Multi-Armed Bandits in Neuro-Evolution

Research Project

COMP90070

75pt

Supervisor: Michael Kirley

Der Hann Lai - 754672

Semester 1, 2020

Abstract

This project explores the use of Multi-Armed Bandits (MAB) in NeuroEvolution of Augmenting Topologies (NEAT). The project aims to decrease the generations required to find solutions and provide a generally applicable speedup to NEAT. Three MAB - Softmax, ϵ-Greedy, Thompson Sampling - are compared against each other with varying parameter degrees of greediness for each Bandit. The ratio of positive and negative impacts to fitness by each arm of the MAB is used as rewards. Two reward schemes were used which summarise fitness changes, one uses the ratio of positive and total changes, and another uses a simple binary value indicating positive or negative change. The MAB were tested in continuous control tasks from Open-AI gym and some classification tasks from UCI Machine Learning Repository. The results find that this implementation of MAB is not able to improve over standard NEAT with basic parameters, but are close to matching it. The results show that less greedy MAB were able to outperform greedier MAB simply due to ignoring the reward values. It is found that Thompson Sampling is definitively worse than Softmax and ϵ-Greedywith higher explore parameters. These findings can be used as a starting point for further research using more sophisticated alterations to NEAT.

Declaration

I certify that - this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

The thesis is 13,200 words in length (excluding text in images, table, bibliographies and appendices).

Der Hann Lai – June 26, 2020

Acknowledgements

I would like to express my greatest thanks to A/Prof. Michael Kirley for supervising me throughout my project and for his patience with my problems.

I would also like to thank my friends for listening to my rambles about ideas despite being in completely different fields, particularly Michelle Lee for her insight on how to analyse my data.

Lastly I would like to thank my family for supporting me throughout every facet of life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Neural Networks (ANN) are a general purpose Machine Learning (ML) tool modelled after biological brains consisting of connected computing nodes which process and communicate information. The Architecture of ANN plays a large part in optimising the performance of ANN, so search methods known as Neural Architecture Search (NAS) were developed to tackle this problem. Genetic Algorithms (GA) is an Evolutionary based method of heuristic optimisation which generates a population of candidate solutions, mutates and applies crossover to the candidates until time runs out or a threshold fitness is reached, which corresponds to finding a satisfactory solution for a given task.

NEAT, short for NeuroEvolution of Augmenting Topologies, applies GA to NAS. NEAT incrementally adds structure and adapts the parameters of ANNs to improve networks and find solutions to a given problem. However, NEAT is time consuming to run, requiring each network in every generation be evaluated in the test environment until the threshold fitness is reached. Running for more generations naturally requires more time as more evaluations are required. Other than time required, there are many hyper-parameters which control the rate of applying mutation and crossover operators. The hyper-parameters determine the probability of performing different mutations and crossover, and are static once defined. Rather than remaining static, operators can be selected dynamically based on feedback from the evolutionary process.

Multi-Armed Bandits is a method of selecting a sequence of actions, known as arms, to maximise reward based on the reward observed from selecting each arm. By using MAB in NEAT where each arm represents a genetic operator, we can observe the change in fitness as the reward for an associated operator and use the reward values to better inform decisions regarding which operator to choose.

This project is motivated by the long run times of NEAT [33] and attempts to provide a speed-up in run time. The goal is to integrate MAB into NEAT and adaptively select genetic operators based on improvements to fitness for faster improvement and thus less time to find solutions. This should also bring the benefit that hyper-parameters for NEAT itself become less important and shifts to the parameters of the MAB. As there are many parameters for NEAT, it is difficult to determine optimal parameters without extensive parameter tuning, adding even more to the run time. The main questions for this thesis are:

1. Can MAB decrease the number of generations required to find solutions?
    - 1.1. Which MAB performs the best in this context?
    - 1.2. How do MAB parameters for MAB affect the ability to find solutions?
    - 1.3. How well does fitness work as a reward for MAB?
2. What benefits do MAB provide over static NEAT, if any?

The results show that there is no significant improvement in performance from the proposed Multi-Armed NEAT over standard NEAT. The best performing MAB of the ones selected is the Softmax MAB. The overall poor performance is attributed to the direct use of improvements to fitness as a reward value. The rewards are noisy and provide little meaningful feedback to the chain of mutations typically required to make actual improvements to fitness. This is supported by the fact that MAB with higher exploration parameters were almost able to match the performance of the standard NEAT algorithm, meaning that random exploration is better than being directed by the changes in fitness.

The rest of the report is structured as follows. Chapter 2 introduces the relevant background theory. Chapter 3 explains where and how MAB are integrated into NEAT as well as the MAB selected and the reward schemes devised. Chapter 5 presents and discusses the results of the experiment. Chapter 6 concludes the report with the major findings and suggestions for future research.

Chapter 2

Background

2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are general purpose tools used in Machine Learning (ML) for tasks including classification, regression, robot control, and content generation. ANN are based on the perceptron model, a model based on biological neurons [19]. Each perceptron, referred to as a node, consisting of inputs, activation functions, and an output. The inputs are aggregated and passed through the activation function before being passed to the nodes it is connected to. This is shown in Figure 2.1, each input value x_i including the bias term x_0 is multiplied by the associated weight w_i before being summed up and passed through an activation function f. A node simply computes the function $f(\bar{x} \cdot \bar{w})$. The bias term is a fixed input with value 1 and variable weight w_0 that conceptually shifts the activation function on the x-axis, which may be an important factor for some tasks.

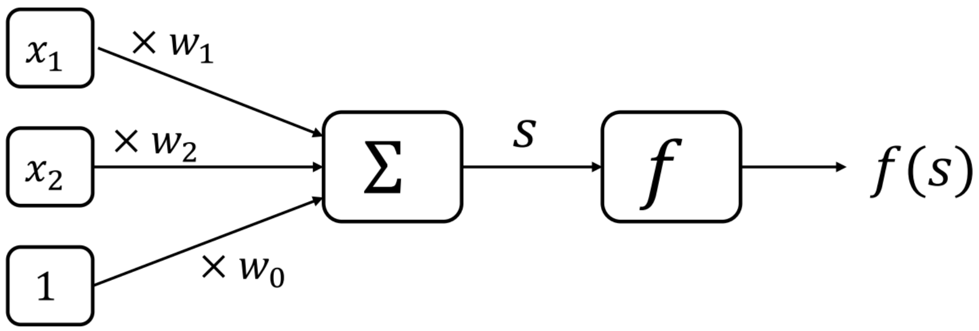


Figure 2.1: Perceptron model. A perceptron with two inputs and one bias term. A weighted sum is calculated and passed through an activation function before being outputted. Image recreated from [20].

ANN consist of a series of nodes, connected by weighted edges which determine the flow of information from node to node. Each edge is weighted and acts as multiplier for the value passed along that edge. A specific class of layout of ANN is the fully-connected feed-forward network, where nodes are arranged in consecutive layers of nodes shown in Figure 2.2. Every node in one layer is connected to every node in the layer directly before it. The layers are input layers, output layers, and anything in between is known as Hidden layers.

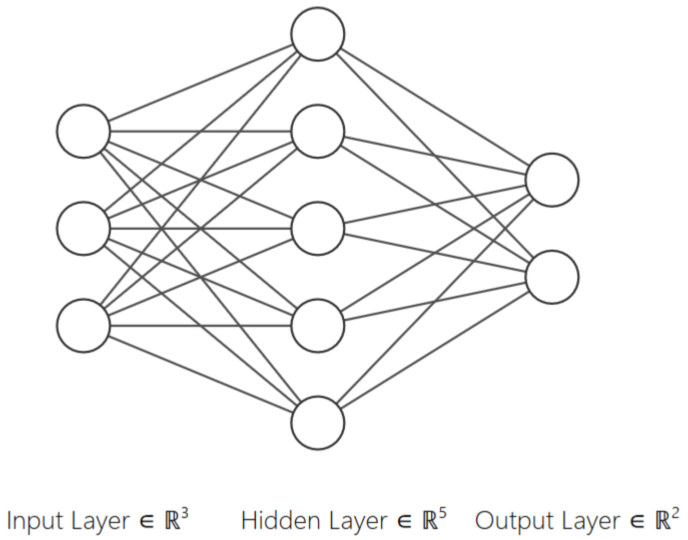


Figure 2.2: Feed-Forward Network with one hidden layer. Information flows strictly from the Input layer to the output layer.

In traditional Machine Learning, the weights of the network are adjusted to minimise the error between the generated output and expected output through backpropogation [21]. This structure is widely used for its ability to approximate functions over small domains to any degree of accuracy, known as the universal approximation theorem [5]. However there is no concrete method of determining the appropriate parameters of the neural network, [23] proves that two layers is sufficient for all general function representations, but does not indicate methods of finding the parameters of the network. Often-times ANN become stuck in local minima and become unable to further improve despite potential lower error elsewhere.

2.2 Neural Architecture

The connectivity of a network is known as its Architecture. It plays an important role in ANN performance. Too few nodes and the ANN will be unable to find any useful representation, too many and the ANN may overfit to the problem at hand, losing its ability to generalise. The process of finding and testing neural architectures is known as Neural Architecture Search (NAS)

When done manually, NAS is called hyper-parameter optimisation, as the number of layers, nodes, activation functions, etc. are effectively the parameters required to define the selected ANN. A limiting factor to manual NAS is that the search space is large, the results take time to compute, and subsequently, requires a lot of a user's time. Manually searching for architecture is also restrictive as non-fully-connected feed-forward networks with individual activation functions are more difficult to justify. Automatically searching for neural architectures is therefore a favourable alternative. By defining the network's representation and how to modify it, search spaces and search strategies can be defined to find ANN that give high performance in a given task.

NAS as been applied with success in many respects. Restarting a network when it gets stuck in local minima with random parameters is shown to be effective for some tasks [11]. Effecient Neural Architecture Search (ENAS) via Parameter Sharing trains a larger super-graph and discovers optimal sub-graphs within the larger graph [18]. It is a subtractive model of NAS, where a larger graph is first found, then reduced while meeting performance expectations. Differentiable Architecture Search (DARTS) tackles the issue of not having a differentiable search the discrete space of NAS by applying a continuous relaxation of the architecture representation, allowing the use of gradient descent [14]. Neural Architecture Search by Hill-climbing (NASH) is a simple Hill-Climbing algorithm which alters a network by applying network morphisms and replacing it with the most promising network produced [8]. The search space for Neural Architectures can be modelled as a tree. By using Upper-Confidence bound applied to Trees (UCT), competitive networks can be found within just one day of GPU time where other methods can take months or years of GPU time [31]. Monte-Carlo Tree Search (MCTS) can be used to search the space of architectures and can also achieve competitive architectures without much need from human experts [16].

One particular approach to NAS is the use of Genetic Algorithms. There are many applications of this, known as Neuro-Evolution [25], but this project will focus only on the Genetic Algorithms itself, and one particular method known as NEAT [27].

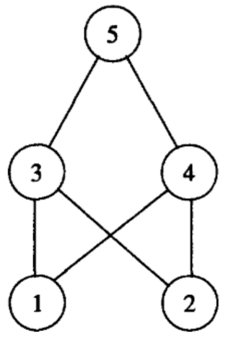
2.3 Genetic Algorithms

Genetic Algorithms (GA) are meta-heuristic optimisation processes loosely based on Darwin's theory of Evolution introduced by Holland (1975) [12]. The idea centers around passing down genes of the fittest members of the population from generation to generation, and applying mutations in hopes of finding fitter genes. Each member in the population is a representation of a solution to a selected problem. The core of GA consists of genomes, genetic operators, evaluation, and selection process.

A genome a member of a larger population, and is a collection of genes representing a phenotype. The terminology used reflects that of natural evolution, but represent simpler concepts. A gene is a singular characteristic of a genome, which altogether represents a solution. The phenotype is the observable behaviour of a genome.

A conection matrix

$$\begin{matrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$



The chromosome for the whole matrix

0011000110000010000100000

The chromosome for the feed forward portion only

0110110011

Figure 2.3: Neural Network represented as a bit string. The connection matrix is defined for a fixed topology and rows are placed end to end to create a string. The feed forward portion refers to the top triangle of the connection matrix only, representing only the feed forward connections. Recreated from [22]

The canonical algorithm uses bit strings which can represent any number of potential solutions. For example, Figure 2.3 represents an ANN with fixed topology and variable connections. In this case, a chromosome is just a different term for a genome. Each row and column is indexed by the number of the node. The connection matrix is serialised into a string by placing each row end to end. In the figure the forward portion refers to the serialised representation of the top right triangle, where the rows of decreasing length are placed end to end. In this case a gene is every bit in the chosen representation. The genome is the total collection of all genes, and the phenotype is the appearance of the network/graph.

The next component are the genetic operators. These are actions which alter a genome's genes. This can be done in two ways, through mutation or crossover. Mutation is an alteration to a gene, such as flipping a bit or if the genome is more flexible in representation, adding more bits. Crossover is the combination of two or more parent genomes to produce an offspring with a mix of genes from the parents.

Next is the evaluation component. As the name suggest, it evaluates the performance of a genome in the given environment. More specifically, it evaluates the phenotype represented by the genome and assigns it a performance value known as fitness. Evaluation can be done looking at each genome individually, or relative to the rest of the population.

Tied closely to evaluation is the selection process. Selection ranks genomes by fitness, offering fitter genomes a higher chance of reproduction either through mutation, crossover, or both. This passes fitter genes to future generations, and by mutating individuals in the population, provides opportunities for genomes to improve over their parents.

A high level overview of GA is shown in Figure 2.4.

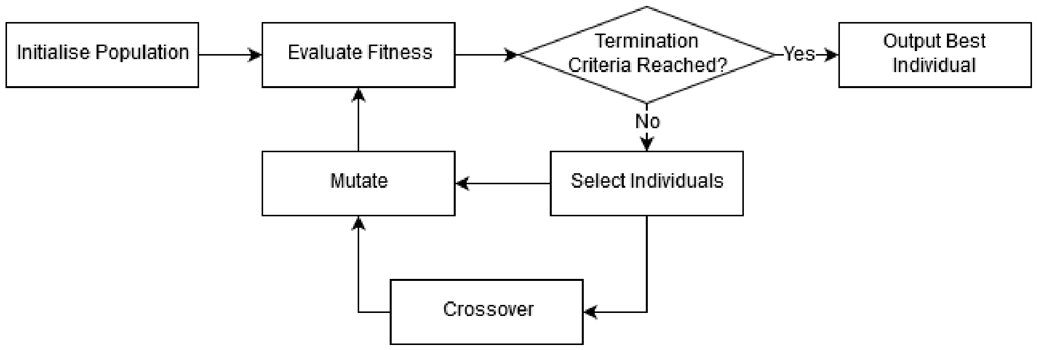


Figure 2.4: High level overview of Genetic Algorithm. Each iteration from evaluating fitness to mutation is one generation.

GA has been applied to ANN, as shown in Figure 2.3. But as research furthers, the bit representation is replaced by more sophisticated means and alternate encoding such as indirect encoding, where the genome represents instructions on how to construct a graph.

The process of creating ANN using GA is known as Neuro-Evolution. In short, Neuro-Evolution performs NAS using GA. A comprehensive review is provided by Stanley et. al (2019) here [25]. The main focus for this project is on NeuroEvolution of Augmenting Topologies (NEAT) [27], which is explored more in depth in the following section.

2.4 NEAT in Depth

The main algorithm this project focuses on is NeuroEvolution of Augmenting Topologies (NEAT). [27]. NEAT incrementally build networks from minimal topology by adding nodes and connections and recombining networks using GA. NEAT implements GA with custom genetic representation, crossover and mutation mechanics, and solves some problems related to naive representations of ANN.

2.4.1 Genetic Representation

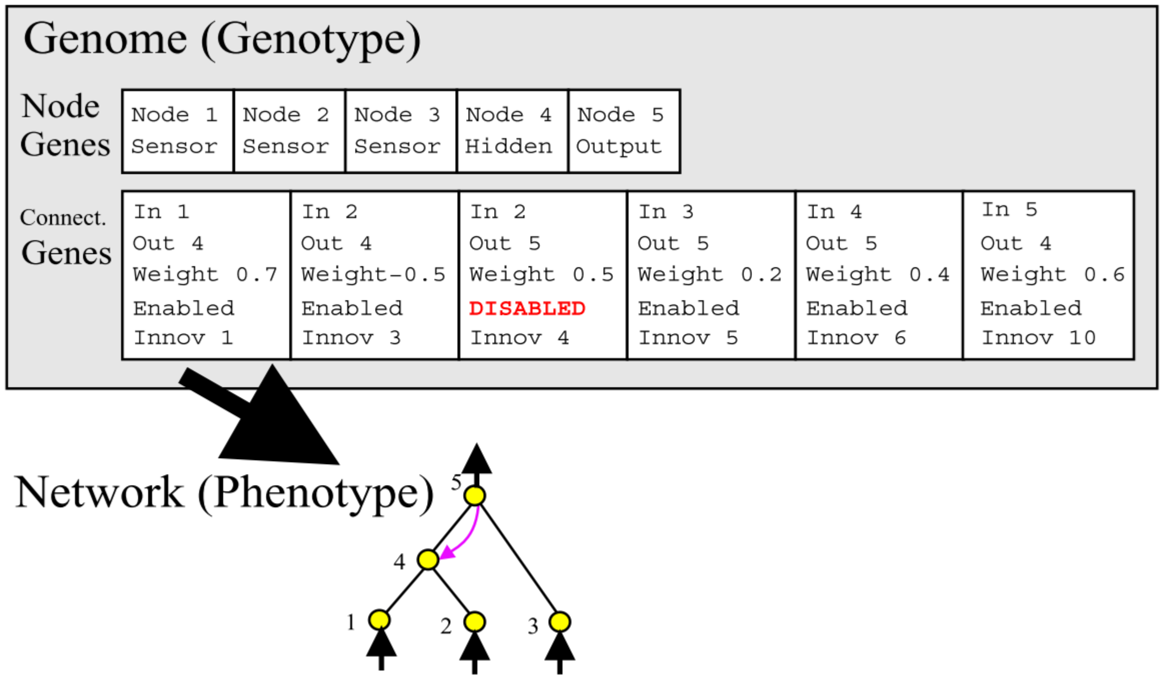

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Hidden
Node 5 Output
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 Enabled Innov 3
In 2 Out 5 Weight 0.5 DISABLED Innov 4
In 3 Out 5 Weight 0.2 Enabled Innov 5
In 4 Out 5 Weight 0.4 Enabled Innov 6
In 5 Out 4 Weight 0.6 Enabled Innov 10
Network (Phenotype)
5
4
1
2
3

Figure 2.5: Genetic Representation in NEAT. Genomes are represented as lists of node and connection genes, identifying properties of each node and connection. The top list shows the genome, and the bottom figure shows the network produced from the genome. Image recreated from [27]

Genomes in NEAT are represented as lists of node and connection genes, shown in Figure 2.5. Node genes contain the identifier of the Node, and its function. Connection genes hold their input/output nodes, their weights, a Boolean "enabled" flag for whether it is enabled or disabled, and an innovation number. Recurrent connections are allowed in this representation.

2.4.2 Genetic Operators

Genetic operators are mutations performed on the genomes and fall under two categories, structural mutations and attribute mutations. There are two types of structural mutation, these mutations add nodes and add connections. Figure 2.6 shows an example of adding a connection and adding a node. *Mutate Add Node* also adds the corresponding node to the genome from Figure 2.5. *Mutate Add Connection* randomly selects two unconnected nodes, and adds a connection between the two nodes. Restrictions can be imposed such that the connections are always feed-forward, but this is optional. *Mutate Add Connection* randomly selects a connection and adds a node across it. Technically this forms two new connections, between the previous input node to the new node, and between the new node to the output node. The connection leading into the new node is given a weight of 1, and the outgoing connection is assigned the same value as the connection it replaces. This supposedly minimises the effect of adding a new node.

Attribute mutations change the attributes of node and connection genes such as weights, enabled flag, and activation function of the node. These mutations occur according to probabilities set as hyper-parameters. For example, if the probability of mutating connections is 0.2, then each connection has a 20% chance of being mutated.

