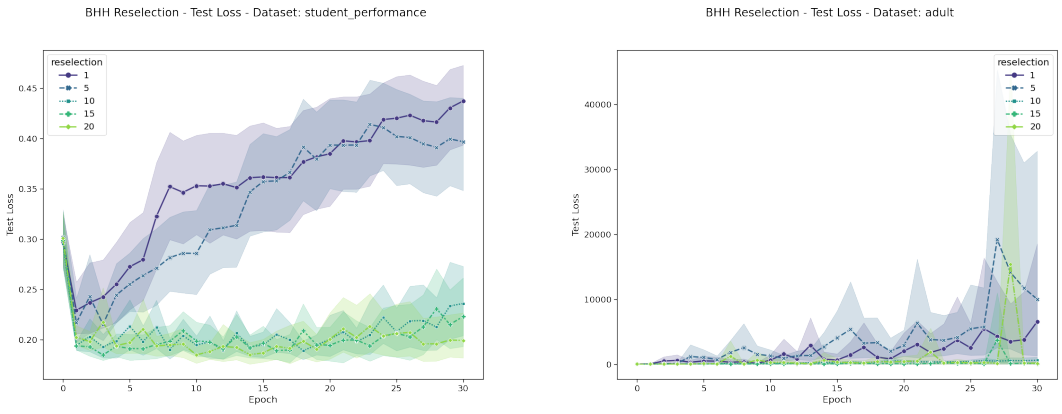
step operations for these heuristics do not map as well between meta-heuristics as they do with gradient-based heuristics and thus an update step that occurs when switching from a PSO to a GA for example might maintain a state of momentum in a certain direction and then as a result of cross-over end up in a totally opposite direction, resulting in the entity vastly overshooting any optima. Van Wyk [188] did a detailed analysis of overfitting that occurs when training FFNNs using PSOs. Although the same velocity clamping techniques were used as suggested in Van Wyk's work, they are not guaranteed to avoid overfitting. Another technique would be to nullify and reset velocites and momentum, but this could then be detrimental to the other heuristics. Since the BHH includes a PSO in the heuristic pool, one could expect similar results.

Another aspect to remember is that when overfitting starts, the BHH has no notion of negative progress, since it merely maps the best of the evidence it observes. A strategy of picking the "lesser of two evils". It will then continue to bias toward the best of the heuristics even though the underlying entities have vastly overshot good solutions.

Credit should however be given to the BHH for its ability to learn positively in such a short timespan. It has been indicated that the update step for the BHH happens at each mini-batch iteration. If a large dataset is presented, that means that the majority of training takes place in a small number of epochs. This leaves a very small window for the BHH to actually learn relevant and useful information from the heuristic-space.

Once overfitting has occurred, it is difficult to undo. Therefore the best recommendation is to just use early stopping criteria. A suggestion is to simply stop training after k-iterations have not yielded better test set generalisation results.

As a last example, Figure 8.45 shows two examples where overfitting is very obviously noticable.



(a) student_performance (b) adult

Figure 8.45: BHH overfitting behaviour for varying selection window sizes and datasets

The natural steps that follow on this empirical process is to apply early stopping conditions and plot for every mini-batch iteration. This is left for future research. The

following section briefly talks about the practical implications for the BHH.

8.14 Computational Requirements

This section shines some light into the practical applications of the BHH. All population-based approach are generally more computationally expensive than individual candidate solution approaches. This is because of the number of function evaluations they have to do. This is even worse for HHs since some HHs require multiple runs, re-evaluations and some have very large memory footprints. This notion is further emphasised when training FFNNs, since training a single NN is already computationally expensive, let alone a whole population.

Specifically, the BHH admittedly has quite a high computational overhead and a relatively large memory footprint. Consider the following aspects that attribute to this fact:

- The BHH contains a population of entities.
- Each entity maintains a set of state variable that represent all the underlying states of various low-level heuristics in its heuristic pool.
- The BHH sometimes have to re-evaluate heuristics. Some heuristics evaluate after the update step while others first step and then evaluate.
- The BHH maintains a global population state.
- The BHH maintains a performance log of variable length that needs to be maintained.
- The BHH has to apply credit assignment strategies multiple times during a single step to ensure that the correct performance state is maintained.
- Thrashing between CPU and GPU execution, such as training the FFNN and then updating the performance log, drastically slows down the training process and calls for rethinking of implementation.

There are many suggestions that could be made as to how the BHH can be improved upon, but those concepts are left for future research. It should just be noted since there is a trade-off involved here. In Chapter 1 it is mentioned that the intent of a HH is to eliminate the need for a tedious process of trial and error to find a heuristic that can train a FFNN sufficiently. The gain in computational time should still be less than the time it would need to follow the trial and error approach for the use of the BHH to be justified.

The next section captures a summary of all of the results.

