HARDWARE IMPLEMENTATION OF ARTIFICIAL EPIGENETIC NETWORKS

Andrew J. Walter MEng (Hons)

PhD

The University Of York

Electronic Engineering

September 2019

Abstract

An extension of Artificial Gene Regulatory Networks (AGRNs), Artificial Epigenetic Networks (AENs) implement an additional layer of bio-inspired control to allow for enhanced performance on certain types of control tasks by facilitating topological self-modification. This work looks to expand the applications of AENs by translating the existent software architecture into a form suitable for implementation on a Field Programmable Gate Array (FPGA). This opens the possibility of AENs being used in applications where high-performance computational resources are impractical, such as robotic control. This thesis develops a more resource efficient architecture for epigenetic networks based on reduced precision integer mathematics, and then translates it into hardware to provide improvements in resource utilisation and execution speed while not sacrificing the unique benefits provided by the epigenetic mechanisms. The application to robotic control is investigated by utilising the hardware AEN to perform various versions of a foraging task, culminating in one designed to replicate a search and rescue scenario. While the AENs did not demonstrate significant performance improvements compared to their non-epigenetic counterparts, this did indicate that not every type of control task benefits from the inclusion of the epigenetic mechanism. In addition, this work investigates another aspect of AENs, specifically the limits of their topological self-modification with respect to reacting to changes in their environment. More specifically, it is asked if an AEN can maintain its ability to perform a specific task when confronted with factors outside of those it has been optimised to handle. While not conclusively demonstrated, there is sufficient evidence that the answer to this question depends on the performance gains imparted by epigenetic behaviours under normal circumstances.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Prof. Andy Tyrrell, Dr. Simon Bale and Dr. Martin Trefzer for their tireless guidance and assistance; Dr. Alex Turner for his help in replicating and building upon his own doctoral work; John Wilson for his pointers on foraging robots; and Richard Redpath, Matthew Rowlings and Dr. John Szymanski for their aid in various technical areas; and finally my partner Finn McLellan, for his tireless support.

Author’s Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, university. All sources are acknowledged as reference.

1 Introduction

The biological world is a perennial source of inspiration for engineers. With millions of years of evolution to devise solutions, engineers have applied ideas from nature to problems that range from energy efficient buildings [1], to advanced optics [2]. An area where bio-inspiration is of great interest is computation, as the natural world possesses a plethora of mechanisms that store, manipulate and process data. A comparative newcomer in this arena is the Artificial Epigenetic Network (AEN), which expands the more traditional Artificial Gene Regulatory Network (AGRN) through the addition of components that mimic the biological mechanisms of epigenetic regulation. This work is centred on translating such networks from their more resource intensive software form, to a hardware architecture that maintains their unique benefits while opening up more possible applications. Along the way, a hypothesis developed about an epigenetically augmented network's ability to react to unanticipated changes in its environment, which fuelled the latter parts of this work.

1.1 Background

The supermajority of biological systems rely on genetic material for the storage of the information needed to construct and maintain their physiologies. However, far from being simply a static storage system, biological genomes are capable of information processing though a number of systems collectively referred to as gene regulatory mechanisms [3]. In recent years however, it has been discovered that these are not the only systems that biological organisms possess that alter their genetic expression. These so called Epigenetic mechanisms also play a role, interacting with the genetic material through alterations to its structure, but without being a part of it [4].

Artificial Gene Regulatory Networks (AGRNs) are computational architectures inspired by these mechanisms. They are created either as a tool for research into their biological counterparts, or as a way to exploit their properties for useful applications, with the latter being the one relevant to this work [5]. While they possess similarities to the more ubiquitous Artificial Neural Networks (ANNs), their basis in genetic, rather than neural mechanisms leads them to possess a number of differences, most notably in their more indirect method of network topology encodement, which leads to networks that are less linear than feed forward ANNs, but without the total interconnectivity of recurrent neural networks; while also reducing the number of parameters needed [5]. While direct performance comparisons between AGRNs and ANNs are currently lacking the field is still developing.

Artificial epigenetic networks expand upon AGRNs in the same way as in nature, adding additional control mechanisms that sit outside the “genome” but alter its behaviour. Devised by Dr. Alex Turner, the addition of these epigenetic elements to a network provide performance benefits over standard gene regulatory networks, particularly with tasks that involve the control of dynamical systems, with explicit results that confirm this for Chirikov’s standard map; the control of single and multiple coupled inverted pendulums; and the control of transfer orbits in gravitational systems [6]. However, the implementation of epigenetics employed previously is computationally intensive: utilising floating-point numbers and having several features that while more biologically accurate are computationally inefficient, such as: replication of gene to protein transcription, causing increased memory requirements; utilisation of single, output applied weights, resulting in different connection weights requiring entirely different to exist; and an absence of dedicated input/output elements, requiring processing elements to be temporally utilised as such, wasting their resources.

Biological systems on the other hand are frequently lauded for their resource efficiency, so once again it inspires this work: the development and testing of a hardware implementation of an epigenetic network, built upon the reduction of the original’s resource utilisation and streamlining of its processes.

While pursuing this objective, a hypothesis was developed about an epigenetic network’s ability to handle unexpected stimulus: under ideal circumstances, a control system (epigenetic or otherwise) can be expected to encounter conditions that have already been accounted for in its design, either by human engineers or by the parameters used by some automated optimisation mechanism, like a genetic algorithm. In reality, it is impossible for every possibility to be factored in to the design process, and such there is a desire for a control paradigm that is able to maintain its performance when confronted with such unforeseen circumstances. The self-modification ability inferred by epigenetic behaviour seems to be ideally suited to providing this kind of performance.

1.2 Hypothesis

As a key strength of the artificial epigenetic network is its ability to dynamically reconfigure its topology in response to the changing state of its environment, is it possible for an artificial epigenetic network to complete a task it has been optimised for, while being confronted with environmental alterations that it has not encountered during its optimisation? Or, to put this in the form of a hypothesis:

"Do the topological self-modification abilities granted to an artificial hardware network by the inclusion of epigenetic elements, henceforth referred to simply as epigenetic mechanisms, grant that network a greater capacity to maintain its functionality when confronted with un-optimised for stimulus compared to a similar artificial network without epigenetic mechanisms?"

1.3 Objectives

The final outcome of this thesis is the hardware implementation of the artificial epigenetic network, which entailed the development of a version of the artificial epigenetic network that would translate well into digital hardware, In the course of perusing this objective, several side goals developed, which are encapsulated in the questions below.

a) Is it possible to switch the underlying logic of the artificial epigenetic network from computationally intensive floating-point maths to more simplistic integer mathematics?

b) Is it then possible to reduce the precision of the artificial epigenetic network's mathematics while maintain its unique functionality?

c) Is it possible to bring the architecture of an epigenetic network more in line with design conventions, without loss of its unique abilities?

d) Taking it as an example of a resource limited application, does the artificial epigenetic network bring any significant benefits to the area of robotic control?

These questions are, of course, in addition to the one posed in Section 1.2, regarding the epigenetic network's ability of react to un-optimised for stimulus.

1.4 Contributions

In the process of carrying out this work, and pursuing the objectives in Section 1.2, the following contributions to knowledge were made:

- Demonstrated that the properties and performance of an epigenetic network are preserved when its underlying logic is translated from floating-point to integer mathematics.
- Demonstrated that the properties and performance of an epigenetic network are preserved when its precision is reduced through the reduction in the bit width of its underlying logic.
- Demonstrated that the properties and performance of an epigenetic network are preserved when its processing elements are re-designed to bring them more in line with design conventions; while also demonstrating that such networks can be more compact than their Turner architecture counterparts.
- Demonstrated that the properties and performance of an epigenetic network are preserved when it's translated from a software to hardware implementation.
- Demonstrated that the translation from a software to hardware implementation of an epigenetic network results in improvements in resource utilisation and speed performance.
- Demonstrated that networks with epigenetic elements will not always utilise them, and that only tasks of a certain level of complexity will trigger the evolution of epigenetic behaviours.
- Produced data on the performance of epigenetic networks when applied to various permutations of a foraging robot task, including a version simulating a search and rescue scenario.
- Produced data showing that it is theoretically possible for the epigenetic elements of a network to react to maintain the network's performance when confronted with stimulus that it has not already been optimised for.

1.5 Thesis Organisation

This thesis is organised into three parts. Chapters 2-3 will cover the background of this work, detailing the origins and biological basis of epigenetic networks and genetic algorithms. Chapters 4-5 will then cover the specific implementations of epigenetic networks created for this work, with Chapter 5 in particular describing the hardware architecture. Chapters 6-9 concern the various experiments carried out to ensure the functionality and demonstrate the capabilities of the hardware epigenetic network.

Chapter 2 Background of epigenetic networks, with a particular focus on the biological mechanisms that inspired them, and their predecessors: the artificial neural and gene regulatory networks.

Chapter 3 Discussion of genetic algorithms, including their biological inspiration and functionality, as well as the specific implementations relevant to this thesis.

Chapter 4 Description of the software epigenetic network architecture used in this work; comparisons between it and the original Turner networks; as well as discussion of the translation from floating-point to integer mathematics; and the reduction of precision via lowing of the network's bit width.

Chapter 5 Description of the hardware epigenetic network architecture used in this work, including the mechanisms to facilitate parametrisation and parallelisation of the network; with the former being in service to investigation of bit width reduction, and the latter being a benefit that dedicated hardware can bring about.

Chapter 6 Initial outline of experiments; the implementation of the inverted pendulum model and simulated robot, as well as the fitness metrics used in each case; then a detailed discussion of the idea mentioned in Section 1.2: that epigenetic networks can maintain performance when encountering un-optimised for conditions, something conceptualised within the idea of a dynamic environment.

Chapter 7 Detailed look at experiments involving the inverted pendulum, and results from the same. This chapter includes the successful demonstration of the hardware network, as well as experiments relating to the reduction of precision; and the first experiments involving dynamic environmental conditions, in the form of impulse injection, which provide promising results.

Chapter 8 Detailed look at experiments involving simulated robots performing a foraging task, and results from the same. These include experiments with both a basic and more complex foraging task, which the former showing how the development of epigenetic behaviour is not a forgone conclusion. The second set of dynamic environment experiments are present in this chapter, with the dynamics taking the form of moving foraging targets. These are not as successful as those in Chapter 7.

Chapter 9 Detailed look at experiments involving simulated robots performing a more complex foraging task, conceptualised as a search and rescue, and results from the same. This includes the initial experiments, which show how the contextualisation of the task brings improvements to the fitness of the networks; and the third set of dynamic environment experiments, which involve the simulation of debris fall. While also not as successful as those in Chapter 7, they are more so than those in Chapter 8.

Chapter 10 Summation of the work conducted in this thesis, the drawing of conclusions, and discussion of the direction of possible future work.

2 Artificial Epigenetic Networks

The ultimate goal of this work is to develop a hardware based epigenetic network architecture, one which is able to bring the advantages such networks have already demonstrated to resource constrained applications. It is therefore vital to first understand the nature of these kind of networks. This chapter details the lineage of Artificial Epigenetic Networks (AENs): first with their predecessors in the form of Artificial Neural and Gene Regulatory Networks (ANNs and AGRNs); then with the epigenetic mechanisms of biology; before finally looking at the creation of AENs themselves by Dr. Alex Turner, and his work applying them to various control problems [6].

2.1 Neural Networks

Tracing their origins back to the neuron models of Warren McCulloch and Walter Pitts [7], artificial neural networks seek to replicate the information processing architecture of the organic brain. In addition to the obvious strength of being the only processing paradigm known to be capable of implementing intelligent thought, the brain's neural networks have other features that are considered highly desirable in computing systems [8]:

- Fault Tolerance and Robustness. Neural cells die regularly without impacting the overall performance of the brain, and significant functionality can be maintained even after extensive damage, such as in the famous case of Phineas Gage [9].
- Flexibility. Not only is the brain able to cope with fuzzy or inconsistent input, but it can adjust to completely new tasks and input sets through learning.
- Highly Parallel. The adult human brain has an average of 86 ± 8 billion neurons, and an additional 85 ± 10 billion non-neuronal cells, each of which is capable of functioning independently [10].
- Low Power. Most estimates place the power consumption of the human brain the region of 12-20W [11].