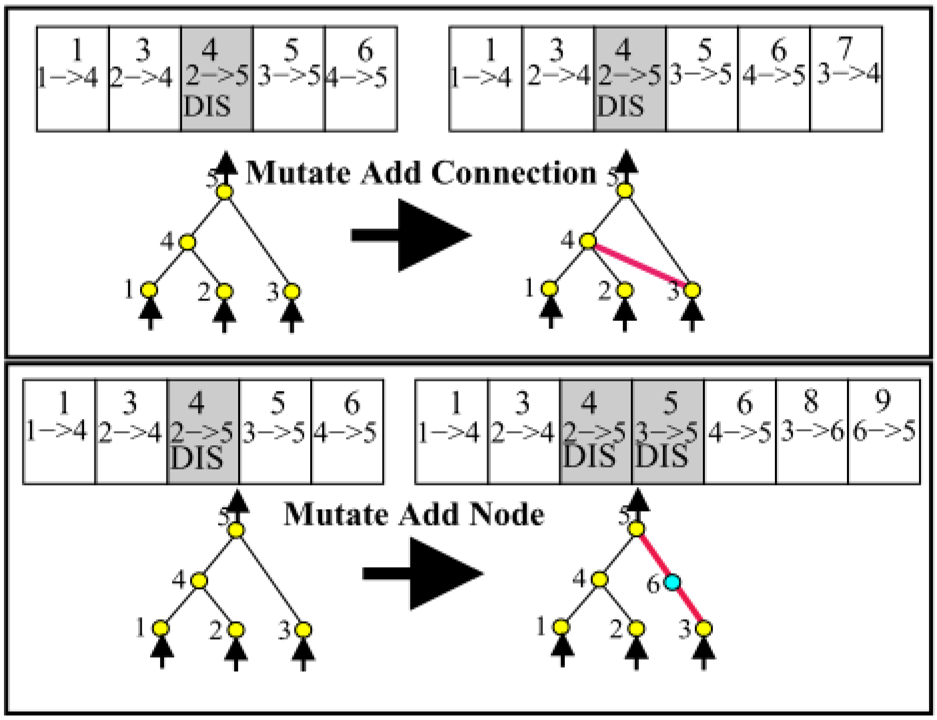


Figure 2.6: The two types of structural mutation in NEAT. Connections are added between previously unconnected nodes. Nodes are added across existing connections, disabling the old connection. The corresponding genome is shown above each network.

The other form of genetic operator is crossover, where parent genomes are combined to create new genomes with properties of the parents. Crossover is shown in Figure 2.7. Genomes are aligned according to their innovation number. Matching genes are inherited randomly, while disjoint and excess genes (genes unique to each parent) are inherited from the fitter parent. The figure assumes parents of equal fitness, hence genes are inherited from both. f

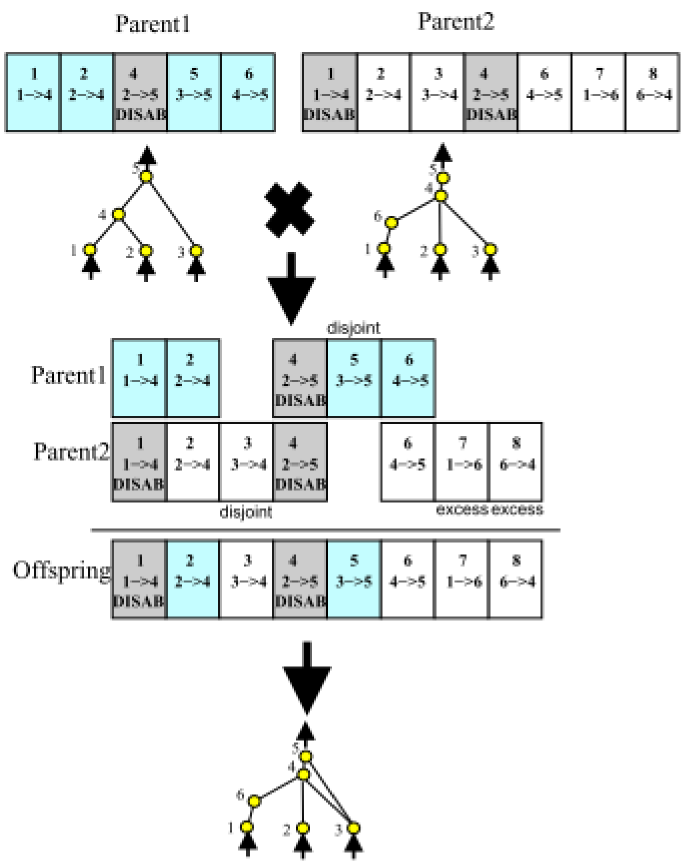


Figure 2.7: Crossover using innovation numbers. Genes are aligned based on innovation number. Matching genes are inherited randomly, while disjoint and excess genes are inherited from the more fit parent. This case shows parents of equal fitness. Image recreated from [27]

2.4.3 Innovation Number and Competing Conventions

All network structures are recorded externally to each genome to track innovation numbers. When a structural mutation occurs, a genome first searches for the structure in the database. If it exists, then the new structure is assigned the same innovation number. If it doesn't exist, the structure is recorded and a new innovation number is assigned.

Perhaps the most important feature of the gene is the innovation number. This is a historical marking that tracks exactly when in evolution a gene is discovered. This solves a problem known as the competing conventions problem, where multiple representations of a genome can represent the same solution but due to problems with interpreting a genome, appear distinct. As seen in Figure 2.8, the two networks are mirror images of each other, but due to their representation, appear as different. If crossover as recombinations of parent genomes are done with Network A and Network B, the result is either $[ABA]$ or $[CBC]$. A degree of information is lost, and the offspring is unlikely to gain benefits from the parents.

Tracking innovation number essentially solves this problem by being able to detect exactly which genes match which. Because innovation numbers are historical markings, indicating when in the mutation it arose, genes of the same innovation number must mean that the genes represent the same structure, although possibly with different attributes. So by lining up genes as described in Figure 2.7, genomes of arbitrary sizes can be meaningfully crossovered.

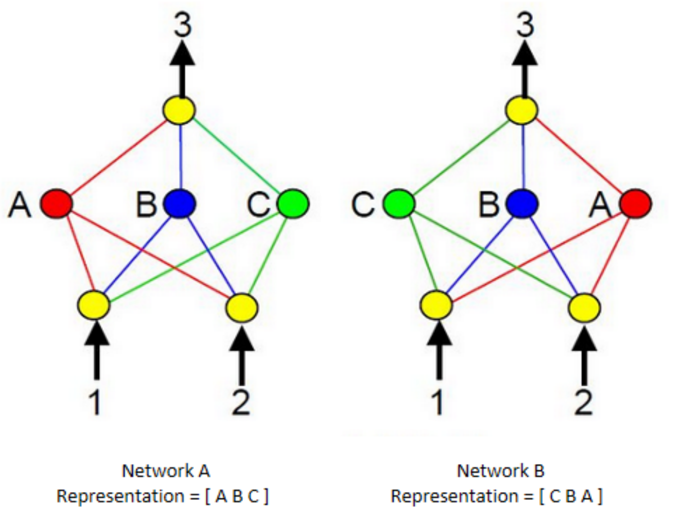


Figure 2.8: Competing Conventions problem. Network A and Network B are fully-connected ANN represented by a list of their hidden nodes. The networks appear to be different due to their representation while being functionally identical. Image recreated from [27].

2.4.4 Speciation

One of the key aspects of GA is maintaining diversity. If only the best structures are allowed to mutate, then the genes which discover locally optimum behaviour earlier on will end up dominating the population. To reduce this from happening, genomes are separated into species based on genetic difference.

A representative is selected for each species, which is used as the sole point of comparison for membership into the species, if the genetic distance is above a particular threshold δ, then it is rejected from the current species and compared to the next. This continues until it is placed in a species or if not placed in any, becomes the representative of a new species. Genetic distance is determined by the formula:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \bar{W} \tag{2.1}$$

Where the distance between two genomes δ is determined by adding the number of excess E and disjoint D genes normalised by the number of genes in the largest of the two genomes, and adding the average weight difference $\bar{W}$. Each term is scaled by their corresponding coefficients c_1, c_2, c_3 set as parameters to indicate the importance of each term.

Once divided into species, genomes are only allowed to reproduce with other members of the species. This allows diversity in the population while protecting structural mutations that may not be immediately viable, but may become viable in the future. The lowest performing members of the species is first eliminated, then replaced by the offpsring of the remaining fitter members. A species is allotted a number of allowed individuals according to the ratio of the species' average fitness over the sum of all species' average fitness.

NEAT has been applied to many aspects of ML. Feature-Selective NEAT (FS-NEAT) is functionally identical to standard NEAT, with the exception of the starting population. Rather than starting with fully connected input/output, networks are initialised with one input connected to one output, placing a bias on the level of connectivity of one input over others [30]. By starting with a bias to one particular input, only networks with useful input will tend to survive. This is shown to be faster than standard NEAT and learns smaller networks. Real-Time NEAT (rtNEAT) evolves networks in real-time by training a group of agents in an on-line fashion. Every n time steps, the weakest genome is replaced with the offspring of fitter genomes while retaining the speciation of standard NEAT [28]. This is demonstrated to work in video game situations where there are groups of agents whose behaviour can be assessed over time. The process is largely invisible to the player, but behaviour is able to change over time in an adaptive manner. Hyper-cube based NEAT (HyperNEAT) uses NEAT to produce Compositional Pattern-Producing Networks (CPPN) which dictate the connectivity and weights between nodes on an underlying substrate (a fixed node and input/output associations) [26]. The CPPN represents a connective *pattern*, and can thus produce ANN of any size by varying the size of the substrate. Some of the applications include image classification as well as image generation. An interesting application of NEAT reduces the search space for NEAT by testing networks on common weights, referred to as shared weights, across all connections [10]. Networks are evaluated each time with all weights being set to a common value. The fitness is obtained as the average performance over each evaluation. The main difference is in the fitness evaluation, and weight setting, other than that the algorithm proceeds the same as regular NEAT. This produces networks with a strong intrinsic bias to solving the problem regardless of weight parameter.

2.5 Multi-Armed Bandit

MAB is a framework for a family of algorithms which selects from available actions in such a way as to maximise rewards. The problems they are applied to are known as sequential decision problems, as the MAB are tasked with making a sequence of decisions to maximise the rewards received. In addition to maximising rewards, MAB are used to balance exploitation and exploration. Exploitation is where the MAB selects the arm with the current optimal arm, the arm with the highest reward, while exploitation is the selection of less optimal actions. The balancing of exploration vs exploitation is to discover the true optimal arms such that it does not become stuck in a sub-optimal selection simply due to bad luck.

The basic model of MAB is a list of possible actions and observed reward. At each time step, the MAB plays an action, known as an arm, based on the current information regarding actions. A reward is received for the action which it then uses to update its reward information. The Bernoulli MAB assumes rewards for actions are 1 with probability p and 0 otherwise. The general MAB does not assume rewards are constrained to any range.

When queried for an action, the MAB assesses and selects an action based on the currently known rewards. Selection mechanisms can range from probabilistic, such as with Boltzmann exploration [13] or deterministic such as Upper-Confidence Bound (UCB) [2]. Thompson Sampling applied to MAB samples the probability of each action resulting in a reward using a beta distribution [1].

The practical applications are in healthcare, finance, recommendation systems and also to enhance ML algorithms [3].

The MAB problem and its solutions are simply mechanisms which aim to exploit the highest rewarding action while maintaining an aspect of exploration. These properties makes it an ideal tool to use in problems with uncertain rewards.

2.6 Adaptive Mutations

Genetic Algorithms by default use fixed probabilities to select mutation and crossover operators. This has the issue in that it relies on initial heuristics and does not adapt with changes to the characteristics of the population. The problem can be tackled by off-line methods such as grid search to adjust parameters externally or between runs. However these methods are very computationally expensive and further adds to the already large computational cost of GA, making it an unattractive option. A preferable method is to adjust parameters on-line such that operator selection is done according to the rewards observed by the algorithm. By moving the operator selection to an on-line scheme, applications of GA essentially become self tuning to the characteristics of the problem.

Adaptive Operator Selection (AOS) is the on-line selection of the best genetic operator, based on rewards received for past selection of operators [9]. The AOS scheme requires a few components. First is measuring effects of operators, second is assigning credit to each operator, and third is using the reward to select the next operators.

To measure the effect of operators, a typical scheme is to compare the fitness of the offspring to its parent. Such as in [32]. Another simpler scheme is also possible where a binary value measuring whether a operator is successful or not in improving fitness over the parent [17].

An early method adjusts the rates of mutation and crossover according to change in fitness of a genome, the best fitness of the population, and the mean fitness of the population [24]. The operator rates are increased when the difference between the best fitness and change to current genome are large, and when the difference between the best fitness and mean fitness is low. This method presumes that a spread in fitness corresponds to wider genetic diversity, which is to some extent true, but it may also be possible that an upper limit to fitness is found.

As far as credit assignment, the most common method is to simply assign the credits to the most recent operator that brought the improvement to fitness. While the sequence of mutations is an undoubtedly important aspect of overall improvement, no clear benefits are found when assigning credits retroactively to the operators applied [9]. Considering only the most recent mutation is typically sufficient.

The last aspect of operator selection is to select the most promising operators to skew mutation to use operators which are more likely to improve fitness, but also to keep using some of the less promising mutations as they may improve as the characteristics of the population changes. This is a case of Exploration vs Exploitation tradeoff, and is ideally tackled by Multi-Armed Bandits, as reviewed above.

An example of the use of MAB is in [6] which handle changes to the reward landscape using UCB in Dynamic Multi-Armed Bandits. The general MAB framework is adapted to handle potential change in rewards by restarting the MAB from scratch when a change is detected. Changes are detected using the Page-Hinkley test on the sequence of previous observations and if the value of the test exceeds some predefined trigger parameter λ, the MAB algorithm restarts and all previously recorded values are discarded. This method can then handle drastic changes to the reward landscape without the delay of upating UCB rewards.

2.7 Adaptive Mutation and NEAT

One issue identified is that the performance of NEAT can be dependent on hyper-parameter settings. As far as the literature reviewed, there is not much work on optimising performance through parameters or on ways to reduce the number of hyper-parameters required. This leaves a lot of guesswork and relying on general heuristics or expensive hyper-parameter tuning which makes NEAT inconsistent and difficult to use. There is however, a body of work on the application of MAB to GA which suggests the possibility of on-line parameter optimisation in NEAT. This motivates the thesis to integrate MAB into NEAT to form Multi-Armed NEAT. It is hypothesised that Multi-Armed NEAT will bring a speed-up to NEAT by using mutations which are more likely to improve fitness while also reducing the number of important parameters to just those of the selected MAB. The idea Multi-Armed NEAT and is further explained in Section 3.

Chapter 3

Models and Application

3.1 Multi-Armed NEAT

The idea of this project is the integration of MAB into the NEAT algorithm/system. this section explains where and how the MAB are integrated into NEAT, replacing the standard mutation and crossover mechanics. This method incorporating MAB and NEAT is known as Multi-Armed NEAT (MA-NEAT). The ideas are based on AOS but does not directly implement any specific algorithm reviewed.

The MAB in MA-NEAT replaces both the Mutation and Crossover functionality of the GA seen in Figure 2.4. Each of the N arms in the MAB corresponds to a mutation operator, referred to interchangeably as arm or action. The possible actions are `add_node`, `mutate_nodes`, `add_connection`, `mutate_connections`, `crossover`. Selecting an action applies the corresponding mutation on the selected genome. In all cases 2 parents are chosen, but if crossover is not selected, then the fittest parent is cloned and mutated. In this application only single-play MABs are considered. Only one mutation is selected for each genome at every time step.

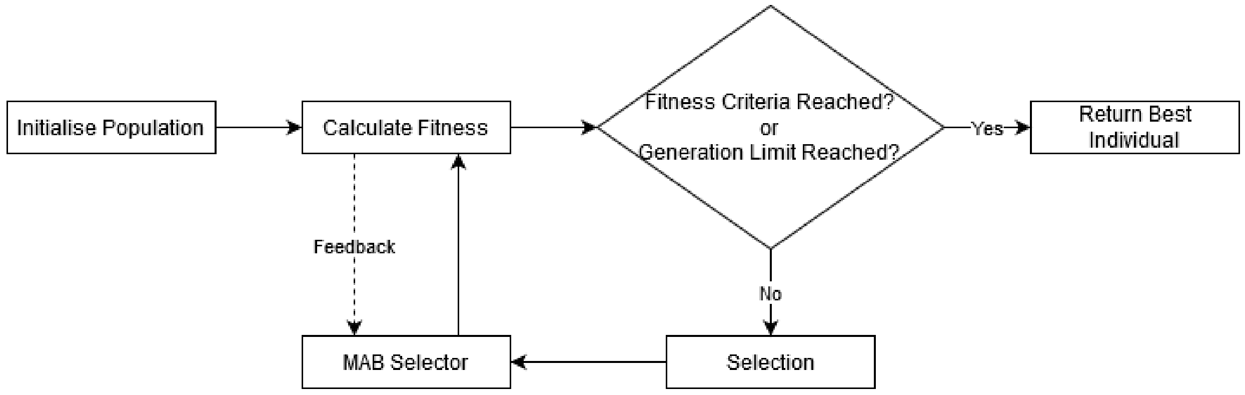


Figure 3.1: Multi-Armed Bandit Integration in NEAT Multi-Armed Bandits replace the previous Mutation and Crossover operations and selectively applies operations based on observed feedback

After being mutated, mutation applied and old fitness are saved for each genome. After evaluating the population, the change to fitness is calculated as the new fitness minus the old fitness. The

MAB is then passed a list containing every application of mutations and the corresponding change in fitness. The MAB then updates itself using the information given.

After the MAB is updated with *all* changes to fitness, then the process of selection, mutation, evaluation, and feedback continues. This system is roughly summarised in Figure 3.1.

All mutations are applied before the start of each generation, then the mutated genomes are evaluated and the change in fitness resulting from each application of a mutation is reported back to the MAB. The result is a batch update, and as such there is a need to select MAB with at least some element of randomness as it is unlikely that applying the same mutation repeatedly results in favourable performance. For this reason, the MAB chosen are Softmax, ϵ-Greedy, and Thompson Sampling. The Static MAB is included as a stand-in for standard NEAT, as it works in exactly the same way as NEAT would without modification.

3.2 MAB Rewards

A central reward scheme is devised for all the selected MAB models. A common aspect between the selected MAB is the use of mean rewards. Although Thompson Sampling seems like an exception, the curves defined by the *Beta* distribution are centered around the mean reward of successful and unsuccessful trials. The central reward scheme is inspired by the idea of *Beta* distributions maintaining positive rewards and negative rewards separately.

In each generation, MABs receive a list of rewards R listing the change in fitness resulting from the actions applied in the previous generation. Each bandit maintains a value of successful rewards $S_i(t)$ and unsuccessful rewards $F_i(t)$ for each arm i obtained up to time step t. Additionally the number of times each action is played $n_i(t)$ is recorded.

The rewards are processed in two ways, numerical and heuristic. Firstly, heuristic rewards estimate the value of a reward by recording if the rewards are positive or negative:

Algorithm 1: Update rule for heuristic rewards

```
foreach $(i, r) \in R(t)$ do
    if $r > 0$ then $S_i(t+1) = S_i(t) + 1$ ;
    else $F_i(t+1) = F_i(t) + 1$ ;
    $n_i(t) = n_i(t) + 1$;
```

Where i is the arm played and r is the reward received for that particular play. An action can be played many times in a round before receiving an update at the end of the generation. For heuristic rewards, this is exactly a count of the number of positive and negative rewards received.

In numerical rewards, the value of the reward is added directly to the (S, F) values of the respective action:

Algorithm 2: Update rule for numerical rewards

```
**foreach** $(i, r) \in R(t)$ **do**
    **if** $r > 0$ **then** $S_i(t+1) = S_i(t) + r$ ;
    **else** $F_i(t+1) = F_i(t) + |r|$ ;
    $n_i(t+1) = n_i(t) + 1$;
```

Where i is the arm played and r is the reward received for that particular play. In order to maintain a ratio of positive to negative rewards, only $|r|$ is kept for unsuccessful trials $F_i(t)$.

The mean reward $\hat{\mu}_i(t)$ for both reward schemes is:

$$\hat{\mu}_i(t) = \frac{S_i(t)}{S_i(t) + F_i(t)} \tag{3.1}$$

and can be described plainly as the portion of changes that are positive.

Softmax and ϵ-Greedyuses the value $\hat{\mu}_i(t)$ directly in calculating which arm to choose, but Thompson Sampling requires a slight modification when using numerical rewards.

Changes have to be made when using $S_i(t)$ and $F_i(t)$ with numerical rewards to Thompson Sampling. MABs with Thompson Sampling expects (S, F) values to be incremented by values in a $[0, 1]$ range , equivalent to α, β in typical notation of the *Beta* function. The other MABs selected do not need modification as they just use the mean reward with no consideration required for its absolute value.

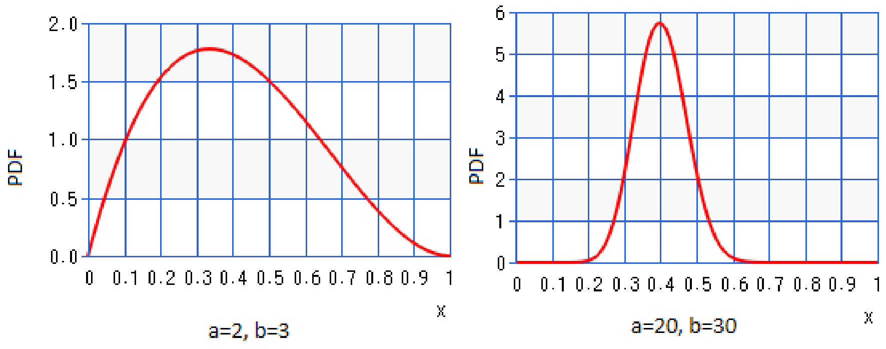


Figure 3.2: Beta distribution with different parameters but same mean. With a=2, b=3 and a=20, b=30. Although representing the same means, with lower a,b values the distributions are wider, representing higher variance and more uncertainty in the mean value. For higher a,b values, the distribution becomes narrower, representing lower variance and higher confidence in the mean value. Graphs generated using https://keisan.casio.com/exec/system/1180573226

Numerical rewards require adaptation as reward values cannot be easily reduced to values within the range [0,1] unless using a sigmoidal function. While possible, a sigmoidal function introduces additional parameters for abstract changes to the system that are difficult to justify. If the raw rewards are used, the variance of the Beta distribution can decrease faster than expected if the rewards lie outside the range [0,1], giving false confidence in the mean. Playing an arm more frequently should result in higher confidence in its mean value, not just because it receives a large reward once. This is demonstrated in Figure 3.2.