8.15 Summary of Results

This section aims to provide a brief summary of the results before conclusions are made in the next chapter. This chapter was split into multiple experimental groups and their findings are given as:

- Section 8.2 presented a case study that was done on the behavour of the BHH during training and testing. It was shown to be able to train FFNNs well. Detailed analysis was done and it was shown that the BHH does indeed learn and that the general learning is positive towards the overall goals. Both the selection mechanism as well as the perturbative component was shown to work well.
- Section 8.3 presented an empirical test that was executed to compare the BHH's performance to that of standalone low-level heuristics. It was found that the BHH performed reasonably well and is able to generalise well on the test set especially considering overfitting conditions that can be avoided in future work. Although the BHH was able to generalise well to the test set, it was not able to perform better than the absolute best low-level heuristic on all occasions. It was mentioned here that the BHH has the advantage to bias prior knowledge towards well known heuristics to aid in the learning process.
- Section 8.4 presented an empirical test to compare different heuristic pool configurations. It was found that the gradient-only heuristic pool performed best.
- Section 8.5 presented an empirical test to evaluate the effect of population size on the BHH. It was found that population size is shown to be problem-dependent.
- Section 8.6 presented an empirical test to evaluate credit assignment strategies and it was found that the choice of credit assignment strategy to use is problem-dependent.
- Section 8.7 presented an empirical test to evaluate different reselection window sizes. It was found that the the BHH prefers larger reselection window sizes to give it more time to gather evidence.
- Section 8.8 presented an empirical test to evaluate different replay window sizes and it was found that the choice of replay window size is problem dependent.
- Section 8.9 presented an empirical test to evaluate different reanalysis window sizes and it was also found to be problem dependent.
- Section 8.10 presented an empirical test to evaluate different burn in window sizes. In general it was found that limited to no burn in was preferred, giving the BHH the opportunity to start learning as soon as possible.

- Sections 8.11 investigated the impacts of normalising pseudo counts for the concentration parameter updates before they are updated. It was found that the choice to enable normalisation or not is problem dependent but generally showed not to be necessary. For that reason the authors deem the impact of this parameter to be negligibly small. Small enough to consider for removal in future work.
- Sections 8.12 investigated the impacts of discounted rewards in the performance logs for events further back in history. Similar outcomes to that of the normalisation flag parameter was found. In future work, this parameter can be removed.

This chapter presented various angles of analysis to ensure that the BHH is properly analysed and studied. As far as was required, all empirical test were analysed for statistical significance and the results of these tests have been provided. Various example plots and figures have been provided as visual aid to try better illustrate the outcomes of the empirical process. Deeper insight into all of the parameters have been given as far as possible. In general, the outcome of the entire empirical process is deemed successful and the BHH shows potential for future research work. The following chapter provides a conclusion of the entire dissertation and the work that is presented here.

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

Finally, the research objectives that have been set out in this dissertation must be concluded. This chapter provides a summary of the research that has been done.

The remainder of the chapter is structured as follows:

- **Section 9.1** 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 recap of the BHH and its implementation.
- **Section 9.4** provides a brief recap of the methodology and the empirical process that was followed.
- **Section 9.5** summarises the research findings.
- **Section 9.6** discuss further research opportunities.
- **Section 9.7** briefly discusses some supplementary and supporting material related to the research, as made available by the authors either as external sources or in the appendices of this dissertation.
- Finally, **Section 9.8** provides the closing statements of the dissertation.

9.1 Reviewing Research Objectives

This section briefly reviews the problem statement, motivations and research objectives that was set out in Chapter 1 of this dissertation.

The main area of focus for this research stems from the problem statement which identified the difficult and tedious problem of heuristic selection for FFNN training. Often

times this process is non-trivial and very time-consuming. It has been identified that 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 careful consideration and selection, often at the expense of the researcher's resources. This research has identified the possibility of using a relatively new approach, referred to as HHs to automate this selection process. HH has been shown to provide a general solution to this problem. The consequences of such approach being successful could provide a mechanism that saves a lot of time and could provide a more generalised solution to heuristic selection.

Investigation was done into the landscapes of existing solutions to train FFNNs as well as HHs. Though there exists many applications of HHs there are little to no published work on using HHs to train FFNNs. The only work that was found include the work by Nel [131].

Other existing techniques from other problem domains were investigated. RL, meta-learning, online, one-shot and probabilistic learning approached were considered. Bayesian probability theory showed promise and the BHH was conceptualised.

The research objectives were defined and are summarised as follows:

- Conduct literature studies on all related topics that are relevant to the development of the BHH. These include ANN, heuristics/optimisers, meta-learning, HHs, probability theory and Bayesian statistics.
- Develop a novel BHH that has both a selective and perturbative element. For the selection mechanism, Bayesian statistics and MAP is used. For the perturbative element, proxied low-level heuristic update steps are used.
- Conduct an empirical study to show that the BHH is able to train FFNNs effectively on a number of different datasets.
- Conduct an empirical study to study the behavioural characteristics of the BHH.
- Conduct an empirical study to critically evaluate the performance of the BHH to those of traditional, low-level heuristic when training FFNNs on a number of different datasets.
- Conduct an empirical study to evaluate variants of the BHH to study the impacts of various design decisions on the outcomes of the BHH.
- Provide a statistically sound evaluation of outcomes with statistical significance tests driving the conclusions of the outcomes.

As one can see, the research set out to do in this dissertation was quite extensive and also identifies many different opportunities for future research.

The next sections aim to provide brief summaries of the work that was done, followed by a conclusion of the outcomes of each.

9.2 Summary of Background Information

Chapters 1 - 5 provided the reader with background information that is necessary to understand the components that are discussed and developed in this work. These chapters respectively provided background information, literature reviews and landscape analyses into ANNs, heuristics/optimisers, HHs and meta-learning, probability theory and Bayesian statistics.