2.1.1 Neuron Model

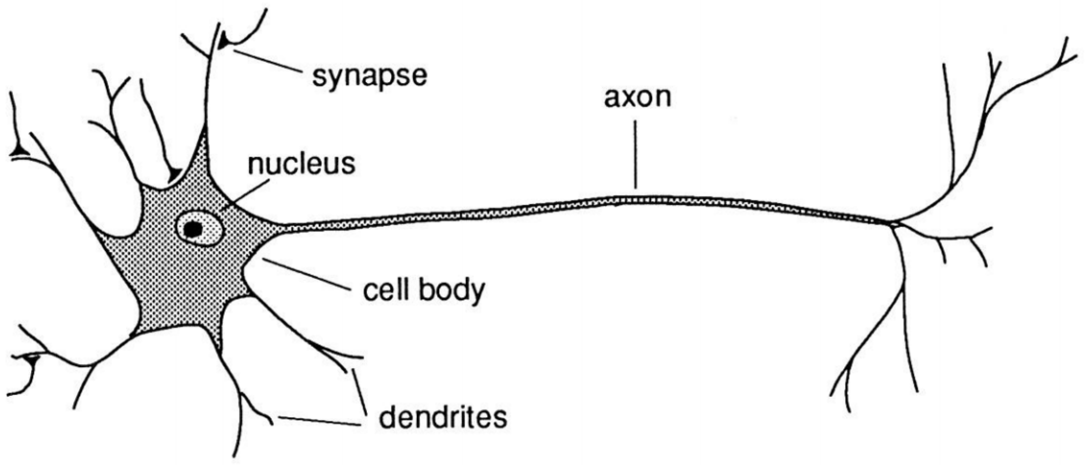


Figure 2.1.1. Schematic of a typical neuron, showing the synapses, dendrites and axon. Taken from [8].

Figure 2.1.1 shows an example of a neuron, the primary repeating unit of an organic brain. Each neuron has a number of dendrites, branching fibres that form connections with other neurons via electro-chemical junctions called synapses. The synapses closest to the cell body of the neuron can be conceptualised as its inputs, with signals that arrive at them from other neurons which alters the electrical potential within the cell itself. Synaptic connections that raise this potential are referred to as excitatory, while those that lower it are inhibitory. This is because when the electrical potential reaches a certain level, known as the threshold voltage, a pulse called the action potential is fired along the neuron’s axon, which can be through of as its output. More synapses are located at the end of the axon, facilitating connections with other neurons [8] [12].

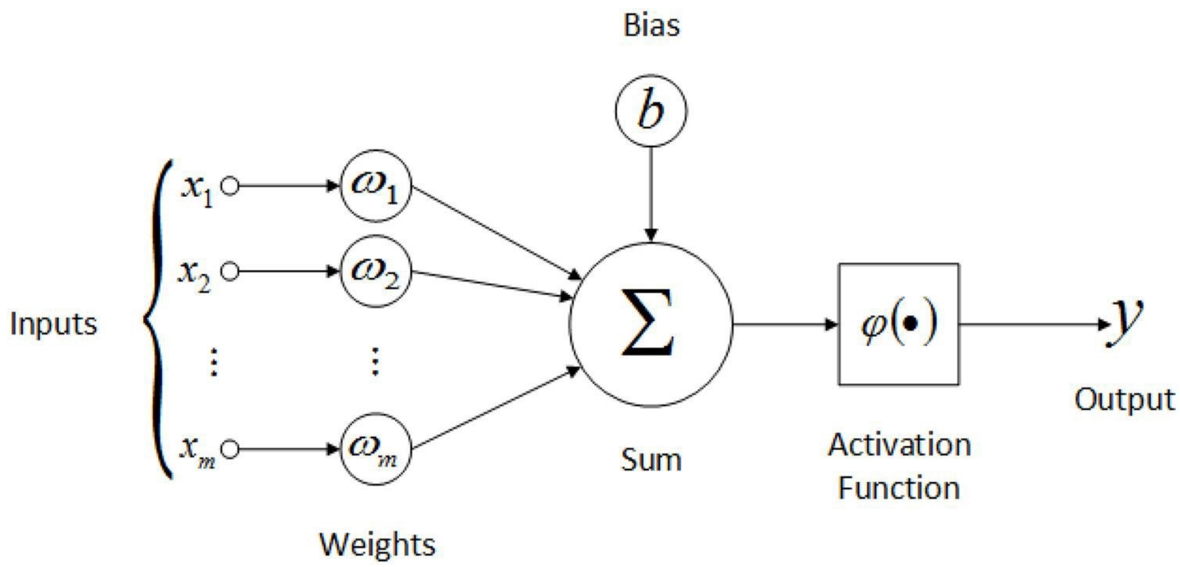


Figure 2.1.2. McCulloch-Pitts neuron model. Taken from [13].

Figure 2.1.2 shows the McCulloch-Pitts model of the neuron, which has become the basis of most artificial neural networks since they proposed it in 1943 [7]. The input synapsis and dendrites are represented by the weighted inputs, with the weights allowing connections to be excitatory or inhibitory. There is also a bias value, which enables adjustment of the threshold at which the neuron fires. The weighted inputs and bias are summed together, producing a value that is fed to the activation function, sometimes also called the firing function. Based on the weighted sum, the function determines if the neuron produces an output. Equations 2.1.1 and 2.1.2 summaris this behaviour, with n being the number of inputs; X_i being inputs; W_i being weights; S being the weighted sum; b being the bias; φ() being the activation function and Y being the output [8].

$$\boldsymbol{S = \sum_{i=1}^{n} X_i W_i} \qquad (2.1.2)$$

$$\boldsymbol{Y = \varphi(S - b)} \qquad (2.1.2)$$

The original McCulloch-Pitts neuron used 1 and 0 as the input/output values; -1 and +1 as weights and a simple step for the firing function, shown in equation 2.1.3.

$$\boldsymbol{Y = \begin{cases} 1 & if\ S \geq 0; \\ 0 & Otherwise. \end{cases}} \qquad (2.1.3)$$

This the far from the only activation function however, and as table 2.1.1 and figure 2.1.3 show there are a range of behaviours that can be achieved.

Table 2.1.1. A selection of neuron activation functions, their equations and the range of neuron outputs they produce.

Activation Function	Equation	Output Range
Sigmoid [14]	$Y = \frac{1}{1 + e^{-S}}$	[0, 1]
Hyperbolic Tangent [15]	$Y = \frac{(e^{S} - e^{-S})}{(e^{S} + e^{-S})}$	[-1, 1]
Rectified Linear Unit [16]	$Y = \begin{cases} S \; if \; S \geq 0 \\ 0 \; if \; S < 0 \end{cases}$	[0, ∞]
Leaky Rectified Linear Unit [17]	$Y = \begin{cases} S & if \; S \geq 0 \\ 0.01S & if \; S < 0 \end{cases}$	[-∞, ∞]
SoftPlus [18]	$Y = \ln(1 + e^{S})$	[0, ∞]

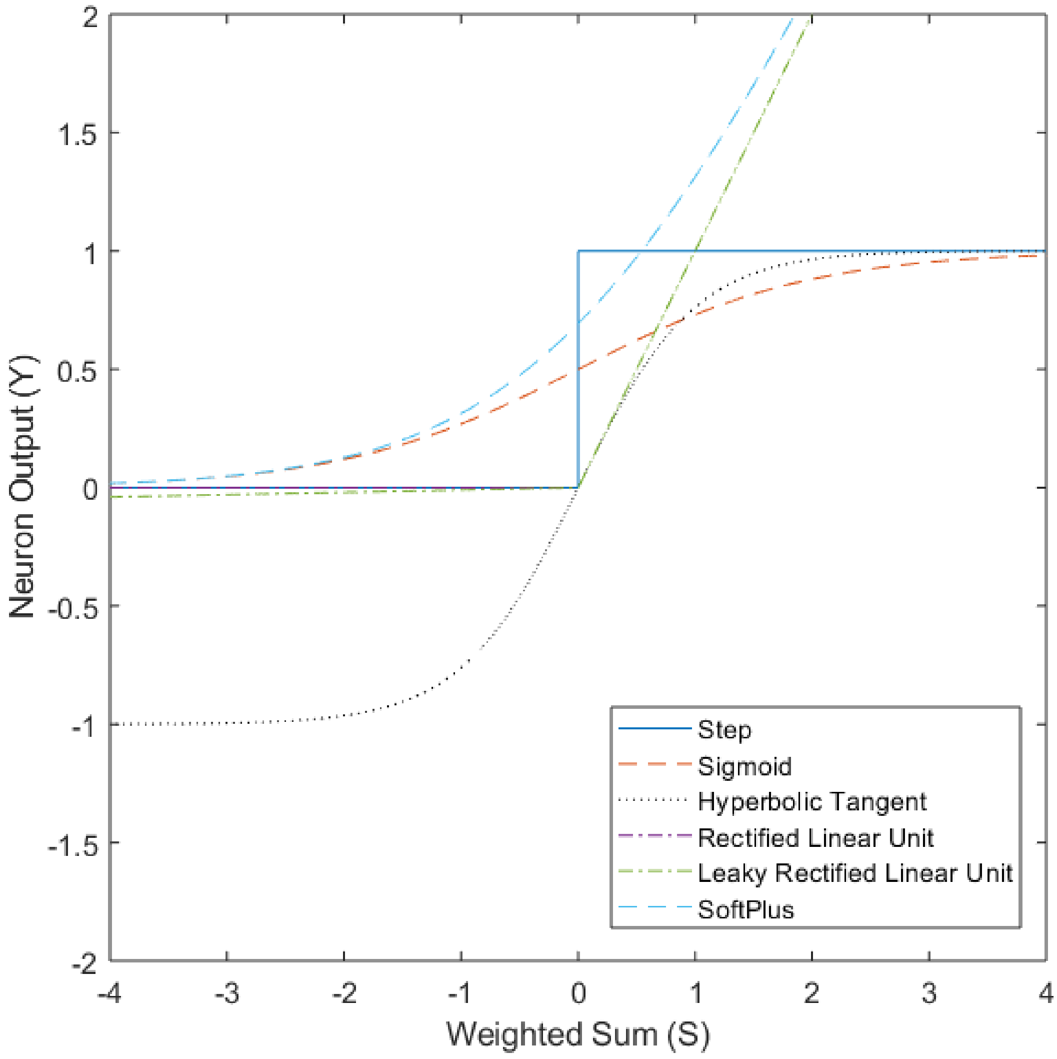


Figure 2.1.3. Plots of the activation functions from table 2.1.1, as well as the step function from equation 2.1.2. S range ± 4.

2.1.2 Multi-Neuron Networks

In both biological and artificial contexts, a single neuron is limited in its utility. For example, in 1958 the Perceptron neuron, an improved version of the McCulloch-Pitts neuron that substitutes the Boolean weights for continuous values in the range -1 to +1, thus allowing for inputs to have different degrees of importance [19] was devised. However, in 1969 it was shown to be incapable of computing the XOR function [20].

Table 2.1.2. Truth table for an XOR function with two inputs

Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1.2 shows the possible input combinations and the outputs they should produce. Taking this and the activation function of the neuron, the step function from equation 2.1.2, it is possible to express the weights as the inequalities shown in equations 2.1.3 through 2.1.6.

$$\boldsymbol{Input_1 = 0,\ Input_2 = 0 \Rightarrow W_1 In_1 + W_2 In_2 = 0 \Rightarrow 0 < b} \qquad (\mathbf{2.1.3})$$

$$\boldsymbol{Input_1 = 0,\ Input_2 = 1 \Rightarrow W_1 In_1 + W_2 In_2 = W_2 \Rightarrow W_2 \geq b} \qquad (\mathbf{2.1.4})$$

$$\boldsymbol{Input_1 = 1,\ Input_2 = 0 \Rightarrow W_1 In_1 + W_2 In_2 = W_1 \Rightarrow W_1 \geq b} \qquad (\mathbf{2.1.5})$$

$$\boldsymbol{Input_1 = 1,\ Input_2 = 1 \Rightarrow W_1 In_1 + W_2 In_2 = W_1 + W_2 \Rightarrow W_1 + W_2 < b} \qquad (\mathbf{2.1.6})$$

From the inequalities in equations 2.1.4, 2.1.5 and 2.1.6; it is clear to see why the XOR function isn't implementable, as it requires the weights to be greater than zero on their own, but less than it if added together [21].

However, if additional neurons are introduced the XOR function becomes trivially computable.

Table 2.1.3. Truth tables for neurons acting as NAND, OR and AND gates.