Looking back at Thompson Sampling as described in [1], $S_i(t)$ and $F_i(t)$ can be formulated as the number of plays weighted by the mean rewards for each arm:

$$\begin{aligned} S_i(t) &= \hat{\mu}_i(t) \times n_i(t) \\ F_i(t) &= (1 - \hat{\mu}_i(t)) \times n_i(t) \end{aligned} \tag{3.2}$$

This formulation is applied to numerical rewards for Thompson Sampling, representing a weighted count of arms played, weighted by the ratio of positive to negative reward.

3.3 Static

The static bandit chooses actions probabilistically, according to relative weight set by hyperparameters:

$$p_i(t) = \frac{w_i}{\sum_j^N w_j} \tag{3.3}$$

3.4 Softmax

Softmax bandits select arms with probability proportional to its average reward. Arms with greater means are therefore selected more likely to be selected. The particular Softmax method used is Boltzmann Exploration, as described in [13]. Arms are selected according to probability:

$$p_i(t+1) = \frac{e^{\hat{\mu}_i/\tau}(t)}{\sum_j^N e^{\hat{\mu}_j/\tau}(t)} \tag{3.4}$$

The hyperparmeter τ, called the temperature parameter, controls the amount of exploration vs exploitation. As τ approaches 0, the tendency to exploit increases, the tendency to explore increases as τ approaches infinity. Intuitively, τ can be thought of as the parameter that controls exploration, the higher it is the more exploration, the lower it is the less exploration (and more exploitation).

This MAB is the closest to pure adaptive parameter control, as the probability of selecting an action is directly adjusted according to the reward of an action.

3.5 Epsilon-Greedy

In Epsilon-greedy (ϵ-Greedy), each round has an ϵ chance of selecting arms uniformly at random, and a $1 - \epsilon$ chance of selecting the arm i with the greatest mean reward:

$$i(t+1) = argmax_i \hat{\mu}_i(t) \tag{3.5}$$

Exploration is controlled by changing the ϵ parameter. Lower values of ϵ cause the MAB to be more greedy, while higher values causes more exploration. The benefit of exploration/exploitation is up to the problem domain. For example, a domain where a subset of actions are much more likely to results in rewards, a low ϵ might be preferable. Other domains where rewards are more evenly distributed would be better with higher ϵ values.

3.6 Thompson Sampling

Thompson Sampling as described in [1], maintains a *Beta* distribution for each action, recording the "successful" trials S and "unsuccesful" trials F. When selecting an action, a value is sampled from the *Beta* distribution of each arm and the arm with the highest sampled value is selected as the action for that round:

$$\begin{gathered}\theta(t) = [Beta(S_i(t) + 1, F_i(t) + 1 | i \in 1, \ldots, N], \\ i(t+1) = \arg\max_i \theta_i(t)\end{gathered} \tag{3.6}$$

Where $\theta(t)$ is the list of random values sampled from the Beta distribution for each arm. This method is somewhat similar to the Softmax bandit as the probability of an arm being chosen is determined by the mean reward of each arm. The main difference is arms that are played less frequently will have broader Beta distributions, meaning it has a chance of being sampled as the highest rewarded arm, causing more exploration. As more rounds are played, the mean narrows down as the confidence of the arm's values increases. Because of this, later rounds are more likely to exploit as long as the overlap of distribution between arms decreases and separates as the PDF narrows.

Chapter 4

Experiments

In order to test the proposed changes with MA-NEAT, each MAB is tested in different environments with different parameters. The parameters chosen represent a few distinct configurations when using NEAT and/or MAB. For static MAB, the parameters tested represent biases to the possible mutation operators. These are connection-biased, node-biased, and crossover-biased. For other MABs, they are tested with varying degrees of exploration, these are "balanced", greedy, and explore.

The environments chosen are continuous control tasks from Open-AI gym [4], and classification tasks using UCI Machine Learning repository [7]. Continuous control environments cover the tasks typically suited to NEAT, that is reinforcement learning tasks where loss functions may not be well-defined and backpropogation for standard ANN typically cannot be used. Classification is a task where the broader category of ANN is generally applicable, so these tasks are also used to see how applicable NEAT is in classification in addition to continuous control.

4.1 Test Environments

Each test consists of NEAT running for 200 generations or until the specified fitness threshold is reached. This number of generations is chosen as an estimated balance between sufficient time for the algorithm to reach the fitness threshold and time required to run each test. If the fitness criteria is not found, then the MAB models should be able to differentiate from each other to see which one results in higher fitness. NEAT is a non-deterministic algorithm, so each test is run 32 times to obtain an average performance which should be indicative of its overall performance.

4.1.1 Pendulum

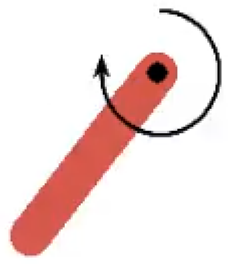


Figure 4.1: Pendulum Environment An agent applies a torque clockwise or counterclockwise to a pendulum with the goal of balancing it in the inverted state. Source: https://gym.openai.com/envs/Pendulum-v0/

Index	Description	Minimum	Maximum
0	Cos(pole angle)	-1.0	1.0
1	Sin(pole angle)	-1.0	1.0
2	Pole angular velocity	-8.0	8.0

Table 4.1: Input features for Pendulum task.

Index	Action	Min	Max
0	Apply Torque	-2.0	2.0

Table 4.2: Output actions for Pendulum task.

Figure 4.1 shows the Pendulum environment from Open-AI gym. An agent tries to balance a pendulum upright in the inverted state by applying torque to pendulum pivot. The pendulum starts at a random angle and a random velocity. The agent is only ever penalised for having the pendulum away from perfectly vertical according to the formula:

$$-(\theta^2 + 0.1 * \omega^2 + 0.001 * action^2) \tag{4.1}$$

Where θ is the angle away from vertical, ω is the angular velocity of the pendulum and *action* is the amount of force applied. A value closer to 0 is preferable. The fitness threshold is set at -1. The agent receives 3 inputs and 1 output. The details can be found below in Table 4.1 and Table 4.2. This is a continuous control task with only one degree of freedom. The agent controls the sign and magnitude of the action performed.

4.1.2 Bipedal Walker

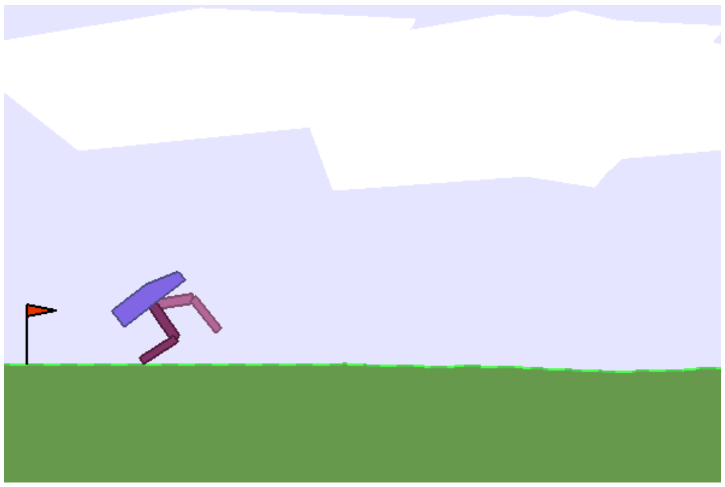


Figure 4.2: Bipedal Walker Environment An agent controls the joints of a robot aiming to move it from the left to the right. Source: https://gym.openai.com/envs/BipedalWalker-v2/

Figure 4.2 shows the Bipedal Walker environment from Open-AI gym. The agent controls a bipedal robot with the goal of moving from the left of the environment to the right by controlling the joints of the robot. The robot starts in a slightly random position at the left of the environment, with legs mostly pointing downwards and the body generally upright. The round finishes when the hull touches the ground or the robot manages to reach the end of the environment within the time period. The agent is rewarded for moving forward, and penalised proportional to the amount of motor force used and also proportional to how tilted the main body is. A full run to the end of the environment receives approximately 300 reward. A penalty of -100 is applied if the hull touches the ground. A common behaviour observed in early testing is for agents to converge on some variant of a kneeling pose, where the hull is supported by the joints with minimal force required to maintain. Penalising this behaviour slightly less than falling should "encourage" agents to walk. Therefore a custom penalty of -80 was added and applied when the agent is detected to be stationary but not fallen over. The fitness threshold is set at 100. The agent receives 24 inputs and 4 outputs. The details can be found below in Table 4.3 and Table 4.4. This is a continuous control task, each output controls the motor force of one joint, and all are applied simultaneously.

Index	Description	Minimum	Maximum
0	Hull Angle	0	2pi
1	Angular Velocity of Hull	-inf	Inf
2	Hull x-velocity	-1.0	1.0
3	Hull y-velocity	-1.0	1.0
4	Hip 1 joint angle	-pi	Pi
5	Hip 1 joint speed	-inf	Inf
6	Knee 1 joint angle	-pi	Pi
7	Knee 1 joint speed	-inf	Inf
8	Leg 1 ground contact	0	1
9	Hip 2 joint angle	-pi	Pi
10	Hip 2 joint speed	-inf	Inf
11	Knee 2 joint angle	-pi	Pi
12	Knee 2 joint speed	-inf	Inf
13	Leg 2 ground contact	0	1
14-23	Lidar readings	0	inf

Table 4.3: Input features for Bipedal Walker task.

Index	Action	Minimum	Maximum
0	Rotate Hip 1	-1.0	1.0
1	Rotate Knee 1	-1.0	1.0
2	Rotate Hip 2	-1.0	1.0
3	Rotate Knee 2	-1.0	1.0

Table 4.4: Output actions for Bipedal Walker task.

4.1.3 Mountain Car

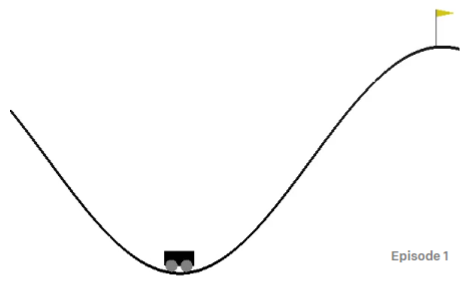


Figure 4.3: Mountain Car Environment An agent controls a car with insufficent power to climb the hill on the right. The goal is to reach the flag at the top of the hill. Source:https://gym.openai.com/envs/MountainCar-v0/

Index	Description	Min	Max
0	Position	-1.2	0.6
1	Velocity	-0.07	0.07

Table 4.5: Input features for Mountain Car task.

Index	Action
0	push left
1	no push
2	push right

Table 4.6: Output actions for Mountain Car Task. Note these are exclusive choices. Only one is chosen at any time, decided by the index with the highest output value.

An agent controls a car that has insufficient power to traverse the hill through one continuous application of power. The agent must alternately push the car backwards and forwards to gain momentum in order to completely scale the hill. The goal is to reach the flag at the top of the hill on the right side. The car starts at the middle of the dip as shown in Figure 4.3 with very slight variation in positioning. The cart is penalised -1 reward for every timestep that it is not in the goal position. However this makes it impossible to distinguish between agents when they are unable to reach the goal. The reward is boosted with the highest absolute velocity reached, difference between the highest and lowest velocity reached, and maximum distance travelled to the right. This should make agents that are able to move faster and further to the right more likely to reproduce, giving a better chance of finding agents that are able to complete the task. Agents receive 2 inputs and 3 outputs, the names of the inputs can be found in Table 4.5 and Table 4.6 The task is a continuous control task with singular actions at each time step.

4.1.4 Banknote Authentication

This task is to classify banknotes as authentic or forged given features extracted from image files. The data consists of features extracted from images a using Wavelet Transform tool. There are 1372 instances in the dataset. Rewards are a sum of the number of correct estimates minus the mean squared error for each data point. As the data is a classifacation task, the outputs are in a one-hot encoding. The reward is to encoura The fitness threshold is set at 1300. The data has 4 features and 2 classes, with the features named in Table 4.7 and Table 4.8. The class with the highest output is selected as the estimated label. The number of each class are approximately equal. This is selected as a simple dataset with fewer number of inputs/outputs. Data sourced from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/banknote+authentication.

Index	Feature	Minimum	Maximum
0	variance of Wavelet Transformed Image	-7.0421	6.8248
1	Skew of Wavelet Transformed Image	-13.7731	12.9516
2	Curtosis of Wavelet Transformed Image	-5.2861	17.9274
3	Entropy of Image	-8.5482	2.4495

Table 4.7: Input features for Banknote Authentication task.

Index	Class
0	Genuine
1	Forged

Table 4.8: Output for Banknote Authentication task.

4.1.5 Wine Quality Classification

This task is to assign a quality rating to wines based on quantitative measurements of a range of wines. The data consists of measurements taken from the wines. The data has 11 features and 6 classes. The features are named in Table 4.9 and Table 4.10. The class with the highest output is selected as the estimated label. The number of each class is heavily unbalanced, with the vast majority being 5 and 6. Data sourced from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/wine

Index	Feature	Min	Max
0	Fixed Acidity	4.6	15.9
1	Volatile Acidity	0.12	1.58
2	Citric Acid	0	1
3	Residual Sugar	0.9	15.5
4	Chlorides	0.01	0.61
5	Free SO2	1	72
6	Total SO2	6	289
7	Density	0.99	1
8	pH	2.74	4.01
9	Sulphates	0.33	2
10	Alcohol	8.4	14.9

Table 4.9: Input features for Wine Quality Classification task.

Index	Class
0	3
1	4
2	5
3	6
4	7
5	8

Table 4.10: Input features for Wine Quality Classification task.

4.2 Additional Parameters

The NEAT package chosen is `neat-python` [15]. It includes many more parameters that may affect the performance of the algorithm. The following is a brief description of the important parameters. NEAT networks are initialised using the FS-NEAT scheme [30]. When the options to mutate nodes or connections is selected by the MAB, it is an indication for the mutations of nodes and connections be carried out with the following probabilities, set as hyper-parameters. The probability of Connection weights being mutated (`weight_mutate_rate`) or replaced (`weight_replace_rate`) is 0.8 and 0.1 respectively. `bias_mutate_rate` and `bias_replace_rate` also function in the same way as weights but for the bias of the node, these are set to 0.8 and 0.1 respectively. Each node can mutate its activation function and bias. `activation_mutate_rate` is the chance that the node changes its activation function, this is set to 0.2. The possible activation functions are listed in the `activation_options` parameter. The compatibility coefficients, `compatibility_disjoint_coefficient` and `compatibility_weight_coefficient`, were set to 1.0 and 0.2 respectively. `neat-python` differs slightly from the literature as `compatibility_disjoint_coefficient` covers both disjoint and excess genes, while the parameter `compatibility_weight_coefficient` is also applied to bias values in addition to weight values, and added if there are different activation function. These values are set to be similar to the parameters set in the initial paper for NEAT [27]. As there were more parameters in nodes than specified in the literature, `compatibility_weight_coefficient` was lowered to accommodate this.

The parameters of the bandits are also altered slightly to test the effects of exploitation vs exploration. Thompson Sampling does not have any parameters to set, so is excluded from parameter testing. There are also parameter schemes where the greediness increases over time, however this is unlikely to be beneficial as diversity of mutations is an important aspect of evolution, furthermore it is shown to have no practical benefit [29], so only static parameters are considered. The parameter τ for the Softmax bandit is tested with 0.05, 0.01 and 1.0 to cover balanced, greedy, and exploration. The parameter ϵ for the ϵ-Greedybandit is set as 0.2, 0.01, and 0.8 to cover the same range as Softmax.

The Static bandit representing base NEAT has parameters `node_add`, `node_mutate`, `connection_add`, `connection_mutate`, `crossover_rate` set to cover a few mutation biases, listed in the same order mentioned. The first is to be connection-biased with parameters $[0.3, 0.1, 0.7, 0.5, 0.1]$, secondly is node-biased with parameters $[0.7, 0.5, 0.3, 0.1, 0.1]$ and lastly increased crossover rate with balanced node and connection parameters $[0.7, 0.3, 0.7, 0.3, 0.5]$.

The Static MAB are named according to their biases, Static-Conn, Static-Node, and Static-Cross, respectively. The MA-NEAT implementations are named after their reward, MABs, and parameter. For example Softmax with heuristic rewards and $\tau = 0.05$ is named H-Softmax($\tau = 0.05$)

The main parameters are listed again in Table 4.11.

MAB Parameters		
MAB	Parameters	Purpose
Static	$rates = [0.3, 0.1, 0.7, 0.5, 0.1]$	Connection Biased
	$rates = [0.7, 0.5, 0.3, 0.1, 0.1]$	Node Biased
	$rates = [0.7, 0.3, 0.7, 0.3, 0.5]$	Increased Crossover
Softmax	$\tau = 0.05$	Balanced
	$\tau = 0.01$	Greedy
	$\tau = 1.0$	Explore
ϵ-Greedy	$\epsilon = 0.2$	Balanced
	$\epsilon = 0.01$	Greedy
	$\epsilon = 0.8$	Explore
Thompson Sampling	N/A	N/A

Table 4.11: Parameters used for MAB in tests. Additionally, each MAB is tested with Heuristic and Numerical rewards.

4.3 Performance Metrics

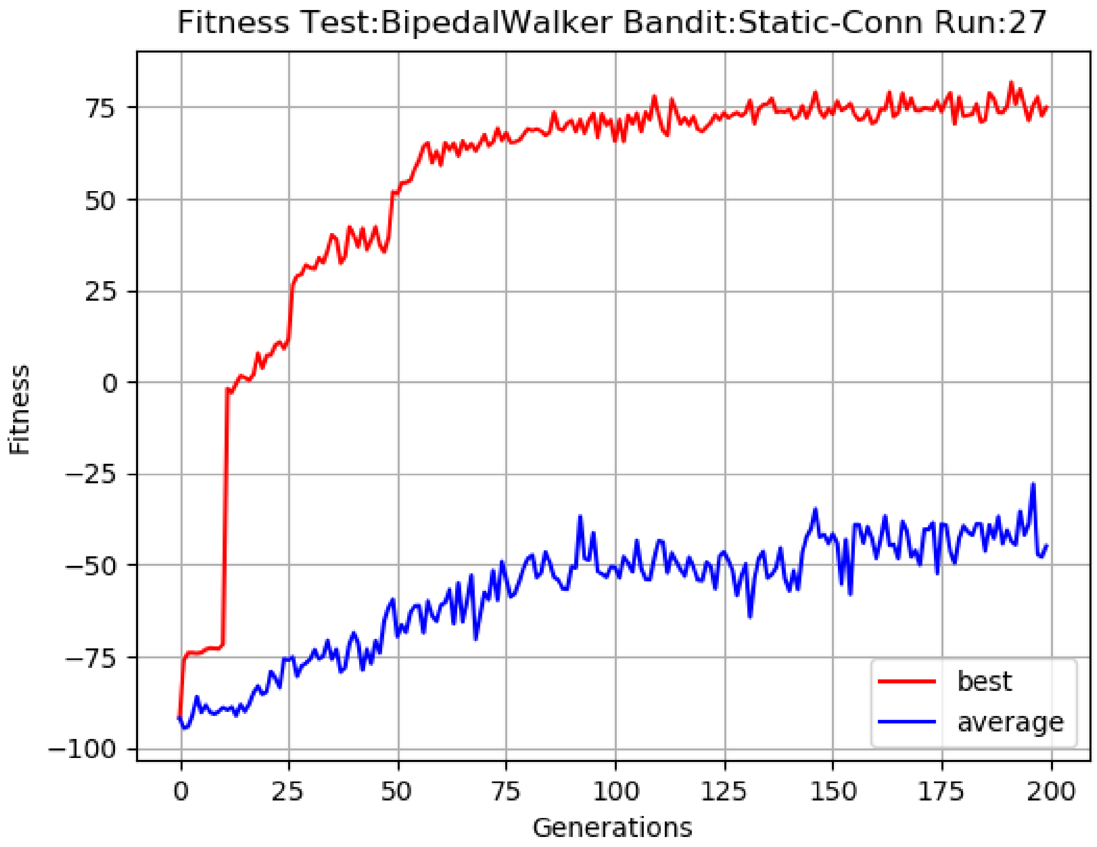


Figure 4.4: Fitness per Generation graph from one run of a test. On the x-axis is the generation of the test, and the y-axis is the fitness at that generation. This shows the improvement of the popultion over time.

Each run of a test produces a fitness per generation curve as shown in example in Figure 4.4. Only the best fitness is used for performance analysis, shown as the red line (top line). In this case the fitness is approximately 75 and the generations is 200. Data collected from the experiments are the number of generations to termination and the fitness reached at termination. These two figures are referred to as the performance metrics.

Gathering these results for each run of a test gives 32 data points for each test, which can be plot on a scatter plot with generations as the x-value and fitness as the y-value.