It should be mentioned that there are many resources available on the above mentioned topics in their own right. However, the intersection of these topics, especially ANNs and HHs are close to non-existent and calls for much more research and publication. This dissertation aims to provide a manifesto of these topics and how they are bring together to show that HHs can be used to train FFNNs.

From these topics, it can be concluded that the necessary background information that was provisioned for in the objectives for the dissertation have been provided. The next section gives a summary of the novel BHH that was developed.

9.3 Summary of Bayesian Hyper-Heuristics

The main takeaway point from this dissertation is the development of a novel BHH. Chapter 6 provides the details and implementation of the BHH and provides the reader with various design decisions that could control/affect the outcomes of the BHH.

It was shown how the BHH implements a Bayesian probabilistic model and makes use of online learning to select heuristics dynamically throughout the training process. This probabilistic model forms the selection mechanism for the BHH.

Furthermore, it is shown that selection alone is not enough and that a measure of transition between heuristics is needed. Entity and heuristic state and state-manipulation formed the cornerstone of this topic. A technique that deconstructs heuristics into their initialisation and update steps is proposed. This technique then allows for the proxying of update step operations as they are required by different heuristics. Though this solves the transition problem between heuristics, is has been indicated that does not guarantee stable search processes. This does raise opportunities for further research which is summarised below in Section 9.6.

The simplicity of the BHH and various challenges related to it have been highlighted. Of these challenge include:

- Catering for overfitting by early stopping conditions.
- Catering for invalid selections.
- Catering for invalid or stale entity and heuristic states.

- Catering for mode-collapse where a heuristic is either always selected or never selected.
- Catering for a trade-off between exploration and exploitation.

After the development of the BHH, the authors presented the methodology used to execute the empirical process. The details of which is summarised in the next section.

9.4 Summary of Methodology

As discussed above, 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 is listed below:

- A case-study on the behaviour of the BHH during training and testing to understand the workings of the BHH and to see if the BHH is able to train FFNNs.
- A comparative analysis between the performance of standalone heuristics and the baseline BHH when training FFNNs.
- A comparative analysis between variants of the BHH that include:
  - Different heuristic pool configurations.
  - Different population sizes.
  - Different credit assignment strategies.
  - Different reselection window sizes.
  - Different replay window sizes.
  - Different reanalysis window sizes.
  - Different burn in window sizes.
  - Enabling and disabling normalisation of pseudo counts in the performance log.
  - Enabling and disabling discounted rewards of pseudo counts in the performance log.

Each experimental group is run against a set of 15 datasets, consisting of regression and classification problems, split between uni- and multimodal problems. Each experimental group is repeated over 30 runs for statistical certainty. The statistical analysis process was discussed and ANOVA, post hoc Tukey, independent T-test and Kruskal-Wallis formed part of the statistical test repertoir.

In order to allow for the studying of the BHH even after training has saturated and overfitting has occured, all experimental groups are run for 30 epochs, way more than

what has been empirically found to be necessary. This also meant that no early stopping conditions were used.

The next section provides a summary of all the results, findings and conclusions related to the empirical process is given next.

9.5 Summary of Results

All empirical tests where successfully executed and statistically evaluate as described in the methodology presented in Chapter 7. From these evaluations a number of conclusions are made, each described briefly in the sub-sections below:

9.5.1 BHH behavioural case study

For this experimental group it was found that the BHH is able to train FFNNs relatively well. It was shown how the BHH is able to learn and that the learning is in a positive direction towards the overall objectives of the BHH. Furthermore, it was shown in isolation that both the selection mechanism as well as the perturbative component of the BHH was successful in their own rights, resulting in a HH that is able to select and combine heuristics as needed.

It was shown how the BHH drastically overfits at some point during training for a number of datasets. Although it is known that early stopping conditions would prohibit this from happening, other solutions have also been proposed. These include:

- Heuristic selection acceptance criteria: where a selection is only deemed valid if it does indeed improve on performance.
- State reset: to avoid overshooting solutions due to momentum.
- Various techniques to balance exploration and exploitation.

Various example plots on training and test datasets have been provided and detailed plots of the inner-workings of the BHH have been presented to support the findings and arguments.

9.5.2 Standalone vs. BHH Baseline

For this experimental group the baseline BHH was found to perform relatively well in comparison to the standalone low-level heuristics. Overall the BHH ranked 5th out of 11 with performances close to that of the best heuristics. Unfortunately, the BHH was not entirely able to beat the best heuristics on all occasions, but did perform comparatively well in all cases. The empirical results, statistical tests, descriptive plots and plots of example runs are given to support findings and conclusions.

9.5.3 BHH Variant: Heuristic pool

For this experimental group, different configurations of heuristic pools were considered. Three different configurations were evaluated. It was found that the gradient-only BHH variant performed significantly better than the configurations that included all of the heuristics and one that included only the meta-heuristics. It was shown that the inclusion of meta-heuristics cause the BHH to overfit much more drastically due to the nature of their update step operations.

9.5.4 BHH Variant: Population Size

For this experimental group, various different population sizes are evaluated. It was found that the population size is largely problem-dependent. An interesting observation was made where some cases where shown where the relationship between population size and performance is not linear, but rather parabolic or even polynomial in some cases. This warrants for further investigation and could serve as a good opportunity for future research.