Input 1	**Input 2**	**Neuron 1 (NAND)**	**Neuron 2 (OR)**	**Neuron 3 (AND)**
0	0	1	0	0
0	1	1	1	0
1	0	1	1	0
1	1	0	1	1

$$\boldsymbol{Input_1 = 0,\ Input_2 = 0 \Rightarrow W_1In_1 + W_2In_2 = 0 \Rightarrow 0 > b} \qquad (\mathbf{2.1.7})$$

$$\boldsymbol{Input_1 = 0,\ Input_2 = 1 \Rightarrow W_1In_1 + W_2In_2 = W_2 \Rightarrow W_2 \geq b} \qquad (\mathbf{2.1.8})$$

$$\boldsymbol{Input_1 = 1,\ Input_2 = 0 \Rightarrow W_1In_1 + W_2In_2 = W_1 \Rightarrow W_1 \geq b} \qquad (\mathbf{2.1.9})$$

$$\boldsymbol{Input_1 = 1, Input_2 = 1 \Rightarrow W_1In_1 + W_2In_2 = W_1 + W_2 \Rightarrow W_1 + W_2 < b} \ (\mathbf{2.1.10})$$

Equations 2.1.7 through 2.1.10 are the weight/bias inequalities for neuron 1, which is acting as a NAND gate. Solving them gives $W_1 = -1$, $W_2 = -1$, and b = -1.

$$\boldsymbol{Input_1 = 0, Input_2 = 0 \Rightarrow W_1In_1 + W_2In_2 = 0 \Rightarrow 0 < b} \qquad (\mathbf{2.1.11})$$

$$\boldsymbol{Input_1 = 0,\ Input_2 = 1 \Rightarrow W_1In_1 + W_2In_2 = W_2 \Rightarrow W_2 \geq b} \qquad (\mathbf{2.1.12})$$

$$\boldsymbol{Input_1 = 1,\ Input_2 = 0 \Rightarrow W_1In_1 + W_2In_2 = W_1 \Rightarrow W_1 \geq b} \qquad (\mathbf{2.1.13})$$

$$\boldsymbol{Input_1 = 1, Input_2 = 1 \Rightarrow W_1In_1 + W_2In_2 = W_1 + W_2 \Rightarrow W_1 + W_2 > b} \ (\mathbf{2.1.14})$$

Equations 2.1.11 through 2.1.14 are the weight/bias inequalities for neuron 2, which is acting as an OR gate. Solving them gives $W_1 = 1$, $W_2 = 1$, and b = 1.

$$\boldsymbol{Input_1 = 0, Input_2 = 0 \Rightarrow W_1 In_1 + W_2 In_2 = 0 \Rightarrow 0 < b} \qquad (\mathbf{2.1.15})$$

$$\boldsymbol{Input_1 = 0, Input_2 = 1 \Rightarrow W_1 In_1 + W_2 In_2 = W_2 \Rightarrow W_2 < b} \qquad (\mathbf{2.1.16})$$

$$\boldsymbol{Input_1 = 1, Input_2 = 0 \Rightarrow W_1 In_1 + W_2 In_2 = W_1 \Rightarrow W_1 < b} \qquad (\mathbf{2.1.17})$$

$$\boldsymbol{Input_1 = 1, Input_2 = 1 \Rightarrow W_1 In_1 + W_2 In_2 = W_1 + W_2 \Rightarrow W_1 + W_2 > b} \ (\mathbf{2.1.18})$$

Finally, Equations 2.1.15 through 2.1.18 are the weight/bias inequalities for neuron 3, which is acting as an AND gate. Solving them gives $W_1 = 1$, $W_2 = 1$, and b = 1.5. By combining these three neurons together in the manor shown in figure 2.1.4, the XOR function is implemented.

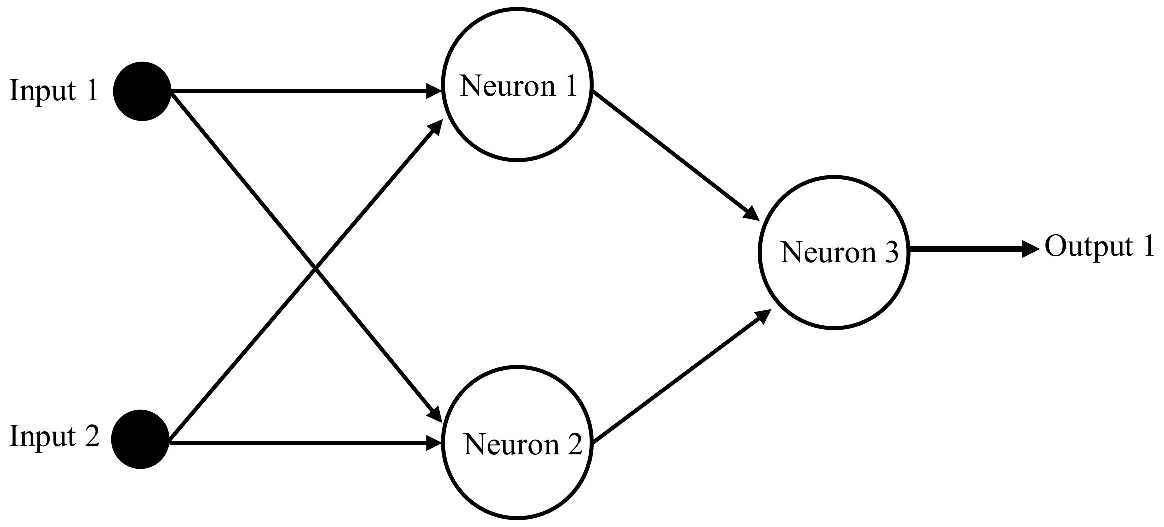


Figure 2.1.4. 3 neuron network for computing the XOR function.

This simple example demonstrates how combining neurons together in networks allows more complex functions to be implemented.

2.2 Gene Regulatory Networks

The brain is not the only aspect of biological organisms that performs information processing. Although it does so on a different timescale and using significantly different mechanisms, the genetic material that shapes organic creatures physiology is also a rich source of inspiration for computational architectures.

2.2.1 Biological Genomes

The genomes of most biological life are comprised of long chains polymers called DeoxyriboNucleic Acid (DNA). The repeating unit of these chains, shown in figure 2.2.1, are called nucleotides and consist of a "backbone" of deoxyribose sugar and phosphoric acid, along with one of four complex bases: Adenine, Guanine, Thymine and Cytosine (abbreviated to A, G, T and C) [22].

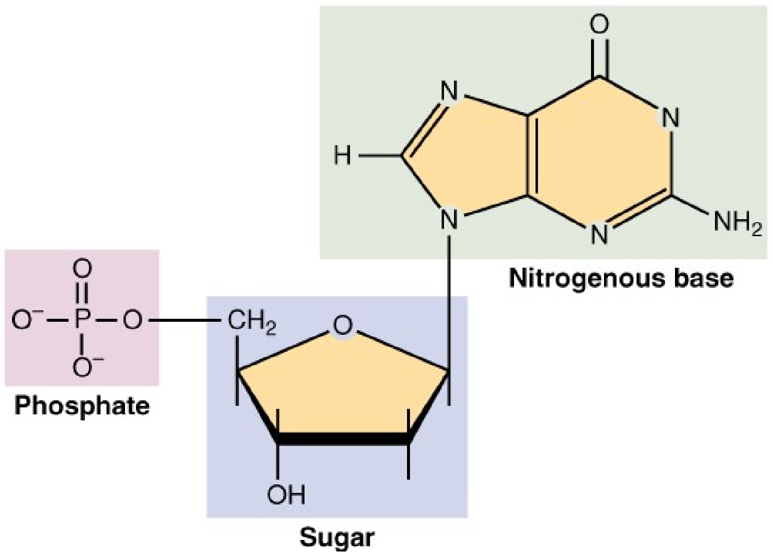


Figure 2.2.1. A single DNA nucleotide, consisting of phosphate/sugar backbone and a base, in this case Adenine. Taken from [23].

The nucleotides are joined together by bonds between the phosphate/sugar backbones, creating the polymer. Each strand of DNA is comprised of two of these polymer chains, jointed together by bonds between the bases, as illustrated in figure 2.2.2.

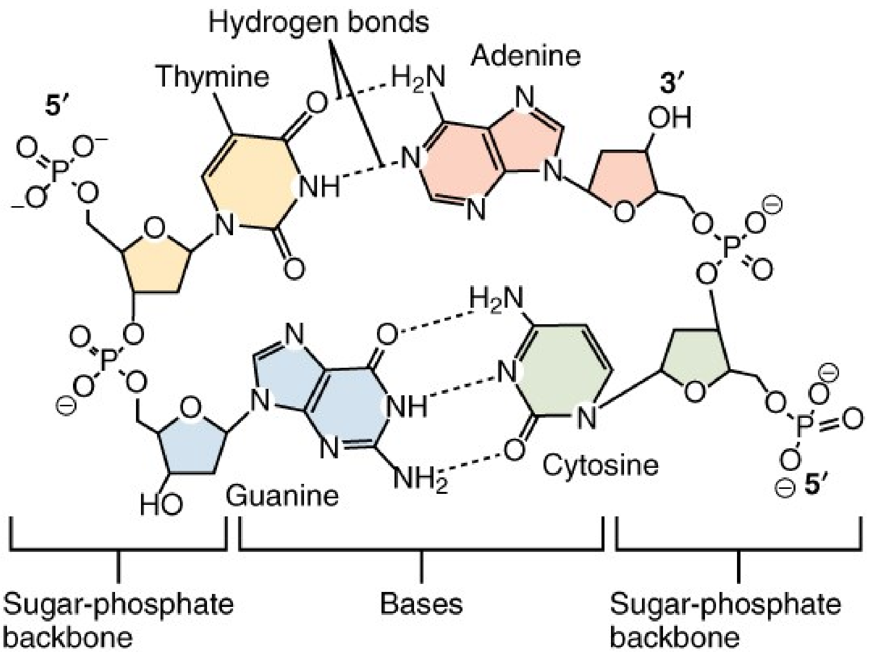


Figure 2.2.2. A small section of a complete DNA strand, consisting of two nucleotide polymer chains joined by bonds between their respective bases. Taken from [23].

It should be noted that the bases can only bond in two configurations: A with T and C with G. The sequence of bases along a DNA strand form a code, which in simple terms is often described as a blueprint for a particular organism. More specifically, however, DNA encodes the instructions needed to create proteins. Proteins are complex macromolecules that are vital for biological life, as they perform a vast array of functions [24]:

- Structural proteins form frameworks, such as collagen in tendons and cartilage in skeletons.
- Contractile proteins, such as actin and myosin in muscles, facilitate movement.
- Transport proteins, carry molecules throughout organism's tissues, with haemoglobin being the most well-known.
- Regulatory proteins, such the hormone insulin, control biochemical reactions.
- Protective proteins, such as antibodies and clotting factors, combat infective agents and facilitate healing.
- Storage proteins hold molecules for future use, such as ferritin which stores iron.
- And perhaps the most important, Enzymes, which act as catalysts for biochemical reactions like the release of energy from glucose; or the transcription of DNA itself.

Like DNA, proteins are also polymers, with their repeating monomer being the amino acid. Amino acids are themselves quite complex, consisting of a shared core of a hydrogen atom, a carboxyl group and an amino group; bonded to a so-called R group, which is different for every amino acid. Each gene with a genome encodes the complete sequence of amino acids that comprise a single protein, as well as other information that will be discussed in section 2.2.3. Each amino acid is represented in this code by a sequence of three bases, referred to as a codon. With the four possible base pairs, there are enough codons to represent 64 amino acids. In reality however most organisms (including humans) only have 20 of them, along with their respective codons [23].

Note that, even with some amino acids having multiple codons, there are only 61 base combinations accounted for. The remaining three are referred to as the termination or stop codons, as they indicate the end of an amino acid sequence.

2.2.2 Protein Synthesis

Protein synthesis is the process of transforming the “blueprint” of DNA into the actual protein. It consists of two stages, shown in figure 2.2.3: Transcription, during which the section of DNA coding for a particular gene is used to create a corresponding section of mRNA; and Translation, during which tRNA segments manoeuvre the amino acids into position.