Chapter 5

Results and Discussion

5.1 Statistical Testing

To consider statistical difference, one-tailed Mann-Whitney U Test is chosen with different H_0 for each metric of generations and fitness. To be considered better, the MA-NEAT models should have a lower distribution for generations and a higher distribution for fitness.

Therefore, for generations:

$$\begin{aligned} H_0(\mathit{Generations}) &: MAB_i \geq MAB_j, i \neq j \\ H_1(\mathit{Generations}) &: MAB_i < MAB_j, i \neq j \end{aligned} \tag{5.1}$$

and for fitness:

$$\begin{aligned} H_0(Fitness) &: MAB_i \leq MAB_j, i \neq j \\ H_1(Fitness) &: MAB_i > MAB_j, i \neq j \end{aligned} \tag{5.2}$$

To consider one model as better than another, we want to reject H_0 for both metrics. The main concern however, is still with Generations, so rejecting $H_0(\mathit{Generations})$ while not rejecting $H_0(Fitness)$ is still acceptable.

One characteristic of the one-tailed Mann-Whitney U test is that reversing the inputs reverses the inequality, and the new P-value P' is $1-P$. Therefore, $P >= 0.95$ is is equivalent to $P' <= 0.05$ with reversed input compared to P. This is used in comparing the MAB to determine the direction of statistical difference when comparing any arbitrary pair of MAB.

5.2 Results Analysis

Each test is reported separately and contains 3 parts. The first is a qualitative look at the scatter plot of terminal number generations and fitness achieved of each MAB. Second is the box plots showing the distribution of each metric for each MAB separately. Lastly is a summary of the full pairwise comparison of the MABs.

The full pairwise comparisons can be found in the Appendix from Table 7.1 to Table 7.5. They were deemed too cumbersome to display in the main text but are included for reference. But in terms of comparisons, the summaries are sufficient. The full pairwise tests record the number of times an MAB is statistically better than others, and the number of times it is statistically worse. The symmetry of the Mann-Whitney U test is helpful as we only need one comparison to determine which MAB is better. This allows each metric to occupy half the amount of space. The upper right triangle indicates the comparisons for fitness and the bottom left triangle indicates the comparisons for generations. The diagonal is comparing an MAB to itself so is useless.

Counting the number of times each MAB is better than others gives each MAB a ranking used to discuss which one has the best performance. The pairwise comparison can be imagined as a hierarchy for the MAB. If MAB A outperforms MAB B, then MAB A must also outperform all the MAB that MAB B outperforms. Thus, a fairly strict hierarchy can be made by ranking the MAB by the number of times it outperforms other MAB.

5.2.1 Pendulum

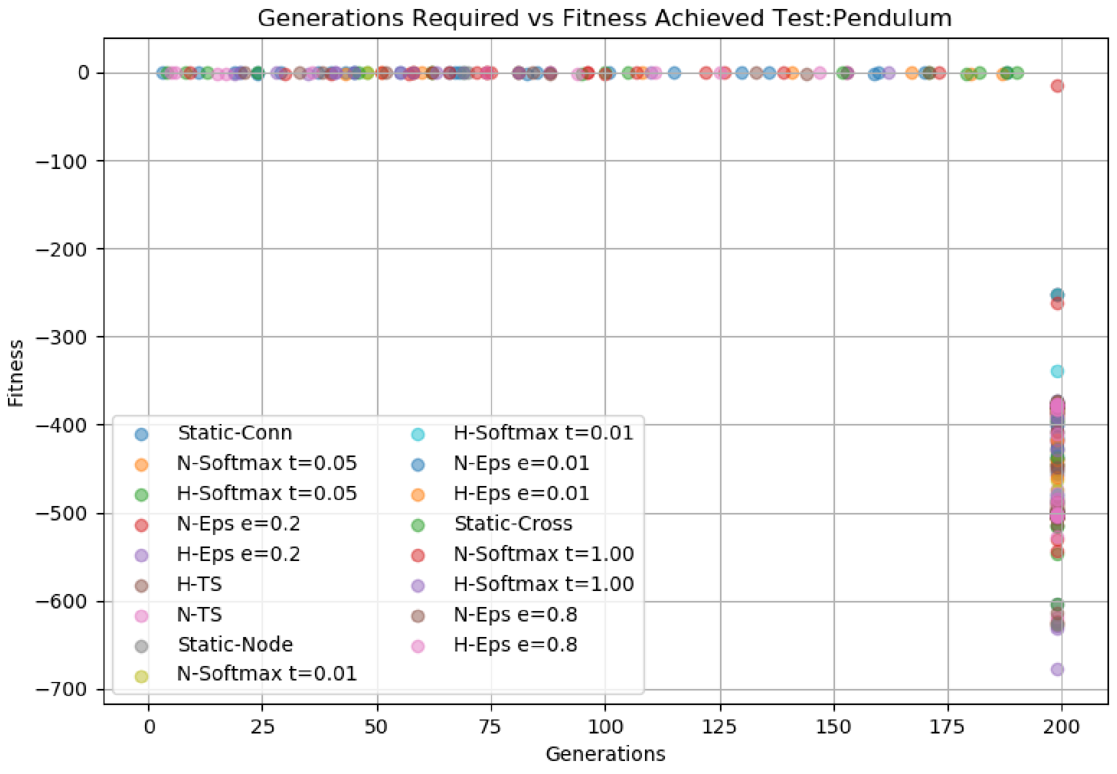


Figure 5.1: Generations vs Fitness Achieved for all MAB in Pendulum environment. Each point represents the generations required to reach the terminal fitness. The x-axis is the number of generations required and the y-axis is the fitness achieved for each run.

The Pendulum environment is a small problem in terms of input/output actions. Despite this, not all MAB were able to find solutions that reach the fitness threshold consistently. Figure 5.1 shows that there is a wide spread of generations required to reach the fitness threshold. The noticeable "wall" on the right of the figure indicates many MAB were unable to find a solution that is able to reach the fitness threshold. The large divide in fitness between runs that are able to find a solution and ones that aren't indicate that this problem is one that tends to be "suddenly" solved. More generations may be required to guarantee all runs achieve the fitness thresholds.

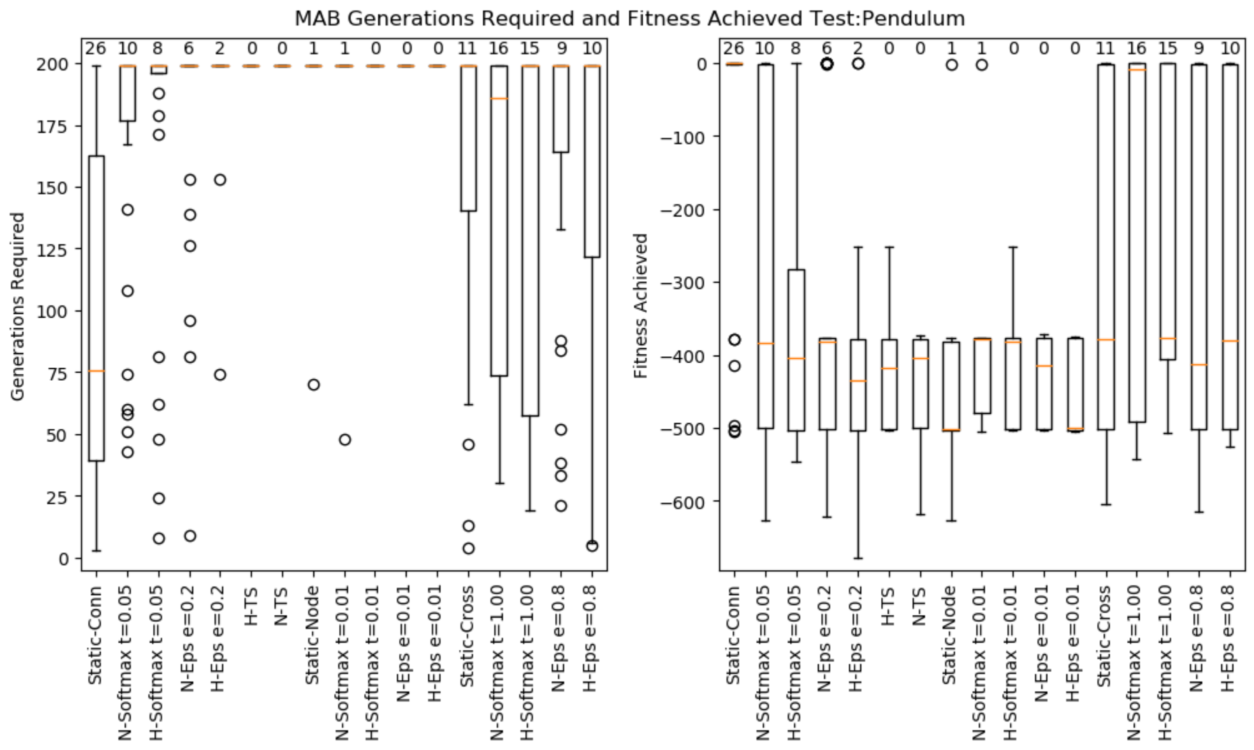


Figure 5.2: Box and whisker plot for Pendulum Test. Outliers are included as circles. Shows the expected range of performance of each MAB with respect to each metric individually. At the top of each box plot is the number of times the MAB reaches the threshold fitness. The left plot is generations and the right plot is fitness.

Looking at the box plots in Figure 5.2, it is immediately noticeable that the Static-Conn MAB, representing static NEAT, reaches the fitness threshold more frequently than others as shown by the figures above the box plots. The fitness is almost entirely concentrated at the top as almost all runs reached the fitness threshold, with only a few outliers not reaching the threshold. The generations is also noticeably lower than others, meaning a faster time to find a solution.

The next note worthy MAB is N-Softmax($\tau = 1.00$) and H-Softmax($\tau = 1.00$), intended to test the importance of exploration in MAB. The fact that it performs well especially in comparison to the greedier parameters of the other MAB suggests that exploration is an important factor when using the MAB in NEAT. The importance of exploration is also supported by the fact that the Softmax with greedier parameters ($\tau = 0.01$) were not able to consistently find solutions. ϵ-Greedyis also more likely to find solutions given less greedy parameters compared to the greedy parameters, but not by a large margin.

Summary of Pendulum pairwise comparisons						
	Generations			Fitness		
MAB Name	n Better	n Neutral	n Worse	n Better	n Neutral	n Worse
Static-Conn	16	0	0	16	0	0
H-Softmax t=1.00	12	3	1	11	4	1
N-Softmax t=1.00	12	3	1	10	5	1
N-Softmax t=0.05	8	5	3	2	13	1
N-Eps e=0.8	8	5	3	2	12	2
Static-Cross	8	7	1	2	13	1
H-Eps e=0.8	8	7	1	3	11	2
H-Softmax t=0.05	8	5	3	1	13	2
N-Eps e=0.2	7	6	3	1	12	3
Static-Node	0	7	9	0	2	14
N-TS	0	7	9	1	12	3
H-Softmax t=0.01	0	7	9	2	11	3
N-Eps e=0.01	0	7	9	1	12	3
H-Eps e=0.01	0	7	9	0	7	9
H-TS	0	7	9	1	12	3
H-Eps e=0.2	0	8	8	0	11	5
N-Softmax t=0.01	0	7	9	3	10	3
Total bandits	17					

Table 5.1: Summary of pairwise comparison between MAB for Pendulum test , sorted by n Better Generations. Counts the number of times one MAB is statistically better, no different, or worse than another for both metrics compared separately. A higher figure for n Better and lower figure for n Worse is favourable.

Table 5.1 shows the number of times an MAB is statistically better than other MABs. This suggests that Static-Conn is better in all aspects, being statistically better compared to all other bandits. The MAB with the second best generations and fitness, N-Softmax($\tau = 1.00$) and H-Softmax($\tau = 1.00$), is also statistically better than most other bandits in both metrics although to a lesser degree.

There is a after N-Eps($\epsilon = 0.2$), although arguably it should not be included in this list as it is only better due to outliers, and thus the central tendency of its performance is equally as bad as the others. This is noticeable particularly when looking at figure 5.2. The other MAB were able to find solutions more consistently. The feature that divides the two levels of performance is primarily their parameters, with the more greedy parameters resulting in low performance.

5.2.2 Bipedal Walker

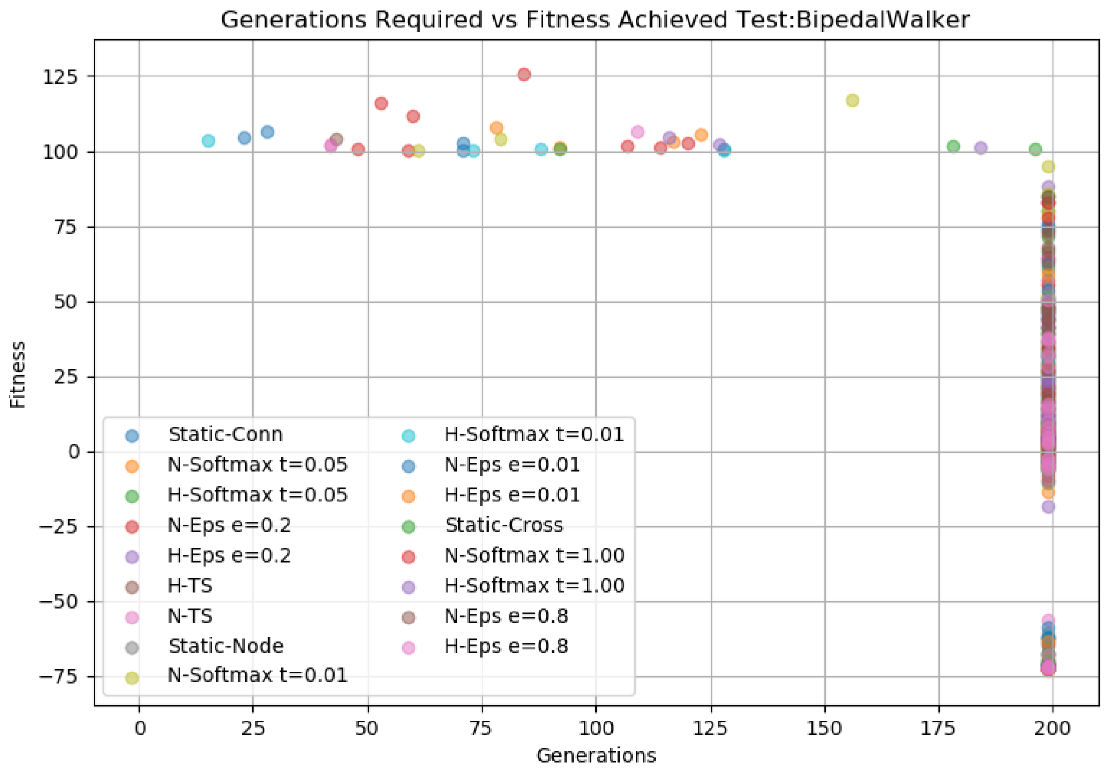


Figure 5.3: Generations vs Achieved Fitness for all MAB in Bipedal Walker environment. Each point represents the generations required to reach the terminal fitness. The x-axis is the number of generations required and the y-axis is the fitness achieved for each run.

The Bipedal Walker is a more challenging environment to solve in a shorter time, as shown in Figure 5.3 by the significant wall at the final generation. However, some MAB were able to reach the fitness threshold in some runs, and many runs appear to reach a positive fitness, meaning most of the networks found have at least learnt to not stand still. However, there is also a noticeable gap at the lower end of fitness. This can be identified as a local optimum where the networks found have learnt not to fall, but have yet not yet learned to move. From earlier inspection of the environment, a completely stable position can be found using only very little motor input. The gap in fitness suggests many many networks were able to exploit this position and were unable to move out of it.

As the terrain does not change much, once an agent learns to move forward while maintaining upright, it is unlikely to fall over until the time limit is reached. The main way of reaching higher fitness before time expires is to move faster. However, even learning to walk is a difficult task that takes many generations, let alone walking fast. Thus the test should be run for many more generations to allow better separation between MAB.

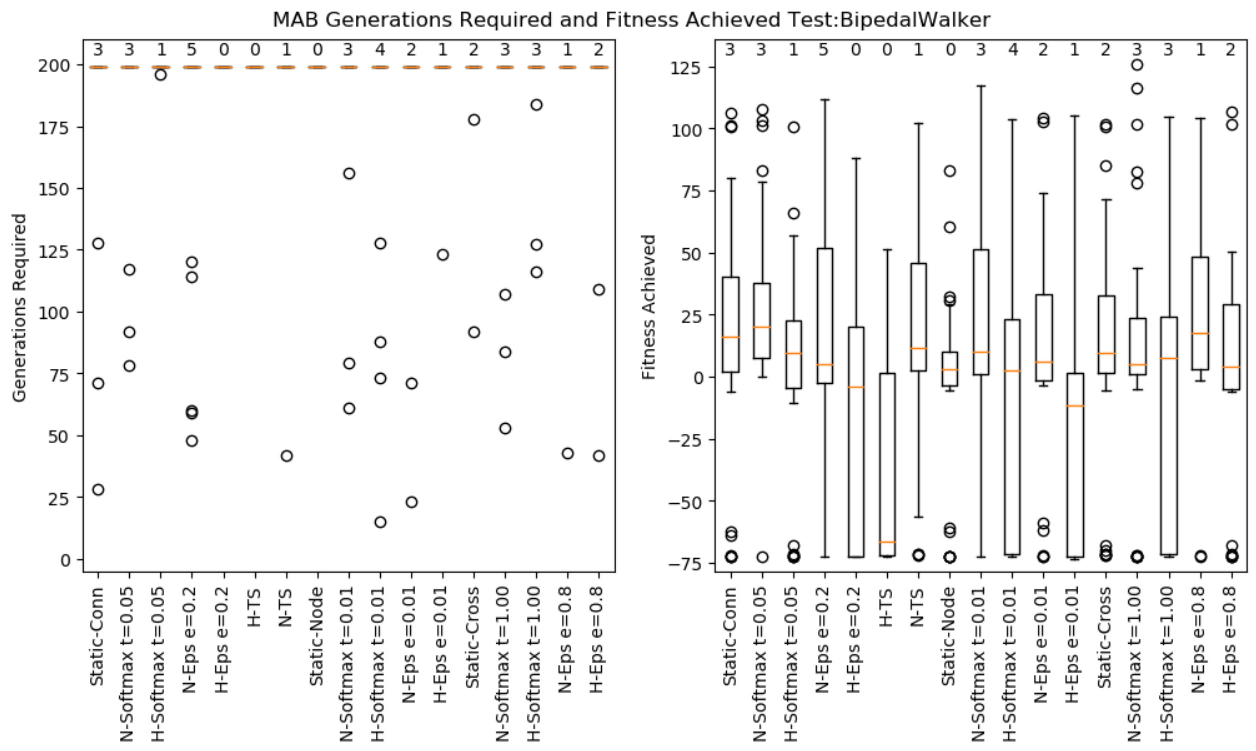


Figure 5.4: Box and whisker plot for Bipedal Walker Test. Outliers are included as circles. Shows the expected range of performance of each MAB with respect to each metric individually. At the top of each box plot is the number of times the MAB reaches the threshold fitness. The left plot is generations and the right plot is fitness.

The fitness threshold was reached by a few MAB on some instances, but those are noticeably outliers, as shown in Figure 5.4. The comparison then should focus on fitness. Firstly, the minimum fitness for all MAB is around -75, corresponding to the penalty of standing still plus some reward of moving forward. Being able to maintain a stable position is arguably an important stepping stone to being able to walk, as the positions tend to be some variant of kneeling. With a bit of additional input, the position can be converted to walking. This represents a significant local optimum which agents can optimise minimally. There is however a noticeable gap between the minimum score and the next lower score of 0 for most of the MABs. This just signifies that the next best solution was to move just slightly as to avoid standing still and avoid the penalty for being stationary. This is a bit of reward hacking on the part of the MAB and can be reduced by increasing the minimum speed required, and also penalising being stationary at the same rate as falling over. Looking at this it is difficult to determine which MABs perform better, but is easy to see which ones perform worse. All the MAB with distributions significantly below 0 fitness can be considered worse. These are mostly the more greedy MABs, as it occurs instead as outliers for the less greedy MAB rather than part of the main distribution.

A wide range of fitness was encountered, with almost all MABs producing one run with the minimum fitness of being stationary, and also many producing fairly high fitness almost reaching the fitness threshold. Based on these results, more generations are needed to observe the proper solving capabilities of each MAB.

Summary of Bipedal Walker pairwise comparisons						
	Generations			Fitness		
MAB Name	n Better	n Neutral	n Worse	n Better	n Neutral	n Worse
N-Softmax t=0.05	3	13	0	11	5	0
N-Eps e=0.2	7	9	0	3	13	0
N-Softmax t=0.01	3	13	0	5	11	0
N-Eps e=0.8	0	15	1	7	9	0
Static-Conn	3	13	0	4	12	0
N-Softmax t=1.00	3	13	0	3	12	1
H-Softmax t=0.01	3	13	0	2	10	4
H-Softmax t=1.00	3	13	0	2	12	2
N-TS	0	15	1	5	11	0
N-Eps e=0.01	0	16	0	4	11	1
Static-Cross	0	16	0	4	11	1
H-Eps e=0.8	0	16	0	2	12	2
H-Softmax t=0.05	0	15	1	2	13	1
Static-Node	0	9	7	2	7	7
H-Eps e=0.01	0	15	1	0	2	14
H-Eps e=0.2	0	9	7	0	7	9
H-TS	0	9	7	0	2	14
Total bandits	17					

Table 5.2: Summary of pairwise comparison between MAB for Bipedal Walker test , sorted by sum of n Better Generations and n Better Fitness. Counts the number of times one MAB is statistically better, no different, or worse than another for both metrics compared separately. A higher figure for n Better and lower figure for n Worse is favourable.