9.5.5 BHH Variant: Credit

For this experimental group, various different credit assignment strategies were evaluated. It was found that the choice of credit assignment strategy to use is largely problem-dependent. Two cases were highlighted where symmetric credit assignment was preferred, which suggests that for that dataset heuristic selection doesn't matter, only the proxied update step operations are sufficient.

9.5.6 BHH Variant: Reselection

For this experimental group different reselection window sizes were evaluated. It was found that a larger reselection size is generally preferred. A bigger reselection window sizes lead to a longer timeframe to learn before reselection has to occur. The increase in number of samples of evidence, per entity-heuristic combination in the performance log is argued to be the reason for this. However, that means that the reselection and replay window size parameter are related. If the reselection window size is large, the replay window size should be large as well.

9.5.7 BHH Variant: Replay

For this experimental group, different replay windows sizes were evaluated. It was found that the replay window is problem-dependent with some dataset preferring short memory and other preferring long memory. Take note of the relationship between the reselection, replay and reanalysis window sizes. The value of the one does influence the value of other and should be carefully selection together.

9.5.8 BHH Variant: Reanalysis

For this experimental group different reanalysis window sizes were evaluated. It was also found that the reanalysis window size is problem-dependent. Given the relationship between the reselection, replay and reanalysis window sizes, a suggestion to eliminate hyper-parameters is to remove the notion of reanalysis and merge with reselection. Reselection is shown to correlate with reanalysis and thus can be removed by just setting the reanalysis window size equal to the reselection size.

9.5.9 BHH Variant: Burn In

For this experimental group different burn in window sizes were evaluated. It was found that in general no burn in is preferred, but some exceptions preferring just a very small burn in. In general is was shown that large burn in values are not good.

9.5.10 BHH Variant: Normalise

For this experimental group an empirical test was conducted to determine the effects of normalisation of pseudo counts in the performance log. The choice of this parameter was shown to be problem depedent.

9.5.11 BHH Variant: Discounted Rewards

It can be concluded that the outcomes of the empirical process were successful and that the scientific process has been followed accordingly. From this, it can be concluded that all of the objective of the dissertation have been met and that the research can be concluded. The next section provides a brief discussion of further research opportunities that stem from this work.

9.6 Further Research

Throughout this dissertation, many different opportunities have been identified for future research. This section contains a summmary of these topics.

Prior Knowledge

The BHH initialises the concentrations for its distributions symmetrically (a value of 1 for all indices). Symmetric initialisation contains no prior 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 concentration for the distributions. This research would then evaluate if such prior information has a positive impact on the outcomes of the BHH.

Early Stopping

It has been shown that the BHH suffers drastically from overfitting as the majority of learning happens during the first few epochs (large datasets have more mini-batch iterations per epoch) resulting in the majority of the training process, which span 30 epochs, to try recover from overfitting. Early stop was purposefully not utilised in this research in order to see how the BHH would behave after training has saturated. Early stopping could provide valuable insight into the actual ranked performance of the BHH since overfitting would then no longer be present and a ranked performance evaluation can be done at some point in time.

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, but rather keeps the entire entity pool and just switch allocated heuristics. 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, more often, which should have a positive effect on the outcomes of the BHH

Models

For this dissertation, focus was put on training FFNNs. However, in theory one should be able to swap out FFNNs for any other mathematical model that can be optimised. Suggestions include *deep neural network*s (DNNs), *recurrent neural network*s (RNNs) and dynamic optimisation problems.

Credit Search

Credit assignment in the context of this dissertation was has 2 outcomes, true and false. Therefore, credit assignment and by definition performance evaluation is modeled as a distribution with a beta probability distribution as a prior probability distribution. Heuristic selection has multiple possible outcomes, yielding a index associated with a heuristic. Therefore, heuristic selection is modeled as a multinomial/categorical with a dirichlet probability distribution as a prior probability distribution. Since the dirichlet-multinomial distributions are the multivariate extensions of the beta-binomial distributions, it show be possible to extend credit assignment to a multivariate version. A suggestion then is to learn which credit assignment to select. The results of this have show that credit assignment is problem-dependent. Therefore, include credit selection as part of the selection pool implemented by the BHH. One could then include multiple strategies, each with varying exploration-exploitation trade-off.

Selection Validation

The intent behind a selection validation mechanism is to confirm that selections are successful before they are utilised. In other words, a small verification window where it is decided whether the outcomes of a selection mechanism was beneficial to the learning outcomes or not. These selection validations are based on an "acceptance criterion" that a selection has to pass before it is validated. Inclusion of a selection validation mechanism for the BHH could yield interesting outcomes.

Dynamic Reselection Window Size Strategies

It was shown that generally larger reselection window sizes are required. It was shown that there is a relationship between the reselection, replay and reanalysis window sizes. There is an inherent trade-off between exploration and exploitation and therefore, investigation into dynamic sizing strategies could be prove to yield interesting outcomes.

Heuristic Pools

This dissertation only considered 3 different configurations of the heuristic pool. The following alternative configurations could be considered:

- Multiple of the same heuristic, but with different hyper-parameter values.
- Very large or very small heuristic pool sizes.
- Inclusion of black-box optimisers such as CMA-ES.
- Ensemble pools where the heuristic selection is a selection of a ensemble configuration.