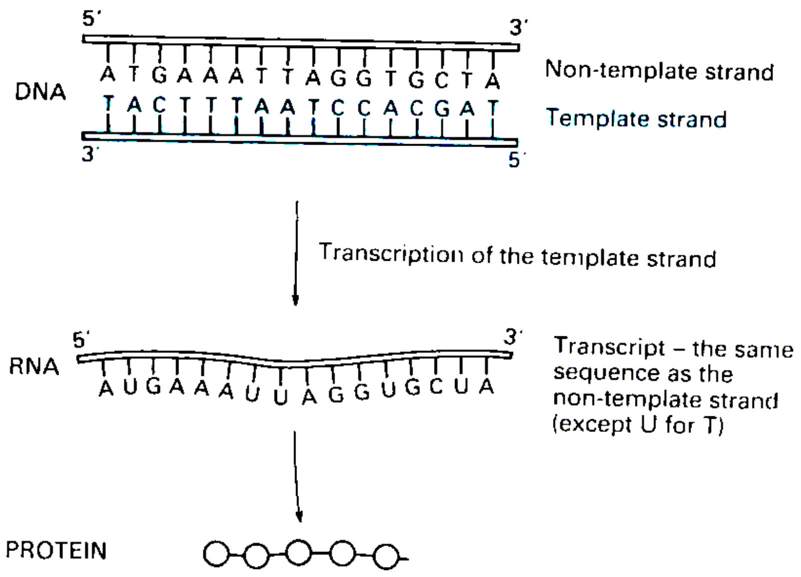
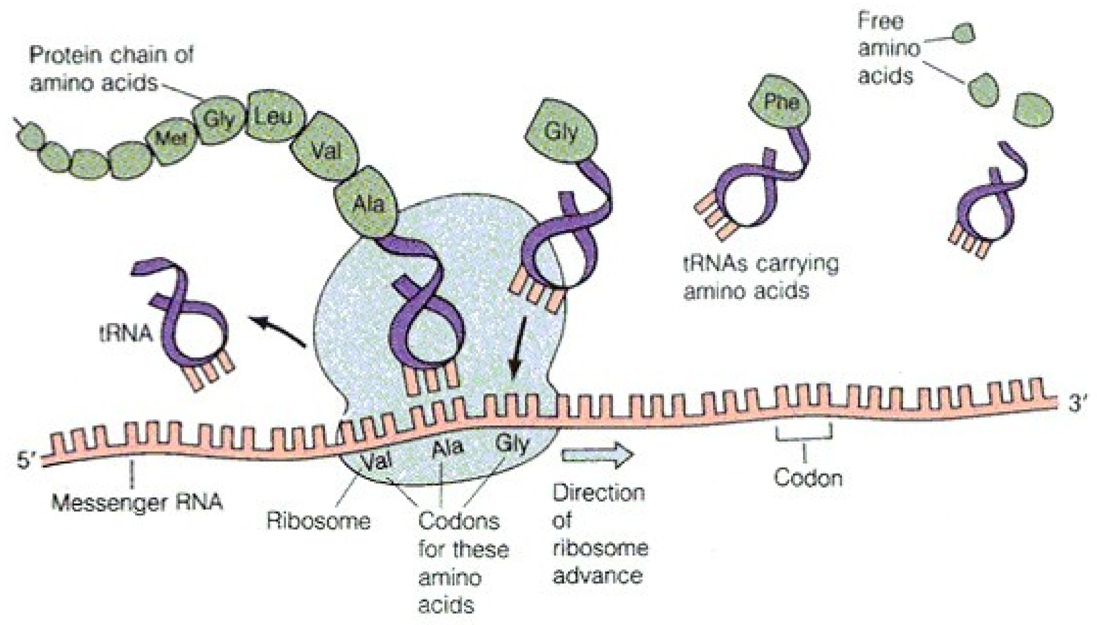

DNA
5'
3'
A T G A A A T T A G G T G C T A
T A C T T T A A T C C A C G A T
3'
5'
Non-template strand
Template strand
Transcription of the template strand
RNA
5'
3'
A U G A A A U U A G G U G C U A
Transcript – the same sequence as the non-template strand (except U for T)
PROTEIN

Figure 2.2.3. Simplified illustration of the protein synthesis process. Taken from [25].

RNA is RiboNucleic Acid, another complex polymer that is involved in these cellular processes. As its name would suggest, it is similar in structure to DNA, but with two differences: first, the base Thymine is replaced by the base Uracil; and second, it only consists of a single polymer chain. It is this second difference that is the most important, as it allows RNA to form bonds and adopt more complex shapes than DNA [25]. mRNA, or messenger RNA, is perhaps the simplest of these shapes, as it is an RNA strand intended to carry the amino acid sequence to the ribosome, the cell’s protein synthesis machinery. It is produced by a protein called RNA polymerase, which latches onto a single strand of DNA at the start of a gene (a location called a promotor). The polymerase then moves down the strand, bonding together RNA nucleotides to produce the RNA strand. Once completed, mRNA has a number of additional features added, the most significant of which is a special non-codon sequence of bases called the ribosome binding site. This provides a point for the ribosome, the protein that mediates the translation process, to latch onto the mRNA strand.


Free amino acids
Protein chain of amino acids
Met
Gly
Leu
Val
Ala
Gly
Phe
tRNAs carrying amino acids
tRNA
5′
3′
Val
Ala
Gly
Messenger RNA
Ribosome
Codons for these amino acids
Direction of ribosome advance
Codon

Copyright © 2000 Benjamin/Cummings, an imprint of Addison Wesley Longman, Inc.

Figure 2.2.4. The process of translation, where an amino acid polymer is formed based on the sequence contained within a strand of mRNA. Taken from [26].

Once the ribosome is connected to the mRNA, it moves along it codon by codon. It is at this stage that tRNA, or transport RNA, is involved. As shown in figure 2.2.4, tRNA collects amino acids, transports them to the ribosome. Each tRNA has a nucleotide sequence on it that corresponds to the codon for the amino acid they carry, so they can connect to the mRNA. This hold the amino acid in place, ready for the ribosome to facilitate the bond between it and the next amino acid in the sequence. When a stop codon is reached, instead of a tRNA section, a protein called a release factor connects, which disconnects the ribosome from the mRNA strand, and releases the completed amino acid polymer [25].

2.2.3 Gene Regulation

A complete biological genome is present in every cell in an organism's body, but not every cell requires all of the proteins it codes for all of the time. Some proteins, like ribosomes and others involved in DNA transcription, are always needed, so their respective genes, called housekeeping genes, are always active or expressed. Non-housekeeping genes however, have their expression controlled through a variety of mechanisms, general collectively referred to as gene regulation mechanisms.

An excellent example of one such mechanism is the so called regulatory circuit of the lac operon of *E. coli*. *Escherichia coli* is a bacterium that is an example of a model organism, a lifeform that is well understood by biologists and hence utilised widely in experiments (other model organisms include *D. melanogaster*, the fruit fly; and *C. elegans*, the nematode worm). Operons are clusters of genes in bacteria and other prokaryotic organisms that code for sets of proteins that work together to complete some larger task [24]. The lac operon codes for three proteins: permease, β-galactosidase and transacetylase; which together form the mechanism, shown in figure 2.2.5, by which *E. coli* breaks down the complex sugar lactose into the simpler galactose and glucose sugars [3].

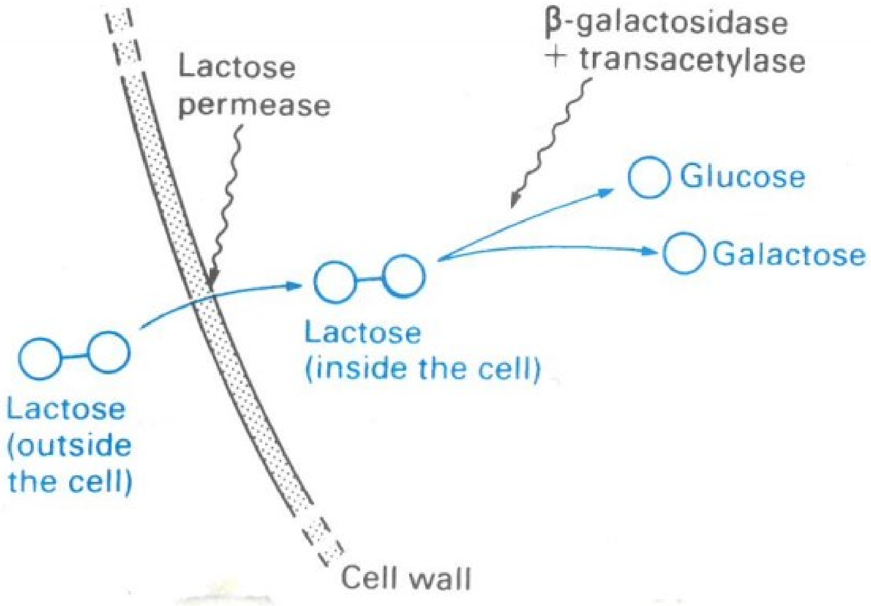


Figure 2.2.5. The lactose utilisation mechanism of E. coli. Permease transports lactose past the cell wall, where β-galactosidase and transacetylase break it down into its constituent sugars. Taken from [25].

The amount of these proteins within the cell varies with the amount of lactose present, something which is caused by a gene slightly before the lac operon called *lacI*. *LacI* codes for a protein called the lactose repressor, which under normal circumstances bonds to the promotor of the lac operon, as shown in figure 2.2.6. This prevents the RNA polymerase from connecting there to initiate transcription of the genes, thus preventing the synthesis of the lactose breakdown proteins.

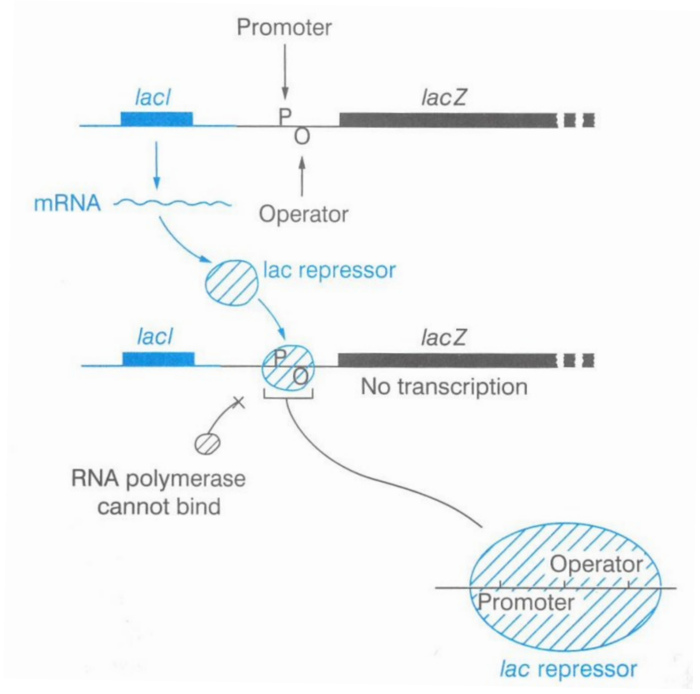


Figure 2.2.6. Action of the lacl gene and lac repressor protein under normal conditions. Taken from [25].

However, as figure 2.2.6 shows, something interesting occurs when lactose in added to the equation. When *E. coli* encounters a quantity of lactose, a small amount is able to permeate the cell wall and is converted to a slightly altered form called allolactose. Allolactose is able to bond with the lac repressor protein before it can bond to the lac operand promotor. This results in the transcription of the genes proceeding, creating the proteins needed to utilise the newly found lactose [3].

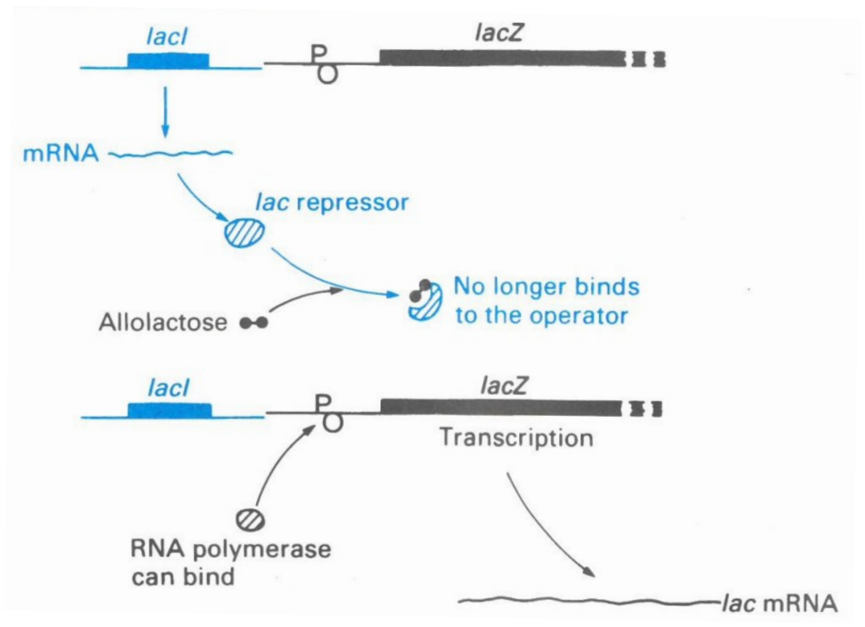


Figure 2.2.7. Blocking of the lac repressor protein by allolactose. Taken from [25].

While other mechanisms of gene regulation exist, this regulatory circuit illustrates the process. It is easy to see how more complex regulatory mechanisms could be built up, though the addition of more parts.

2.2.4 Artificial Gene Regulatory Networks

As alluded to at the start of this section, gene regulation mechanisms have been taken as inspiration for computational methods. One of the first examples is Kauffman's random Boolean networks [27], which modelled genes as binary units which were either on or off. Each gene was randomly connected to other genes, and possessed a random truth table that specified its on/off state based on the state of the other genes. Despite their simplicity, it is easy to see how such a network would be able to replicate the behaviour of a gene regulatory circuit like the one detailed in section 2.2.3.