The pairwise comparisons give the number of times one MAB is statistically better when compared to other MAB. N-Eps($\epsilon = 0.2$) is the best in terms of generations simply by virtue of having the most number of runs which reach the fitness threshold. However the actual fitness achieved on other runs brings its rating down when comparing fitness. Furthermore the number of times the threshold is reached is largely insignificant as it was not consistent enough.

N-Softmax($\tau = 0.05$) performs the best in fitness compared to all other MABs with the next best being H-Eps($\epsilon = 0.8$). However, despite there being statistical difference, the magnitude of the improvement over others is actually minimal. This is visually apparent in the plots in Figure 5.4.

5.2.3 Mountain Car

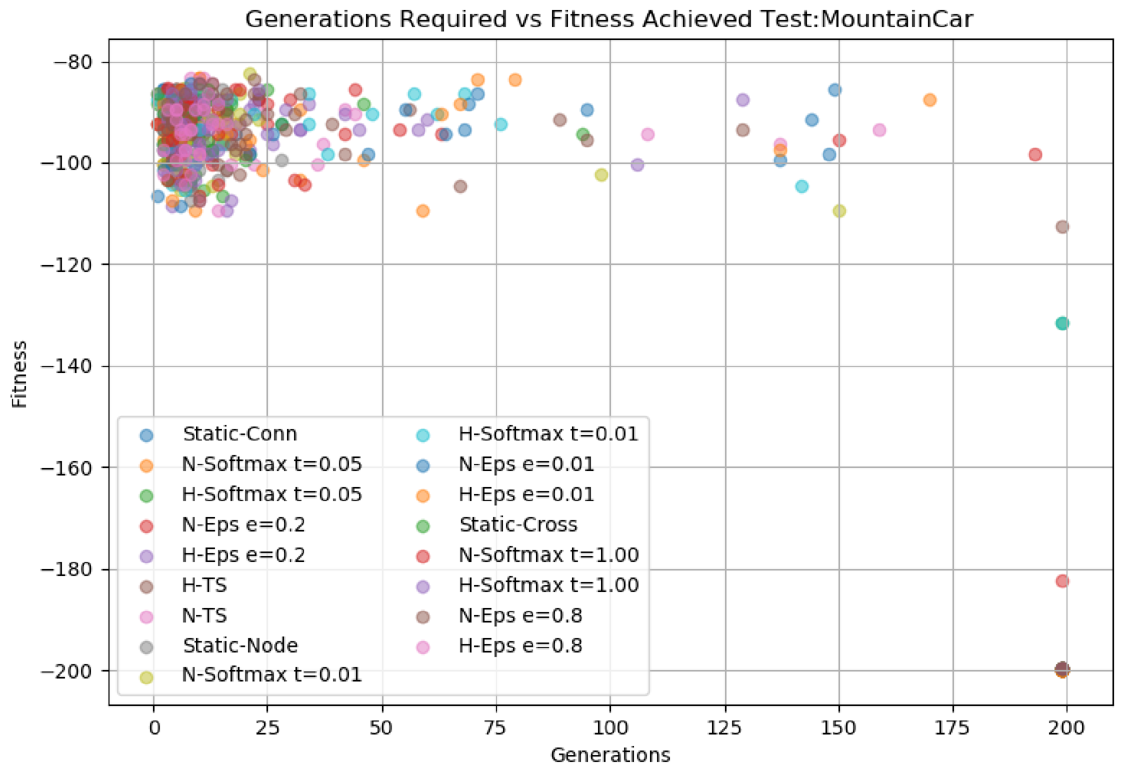


Figure 5.5: Generations vs Achieved Fitness for all MAB in Mountain Car environment. Each point represents the generations required to reach the terminal fitness. The x-axis is the number of generations required and the y-axis is the fitness achieved for each run.

The environment here is clearly simple enough where a large portion of MAB were able to find solutions very quickly, and the vast majority were able to find solutions given more time. The generations are heavily clustered around the earlier generations with a long tail towards later generations. There is also some spread in the final fitness achieved. A few outliers exist which did not manage to solve the environment, but these are very rare compared to the number which managed to find solutions. Importantly it is not the same MAB that consistently failed at finding a solution, meaning poor performance is due to randomness in the environment and initial conditions rather than inherent to an MAB itself.

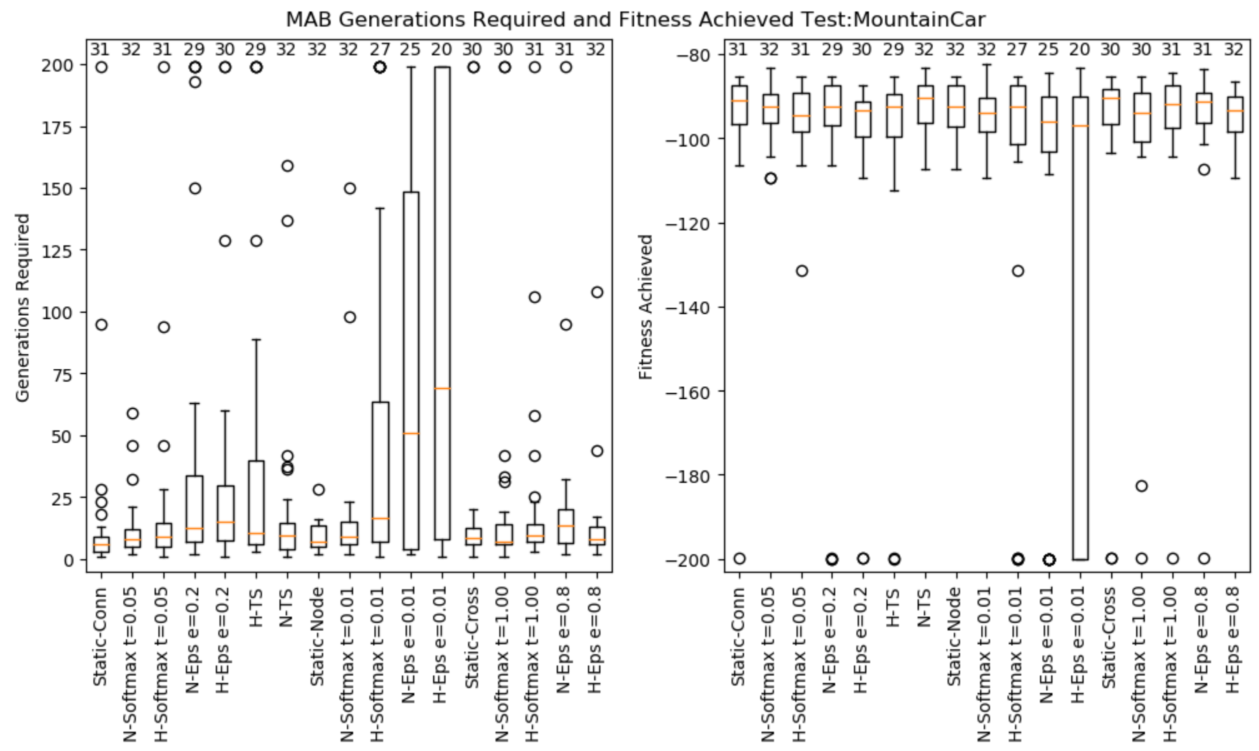


Figure 5.6: Box and whisker plot for Mountain Car test. Outliers are included as circles. Shows the expected range of performance of each MAB with respect to each metric individually. At the top of each box plot is the number of times the MAB reaches the threshold fitness. The left plot is generations and the right plot is fitness.

Perhaps unsurprisingly from the scatter plot, most of the MAB have very similar box plots meaning their performances is very similar. The noticeable exceptions are the the generations of N-Eps($\epsilon = 0.01$), H-Eps($\epsilon = 0.01$) and maybe H-Softmax($\tau = 0.01$) MABs. The MAB still manage to find a solution most of the time however, so it is simply less consistent compared to other MAB. Static-Conn MAB appears to have the tightest distribution but still has a few outliers. The statistical comparisons should lead to more definitive conclusions.

Summary of Mountain Car pairwise comparisons						
	Generations			Fitness		
MAB Name	n Better	n Neutral	n Worse	n Better	n Neutral	n Worse
Static-Conn	11	5	0	4	12	0
Static-Node	8	8	0	3	13	0
N-Softmax t=0.05	7	9	0	1	15	0
N-Softmax t=1.00	7	9	0	1	14	1
Static-Cross	6	9	1	4	12	0
H-Eps e=0.8	6	9	1	0	13	3
H-Softmax t=0.05	4	12	0	1	14	1
N-TS	4	12	0	6	10	0
N-Softmax t=0.01	3	12	1	1	15	0
H-Softmax t=1.00	2	12	2	3	13	0
N-Eps e=0.8	2	8	6	3	13	0
N-Eps e=0.2	1	8	7	1	15	0
H-Eps e=0.2	1	8	7	0	10	6
H-TS	1	11	4	1	15	0
H-Softmax t=0.01	0	6	10	1	15	0
N-Eps e=0.01	0	6	10	0	10	6
H-Eps e=0.01	0	2	14	0	3	13
Total bandits	17					

Table 5.3: Summary of pairwise comparison between MAB for Mountain Car test , sorted by n Better Generations. Counts the number of times one MAB is statistically better, no different, or worse than another for both metrics compared separately. A higher figure for n Better and lower figure for n Worse is favourable.

Static-Conn is the best performing MAB in this test but by a small margin. The next one is Static-Node, also a static MAB. While the results are statistically significant, the distributions are also very narrow. Meaning that very slight variations result in statistical significance. Both ϵ-GreedyMABs with greedy parameters performed considerably worse than other MAB. Fitness is expected to be fairly evenly distributed as there is not much chance to exceed the threshold.

5.2.4 Banknote Authentication

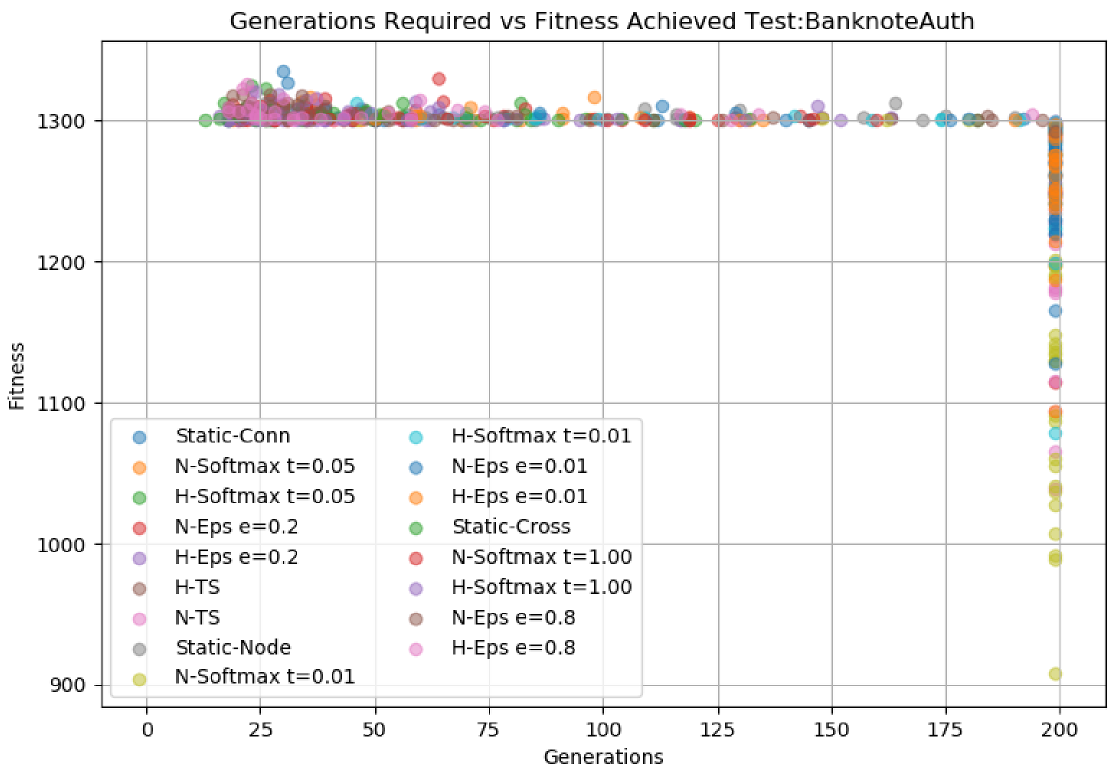


Figure 5.7: Generations vs Achieved Fitness for all MAB Banknote Authentication classification. Each point represents the generations required to reach the terminal fitness. The x-axis is the number of generations required and the y-axis is the fitness achieved for each run.

A decent portion of the runs were able to reach the fitness threshold, but many were unable to. From initial inspection it seems that N-Softmax($\tau = 0.01$) had the lowest fitness overall, and is unable to consistently find a solution. However not much can be said about other MAB as the points are too tightly clustered together. This suggests that the generation limit could be increased to ensure a solution is found, but otherwise is an appropriate limit to determine which MAB were able to find solutions in time. A random baseline is expected to get a fitness of around 760 (out of 1373 total possible fitness), corresponding to an accuracy of 55.5%

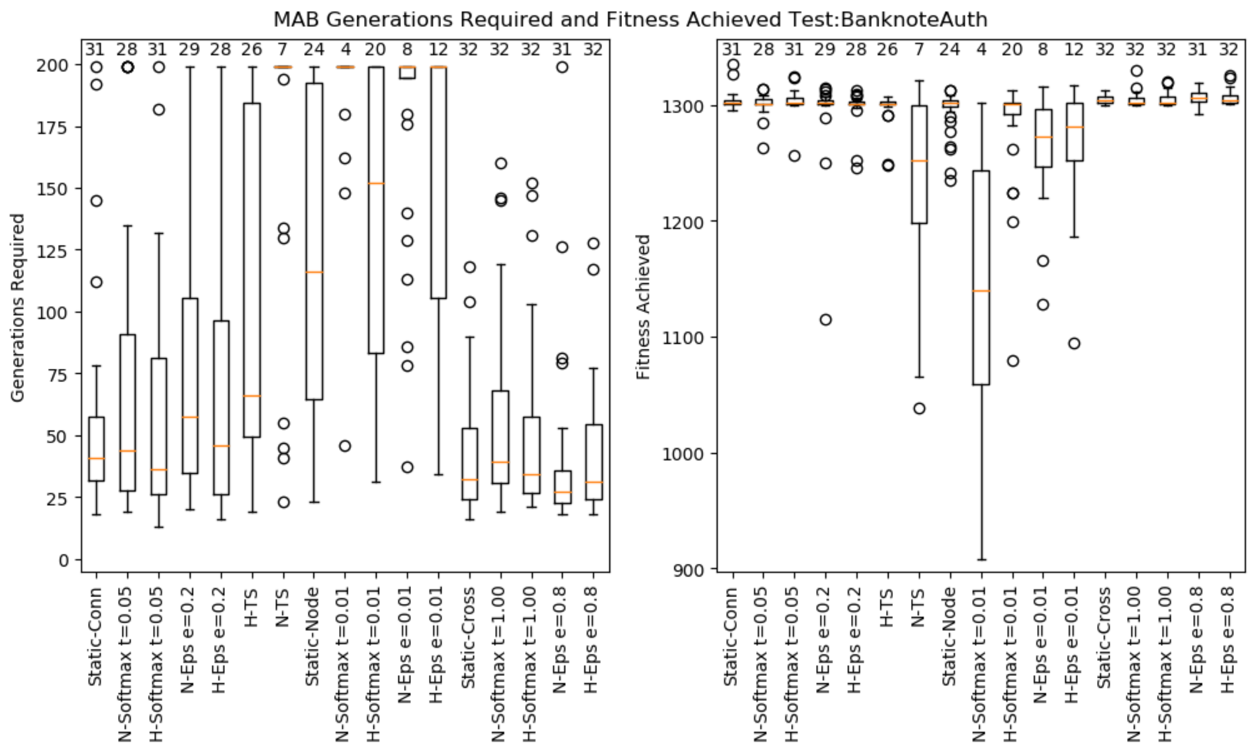


Figure 5.8: Box and whisker plot for Banknote Authentication test. Outliers are included as circles. Shows the expected range of performance of each MAB with respect to each metric individually. At the top of each box plot is the number of times the MAB reaches the threshold fitness. The left plot is generations and the right plot is fitness.

As most MAB were able to find solutions, the main interest is in the number of generations required. N-TS, N-Softmax($\tau = 0.01$), and N-Eps($\epsilon = 0.01$) were the only MABs clearly unable to find solutions which is shown by a much lower n threhold count above the boxplot of the associated MAB. H-Eps($\epsilon = 0.01$) is also similar to some extent but not quite as extremely as the others.. The fitness plots also reflect the inability for those MAB mentioned to consistently find solutions. N-Softmax($\tau = 0.01$) in particular has the widest spread of fitness values. However, all fitnesses are above the baseline, so even the worst classifiers are able to surpass random guessing.

Summary of Banknote Authentication pairwise comparisons						
	Generations			Fitness		
MAB Name	n Better	n Neutral	n Worse	n Better	n Neutral	n Worse
N-Eps e=0.8	14	2	0	15	1	0
H-Eps e=0.8	12	4	0	13	3	0
Static-Cross	10	6	0	11	4	1
H-Softmax t=0.05	8	7	1	6	8	2
H-Softmax t=1.00	8	7	1	8	7	1
N-Softmax t=0.05	7	6	3	5	8	3
N-Softmax t=1.00	7	7	2	7	7	2
Static-Conn	7	7	2	5	8	3
H-Eps e=0.2	7	6	3	4	7	5
N-Eps e=0.2	6	5	5	5	8	3
H-TS	5	2	9	4	6	6
Static-Node	4	2	10	4	8	4
H-Softmax t=0.01	3	2	11	3	4	9
H-Eps e=0.01	1	3	12	2	2	12
N-Eps e=0.01	0	3	13	1	2	13
N-TS	0	3	13	1	1	14
N-Softmax t=0.01	0	2	14	0	0	16
Total bandits	17					

Table 5.4: Summary of pairwise comparison between MAB for Banknote Authentication test , sorted by n Better Generations. Counts the number of times one MAB is statistically better, no different, or worse than another for both metrics compared separately. A higher figure for n Better and lower figure for n Worse is favourable.

In this test, N-Eps($\epsilon = 0.8$) is statistically the best for both generations and fitness, followed closely by the Heuristic version H-Eps($\epsilon = 0.8$). Interestingly, being statistically better in generations also corresponds to being statistically better in fitness, meaning the bandits which are able to find solutions faster also find fitter solutions. This is visually noticeable in Figure 5.7 as the slight "hump" in the early generations. This is also the second time an ϵ-GreedyMAB was able to top the pairwise comparisons, and this time in a much more significant manner.

5.2.5 Wine Quality

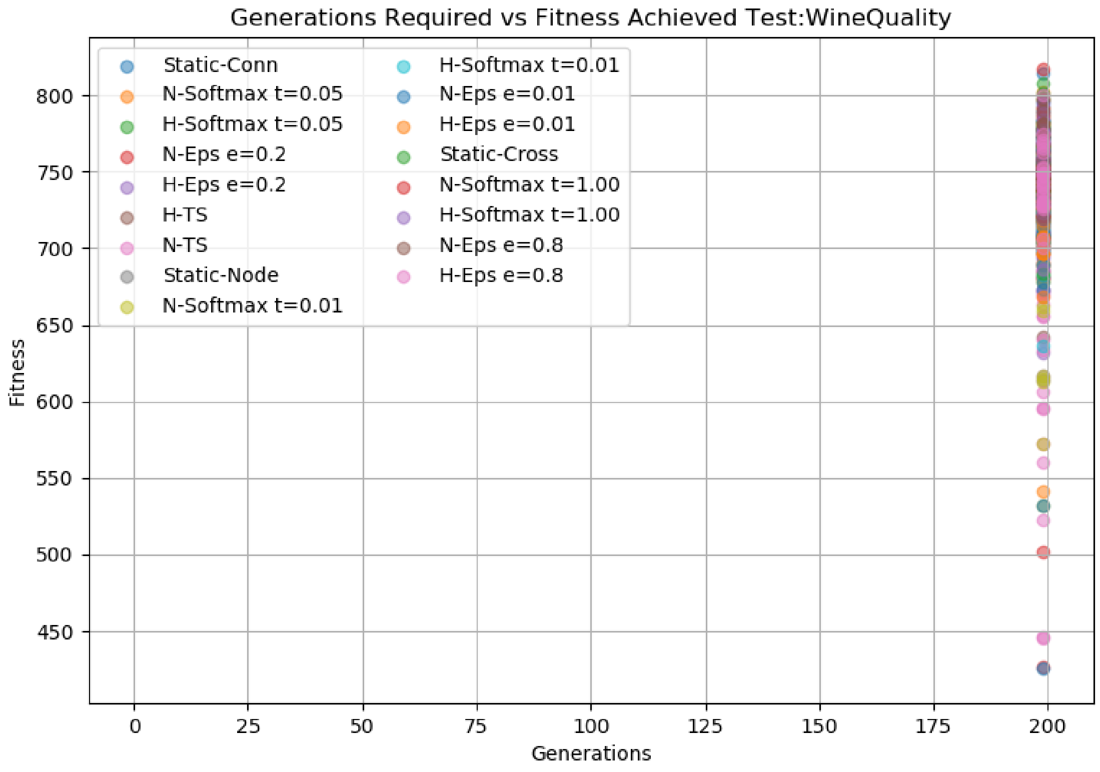


Figure 5.9: Generations vs Achieved Fitness for all MAB in Wine Quality classification. Each point represents the generations required to reach the terminal fitness. The x-axis is the number of generations required and the y-axis is the fitness achieved for each run.