Parameter Pools

Throughout this dissertation, many of the findings have been said to be "problem-dependent". A possible solution to this is to apply the BHH then for all parameters, not just heuristics. This means to include hyper-parameters such as learning rate, which can be "learnt" during training. This eliminates a lot of the burden to tediously fine-tune and optimise hyper-params.

This could also provide opportunity to investigate other FFNN paradigms such as neural architecture search (ref ???) by including various parameters, that describe the NN architecture. The BHH slection mechanism can then include these elements for learning as well.

Models

This dissertation focused on FFNNs as the underlying model to be trained. However, other models can also be considered.

9.7 Documentation and Data

All data, source code, logs, configurations, scripts and analysis is hosted at: https://github.com/arneschreuder/masters.ai with raw empirical data made available on Google Cloud project id: masters-287321.

9.8 Closing Statements

The research that is presented in this dissertation included the development and detailed analysis of a novel BHH. This chapter presented a review of the research objectives. In conclusion it is found that all of the research objectives have been met. The empirical process was sound and the research outcome have been shown to be statistically significant. This concludes the research presented in this document

Bibliography

[1] URL: https://archive.ics.uci.edu/ml/index.php.

[2] Abien Fred Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[3] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. "Data mining: A preprocessing engine". In: *Journal of Computer Science* 2.9 (2006), pp. 735–739.

[4] 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.

[5] Deborah Ashby. "Bayesian statistics in medicine: a 25 year review". In: *Statistics in medicine* 25.21 (2006), pp. 3589–3631.

[6] 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.

[7] Jonathon Shlens Barret Zoph Vijay Vasudevan and Quoc Le. *AutoML for large scale image classification and object detection*. https://ai.googleblog.com/2017/11/automl-for-large-scale-image.html. Blog. 2017.

[8] 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.

[9] Brian Beers. *What P-value tells us*. Jan. 2022. URL: https://www.investopedia.com/terms/p/p-value.asp.

[10] Yoshua Bengio. "Gradient-based optimization of hyperparameters". In: *Neural computation* 12.8 (2000), pp. 1889–1900.

[11] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[12] 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.

[13] 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.

[14] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[15] Christian Blum and Andrea Roli. “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[16] 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.

[17] Hans J Bremermann et al. “Optimization through evolution and recombination”. In: *Self-organizing systems* 93 (1962), p. 106.

[18] Jason Brownlee. “Supervised and unsupervised machine learning algorithms”. In: *Machine Learning Mastery* 16.03 (2016).

[19] Jason Brownlee. *How to One Hot Encode Sequence Data in Python*. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/. Blog. 2017.

[20] Jason Brownlee. “What is the Difference Between Test and Validation Datasets”. In: *Machine Learning Process* (2017).

[21] 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.

[22] 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.

[23] Edmund K Burke, Graham Kendall, and Eric Soubeiga. "A tabu-search hyperheuristic for timetabling and rostering". In: *Journal of heuristics* 9.6 (2003), pp. 451–470.

[24] 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.

[25] James Cannady. "Artificial neural networks for misuse detection". In: *National information systems security conference*. Vol. 26. Baltimore. 1998.

[26] 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.

[27] 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.

[28] Tautvydas Cibas, F Fogelman Soulié, Patrick Gallinari, and Sarunas Raudys. "Variable selection with neural networks". In: *Neurocomputing* 12.2-3 (1996), pp. 223–248.

[29] Wikimedia Commons. *Stochastic Gradient Descent*. 2006. URL: %5Curl%7Bhttps://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png%7D.

[30] Wikimedia Commons. *Beta distribution*. 2014. URL: %5Curl%7Bhttps://upload.wikimedia.org/wikipedia/commons/f/f3/Beta_distribution_pdf.svg%7D.

[31] Wikimedia Commons. *Beta distribution*. 2014. URL: %5Curl%7Bhttps://upload.wikimedia.org/wikipedia/commons/7/74/Dirichlet.pdf%7D.

[32] Wikimedia Commons. *Cumulative Beta distribution*. 2014. URL: %5Curl%7Bhttps://upload.wikimedia.org/wikipedia/commons/1/11/Beta_distribution_cdf.svg%7D.

[33] 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.

[34] Paulo Cortez and Anibal de Jesus Raimundo Morais. "A data mining approach to predict forest fires using meteorological data". In: (2007).

[35] Paulo Cortez and Alice Maria Gonçalves Silva. “Using data mining to predict secondary school student performance”. In: (2008).

[36] 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.

[37] 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.

[38] 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.

[39] Charles Darwin. *On the origin of the species and the voyage of the beagle*. Graphic Arts Books, 2012.

[40] Yann 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: *arXiv preprint arXiv:1406.2572* (2014).

[41] 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.

[42] Abraham De Moivre. *The doctrine of chances: or, A method of calculating the probability of events in play*. W. Pearson, 1718.

[43] 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.

[44] 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.

[45] Thomas G Dietterich et al. “Ensemble learning”. In: *The handbook of brain theory and neural networks* 2.1 (2002), pp. 110–125.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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).

[51] Russ Eberhart, Pat Simpson, and Roy Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., 1996.

[52] 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.

[53] 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.

[54] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[55] Andries P Engelbrecht and Ian Cloete. "A sensitivity analysis algorithm for pruning feedforward neural networks". In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 2. IEEE. 1996, pp. 1274–1278.