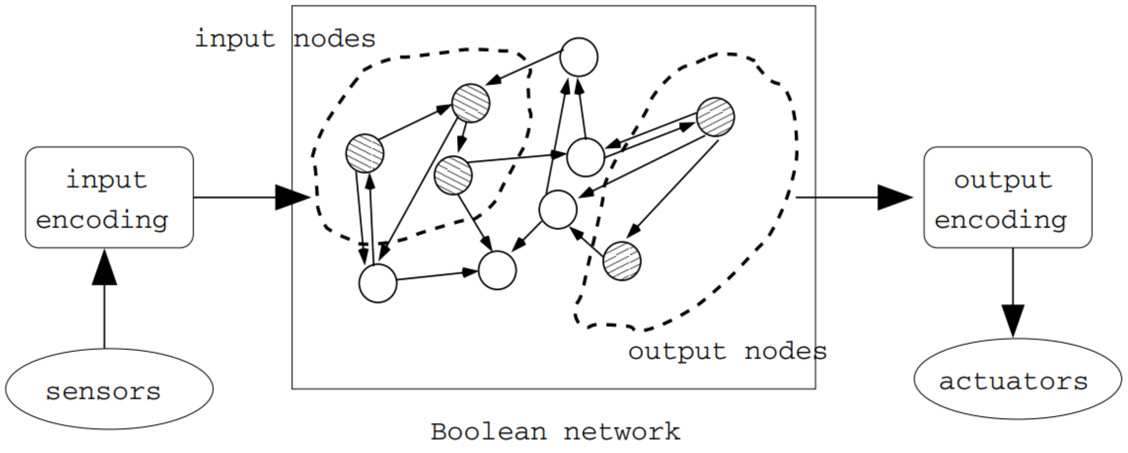


Figure 2.2.8. A Boolean network for robotic control, together with its input/output mechanisms. Taken from [28].

As for computational application, such networks have been successfully used in applications such as robotic control [28], though they are limited by their use of Boolean expression values [29]. Continuous Valued Discreet Time Gene Regulatory Networks (CDGRNs) are an improvement upon Boolean networks, as they solve the limited expression value issue by instead using a continuous value range, such as decimal values between 0.0 and 1.0. Instead of a truth table, CDGRNs use a regulatory function that maps the inputs from other genes to an output expression value [14]. This should sound familiar, as this version of a gene regulatory network is comparable to the neural networks described in section 2.1. In fact the sigmoid activation function, noted in table 2.1.1, is one of the more commonly used regulatory functions [14].

2.3 Epigenetic Networks

Gene regulation as described in section 2.2 is not the only mechanism which alters the expression of genes within an organism. Coming from the Greek and literally meaning "above" or "over" genetics, epigenetic mechanisms operate upon the genome, but they are distinct from it. Dr. Nessa Carey uses the analogy of a play script [4]. The original script is the genome, with dialog being the protein genes and stage directions being the regulatory genes. Several different directors now receive copies of the script, ready to produce the play, and make their alterations. Some are impermanent, directions given to actors, production design decisions; these are non-hereditary epigenetic changes, they alter the expression of the genome, but not the genome itself, and are not carried forward. However, some of the directors write on the script, cross things out, add things in etc. These copies of the script might then pass to other directors, who use them to produce their own versions of the play. These are the hereditary changes, like those discussed at the start of this section: the genome (text of the play) remains the same, but the alterations to their expression are carried forward.

2.3.1 Biological Mechanisms

While often presented as such, DNA does not float freely within the nucleus of the cell, at least not in complex organisms. Instead it is confined to structures called chromosomes, which are shown in figure 2.3.1. As the figure illustrates, chromosomes are formed by tightly coiling together fibers called chromatin. In turn, the chromatin fiber is comprised of units called nucleosomes; which consist of a DNA strand wrapped around a cluster of proteins called histones [30].

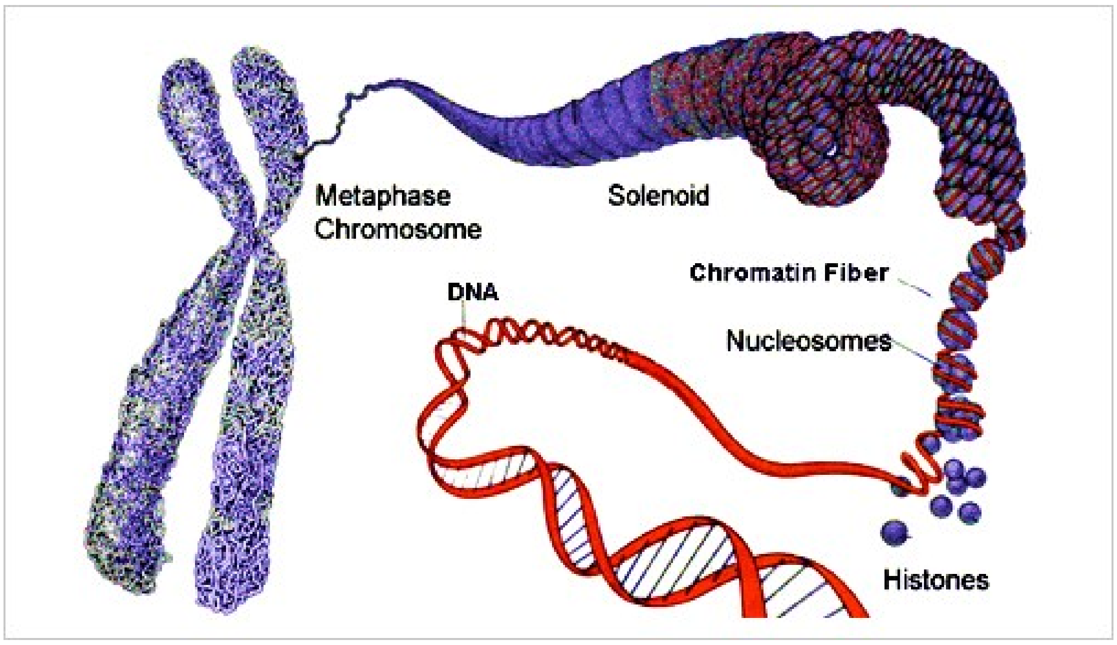


Figure 2.3.1. Structure of a chromosome, showing the coiling of DNA and histones to from chromatin fibers. Taken from [31].

Originally thought to be primarily a packaging method, it is now understood that the structure of chromatin fibers plays a role in altering the expression of the genes wrapped up within them. At the centre of each nucleosome are eight histones, collectively called a histone octamer, of which there a four types: H2A, H2B, H3 and H4 (the histone H1 performs a different function). All of these histones consist of a protein core and two tails of amino acids. These tails allow other molecules to bond to the histones, which can have a variety of different effects from initiating transcription to reacting to DNA damage. Of interest to this work however, is that some modifications such as methylation can repress the transcription process [30].

Occurring in H3 and H4, histones methylation is the addition of one or more methyl groups to instances of the amino acids lysine and arginine in one of the tails of a histone. One of these amino acid instances, referred to as H3K9, has the important property that its methylation forms a binding site for the protein Chromobox Homolog, more commonly called Heterochromating Protein 1 (HP1) [32]. HP1 alters the shape of the chromatin fiber, taking from the more standard Euchromatin to the tightly packed Heterochromatin, both of which are shown in figure 2.3.2. This chromatin compacting has the knock on effect of blocking sections of DNA from being accessed by transcription factors, preventing their expression.

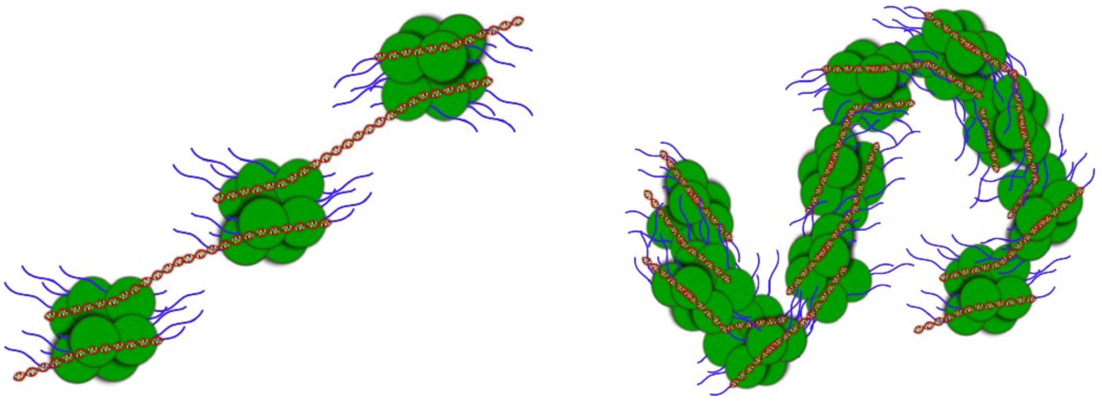


Figure 2.3.2. Non-compact Eurchromatin, left; and tightly compacted Heterochromtin, right. Taken from [33].

2.3.2 Artificial Networks

Inspired by these additional mechanisms of genetic expression control, Dr. Alex Turner created the first artificial epigenetic networks in 2013 [6] [33]. Building upon the CDGRNs described in section 2.2, artificial epigenetic networks add a layer of control that is able to supress the normal functioning of the processing elements, genes, of the network.

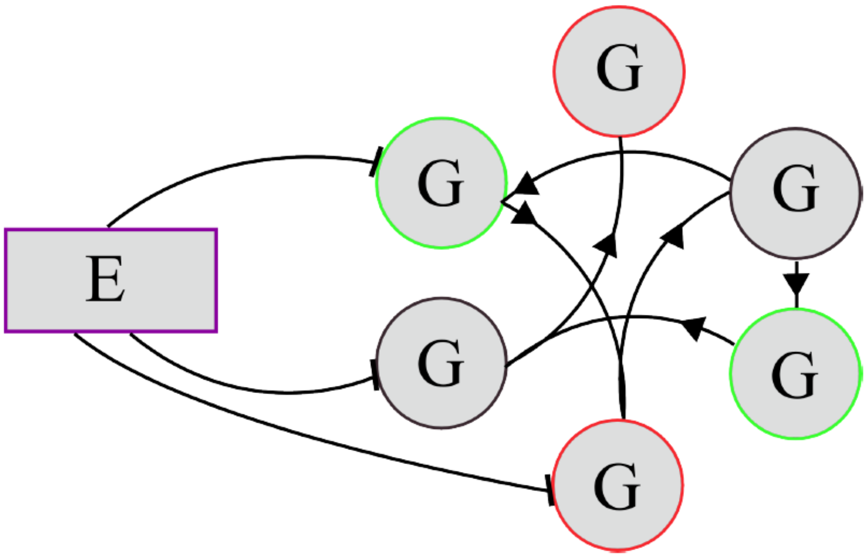


Figure 2.3.3. A simple illustration of a gene regulatory network, with the addition of an epigenetic mechanism. Taken from [6].

The initial version of an epigenetically augmented network, called an Artificial Epigenetic Regulatory Network (AERN), uses Boolean switches as the epigenetic mechanism. A number of genes are connected to each switch, which would have a predetermined condition of activation. When activated, the epigenetic switch would inhibit the activity of its connected genes, as shown in figure 2.3.5.

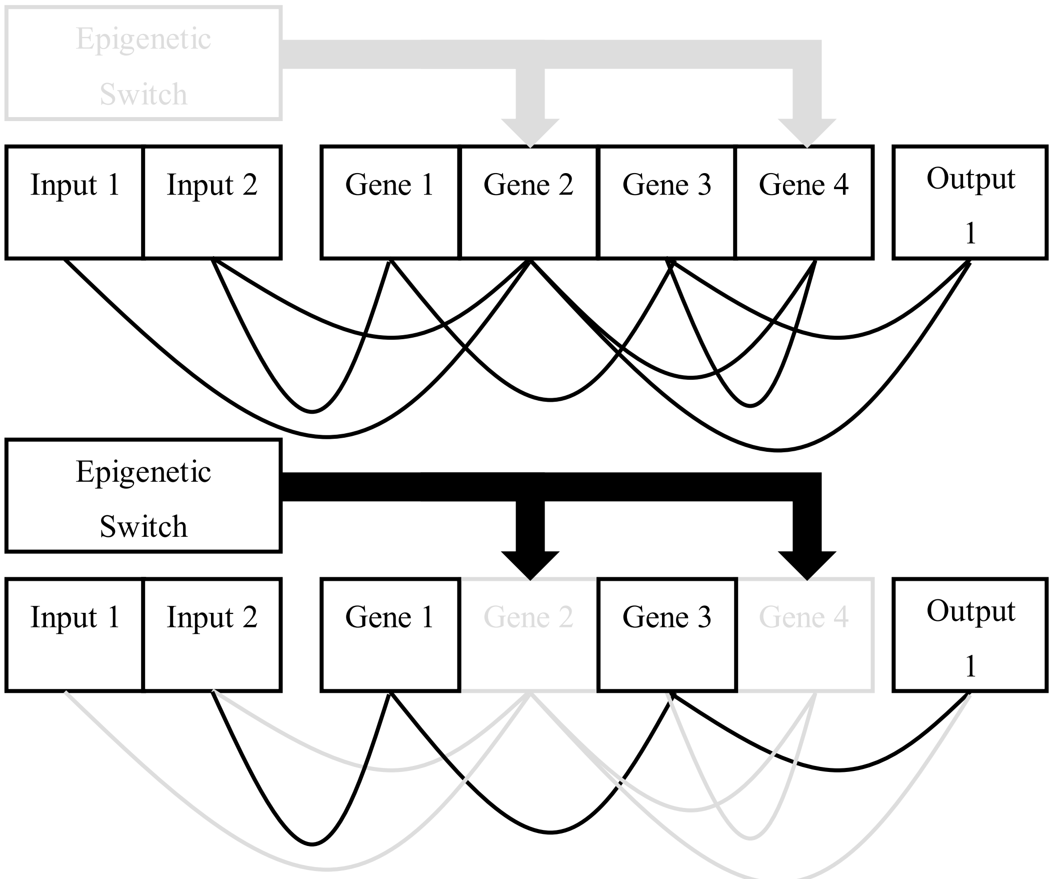


Figure 2.3.4. An AERN with epigenetic switch inactive, top, and active, bottom. Note how the activation of the epigenetic mechanism, and hence suppression of the genes, dramatically changes the characteristics of the network.