This test shows a clear case of insufficient generations or inappropriate fitness threshold. There is absolutely no deviation in generations required as no fitness threshold is reached. As a trend, the fitness rises with each generation, shown by the fitness points being relatively skewed to higher fitness. The cutoff fitness is set at 1200, representing a 75% accuracy. However the highest fitness reached was only slightly above 800, just over 50% accuracy. A random baseline is expected to get a fitness around 680 (out of 1600), or 42.5% accuracy.

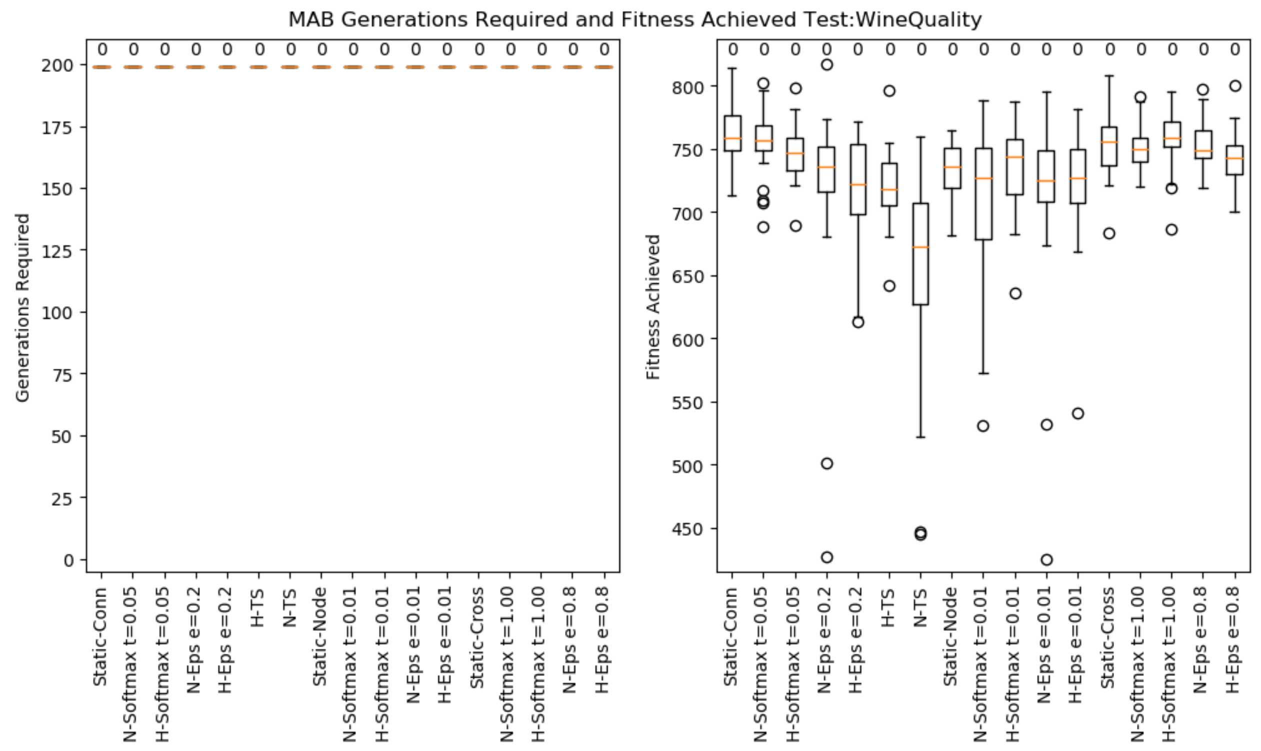


Figure 5.10: Box and whisker plot for Wine Quality test. Outliers are included as circles. Shows the expected range of performance of each MAB with respect to each metric individually. At the top of each box plot is the number of times the MAB reaches the threshold fitness. The left plot is generations and the right plot is fitness. In this case no MAB was able to reach the threshold fitness, so there is no distribution for generations.

As there is no deviation at all for generations, we can only look at fitness and approximate how many generations are required. Static-Conn again has some of the highest fitness reached and additionally a nicely behaved fitness distribution with no outliers, suggesting it is more consistent in finding higher performing solutions overall. Static-Node also exhibits similar behaviour, just with a lower overall distribution.

Summary of Wine Quality pairwise comparisons						
	Generations			Fitness		
MAB Name	n Better	n Neutral	n Worse	n Better	n Neutral	n Worse
H-Softmax t=1.00	N/A	N/A	N/A	13	3	0
Static-Conn	N/A	N/A	N/A	13	3	0
N-Softmax t=0.05	N/A	N/A	N/A	11	5	0
N-Eps e=0.8	N/A	N/A	N/A	10	4	2
N-Softmax t=1.00	N/A	N/A	N/A	10	4	2
Static-Cross	N/A	N/A	N/A	10	6	0
H-Softmax t=0.05	N/A	N/A	N/A	8	5	3
H-Eps e=0.8	N/A	N/A	N/A	6	4	6
H-Softmax t=0.01	N/A	N/A	N/A	3	7	6
Static-Node	N/A	N/A	N/A	2	7	7
H-Eps e=0.01	N/A	N/A	N/A	1	7	8
H-Eps e=0.2	N/A	N/A	N/A	1	7	8
H-TS	N/A	N/A	N/A	1	5	10
N-Eps e=0.01	N/A	N/A	N/A	1	7	8
N-Eps e=0.2	N/A	N/A	N/A	1	8	7
N-Softmax t=0.01	N/A	N/A	N/A	1	6	9
N-TS	N/A	N/A	N/A	0	0	16
Total bandits	17					

Table 5.5: Summary of pairwise comparison between MAB for Wine Quality test , sorted by n Better Fitness. Counts the number of times one MAB is statistically better, no different, or worse than another for both metrics compared separately. A higher figure for n Better and lower figure for n Worse is favourable.

As the Generations were identical for every run for every MAB, the statistical comparisons are not applicable, so this comparison is cut from the test. Looking at the statistical comparisons, from the bottom it is clear that N-TS is the worst performing of all. It has a less favourable distribution compared to every other MAB. As a general trend the more exploration-biased parameters performed better.

5.3 Further Discussion

Summary of All Pairwise Comparisons							
	Generations			Fitness			Average
MAB Name	% Better	% Neutral	% Worse	% Better	% Neutral	% Worse	% Better
Static-Conn	57.81%	39.06%	3.13%	52.50%	43.75%	3.75%	54.86%
H-Softmax t=1.00	39.06%	54.69%	6.25%	46.25%	48.75%	5.00%	43.06%
N-Eps e=0.8	37.50%	46.88%	15.63%	46.25%	48.75%	5.00%	42.36%
N-Softmax t=1.00	45.31%	50.00%	4.69%	38.75%	52.50%	8.75%	41.67%
N-Softmax t=0.05	39.06%	51.56%	9.38%	37.50%	57.50%	5.00%	38.19%
Static-Cross	37.50%	59.38%	3.13%	38.75%	57.50%	3.75%	38.19%
H-Eps e=0.8	40.63%	56.25%	3.13%	30.00%	53.75%	16.25%	34.72%
H-Softmax t=0.05	31.25%	60.94%	7.81%	22.50%	66.25%	11.25%	26.39%
N-Eps e=0.2	32.81%	43.75%	23.44%	13.75%	70.00%	16.25%	22.22%
Static-Node	18.75%	40.63%	40.63%	13.75%	46.25%	40.00%	15.97%
H-Softmax t=0.01	9.38%	43.75%	46.88%	13.75%	58.75%	27.50%	11.81%
N-TS	6.25%	57.81%	35.94%	16.25%	42.50%	41.25%	11.81%
N-Softmax t=0.01	9.38%	53.13%	37.50%	12.50%	52.50%	35.00%	11.11%
H-Eps e=0.2	12.50%	48.44%	39.06%	6.25%	52.50%	41.25%	9.03%
H-TS	9.38%	45.31%	45.31%	8.75%	50.00%	41.25%	9.03%
N-Eps e=0.01	0.00%	50.00%	50.00%	8.75%	52.50%	38.75%	4.86%
H-Eps e=0.01	1.56%	42.19%	56.25%	3.75%	26.25%	70.00%	2.78%

Table 5.6: Total summary of all tests , sorted by Average % Better. Percentage is obtained by dividing the number of times an MAB is deemed better, neutral, or worse in all pairwise comparisons from previous tests. Figures represent percentage of times the MAB is better, neutral, or worse than other MABs.

Table 5.6 shows the aggregate of pairwise comparisons for all tests. Percentages are obtained by dividing the number of times an MAB is deemed Better, Neutral, or Worse from all pairwise comparisons for each tests divided by the total number of comparisons done for each test. The results for Generations from the Wine Quality comparisons (Table 5.5) are discarded as the test run time was insufficient in separating by generations. The average between Generations % Better and Fitness % Better is used to account for both values and used in comparison.

Here each research question and associated sub questions are addressed and discussed directly.

1 **Can MAB decrease the number of generations required to find solutions?**

In over half the cases, the MABs implemented do not perform better than simple static NEAT. Static NEAT is represented by the MABs named Static-Conn, Static-Node, Static-Cross, representing parameters for NEAT with a connection bias, node bias, and even bias with increased crossover respectively.

It is a general correlation that requiring fewer generations is correlated with reaching higher fitness. This is likely due to the fact that the only way to have a run take fewer generations is to reach the fitness threshold, otherwise it receives the maximum generations possible. Perhaps it is unsurprising, as the main source of increase in generations required is being unable to reach the fitness threshold at all. Because the Static-Conn MAB was able to reach the fitness threshold more consistently, its generations and fitness metrics are generally favourable over the other MAB.

From the experimental results obtained, it appears that the particular implementation of MABs in the experiments were generally unable to reduce the number of generations required to find solutions. The rest of the discussion aims to understand why it may be the case that the MAB used were unable to provide speedup.

1.1 Which MAB performs the best in this context?

The best performing MAB is the Softmax(τ = 1.00) MAB, approaching the performance of Static-Conn. Most of the top performing MAB other than the Static MABs were the Softmax MAB, this can be in part due to the similarity of these with the normal mutation selection mechanism of NEAT. Matching the mechanisms of NEAT with simple MAB appear to perform more favourably.

One of the noticeable breaks in the ranking in Table 5.6 is between H-Eps(ϵ = 0.8) and H-Softmax(τ = 0.05), with a 8.33% difference in Average % Better compared to the relatively smaller drops preceding it. One of the key characteristics of MAB above and below this split are the difference in exploration parameters. Those with higher exploration tend to perform more favourably than those with more greedy parameters.

N-Softmax(τ = 0.05) and H-Softmax(τ = 0.05) also performed differently based on Average % Better despite being the same MAB with the same parameters. The main difference is the heuristic or numerical reward system. However this trend is not apparent across other MAB (i.e. the trend that numerical rewards outperform heuristic rewards). It is contradicted by H-Softmax(τ = 1.00) being the second best performing MAB while N-Eps(ϵ = 0.8) being the third. From there it seems that attributing performance to numerical or heuristic reward has little point as the problem becomes whether the MAB is able to find a solution at all rather than how quickly a solution can be found. It is much more likely that parameter settings are the major influence in performance. However, the answers to the research questions later discuss the use of fitness as a reward in general.

1.2 How do MAB parameters for MAB affect the ability to find solutions?

A common characteristic of the top performing MAB is that they use parameters that promote exploration over exploitation. An easy example is N-Eps(ϵ = 0.8), where it explores 80% of the time when selecting mutations. The Softmax exploration/exploitation parameter τ is essentially an exponential reward divider. With lower values, the exponent is increased, while a larger value brings the exponent closer to zero. The rewards are scaled exponentially so a larger exponent results in greater separation, while a smaller exponent brings all values much closer together.

H-Softmax will be used as the example to compare the effects of exploration parameters.

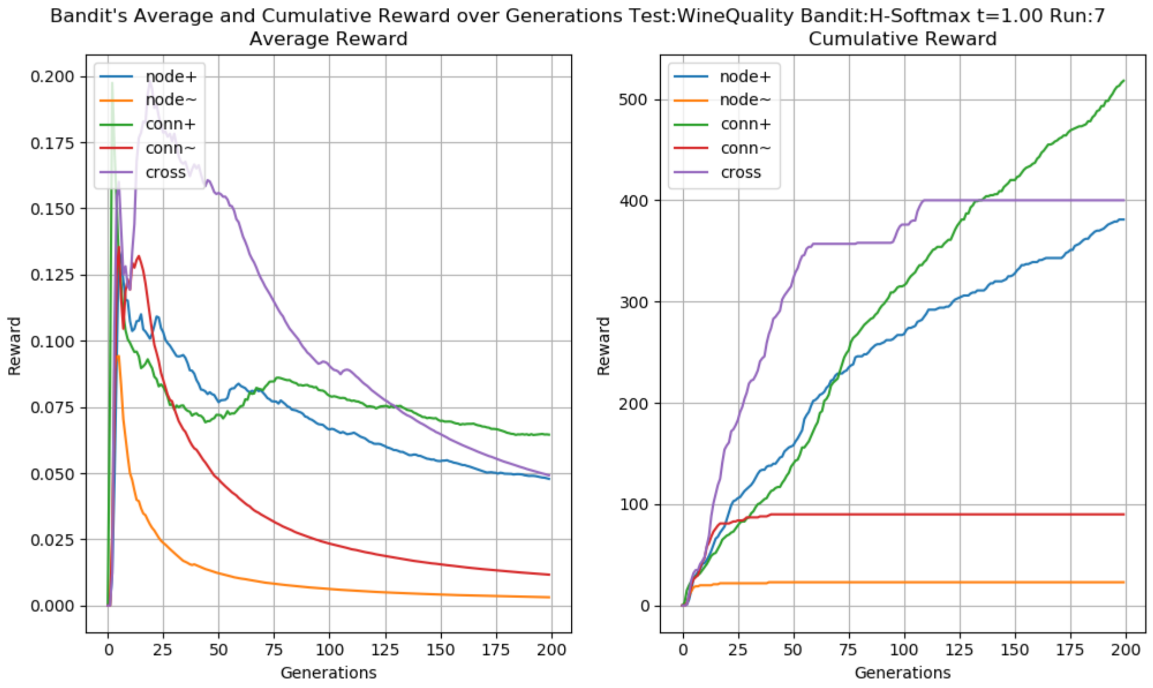


Figure 5.11: Rewards for run 7 of H-Softmax(t=1.00) in Wine Quality test. The left chart shows the average reward and the right chart shows cumulative reward.

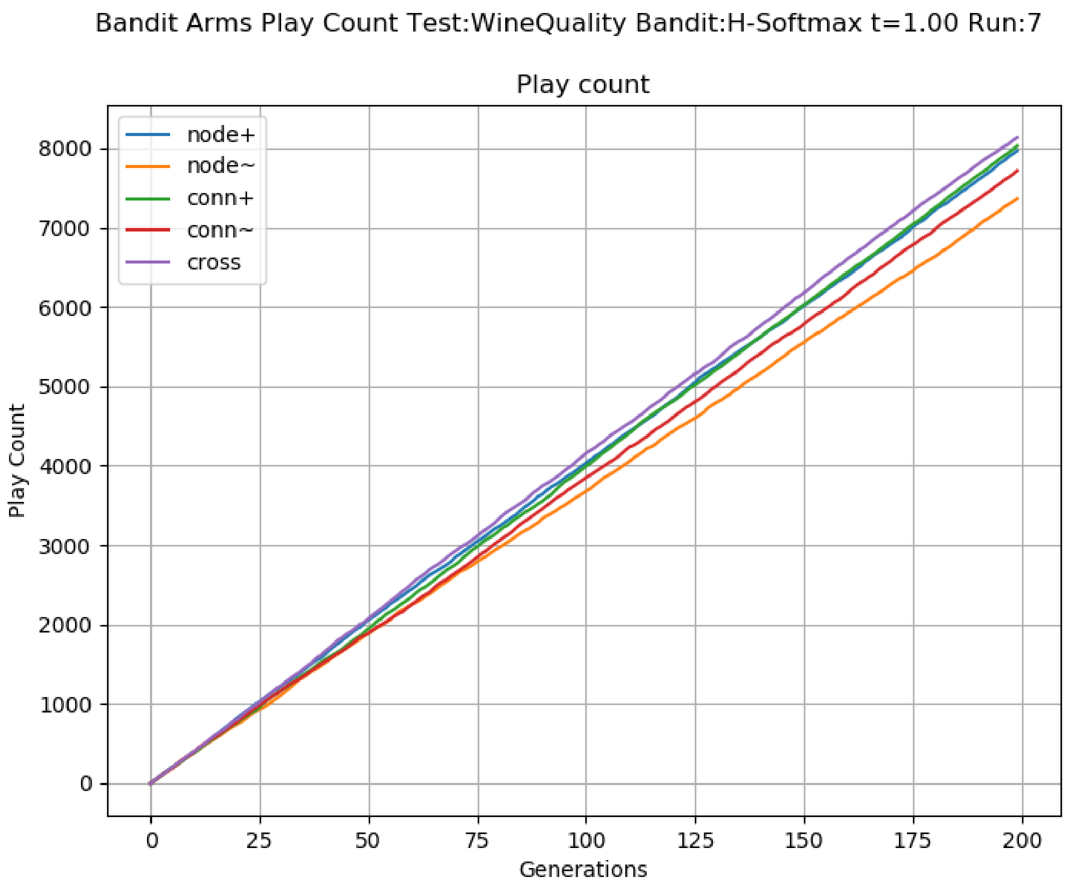


Figure 5.12: Selection count for run 7 of H-Softmax(t=1.00) in Wine Quality test. Shows the cumulative number of times each arm/mutation is played each generation.

The important point to note in Figure 5.11 is the value of the average reward, shown on the left graph. As the run progresses, the rewards diminish to very small values. When using these values in the formula for Softmax in Equation 3.4 in Chapter 3.4, the value for each arm converges to the homogeneous value of 1, later normalised to around 0.2. Therefore they begin to appear uniform when being selected, essentially becoming a uniformly random selector with very small differentiation.

The corresponding number of times each arm is selected for Figure 5.11 is shown in Figure 5.12. The arms are played essentially uniformly, with the difference in number of times selected being essentially negligible.

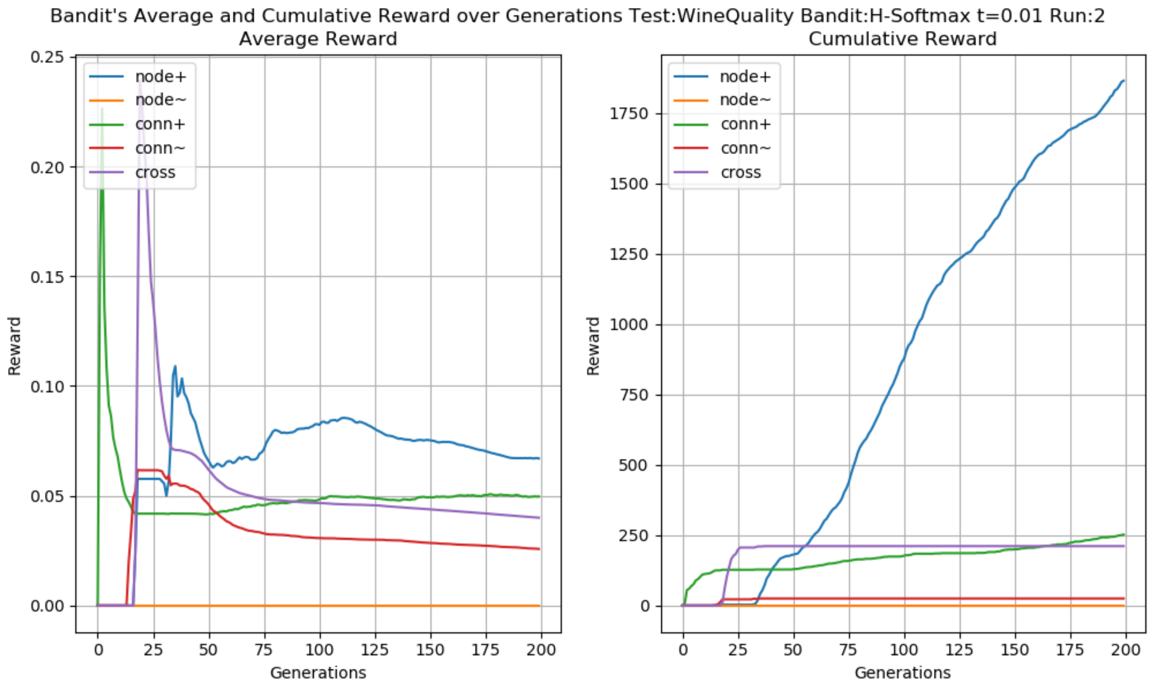


Figure 5.13: Rewards for run 2 of H-Softmax(t=0.01) in Wine Quality test. The left chart shows the average reward and the right chart shows cumulative reward.