[56] Andries Petrus Engelbrecht, Ian Cloete, Jaco Geldenhuys, and Jacek M Zurada. "Automatic scaling using gamma learning for feedforward neural networks". In: *International Workshop on Artificial Neural Networks*. Springer. 1995, pp. 374–381.

[57] 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.

[58] 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.

[59] Huiyuan Fan. “A modification to particle swarm optimization algorithm”. In: *Engineering Computations* (2002).

[60] 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.

[61] Mercedes Fernández-Redondo and Carlos Hernández-Espinosa. “Weight initialization methods for multilayer feedforward.” In: *ESANN*. 2001, pp. 119–124.

[62] Matthias Feurer and Frank Hutter. “Hyperparameter optimization”. In: *Automated machine learning*. Springer, Cham, 2019, pp. 3–33.

[63] Ronald A Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[64] GP Fletcher and CJ Hinde. “Learning the activation function for the neurons in neural networks”. In: *ICANN94*. Springer, 1994, pp. 611–614.

[65] Louise Francis. “Neural networks demystified”. In: *Casualty Actuarial Society Forum*. 2001, pp. 253–320.

[66] 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.

[67] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O’Reilly Media, Inc.", 2017.

[68] 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.

[69] 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.

[70] Fred Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[71] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[72] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[73] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.

[74] Jacomine Grobler. "The heterogeneous meta-hyper-heuristic: from low level heuristics to low level metaheuristics". PhD thesis. University of Pretoria.

[75] 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.

[76] 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.

[77] 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.

[78] Branimir K Hackenberger. "Bayes or not Bayes, is this the question?" In: *Croatian medical journal* 60.1 (2019), p. 50.

[79] Jacques Hadamard. *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*. Vol. 33. Imprimerie nationale, 1908.

[80] Boris Hanin. "Which neural net architectures give rise to exploding and vanishing gradients?" In: *Advances in Neural Information Processing Systems*. 2018, pp. 582–591.

[81] David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[82] 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.

[83] Babak Hassibi, David G Stork, and Gregory Wolff. “Optimal brain surgeon: Extensions and performance comparisons”. In: *Advances in neural information processing systems*. 1994, pp. 263–270.

[84] 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).

[85] 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.

[86] 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.

[87] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[88] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[89] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. “Meta-learning in neural networks: A survey”. In: *arXiv preprint arXiv:2004.05439* (2020).

[90] Yanbo Huang. “Advances in artificial neural networks–methodological development and application”. In: *Algorithms* 2.3 (2009), pp. 973–1007.

[91] 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.

[92] 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.

[93] Kenneth E Iverson. “A programming language”. In: *Proceedings of the May 1-3, 1962, spring joint computer conference*. 1962, pp. 345–351.

[94] Anil Jain, Karthik Nandakumar, and Arun Ross. "Score normalization in multimodal biometric systems". In: *Pattern recognition* 38.12 (2005), pp. 2270–2285.

[95] Anil K Jain, Jianchang Mao, and KM Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 3 (1996), pp. 31–44.

[96] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[97] 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.

[98] 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.

[99] Raymond Dennis Keene, Byron A Jacobs, and Tony Buzan. *Man v Machine: The ACM Chess Challenge: Garry Kasparov v IBM's Deep Blue*. BB Enterprises, 1996.

[100] Troy D Kelley and Lyle N Long. "Deep Blue cannot play checkers: The need for generalized intelligence for mobile robots". In: *Journal of Robotics* 2010 (2010).

[101] 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.

[102] 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.

[103] Mary B Kennedy. "Synaptic signaling in learning and memory". In: *Cold Spring Harbor perspectives in biology* 8.2 (2016), a016824.

[104] 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.

[105] KhanAcademy. *The Synapse*. https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse.

[106] 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.

[107] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[108] Ron Kohavi et al. "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." In: *Kdd*. Vol. 96. 1996, pp. 202–207.

[109] Anders Krogh and John A Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[110] Sandeep Kumar and Eugene H Spafford. "A pattern matching model for misuse intrusion detection". In: (1994).

[111] 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.

[112] 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. DOI: 10.1109/COMITCon.2019.8862255.

[113] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[114] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[115] 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.

[116] Claude Lemaréchal. "Cauchy and the gradient method". In: *Doc Math Extra* 251.254 (2012), p. 10.

[117] Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra. "Deal or no deal? end-to-end learning for negotiation dialogues". In: *arXiv preprint arXiv:1706.05125* (2017).

[118] 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.

[119] Kaifeng Lv, Shunhua Jiang, and Jian Li. "Learning gradient descent: Better generalization and longer horizons". In: *arXiv preprint arXiv:1703.03633* (2017).

[120] 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.

[121] Rinat Maksutov. *Deep study of a not very deep neural network. Part 2: Activation functions*. https://towardsdatascience.com/deep-study-of-a-not-very-deep-neural-network-part-2-activation-functions-fd9bd8d406fc. Blog. 2018.

[122] 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.

[123] 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.

[124] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. "Designing Neural Networks Using Genetic Algorithms." In: *ICGA*. Vol. 89. 1989, pp. 379–384.

[125] 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.

[126] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[127] David J Montana and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms." In: *IJCAI*. Vol. 89. 1989, pp. 762–767.

[128] 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.

[129] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *ICML*. 2010.