A more sophisticated version, simply called the Artificial Epigenetic Network (AEN), replaces the Boolean switches with elements similar to the genes themselves. That is to say processing elements that take in a selection of input values, then apply a function to determine their output. As it is this version that will be expanded upon in this work, a more in-depth look at its architecture and mechanisms will be given in Chapter 4.

Turner was able to demonstrate that these kinds of networks are able to outperform their more standard gene regulatory network counterparts in various example problems, such as: navigating a Chirikov’s standard map; controlling single and multiple coupled inverted pendulums; and the control of transfer orbits in gravitational systems [6].

2.4 Summary

This chapter has traced the development of the epigenetic network architecture, from both its biological origins and the computing paradigms that inspired it. As already mentioned, Chapter 4 will further expand upon the AENs introduced in this chapter, while Chapter 3 continues to look at the aspects of this work that owe their origins biological inspiration, by covering the subject of evolution and genetic algorithms.

3 Genetic Algorithms

Given the expansive range of parameters that an Artificial Epigenetic Network (AEN) possess (as Chapter 4 will show, a small network with 2 inputs, 4 genes, 2 molecules and 2 outputs has at least 40 parameters to set), manual design is not a realistic option. Artificial Neural Networks (ANNs) can solve this problem by making use of learning algorithms, a wide range of different methods that allows an ANN's parameters to gradually be adjusted until the network is trained to solve a particular problem [34]. Unfortunately, no such algorithms exist for Artificial Gene Regulatory Networks (AGRNs), so for one to be applied to an AEN would require development from first principles. However, a more straightforward solution exists in the form of genetic algorithms; a method that can be used to solve complex optimisation problems by evolving a population of individuals towards a solution.

This chapter details the background and theory of evolutionary methods, of which genetic algorithms are an example. It will then detail the particular instances that are employed within this work: a simple, single objective implementation called the Simple Genetic Algorithm; the slightly more complex Single Sample Generational Evolution algorithm; and finally the multi-objective Non-dominated Sorting Genetic Algorithm II (NSGA II).

3.1 Background

3.1.1 Biological Evolution

In a very broad sense, evolution simply means the gradual change of something over time, although nowadays it is far more commonly used to refer to the specific process of biological evolution through natural selection. Evolution through natural selection is the process by which biological life diversifies and adapts itself to its environment [35]. The core of this theory was summarised by biologist Ernst Mayr as follows [36]:

- Every species is fertile enough that if all offspring survived to reproduce, the population would grow.
- Despite periodic fluctuations, populations remain roughly the same size.
- Resources such as food are limited and are relatively stable over time.
- Thus, a struggle for survival ensues.
- Individuals in a population vary significantly from one another.

- Much of this variation is heritable.
- Individuals less suited to the environment are less likely to survive and less likely to reproduce; individuals more suited to the environment are more likely to survive and more likely to reproduce and leave their heritable traits to future generations, which produces the process of natural selection.
- This slowly effected process results in populations changing to adapt to their environments, and ultimately, these variations accumulate over time to form new species.

To elaborate upon this an example will be used, in the form of Darwin's finches; four of which are shown in figure 3.1.1. In numerical order they are: a Large Ground Finch, a Medium Ground Finch, a Small Tree Finch and a Green-Warbler Finch.

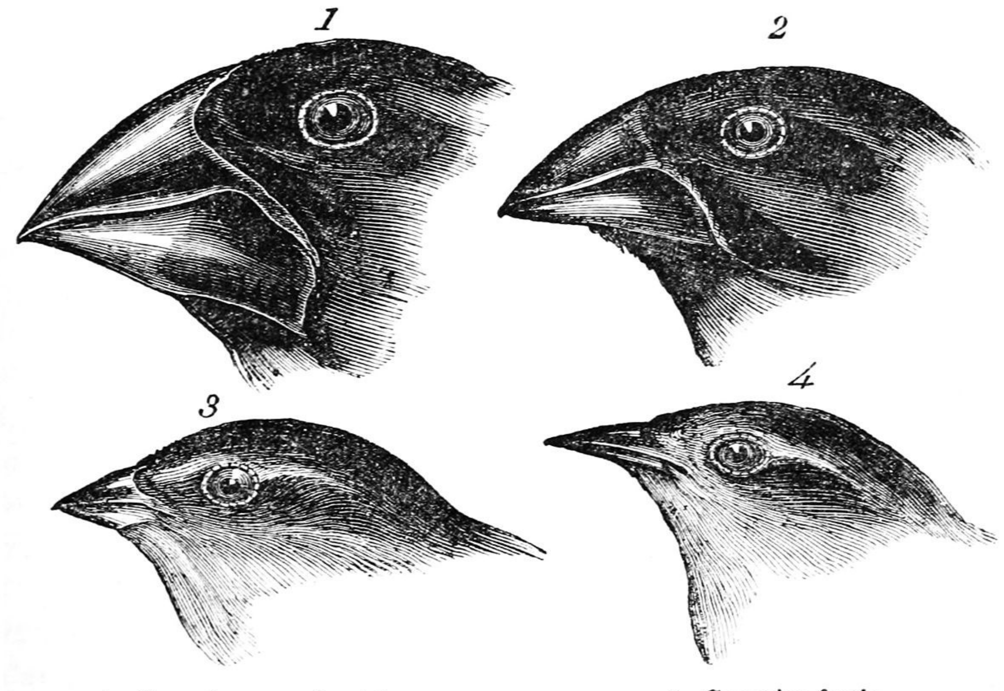


Figure 3.1.1. Four out of the fifteen species of Galapagos finches, more commonly known as Darwin's finches. Drawn by John Gould and taken from [37].

All the finches shown in figure 3.1.1 share a common ancestor, a historical species that was the first to spread to the Galapagos Islands. As this species multiplied, it encountered a variety of differing environmental niches, especially with respect to available food sources. Due to the natural degree of variation within a species, say for example in beak size, some finches happened to be more adept at exploiting some of these sources: a slightly stronger beak might enable a finch to open nuts; a thinner beak to reach grubs within trees; and so on. A finch better equipped to feed itself would be more likely to breed, as it would be both healthier and more able to provide food for its offspring; offspring which would inherit the same physiological differences that provided their parent with an advantage. Over hundreds, if not thousands of generations, these small changes accumulate to create completely different species of finch which look quite different from one another: the slightly stronger beak is now much larger and perfectly adapted for nut cracking, as with the large ground finch, or possibly the consumption of hard seeds like its medium counterpart; while the grub eaters like the Green-Warbler finch possess beaks almost comparable to a spear.

This diversification also illustrates a very important fact about evolution through natural selection: none of the species in figure 3.1.1 are better than the others overall, because each one of them is optimised for the particular environmental niche that they occupy. It is because of this that evolutionary biologists decry the phrase "Survival of the Fittest" as basically meaningless, as more accurate description might be "Proliferation of the Suitable". It is also important to note that, in biological evolution, there is no guiding force, either in the introduction of changes or the elimination of less suitable individuals/promotion of suitable ones. It is on this point that artificial evolutionary methods diverge from biological evolution by natural selection, in that evolutionary methods do possess such a guiding force: the engineer employing them.

3.1.2 History of Evolutionary Methods

Credit for the idea to translate the principles of evolution via natural selection into the field of engineering can be shared between two individuals: Prof. John Henry Holland of the University of Michigan and Prof. Ingo Rechenberg of the Technical University of Berlin. During the 1960s and 70s, both of them developed the idea of employing genetic and evolutionarily inspired methods of designing and adapting artificial systems, with Rechenberg's work being his doctoral thesis, built around the utilisation of an early genetic algorithm to design aircraft wings [38].

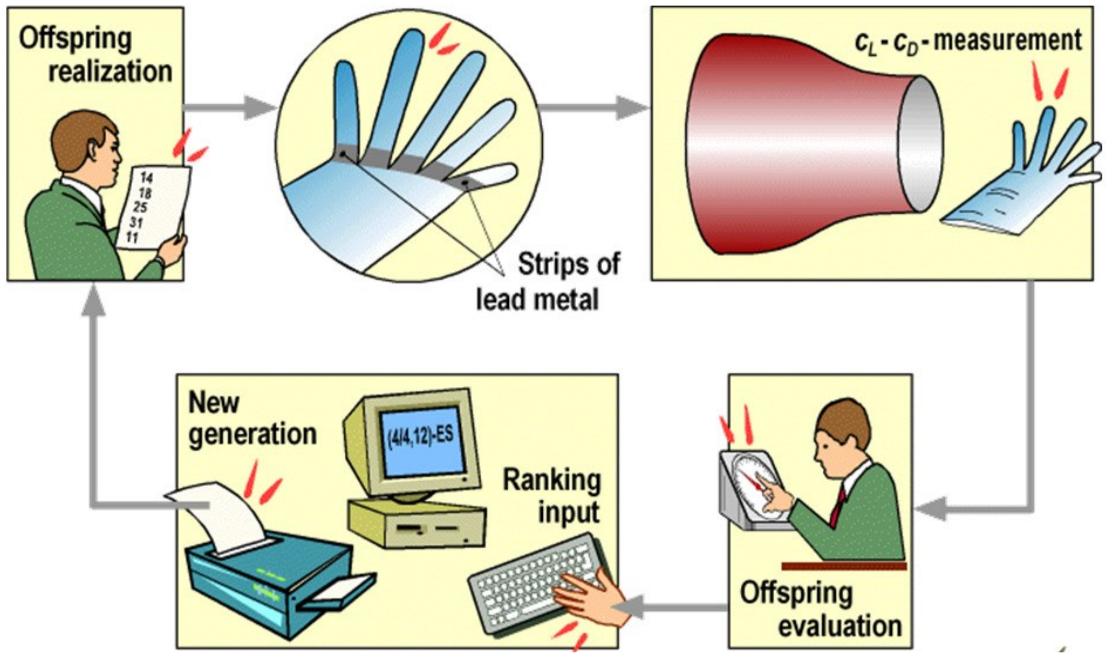


Figure 3.1.2. Rechenberg's evolutionary algorithm, applied to the design of aircraft winglets. Picture taken from [38].

Figure 3.1.2 shows the evolutionary process employed by Rechenberg, which can very clearly be compared directly to biological evolution. In place of, say, a finch, there is instead a model of an aircraft wing, with variations in the angle of its wingtip elements. Instead of the acquisition of food, there is aerodynamic efficiency, and natural selection is replaced by the intelligent selection of individuals that better meet design requirements. These are then used to create a new generation of possible designs, and the process repeats.

This process: creating a population of individuals, evaluating them, selecting the best, creating a subsequent generation and then iterating; is still the core of generic algorithm design. As technology has developed, the automation of the algorithms has improved: while Rechenberg needed to manually alter the shape of the wing model for each variation, put it into a physical wind tunnel and record the results; a modern version of this experiment could be performed entirely within a computer, utilising simulation tools able to automatically produce and analyse the possible wings. Indeed, Akira Oyama utilised this approach in 2000 to optimise the aerodynamics of transonic aircraft wings [39]. Because of the flexibility afforded by this more modern methodology, Oyama was able to modify 11 different parameters of his wing design, something that would have been completely impractical for Rechenberg.

3.1.3 Structure of a Genetic Algorithm

With a basic grounding established, it is now possible to look at the specific stages of a genetic algorithm in greater detail. In order, these are:

1. Genotype Definition
2. Generation of Initial Population
3. Genotype/Phenotype Mapping
4. Evaluation
5. Selection Operation
6. Generation of Offspring

With stages 3 through 6 repeating until an ending condition is met, usually either a number of generations, for more open ended problems; or an individual reaching a target evaluation result [40].