Compared to the greedier parameter settings of H-Softmax with $\tau = 0.01$. The magnitudes of the average reward are approximately the same, but when looking at the play count chart in figure Figure 5.14, the distribution reflects the cumulative reward but now a few of the mutations are rarely used. In fact, this is a common behaviour across many of the runs with greedy Softmax parameters where only a limited subset of mutations are regularly used.

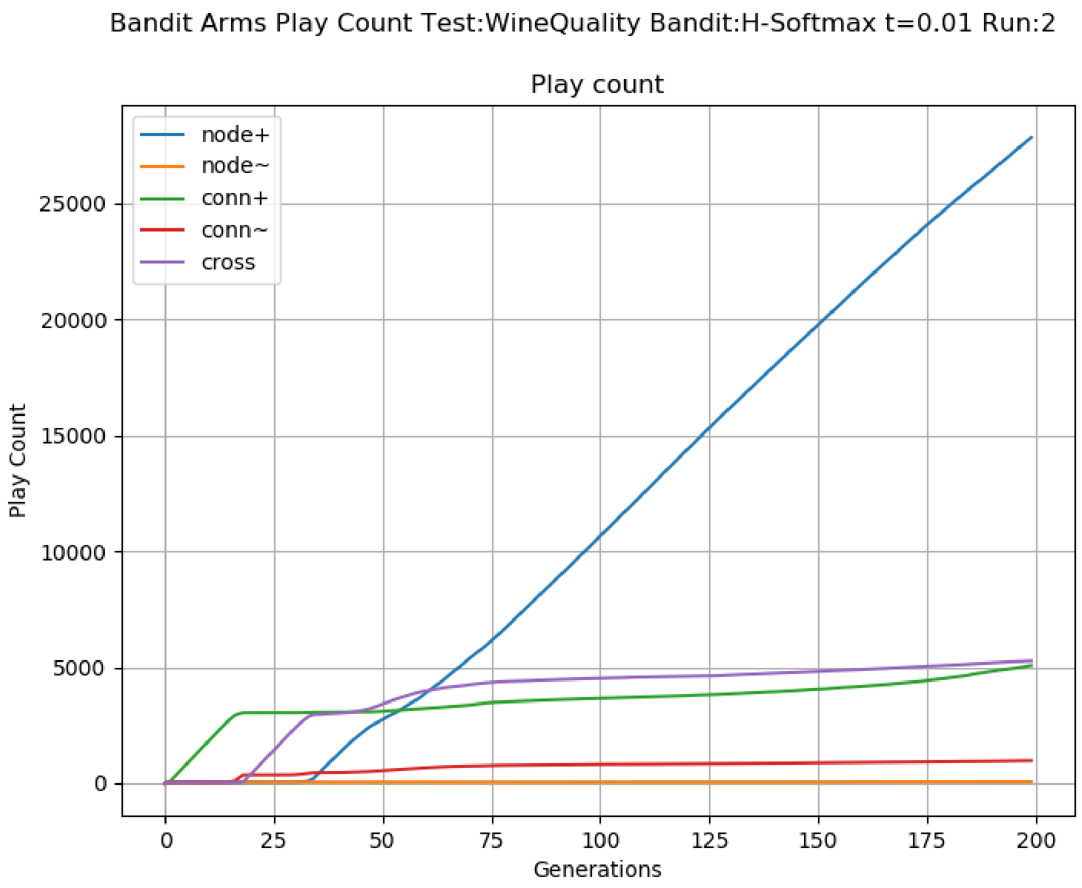


Figure 5.14: Selection count for run 2 of H-Softmax(t=0.01) in Wine Quality test. Shows the cumulative number of times each arm/mutation is played each generation.

Interestingly, Thompson Sampling exhibits much of the same behaviour as the greedy parameters. This may be due to the sheer number of rewards received, causing the PDF distributions to narrow perhaps too quickly, settling on a suboptimal reward. As the PDF distribution is now greatly narrowed, the chances of selecting anything but the current maximum is greatly reduced, practically non-existent. The play count is shown in Figure 5.15. This is the play count from a different test, yet mirrors much of the same properties and can easily be mistaken as being identical. The over-eager convergence can be reduced if a parameter was introduced to the Thompson Sampling MAB that scales the α, β parameters to represent a slower learning rate, essentially lowering the value of each reward to maintain exploration for longer.

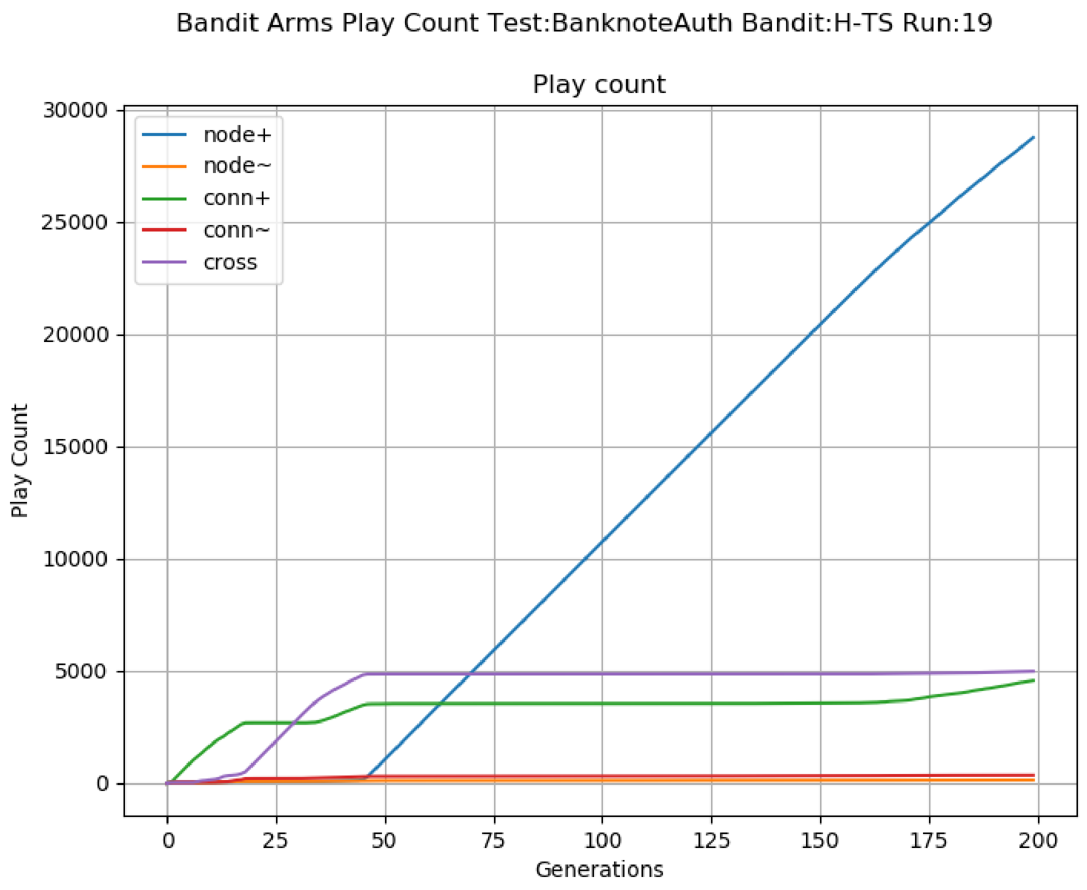


Figure 5.15: Selection count for run 10 of H-TS in Banknote Authentication Test. Shows the cumulative number of times each arm/mutation is played each generation.

A conclusion can be made that not only is diversity in the population required, and protected by speciation, but also a diversity in genetic operator selection is also required. Without a diverse selection of genetic operators, there is less likely to be room for genetic variation. As the ablation tests in [27] show, speciation is one of the key reasons to the performance of NEAT, but must be made use of by the genetic operators to be useful. Thus, when selecting parameters to use for MAB in NEAT, parameters favouring higher exploration are almost essential.

1.3 How well does fitness work as a reward for MAB?

One important aspect in many machine learning systems are proper reward or loss functions. These experiments just used a very basic reward system that only considers instantaneous reward from every application of a Mutation. Given that the parameters for the static MAB were chosen with very little reasoning, the connection biased parameters still managed to outperform most other MAB. So it seems that the direct use of individual fitness changes does rewards does not provide much benefit.

As can be seen in Figure 5.16, there is a slight difference in the numerical and heuristic rewards, top left and bottom left respectively. While there is a difference in the scale of these rewards, the final rankings end up approximately the same. As a result, the Cumulative rewards also follow approximately the same rankings.

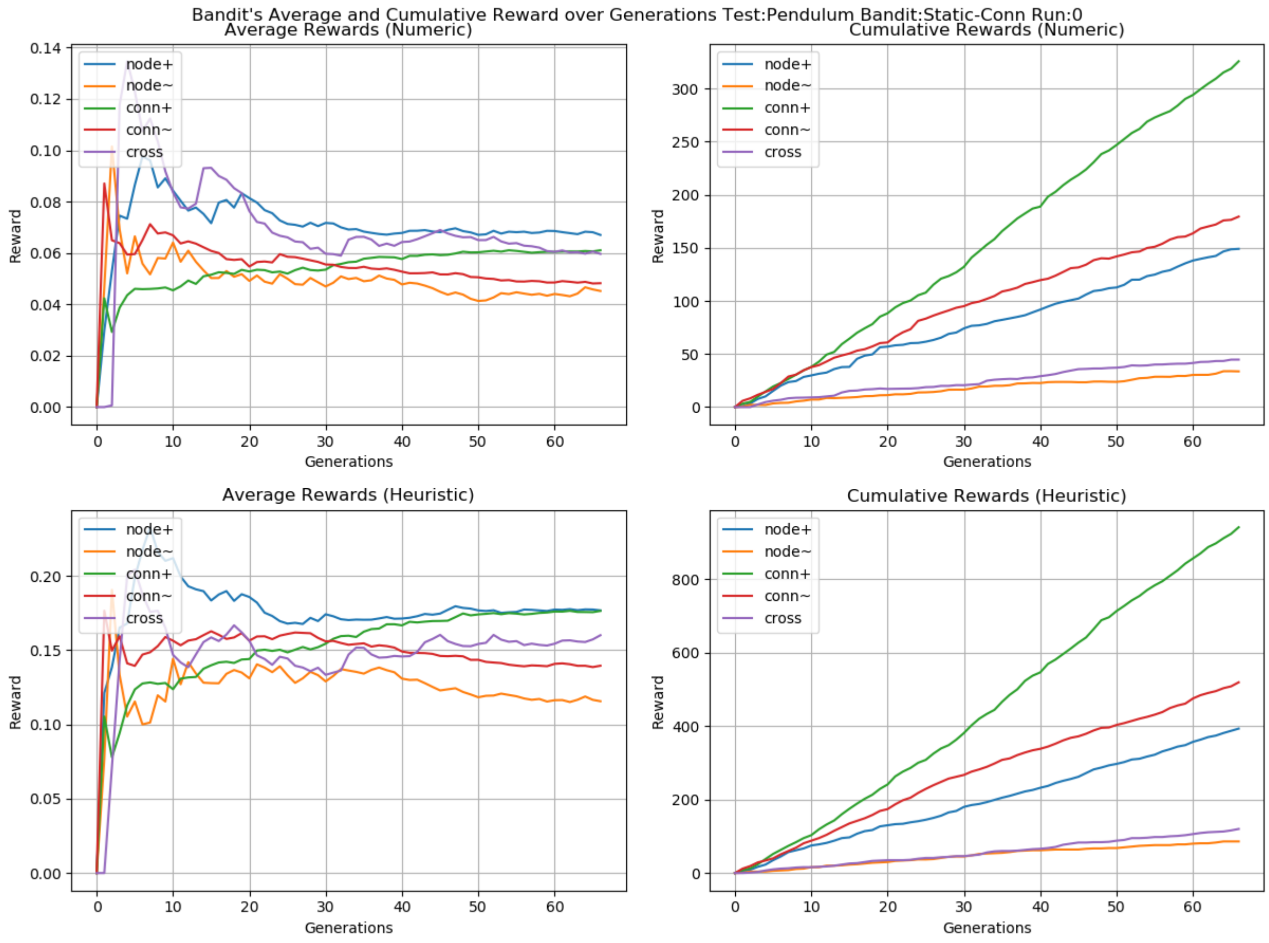


Figure 5.16: Rewards observed by static MAB. Static MAB were used to observe rewards of playing arms according to hyper-parameters. The overall observed reward is very low, and subject to noise. Furthermore the difference between rewards is not nearly as great as the typical difference between hyper-parameterised probabilities.

The values are noisy and for the most part have very little to differentiate between each mutation. The values seen in Figure 5.16 are the observed reward for the Static-Conn MAB but is typical for most other MAB. These values are not acted on by the MAB as it was used essentially as an observation tool.

It is also telling that the MAB that performed better are the ones with higher exploration parameters. These MAB practically ignore the fitness and explore the majority of the time. Looking back at Figure 5.12, the play count for each arm is practically equal, with very slight variation.

There are mechanisms that allow the population to improve with frequent and diverse mutations. These are speciation and elitism. The former is identified as vital for performance in [27]. While speciation protects structural innovation from being removed immediately due to lowered fitness, elitism reduces genetic drift where the overall performance within a species shifts due to frequent crossover and mutation unintentionally dropping fitness of the species. By enforcing that at least a few of the fittest members of the species remain from generation to generation, genetic operations can be considered as repeated attempts to improve the fittest few members. As such, applying a wide range of mutations with slight preferences over which to apply is more favourable than repeatedly applying the same mutation even tho it may have received high reward in the past.

2 What benefits do MAB provide over static NEAT, if any?

In most of the tests the Static MAB was the best performer. However, the tests are subject to a high degree of randomness, and the statistical differences found are largely due to outliers. Using a probability matching MAB such as the Softmax MAB oftentimes matches the performance of static NEAT. With the matched performance, parameter tuning is much simpler as the main variable dictating performance is reduced to a single hyper-parameter.

Since there is no appreciable difference to fitness when aiming for a threshold, the main concern is with how quickly the MAB are able to reach the fitness threshold. While there is statistical siginificance when comparing some MAB, these are usually the result of very poor performance and normal performance. The actual difference between the higher performing MAB is much smaller. The central tendencies are negligibly close, just by inspection of the box plots.

Chapter 6

Conclusion

This project is motivated by the problem of parameter optimisation when using NEAT, namely to remove the need for setting some of the probability based hyper-parameters at all. The implementation of MAB in NEAT attempts to adaptively select genetic operators based on improvements to fitness. The hypothesis is that observing feedback for operators allows the MAB to make more informed decisions that are more likely to improve the fitness of the networks and thus faster to reach the fitness threshold. The experiments test a selection of MAB with different reward schemes and differing parameters in multiple test environments. The results are used to discuss what benefits this brings over standard NEAT.

This project attempts to speed up NEAT by using MAB to select mutations that are more likely to improve fitness. The 4 MABs used are Static, Softmax, ϵ-Greedy, and Thompson Sampling. Each MAB is also tested with different parameters to test the benefit of exploration and exploitation in the NEAT environment. Additionally, 2 reward schemes are used for each MAB, heuristic rewards which reduce fitness changes to Bernoulli success/fail trials, and numerical rewards which directly utilise changes to fitness as the reward for actions.

The MAB are tested in the Pendulum, Bipedal Walker, and Mountain Car environments from Open AI gym and the Banknote Authentication and Wine Quality datasets from UCI Machine Learning Repository.

The results show that Static MAB most consistently outperforms other MAB. As the static MAB was a stand-in for regular NEAT, it indicates that no improvement was achieved by adaptively selecting mutation operators. Of the MAB tested, the best performing ones are the Softmax MAB with less greedy parameters. The parameters tested indicate that the less greedy the parameters are, the more likely it is to perform well. There is not much appreciable difference between the heuristic and numerical rewards based on the performance of the MABs, and differences can be attributed to randomness. While the MAB does not perform well compared to static NEAT, it potentially reduces the number of parameters required to be tuned if an improved implementation were successful.

An unintended side effect of the tests is that it shows NEAT is actually quite robust to frequent mutations. Given that the less greedy parameters are essentially uniformly random selectors, NEAT is still able to filter out the poor mutations and keep the best performing genomes. However, if mutations are poorly distributed such as with greedy parameters, then the chances of beneficial mutations being encountered at all are much lower and NEAT cannot help against that. That being said, it may still be more beneficial if an appropriate spread of mutations are selected.

6.1 Limitations and Future Work

One of the limitations or shortcomings, related to this work is in the experiments. The high variability of the results indicates that the number of runs per test per MAB is insufficient and more runs are needed to gain more definitive results for comparing the MAB. Additionally, the tests would have to be run for more generations, as some tests were unable to reach the threshold fitness. By not reaching the threshold fitness, we are left speculating how quickly it *may* have performed based on peak fitness.

Another limitation is in the update and reward mechanics to the MAB. The batch update and delayed reward used for the MAB prevents the use of more deterministic MAB such as UCB bandits. The implementation was naively done as a direct drop-in replacement to mutation and crossover. A better mechanism which provides immediate feedback for mutations on an individual basis would be more ideal. Doing so would require careful consideration of the selection and speciation mechanics of NEAT and would require a a change to the GA portion of the algorithm.

Only single-play MAB were used in this test, resulting in slower mutation overall, and also excludes more complex developments from occurring such as crossover and adding a connection simultaneously. Multi-play bandits could be explored in the future as NEAT originally has no restrictions on the number of mutations applicable. It was consciously avoided in these experiments as it complicates the allocation of rewards. Furthermore it was uncertain if MAB would provide any benefit in run time, so starting minimally was preferred.

One key idea for future work could be the use of *contextual MAB* which uses the network properties of the ANN produced as the context, selecting mutations based on these properties. Doing so could identify, for example, that a network with significant diameter and low degree may need a connection added rather than a new node - purely speculative at this point. Applying context could potentially lead to more insight in the construction and resulting network, justifying the addition of network features by analysing the network itself.

Bibliography

[1] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. *Journal of Machine Learning Research*, 23:1–26, 2012.

[2] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(3):397–422, 2003.

[3] Djallel Bouneffouf and Irina Rish. A Survey on Practical Applications of Multi-Armed and Contextual Bandits. apr 2019.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *OpenAI Gym*, pages 1–4, 2016.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, dec 1989.

[6] Luis DaCosta, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. *GECCO'08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation 2008*, pages 913–920, 2008.

[7] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, pages 1–14, 2018.

[9] Álvaro Fialho and Roberto Silvestre. Adaptive Operator Selection for Optimization. *Computers and Structures*, 85:1547–1561, 2010.

[10] Adam Gaier and David Ha. Weight agnostic neural networks. 2019.

[11] Faustino J. Gomez and Risto Miikkulainen. Solving non-Markovian control tasks with neuroevolution. *IJCAI International Joint Conference on Artificial Intelligence*, 2:1356–1361, 1999.

[12] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[13] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *Journal of Machine Learning Research 1*, 1:1–32, feb 2014.

[14] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–13, 2019.

[15] Alan McIntyre, Matt Kallada, Cesar G. Miguel, and Carolina Feher da Silva. neat-python. `https://github.com/CodeReclaimers/neat-python`.

[16] Renato Negrinho and Geoff Gordon. DeepArchitect: Automatically Designing and Training Deep Architectures. 2017.

[17] Jens Niehaus and Wolfgang Banzhaf. Adaption of operator probabilities in genetic programming. In Julian Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming*, pages 325–336, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient Neural Architecture Search via parameter Sharing. *35th International Conference on Machine Learning, ICML 2018*, 9:6522–6531, 2018.

[19] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[20] Ben Rubinstein. Statistical machine learning, lecture 6. perceptron, university of melbourne, 2019.

[21] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[22] J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. *COGANN 1992 - International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, 1992.

[23] E.D. Sontag. Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, 3(6):981–990, 1992.

[24] M. Srinivas and L. M. Patnaik. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

[25] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[26] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, apr 2009.

[27] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[28] K.O. Stanley, B.D. Bryant, and Risto Miikkulainen. Real-Time Neuroevolution in the NERO Video Game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, dec 2005.

[29] Joannès Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3720 LNAI:437–448, 2005.

[30] Shimon Whiteson, Peter Stone, Kenneth O Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection in neuroevolution. In *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, number June, page 1225, New York, New York, USA, 2005. ACM Press.

[31] Martin Wistuba. Practical Deep Learning Architecture Optimization. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 263–272. IEEE, oct 2018.