[130] Radford M Neal. "Bayesian learning via stochastic dynamics". In: *Advances in neural information processing systems*. 1993, pp. 475–482.

[131] Gerrit Stephanus Nel. "A hyperheuristic approach towards the training of artificial neural networks." PhD thesis. University of Stellenbosch, 2021.

[132] 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.

[133] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.

[134] 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.

[135] 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.

[136] 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.

[137] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. "A comprehensive analysis of hyper-heuristics". In: *Intelligent data analysis* 12.1 (2008), pp. 3–23.

[138] 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.

[139] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984. ISBN: 9780201055948.

[140] N. Pillay and R. Qu. *Hyper-Heuristics: Theory and Applications*. Natural Computing Series. Springer International Publishing, 2018. ISBN: 9783319965147.

[141] Nelishia Pillay. "Intelligent system design using hyper-heuristics". In: *South African Computer Journal* 56.1 (2015), pp. 107–119.

[142] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[143] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[144] Mike Schuster Quoc V. Le. *A Neural Network for Machine Translation, at Production Scale*. https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html. Blog. 2016.

[145] Anastassia S Rakitianskaia and Andries P Engelbrecht. "Training feedforward neural networks with dynamic particle swarm optimisation". In: *Swarm Intelligence* 6.3 (2012), pp. 233–270.

[146] 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.

[147] R. Reed and R.J. MarksII. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. A Bradford Book. MIT Press, 1999. ISBN: 9780262181907.

[148] Colin R Reeves and Stewart J Taylor. "Selection of training data for neural networks by a genetic algorithm". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1998, pp. 633–642.

[149] CR Reeves. "Modern heuristic techniques for combinatorial problems Blackwell Scientific Press". In: *Oxford, UK* 1.99 (1993), p. 2.

[150] Bob Ricks and Dan Ventura. "Training a quantum neural network". In: *Advances in neural information processing systems*. 2004, pp. 1019–1026.

[151] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[152] 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.

[153] Marc HJ Romanycia and Francis Jeffry Pelletier. "What is a heuristic?" In: *Computational Intelligence* 1.1 (1985), pp. 47–58.

[154] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[155] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[156] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[157] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[158] Anders Sandberg and Nick Bostrom. "Whole brain emulation". In: (2008).

[159] Choe Sang-Hun. *Googleś Computer Program Beats Lee Se-dol in Go Tournament*. https://www.nytimes.com/2016/03/16/world/asia/korea-alphago-vs-lee-sedol-go.html. Blog. 2016.

[160] Jeffrey Curtis Schlimmer. "Concept acquisition through representational adjustment". PhD thesis. University of California, Irvine, 1987.

[161] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[162] Yuhui Shi et al. "Particle swarm optimization: developments, applications and resources". In: *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*. Vol. 1. IEEE. 2001, pp. 81–86.

[163] 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.

[164] 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.

[165] 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.

[166] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550 (Oct. 2017), pp. 354–. URL: http://dx.doi.org/10.1038/nature24270.

[167] Yashpal Singh and Alok Singh Chauhan. “NEURAL NETWORKS IN DATA MINING.” In: *Journal of Theoretical & Applied Information Technology* 5.1 (2009).

[168] 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.

[169] 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.

[170] Donald F Specht. “A general regression neural network”. In: *IEEE transactions on neural networks* 2.6 (1991), pp. 568–576.

[171] Sunny Srinidhi. *Label Encoder vs. One Hot Encoder in Machine Learning*. https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621. Blog. 2018.

[172] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[173] Rainer Storn. “On the usage of differential evolution for function optimization”. In: *Proceedings of North American Fuzzy Information Processing*. IEEE. 1996, pp. 519–523.

[174] 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.

[175] 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).

[176] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

[177] 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.

[178] 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.

[179] Gilbert Syswerda et al. "Uniform crossover in genetic algorithms." In: *ICGA*. Vol. 3. 1989, pp. 2–9.

[180] 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.

[181] 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.

[182] Georg Thimm and Emile Fiesler. "Neural network initialization". In: *International Workshop on Artificial Neural Networks*. Springer. 1995, pp. 535–542.

[183] 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.

[184] Frans Van Den Bergh et al. "An analysis of particle swarm optimizers". PhD thesis. University of Pretoria, 2007.

[185] 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.

[186] 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.

[187] 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).

[188] Andrich Benjamin Van Wyk. "An Analysis of Overfitting in Particle Swarm Optimised Neural Networks". PhD thesis. University of Pretoria, 2014.

[189] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial intelligence review* 18.2 (2002), pp. 77–95.

[190] Dennis Wackerly, William Mendenhall, and Richard L Scheaffer. *Mathematical statistics with applications*. Cengage Learning, 2014.

[191] Kwok Ho Wan, Oscar Dahlsten, Hlér Kristjánsson, Robert Gardner, and MS Kim. "Quantum generalisation of feedforward neural networks". In: *npj Quantum Information* 3.1 (2017), p. 36.

[192] 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.

[193] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Vol. 1. John Wiley & Sons, 1994.

[194] 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.

[195] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[196] Saurabh Yadav. *Weight Initialization Techniques in Neural Networks*. https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78. Blog. 2018.

[197] 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.

[198] Edward N Zalta. "Metaphysics Research Lab". In: *Center for the Study of Language and Information, Stanford University, Stanford, CA* (2015).