3.1.3.1 Genotype Definition

In the biological world, the element responsible for the transmission of traits from one generation to the next is its genome, stored within its DNA. As already discussed in section 2.2, DNA is a complex molecule that is able to encode information in the pattern of its sub-elements, called base pairs: Adenine (A), Thymine (T), Cytosine (C), and Guanine (G). Each sequence of three is referred to as a codon, and each codon relates to either a specific amino acid, the building blocks of proteins; or functions as an "instruction" to the protein synthesis "machinery". A complete set of control and amino acid codons encodes the needed information to create a single protein, with multiple proteins combining together to build a part of an organism. This total summation of an organism's genetic information is referred to as its genotype, with the morphology that results from it being the phenotype [35]. The first step in creating a genetic algorithm is to define how the genotype, which is what will be carried down the generations, encodes its phenotype, the thing that is actually being designed. In the case of both Rechenberg and Oyama's work, the phenotype was wing geometry, and the genotype consisted of a number of measurements of described that geometry e.g. winglet angles for Rechenberg [38] [39]. Note that in these cases, the genotype directly describes some aspect of the phenotype, whereas in biological systems the mapping is much more indirect.

3.1.3.2 Generation of Initial Population

With genotype structure defined, the first step of the algorithm proper is the generation of an initial population. This will consist of a number of genotypes, produced in a number of possible ways, though the most usual is to simply generate them randomly. Other possibilities include: seeding the population with genotypes that relate to known good phenotypes, such as the best existing design; or building a population out of copies, or slightly altered copies, or the same genotype. This does limit the diversity of the population, whether or not this is a bad thing depends on the desires of the algorithm's designer [40].

3.1.3.3 Genotype/Phenotype Mapping

In order to evaluate the individual solution that the genotypes within the population relate to, they need to be translated into their phenotype form. Referring back to Rechenberg and Oyama: Rechenberg's translation required the manual adjustment of a physical model of an aircraft wing; while Oyama was able to offload this work to software which generated a simulated model based on the parameters encoded by the genotype [38] [39].

3.1.3.4 Evaluation

Now that the phenotype exists, it can be evaluated to determine its suitability for the task. This can be broken down into two stages: testing and assessment. In the testing stage, the phenotype is exposed to some range of environments or stimuli, while its performance and responses are measured. With Rechenberg, this involved placing the physical wing model within a wind tunnel, and then measuring its aerodynamic performance with various instruments; Oyama's testing was the same, but performed using simulated wind tunnel [38] [39]. This, together with the equivalent comparison from section 3.1.3.3 illustrates why most current genetic algorithms utilise computer models and simulations, as it dramatically reduces resource and time requirements, while increasing automatability and parallelisability. However, there are still arguments for utilising physically realised phenotypes, particularly in fields like robotics, as it is often impractical to encapsulate all the factors and variations of the real world within a simulator. This can lead to designs created using computer simulation for evaluation having reduced performance, or even failing completely, when taken into the real work, an issue called the "reality gap" [41].

With data gathered from the evaluation, whichever form it has taken, the suitability of the individual can now be assessed and a "score" given. This "score" is referred to as the individual's fitness, and the means of calculating it as a fitness function. The creation of a fitness function for a particular application is complex, especially in more open-ended situations. There are tasks, such as optical character recognition, where the finished design must produce a particular response to a particular stimulus, and as such the fitness directly relates to this. However, consider a more complex problem, such as evolving a simulated robot able to walk, an example of one of the many problems looked at by Karl Sims [42]. Sims' work was centred on simulated "creatures", whose morphology and control systems could be altered via evolution to suit a specific task, specified by the fitness function used. In his initial work on evolving locomotion, the fitness of a simulated creature was based solely on its average horizontal velocity, with Sims' reasoning being that a better locomotion method would be able to move faster. This instead reliably produced creatures like those shown in figure 3.1.3.

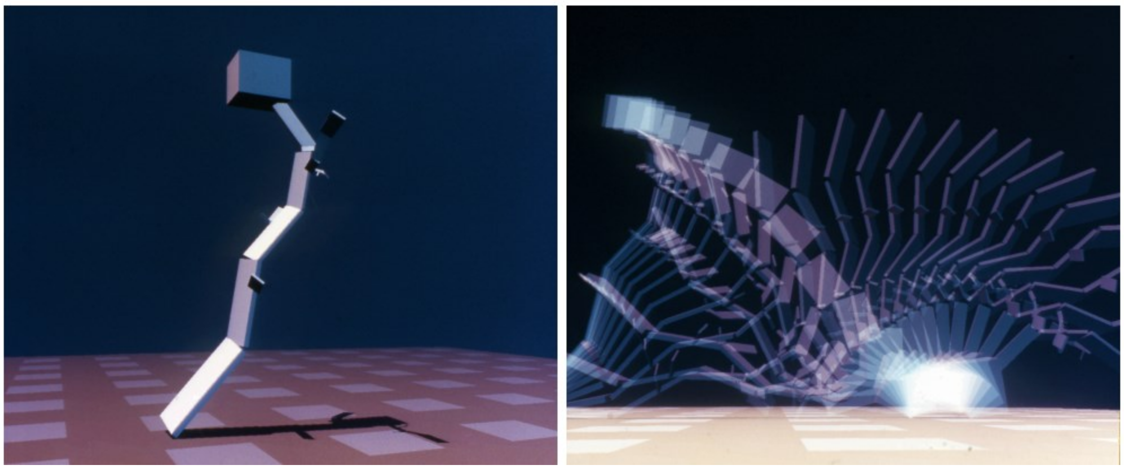


Figure 3.1.3. An example of a flawed creature (left) and its unconventional method of locomotion (right). Taken from [42].

Rather than evolve anything that could be characterised as limbs or a gait, the creatures instead evolved to be very tall and top heavy. When simulated, they would fall forward, sometimes even managing to "somersault" to maintain momentum for longer. While far from the intended outcome, these falling creatures were able to dominate because of their performance exactly met the specification of the fitness function: maximise horizontal velocity [43].

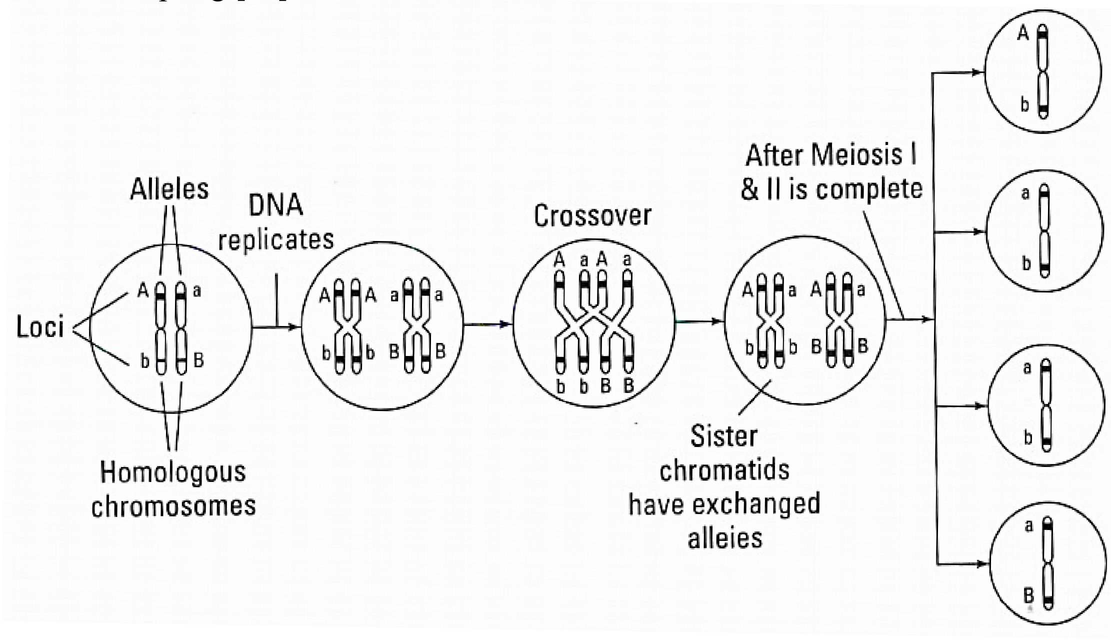
3.1.3.5 Selection Operation

Once every individual in the population has been assessed and evaluated, the next stage is to select which of them will contribute to the next generation: either by being directly carried over to it; or as parents to new individuals (offspring). There are a wide variety of stratagems, of which a few of the most common are:

- Tournament Selection. One of the simplest selection methods, in a tournament selection algorithm, a group of q individuals are taken, often randomly, from the population and compared to one another. The best individual is then either placed directly in the next population, or becomes a parent. In the latter case, the other parent will either be chosen from the same group of q by some other metric; or by a second tournament selection [44].
- Rank-Based Selection. Rather than drawing a small group from the population and then choosing from it, rank-based selection works by ordering the entire population based on fitness, and then weighting the selection of individuals to move the new population/become parents based on this ranking [45].
- Diversity-Based Selection. Similar to rank-based selection, but instead of the weighting being proportional to the fitness of an individual, it is instead based on how different each individual is to one another. This is intended to maintain the diversity of the population, which helps to avoid local optima [46]
- Disruptive Selection. Suggested by Ting Kuo and Shu-Yuen Hwang, disruptive selection penalises individuals whose fitness is close to the population average, instead choosing those with very high, or very low fitnesses. While this stratagem doesn’t work well for all problems, it has been shown to have a positive impact of the diversity of the population [47].
- (μ, λ) Evolution. An example of a deterministic selection method. μ parents will be chosen, which in turn will create λ offspring. The best μ of these offspring will then replace their parents. Note that this does not allow for the possibility of an individual progressing directly from one generation to the next, which means it is possible for the maximum fitness to go down [40].
- ($\mu + \lambda$) Evolution. Like (μ, λ) evolution, but instead of replacing their parents, the offspring and parents are assessed as a group, which allows for parents fitter than their offspring to progress to the next generation in their place [40].

3.1.3.6 Generation of Offspring

Regardless of the method used to select them, the next stage is to use various pairs of parents to create a number of offspring. This involves two processes: crossover, sometimes called recombination; and mutation. In biological evolution, crossover occurs as a part of the process of genetic recombination during the production of gametes during meiosis. The gametes, or sex cells (e.g. eggs and sperm), of most animals are haploid, which means they have half the number of chromosomes as a normal, or diploid, cell. This allows for half a child's genetic information to come from each parent, but would limit the possible combinations of genetic material. However, during the process of meiosis, sections, or alleles, of two comparable, or homologous, chromosomes can swap. This crossing over of material, shown in figure 3.1.4, increases the diversity of a given parent's gametes, and thus their offspring [25].



Alleles
DNA replicates
Crossover
After Meiosis I & II is complete
Loci
Homologous chromosomes
Sister chromatids have exchanged alleies
A
a
b
B

Figure 3.1.4. The biological process of chromosomal crossover during meiosis. Note that without crossover, the only allele combinations would be Ab and aB. With crossover this becomes AB, ab, Ab and aB. Taken from [25].

As the individuals in an artificial system do not have the same sort of variation in chromosomes as a biological organism, the crossover process instead occurs between the two parents and their offspring: sections of the two parent genomes are combined, creating an offspring comprised of aspects of both. Like the selection process, there are various different stratagems for determining which parent will provide which sections:

- Uniform Crossover. A simplistic method, in which the choice between the parents is made independently for each value of the genome. This can either be completely random; or weighted depending on factors like fitness, with the better parent having a greater chance of being chosen [48]. Illustrated in figure 3.1.5.

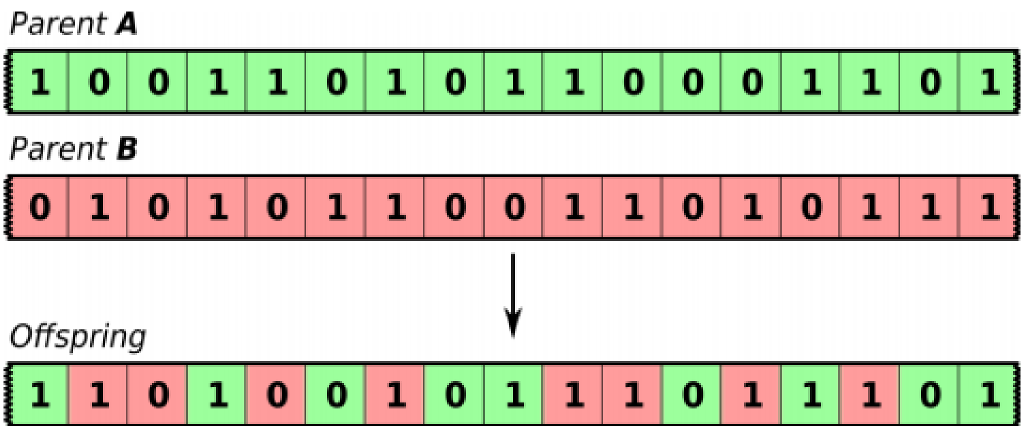


Figure 3.1.5. An example of uniform crossover. Taken from [48].