[32] Y. Y. Wong, K. H. Lee, K. S. Leung, and C. W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Comput.*, 7(8):506–515, August 2003.

[33] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–17, 2019.

Chapter 7

Appendix

7.1 Code Availability

The code has been made available on GitHub: `https://github.com/MD-Lai/ma-neat`

7.2 NEAT Parameters

Parameters used for `neat-python` can also be found in the github link as plain text files with the names `cfg_<test_name>`.

7.3 Full Pairwise Comparisons

The following tables were deemed too cumbersome to fit in the body of the paper, but is important to show where the numbers come from. Used in the summary tables Table 5.1, Table 5.2, Table 5.3, Table 5.4, and Table 5.5. The lower left triangles are the pairwise comparisons of generations, and the upper right are pairwise comparisons of fitness. The statistical tests are formulated in such a way where $P <= 0.05$ indicates the left MAB is statistically "better" and $P >= 0.95$ indicates the upper MAB is statistically "better". The arrows at the end of the P-value are added to "point" to the favourable MAB. The values obtained in the summary tables count the number of times an MAB is pointed towards or away from in each respective triangle. By symmetry, when one is pointed to, the other is pointed away from.

For example in Table 7.1, comparing row "N-Softmax t=0.01" and column "N-Eps e=0.2", $P = 0.97$. Therefore, a count is added to "Generations n Better" for "N-Eps e=0.2" and a count for "Generations n Worse" for "N-Softmax t=0.01", indicating that "N-Eps e=0.2" was the better MAB in that specific comparison. This is done for all MAB pairs in the table.

Pairwise Comparison for Pendulum Test																	
	Static Conn	**N-Softmax t=0.05**	**H-Softmax t=0.05**	**N-Eps e=0.2**	**H-Eps e=0.2**	**H-TS**	**N-TS**	**Static Node**	**N-Softmax t=0.01**	**H-Softmax t=0.01**	**N-Eps e=0.01**	**H-Eps e=0.01**	**Static Cross**	**N-Softmax t=1.00**	**H-Softmax t=1.00**	**N-Eps e=0.8**	**H-Eps e=0.8**
Static Conn	-	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←
N-Softmax t=0.05	1.00↑	-	0.38	0.33	0.06	0.13	0.15	<0.01←	0.44	0.23	0.12	0.03←	0.53	0.91	0.92	0.36	0.47
H-Softmax t=0.05	1.00↑	0.69	-	0.45	0.13	0.26	0.21	0.02←	0.55	0.44	0.29	0.08	0.6	0.95	0.95↑	0.56	0.58
N-Eps e=0.2	1.00↑	0.87	0.74	-	0.09	0.22	0.17	0.01←	0.41	0.35	0.24	0.06	0.66	0.97↑	0.97↑	0.61	0.66
H-Eps e=0.2	1.00↑	0.99↑	0.98↑	0.94	-	0.75	0.75	0.19	0.96↑	0.89	0.81	0.44	0.94	1.00↑	1.00↑	0.92	0.96↑
H-TS	1.00↑	1.00↑	1.00↑	0.99↑	0.93	-	0.52	0.04←	0.88	0.75	0.55	0.15	0.87	1.00↑	1.00↑	0.82	0.9
N-TS	1.00↑	1.00↑	1.00↑	0.99↑	0.93	*	-	0.04←	0.89	0.76	0.59	0.18	0.87	1.00↑	1.00↑	0.83	0.91
Static Node	1.00↑	1.00↑	0.99↑	0.97↑	0.72	0.17	0.17	-	1.00↑	0.99↑	0.97↑	0.76	0.98↑	1.00↑	1.00↑	0.99↑	1.00↑
N-Softmax t=0.01	1.00↑	1.00↑	0.99↑	0.97↑	0.72	0.17	0.17	0.5	-	0.36	0.25	0.04←	0.65	0.99↑	0.99↑	0.62	0.66
H-Softmax t=0.01	1.00↑	1.00↑	1.00↑	0.99↑	0.93	*	*	0.85	0.85	-	0.37	0.05←	0.75	1.00↑	0.99↑	0.73	0.82
N-Eps e=0.01	1.00↑	1.00↑	1.00↑	0.99↑	0.93	*	*	0.85	0.85	*	-	0.1	0.85	1.00↑	1.00↑	0.81	0.9
H-Eps e=0.01	1.00↑	1.00↑	1.00↑	0.99↑	0.93	*	*	0.85	0.85	*	*	-	0.96↑	1.00↑	1.00↑	0.96↑	0.98↑
Static Cross	1.00↑	0.37	0.22	0.07	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	-	0.87	0.9	0.4	0.43
N-Softmax t=1.00	1.00↑	0.04←	0.02←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.09	-	0.5	0.05←	0.06
H-Softmax t=1.00	0.99↑	0.04←	0.02←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.09	0.35	-	0.06	0.05←
N-Eps e=0.8	1.00↑	0.53	0.38	0.18	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.67	0.95↑	0.96↑	-	0.57
H-Eps e=0.8	1.00↑	0.37	0.23	0.09	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.52	0.89	0.88	0.32	-

Table 7.1: Full pairwise statistical comparison for Pendulum Test summarised in Table 5.1. The table shows the P-value when comparing MAB_i to MAB_i using one-tailed Mann-Whitney U test for all i, j where i, j are the row and column index respectively. The bottom triangle compares the generations, and upper triangle compares fitness.of each pair of MAB. $P <= 0.05$ indicates the left MAB is “better” and $P >= 0.05$ indicates the top MAB is “better”.

Pairwise Comparison for Bipedal Walker Test

	Static Conn	**N-Softmax t=0.05**	**H-Softmax t=0.05**	**N-Eps e=0.2**	**H-Eps e=0.2**	**H-TS**	**N-TS**	**Static Node**	**N-Softmax t=0.01**	**H-Softmax t=0.01**	**N-Eps e=0.01**	**H-Eps e=0.01**	**Static Cross**	**N-Softmax t=1.00**	**H-Softmax t=1.00**	**N-Eps e=0.8**	**H-Eps e=0.8**
Static Conn	-	0.83	0.16	0.34	<0.01←	<0.01←	0.49	0.01←	0.52	0.05	0.3	<0.01←	0.34	0.17	0.1	0.65	0.08
N-Softmax t=0.05	0.52	-	0.02←	0.05	<0.01←	<0.01←	0.13	<0.01←	0.15	<0.01←	0.04←	<0.01←	0.05←	0.01←	0.01←	0.28	<0.01←
H-Softmax t=0.05	0.86	0.86	-	0.67	0.05	<0.01←	0.83	0.08	0.84	0.2	0.7	<0.01←	0.72	0.48	0.34	0.93	0.27
N-Eps e=0.2	0.23	0.21	0.04←	-	<0.01←	<0.01←	0.68	0.1	0.74	0.09	0.51	<0.01←	0.57	0.43	0.2	0.84	0.19
H-Eps e=0.2	0.96↑	0.96↑	0.85	0.99↑	-	0.16	0.99↑	0.8	1.00↑	0.89	0.99↑	0.16	0.99↑	0.96↑	0.87	1.00↑	0.91
H-TS	0.96↑	0.96↑	0.85	0.99↑	*	-	1.00↑	1.00↑	1.00↑	0.99↑	1.00↑	0.47	1.00↑	1.00↑	0.99↑	1.00↑	1.00↑
N-TS	0.85	0.84	0.5	0.95↑	0.17	0.17	-	<0.01←	0.55	0.03←	0.26	<0.01←	0.35	0.17	0.07	0.74	0.05
Static Node	0.96↑	0.96↑	0.85	0.99↑	*	*	0.85	-	0.98↑	0.51	0.95↑	<0.01←	0.98↑	0.9	0.78	1.00↑	0.75
N-Softmax t=0.01	0.52	0.5	0.15	0.8	0.04←	0.04←	0.17	0.04←	-	0.03←	0.27	<0.01←	0.36	0.23	0.1	0.69	0.06
H-Softmax t=0.01	0.36	0.33	0.08	0.65	0.02←	0.02←	0.09	0.02←	0.34	-	0.92	0.02←	0.93	0.78	0.58	0.99↑	0.71
N-Eps e=0.01	0.67	0.65	0.27	0.88	0.08	0.08	0.28	0.08	0.66	0.79	-	<0.01←	0.56	0.38	0.18	0.9	0.16
H-Eps e=0.01	0.85	0.86	0.5	0.96↑	0.17	0.17	0.52	0.17	0.85	0.92	0.74	-	1.00↑	1.00↑	0.99↑	1.00↑	1.00↑
Static Cross	0.7	0.7	0.27	0.9	0.08	0.08	0.3	0.08	0.7	0.83	0.53	0.28	-	0.22	0.13	0.78	0.14
N-Softmax t=1.00	0.51	0.49	0.15	0.77	0.04←	0.04←	0.17	0.04←	0.5	0.66	0.35	0.15	0.31	-	0.36	0.9	0.28
H-Softmax t=1.00	0.53	0.54	0.15	0.82	0.04←	0.04←	0.17	0.04←	0.53	0.69	0.36	0.16	0.34	0.55	-	0.96↑	0.5
N-Eps e=0.8	0.85	0.84	0.5	0.95↑	0.17	0.17	0.52	0.17	0.84	0.91	0.73	0.5	0.72	0.84	0.84	-	0.01←
H-Eps e=0.8	0.68	0.67	0.27	0.88	0.08	0.08	0.29	0.08	0.67	0.81	0.52	0.27	0.49	0.68	0.65	0.28	-

Table 7.2: Full pairwise statistical comparison for Bipedal Walker test summarised in Table 5.2. The table shows the P-value when comparing MAB_i to MAB_i using one-tailed Mann-Whitney U test for all i, j where i, j are the row and column index respectively. The bottom triangle compares the generations, and upper triangle compares fitness.of each pair of MAB. $P <= 0.05$ indicates the left MAB is "better" and $P >= 0.05$ indicates the top MAB is "better".

Pairwise Comparison for Mountain Car Test																	
	Static Conn	**N-Softmax t=0.05**	**H-Softmax t=0.05**	**N-Eps e=0.2**	**H-Eps e=0.2**	**H-TS**	**N-TS**	**Static Node**	**N-Softmax t=0.01**	**H-Softmax t=0.01**	**N-Eps e=0.01**	**H-Eps e=0.01**	**Static Cross**	**N-Softmax t=1.00**	**H-Softmax t=1.00**	**N-Eps e=0.8**	**H-Eps e=0.8**
Static Conn	-	0.21	0.1	0.37	0.02←	0.14	0.69	0.41	0.14	0.26	0.01←	<0.01←	0.41	0.08	0.45	0.48	0.03←
N-Softmax t=0.05	0.88	-	0.27	0.56	0.1	0.31	0.87	0.6	0.3	0.42	0.06	0.02←	0.79	0.24	0.66	0.69	0.17
H-Softmax t=0.05	0.95	0.69	-	0.78	0.28	0.57	0.96↑	0.83	0.61	0.65	0.15	0.04←	0.88	0.48	0.87	0.87	0.43
N-Eps e=0.2	1.00↑	0.99↑	0.94	-	0.09	0.38	0.84	0.56	0.31	0.46	0.05	0.01←	0.69	0.22	0.63	0.68	0.15
H-Eps e=0.2	1.00↑	0.98↑	0.95↑	0.47	-	0.82	0.99↑	0.96↑	0.81	0.88	0.28	0.1	0.98↑	0.69	0.96↑	0.97↑	0.64
H-TS	1.00↑	0.98↑	0.94	0.45	0.54	-	0.93	0.7	0.48	0.55	0.12	0.03←	0.87	0.41	0.85	0.86	0.34
N-TS	0.92	0.66	0.48	0.05←	0.07	0.05	-	0.17	0.07	0.12	<0.01←	<0.01←	0.33	0.03←	0.26	0.31	0.02←
Static Node	0.84	0.39	0.2	<0.01←	<0.01←	<0.01←	0.26	-	0.27	0.33	0.04←	<0.01←	0.62	0.14	0.6	0.7	0.13
N-Softmax t=0.01	0.98↑	0.88	0.69	0.06	0.07	0.11	0.73	0.91	-	0.6	0.13	0.03←	0.88	0.39	0.82	0.85	0.37
H-Softmax t=0.01	1.00↑	1.00↑	0.99↑	0.75	0.84	0.77	0.99↑	1.00↑	0.98↑	-	0.08	0.02←	0.77	0.36	0.76	0.78	0.27
N-Eps e=0.01	1.00↑	1.00↑	0.99↑	0.87	0.95	0.84	0.99↑	1.00↑	0.98↑	0.75	-	0.32	0.98↑	0.81	0.98↑	0.98↑	0.82
H-Eps e=0.01	1.00↑	1.00↑	1.00↑	0.98↑	0.99↑	0.98↑	1.00↑	1.00↑	1.00↑	0.95	0.83	-	1.00↑	0.96↑	0.99↑	0.99↑	0.93
Static Cross	0.97↑	0.74	0.45	0.04←	0.03←	0.08	0.54	0.81	0.28	0.01←	0.02←	<0.01←	-	0.1	0.49	0.45	0.03←
N-Softmax t=1.00	0.91	0.51	0.31	0.02←	0.03←	0.03←	0.41	0.61	0.18	<0.01←	<0.01←	<0.01←	0.27	-	0.88	0.89	0.52
H-Softmax t=1.00	1.00↑	0.93	0.77	0.15	0.12	0.22	0.8	0.96↑	0.61	0.05←	0.05	<0.01←	0.77	0.93	-	0.52	0.07
N-Eps e=0.8	1.00↑	0.97↑	0.91	0.26	0.25	0.28	0.89	0.99↑	0.86	0.09	0.03←	<0.01←	0.96↑	0.95↑	0.78	-	0.08
H-Eps e=0.8	0.97↑	0.74	0.45	0.02←	0.02←	0.07	0.55	0.8	0.24	<0.01←	0.01←	<0.01←	0.47	0.73	0.2	0.04←	-

Table 7.3: Full pairwise statistical comparison for Mountain Car test summarised in Table 5.3. The table shows the P-value when comparing MAB_i to MAB_i using one-tailed Mann-Whitney U test for all i, j where i, j are the row and column index respectively. The bottom triangle compares the generations, and upper triangle compares fitness.of each pair of MAB. $P <= 0.05$ indicates the left MAB is "better" and $P >= 0.05$ indicates the top MAB is "better".

Pairwise Comparison for Banknote Authentication Test																	
	Static Conn	**N-Softmax t=0.05**	**H-Softmax t=0.05**	**N-Eps e=0.2**	**H-Eps e=0.2**	**H-TS**	**N-TS**	**Static Node**	**N-Softmax t=0.01**	**H-Softmax t=0.01**	**N-Eps e=0.01**	**H-Eps e=0.01**	**Static Cross**	**N-Softmax t=1.00**	**H-Softmax t=1.00**	**N-Eps e=0.8**	**H-Eps e=0.8**
Static Conn	-	0.26	0.6	0.65	0.16	0.11	<0.01←	0.16	<0.01←	<0.01←	<0.01←	<0.01←	0.99↑	0.81	0.83	1.00↑	0.99↑
N-Softmax t=0.05	0.67	-	0.84	0.8	0.42	0.25	<0.01←	0.28	<0.01←	0.03←	<0.01←	<0.01←	0.99↑	0.94	0.94	1.00↑	1.00↑
H-Softmax t=0.05	0.24	0.18	-	0.48	0.07	0.04←	<0.01←	0.1	<0.01←	<0.01←	<0.01←	<0.01←	0.95	0.72	0.72	1.00↑	0.97↑
N-Eps e=0.2	0.92	0.81	0.96↑	-	0.11	0.08	<0.01←	0.15	<0.01←	<0.01←	<0.01←	<0.01←	0.99↑	0.64	0.78	1.00↑	0.99↑
H-Eps e=0.2	0.73	0.51	0.82	0.26	-	0.36	<0.01←	0.48	<0.01←	0.06	<0.01←	<0.01←	1.00↑	0.97↑	0.97↑	1.00↑	1.00↑
H-TS	1.00↑	0.98↑	1.00↑	0.94	0.97↑	-	<0.01←	0.55	<0.01←	0.09	<0.01←	<0.01←	1.00↑	0.98↑	0.99↑	1.00↑	1.00↑
N-TS	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	-	1.00↑	<0.01←	1.00↑	0.91	0.98↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
Static Node	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	0.88	<0.01←	-	<0.01←	0.08	<0.01←	<0.01←	1.00↑	0.93	0.98↑	1.00↑	1.00↑
N-Softmax t=0.01	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	0.87	1.00↑	-	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
H-Softmax t=0.01	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	0.99↑	<0.01←	0.89	<0.01←	-	<0.01←	0.05	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
N-Eps e=0.01	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	0.44	1.00↑	0.09	1.00↑	-	0.84	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
H-Eps e=0.01	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	0.09	0.99↑	<0.01←	0.93	0.09	-	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
Static Cross	0.06	0.04←	0.22	<0.01←	0.04←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	-	0.07	0.24	0.96↑	0.7
N-Softmax t=1.00	0.51	0.37	0.76	0.1	0.34	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.95	-	0.51	0.99↑	0.95↑
H-Softmax t=1.00	0.16	0.14	0.45	0.02←	0.13	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.82	0.17	-	0.97↑	0.9
N-Eps e=0.8	<0.01←	<0.01←	0.03←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.16	<0.01←	0.02←	-	0.16
H-Eps e=0.8	0.03←	0.02←	0.14	<0.01←	0.03←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.43	0.03←	0.1	0.8	-

Table 7.4: Full pairwise statistical comparison for Banknote Authentication test summarised in Table 5.4. The table shows the P-value when comparing MAB_i to MAB_i using one-tailed Mann-Whitney U test for all i, j where i, j are the row and column index respectively. The bottom triangle compares the generations, and upper triangle compares fitness.of each pair of MAB. $P <= 0.05$ indicates the left MAB is "better" and $P >= 0.05$ indicates the top MAB is "better".

Pairwise Comparison for Wine Quality Test																	
	Static Conn	**N-Softmax t=0.05**	**H-Softmax t=0.05**	**N-Eps e=0.2**	**H-Eps e=0.2**	**H-TS**	**N-TS**	**Static Node**	**N-Softmax t=0.01**	**H-Softmax t=0.01**	**N-Eps e=0.01**	**H-Eps e=0.01**	**Static Cross**	**N-Softmax t=1.00**	**H-Softmax t=1.00**	**N-Eps e=0.8**	**H-Eps e=0.8**
Static Conn	-	0.33	0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.13	0.03←	0.44	0.05←	<0.01←
N-Softmax t=0.05	*	-	0.04←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.26	0.05	0.67	0.08	<0.01←
H-Softmax t=0.05	*	*	-	0.01←	<0.01←	<0.01←	<0.01←	<0.01←	<0.01←	0.09	<0.01←	<0.01←	0.89	0.76	0.99↑	0.83	0.21
N-Eps e=0.2	*	*	*	-	0.24	0.05	<0.01←	0.59	0.17	0.78	0.2	0.28	1.00↑	1.00↑	1.00↑	1.00↑	0.93
H-Eps e=0.2	*	*	*	*	-	0.33	<0.01←	0.89	0.33	0.94	0.58	0.62	1.00↑	1.00↑	1.00↑	1.00↑	0.99↑
H-TS	*	*	*	*	*	-	<0.01←	0.99↑	0.46	0.99↑	0.74	0.84	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
N-TS	*	*	*	*	*	*	-	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑	1.00↑
Static Node	*	*	*	*	*	*	*	-	0.09	0.71	0.1	0.14	1.00↑	1.00↑	1.00↑	1.00↑	0.92
N-Softmax t=0.01	*	*	*	*	*	*	*	*	-	0.96↑	0.66	0.76	1.00↑	1.00↑	1.00↑	1.00↑	0.99↑
H-Softmax t=0.01	*	*	*	*	*	*	*	*	*	-	0.07	0.08	0.99↑	0.97↑	1.00↑	0.98↑	0.74
N-Eps e=0.01	*	*	*	*	*	*	*	*	*	*	-	0.56	1.00↑	1.00↑	1.00↑	1.00↑	0.99↑
H-Eps e=0.01	*	*	*	*	*	*	*	*	*	*	*	-	1.00↑	1.00↑	1.00↑	1.00↑	0.99↑
Static Cross	*	*	*	*	*	*	*	*	*	*	*	*	-	0.28	0.82	0.42	0.02←
N-Softmax t=1.00	*	*	*	*	*	*	*	*	*	*	*	*	*	-	0.97↑	0.58	0.03←
H-Softmax t=1.00	*	*	*	*	*	*	*	*	*	*	*	*	*	*	-	0.03←	<0.01←
N-Eps e=0.8	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	-	0.02←
H-Eps e=0.8	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	-

Table 7.5: Full pairwise statistical comparison for Wine Quality test summarised in Table 5.5. The table shows the P-value when comparing MAB_i to MAB_i using one-tailed Mann-Whitney U test for all i, j where i, j are the row and column index respectively. The upper triangle compares fitness of each pair of MAB. $P <= 0.05$ indicates the left MAB is “better” and $P >= 0.05$ indicates the top MAB is “better”. In this case, the bottom triangle cannot be compared as all values are at the maximum generation limit 200 (i.e. Every run of each MAB did not reach the threshold fitness in 200 generations).