[199] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[200] Andreas Zell. *Simulation neuronaler netze*. Vol. 1. 5.3. Addison-Wesley Bonn, 1994.

[201] 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.

[202] 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.

Appendix A

Datasets

Appendices follow exactly the same structure as chapters. They are used to describe aspects that are not central to the dissertation (for example, an algorithm that you benchmark against, but do not focus on in the main text, or a discussion on the sample datasets you test your algorithm on).

A.1 Summary

As always, provide a summary at the end.

Appendix B

Statistical Analysis

Appendices follow exactly the same structure as chapters. They are used to describe aspects that are not central to the dissertation (for example, an algorithm that you benchmark against, but do not focus on in the main text, or a discussion on the sample datasets you test your algorithm on).

B.1 Summary

As always, provide a summary at the end.

Appendix C

Acronyms

Provide a short introduction to the appendix here. Mention that acronyms are listed alphabetically and typeset in bold, with the meaning of the acronym alongside. Also note that you must include acronyms using `gls or glspl` for them to show up here. If there are no acronyms defined in your text, this introduction will also not be displayed:

Acronyms

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 *back-propagation*

CatEX *categorical cross entropy*

CLT *central limit theorem*

CSP *Constraint Satisfaction Problem*

DE *differential evolution*

DNN *deep neural network*

EA *evolutionary algorithm*

EC *evolutionary computation*

FFNN *feedforward neural network*

GA *genetic algorithm*

GD *gradient descent*

HH *hyper-heuristic*

HMM *Hidden Markov Model*

LReLU *leaky rectified linear unit*

MAE *mean absolute error*

MAP *maximum a posteriori estimation*

MCMC *Markov Chain Monte Carlo*

MH *meta-heuristic*

ML *machine learning*

MLE *maximum likelihood estimation*

Momentum *momentum*

MSE *mean squared error*

NAG *Nesterov accelerated gradients*

NFL *no free lunch theorem*

NN *neural network*

PDF *probability density function*

PMF *probability mass function*

PSO *particle swarm optimisation*

ReLU *rectified linear unit*

RL *reinforcement learning*

RMS *root mean squared*

RMSE *root mean squared error*

RMSProp *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 D

Derived Publications

Explain that the following list includes a list of the publications derived from produced by BibTeX (we will hopefully release an automated way of generating this list some time in the future), and that the references are all correct.

- First reference.
- Second reference.
- Third reference.

Appendix E

Symbols

This appendix summarises the important symbols used throughout the dissertation. For each symbol, a short description is provided as well as the reference to the equation that makes use it. Symbols are divided according to the chapter in which they were introduced and are ordered alphabetically.

E.1 Chapter 2: Artificial Neural Networks

ref usage

α	Control parameter for leaky ReLU that controls non-zero gradients.
β	Batch size.
c	Index for classes.
C	Total number of classes.
E	Total number of epochs.
ϵ	Error value calculated from loss/cost function.
f	The activation function implemented by the AN.
f_{AN}	The non-linear function represented by the AN.
$\vec{h}$	The vector representing the activations of the hidden units.
h_j	The j-th hidden unit and thus its activation.
i	Index for the input dimensions.
I	Total number of dimensions on the input space.
j	Index for the hidden layer.

J The total number of units in the hidden layer.

k Index for the output dimensions.

K The total number of dimensions in the output space.

λ Control parameter for the steepness of sigmoid and hyperbolic tangent activation functions.

net The net input signal.

net' The augmented net input signal which contains the bias term θ.

μ_i The mean over all input patterns at index i.

p Index used for patterns in the set of all training data.

P Total number of patterns in the training data.

ω_{min} The lower bound of the range for random uniform sampling.

ω_{max} The upper bound of the range for random uniform sampling.

$\mathbb{R}$ The real number space.

σ The standard deviation.

σ_i^2 Unit variance.

T The total number of dimensions in the target space.

θ The bias or threshold term.

$\vec{v}$ The vector of weights associated with the input.

v_i The i-th index of the weight vector associated with the input.

$v_{i,j}$ The i, j-th index of the weight vector between the input and hidden layers.

$\vec{w}$ The weight vector between the hidden and output layers.

$w_{j,k}$ The j, k-th index of the weight vector between the hidden and output layers.

$\vec{x}$ The input vector.

$\vec{x}_p$ The input vector for pattern p.

x_i The i-th index of the input vector $\vec{x}$

$x_{i,p}$ The i-th index of the input vector $\vec{x}$ for pattern p.

$x'_{i,p}$ The pre-processed input at the i-th index of the input vector $\vec{x}$ for pattern p.

$x_{i_{min}}$ The minimum value at the i-th index of the input vector $\vec{x}$ for all patterns in P.

$x_{i_{max}}$ The maximum value at the i-th index of the input vector $\vec{x}$ for all patterns in P.

$\vec{y}$ The output vector.

$\vec{y}_p$ The output vector for pattern p.

y_k The k-th dimension of the output vector $\vec{y}$.

y'_k The post-processed output at the k-th dimension of the output vector $\vec{y}$.

$y_{k,p}$ The k-th dimension of the output vector $\vec{y}$ for pattern p.

$y'_{k,p}$ The post-processed output at the k-th dimension of the output vector $\vec{y}$ for pattern p.

$\hat{y}_{k,p}$ The k-th dimension of the target output for pattern p.

Index