- N-Point Crossover. Instead of going value by value, the genome of the offspring is instead divided at N points, creating a number of sections (e.g. 2 points = 3 sections, as shown in figure 3.1.6). Each section then comes from one of the parents, with the choice being made in a similar manor to uniform crossover.

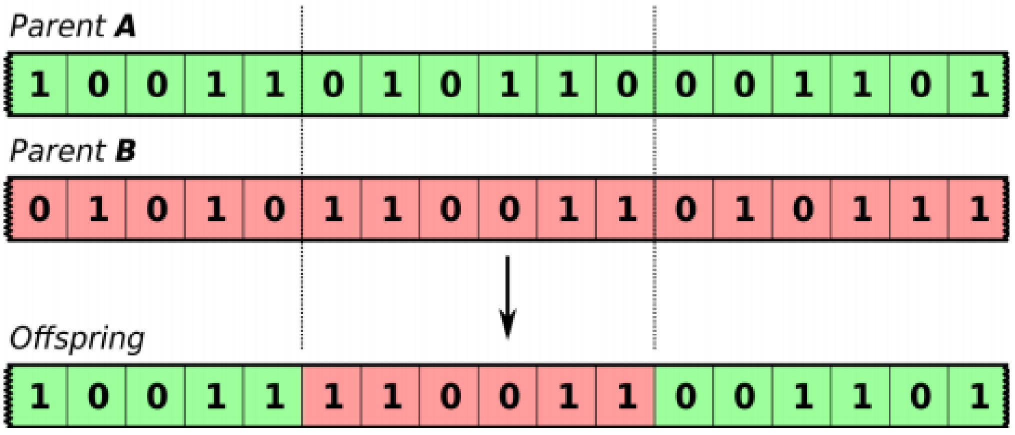


Figure 3.1.6. An example of N-point crossover, where N = 2. Taken from [48].

- Arithmetic Crossover. In arithmetic crossover, neither parent is chosen to contribute to the genome of the offspring. Instead, values are chosen using some function that uses the values from both parents, with the simplest example being the average of the two parents, as shown in figure 3.1.7.

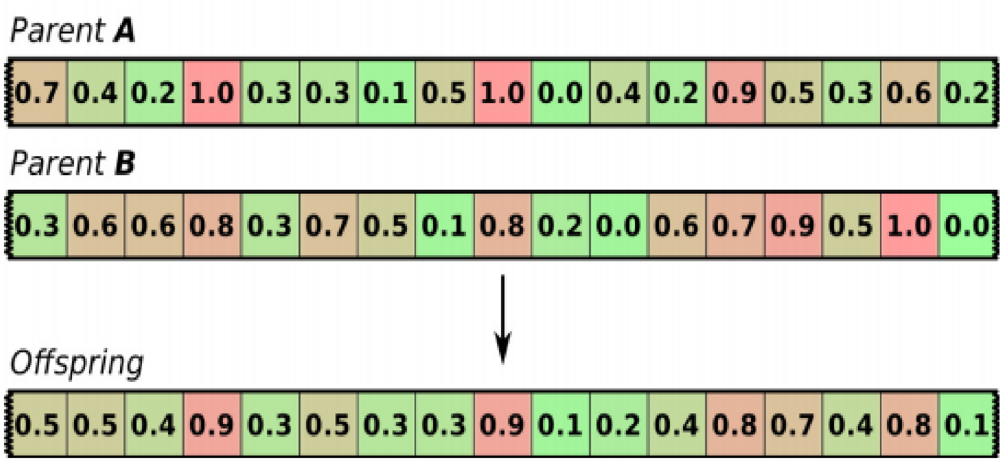


Figure 3.1.7. An example of arithmetic crossover, utilising the mean of the parents. Taken from [48].

After crossover generates the offspring, there then exists the possibility of mutation. In biological organisms, mutation refers to changes in a genome brought about by any one of a huge number of causes, from errors in the process of DNA replication to chemical or radiological exposure. While often associated with negative effects like genetic diseases, mutation is ultimately responsible for all variation in organic life [25]. It is because of this important role that genetic algorithms include a mutation operation, to allow offspring to possess traits beyond those of their parents. Broadly the same method of mutation is used in all cases: a chance exists for each value of the genome to mutate; and if a value is chosen, it is replaced by a different value. The exact value of the mutation chance, frequently called the mutation rate, is important. Too low and very little variation will occur, too high and positive traits won't be passed between parents and offspring. In 1992 Heniz Muhlenbein developed a formula (equation 3.1.1) that posited a relationship between ideal mutation rate, population size and the size of the genome [49].

$$N \times m \times \sqrt{n} = 1.7 \qquad (3.1.1)$$

Where N is the population size, n is the genome length and m is the mutation rate. This broadly states that larger populations with larger genomes should have lower rates of mutation.

With the general structure of a genetic algorithm and its various elements detailed, the following sections will look at the three specific algorithms used within this work, starting with the simple genetic algorithm.

3.2 Simple Genetic Algorithm

A very basic implementation, the simple genetic algorithm is intended for use in the initial experiments to confirm the functionality of the epigenetic networks, as it is similar to the algorithm employed in Turner's work [6], a fact that also allows valid comparisons to be drawn between the two network implementations. An overview of the process is shown in algorithm 3.2.1, while algorithm 3.2.2 elaborates on the new population generation process.

Algorithm 3.2.1 Simple Genetic Algorithm, with a population of M and N generations

```
Generation of initial population of N individuals
for N generations {
    for M individuals in the population {
        Assess fitness of individual
    }
    Generate new population (see algorithm 3.2.2)
}
```

Algorithm 3.2.2 Simple Genetic Algorithm, generation of new population

```
Sort current population by fitness
Copy M/8 best individuals into the new population
for M – M/8 individuals {
    Select two individuals from the best 1/8th to be parents
    Crossover Parents to produce new individual
    Apply mutation operator
    Place new individual into new population
}
```

3.2.1 Selection Method

The simple genetic algorithm uses a form of rank-based selection, with the addition of elitism. The population is ordered by fitness, then the top N individuals are selected, where N is 1/8th of the population size (e.g population size = 128, N = 16). These individuals are copied directly into the new population, while also being used as the pool from which parents are selected to produce offspring. The remaining 7/8th of the new population are new offspring.

3.2.2 Offspring Generation

Uniform crossover is used to produce the initial offspring, with selection between the two parents being based on a crossover rate parameter that biases the fitter parent. For example: a crossover rate of 0.2 means that 80% of the offspring's genome will come from the fitter parent; and 20% from the weaker parent. Following crossover, each offspring passes though the mutation operator, where the mutation rate determines the probability that each value within the genome will be switched with a randomly generated replacement.

3.3 Single Sample Generational Evolution

The single sample generational evolution algorithm is the genetic algorithm integrated into the JBotEvolver simulation package, which is disused in more detail in section 6.2. It is largely the same as the simple genetic algorithm, but with two key differences. First, parents are not directly chosen from the N elite individuals that make up the first portion of the new population, but from a set of elite individuals of size λ. If $N = \lambda$, then the selection functions exactly the same as for the simple genetic algorithm. However, if $N > \lambda$ then the parents are more exclusive, reinforcing the elitism; while $N < \lambda$ allows lower fitness individuals to be parents, aiding genetic diversity. The other difference is that single sample generational evolution has no crossover, with offspring instead being mutated copies of a single parent.

3.4 Nondominated Sorting Genetic Algorithm II

Thus far, all discussion of genetic algorithms, as well as the two implementations in sections 3.2 and 3.3, have been limited to a single objective. That is to say the algorithms are optimising based on only one fitness metric, e.g. aerodynamic performance. However, there are often occasions when multiple factors need to be taken into consideration during the design process. For example, it is possible that a genetic algorithm could produce a design that performed its function perfectly, but which consumed vast amounts of power and cost thousands of pounds to produce. Instead of a perfect design, what these situations call for is something called a pareto-optimal solution: a balance in which all objectives are taken into account [50]. Often, there isn't a single pareto-optimal, rather a set of them that display different trade-offs, much like the old engineering adage: "good, cheap, fast; pick two". In tackling these problems with evolutionary methods, the tool used is called a multi-objective algorithm.

Generally speaking, the primary difference between single objective and multi-objective algorithms is the method by which individuals are selected to proceed to the next generation/produce offspring. Fonseca and Fleming divided the different approaches to this into three broad categories [51]:

- Plain Aggregating Approaches. A sort of half-way approach, plain aggregation works by taking the various different axis of optimisation and combining them into a single fitness, at which point the methods discussed in section 3.1.3.5 can be employed. While considered advantageous for their ability to produce a single solution rather than a group of possible ones, the aggregation approach often requires specialised knowledge of the problem to devise a suitable aggregation function, as the different optimisation axis often do not equally contribute to the final fitness, so weighting must be employed.
- Population-based non-Pareto Approaches. The earliest method applied to the problem of multi-objective optimisation, population-based approaches are built around considering each objective individually, but allowing all of them some contribution to the selection process. An example is the Vector Evaluated Genetic Algorithm (VEGA), devised by J. David Schaffer in 1985 [52]. VEGA divides the new population being produced into a number of sections equal to the number of objectives. Then each section is populated with individuals selected using one of

the objectives, effectively producing a number of single objectively optimised sub-populations. These sub-populations are then shuffled together to produce the final new population. Figure 3.4.1 illustrates this process.

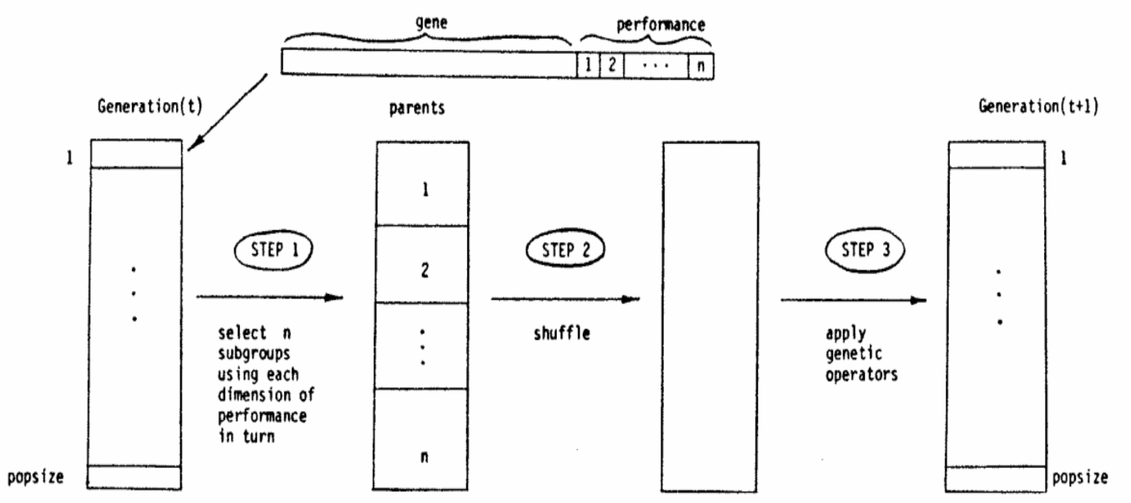


Figure 3.4.1. Population generation method of VEGA. Taken from [52].

Other versions of the population-based approach include: tournament selection where the objective being used for comparison either being randomly selected each time, or being chosen based on a weighting of importance [53] [54]; or the deletion of members of the population with fitness values below a certain threshold for each objective, with the remaining individuals going on to produce offspring [55].

- Pareto-based Approaches. As its name suggest, this approach directly utilises the pareto-optimality of the individuals to assess them, more specifically an aspect of it called domination. In a population, if individual A has a lower fitness in one of its objectives than individual B, then individual A is dominated by individual B. The less individuals that this individual is dominated by, the better it is, with individuals that have the same domination count being grouped together in pareto fronts, as they are considered to be comparably pareto-optimal. The ultimate goal is for a front to be comprised of non-dominated individuals, ones which are better than all others. Selection of individuals is then made based on these fronts, with secondary factors sometimes being employed to improve elements like genetic diversity.

As its name would suggest, the Non-dominated Sorting Genetic Algorithm 2 (NSGA-II) is an example of this final category. Proposed in 2002 by Kalyanmoy Deb et al [56] as an improvement to the previous NSGA [57], NSGA-II starts by assessing the individuals in the population to see how many other individuals dominate them, before sorting them into pareto fronts, also called non-dominated fronts (NDFs) in this instance. It also calculates a value called the crowding distance for each individual, which is based on its proximity to other individuals within the same front, and then sorts each front based on this value. Thus, the algorithm produces a number of groups of individuals of comparable suitability, sorted in order of difference from each other. The algorithm then fills half the new population with individuals selected by working though the fronts in descending order, moving onto the next when the front is depleted. This typically results in all of the first few fronts, and then a fraction of a middling front, with all fronts below it being discarded. Finally, the individuals that have passed into the new population are used as the pool from which parents are chosen, typically at random, to produce the offspring that comprise the second half of the new population. This process is illustrated in figure 3.4.2.

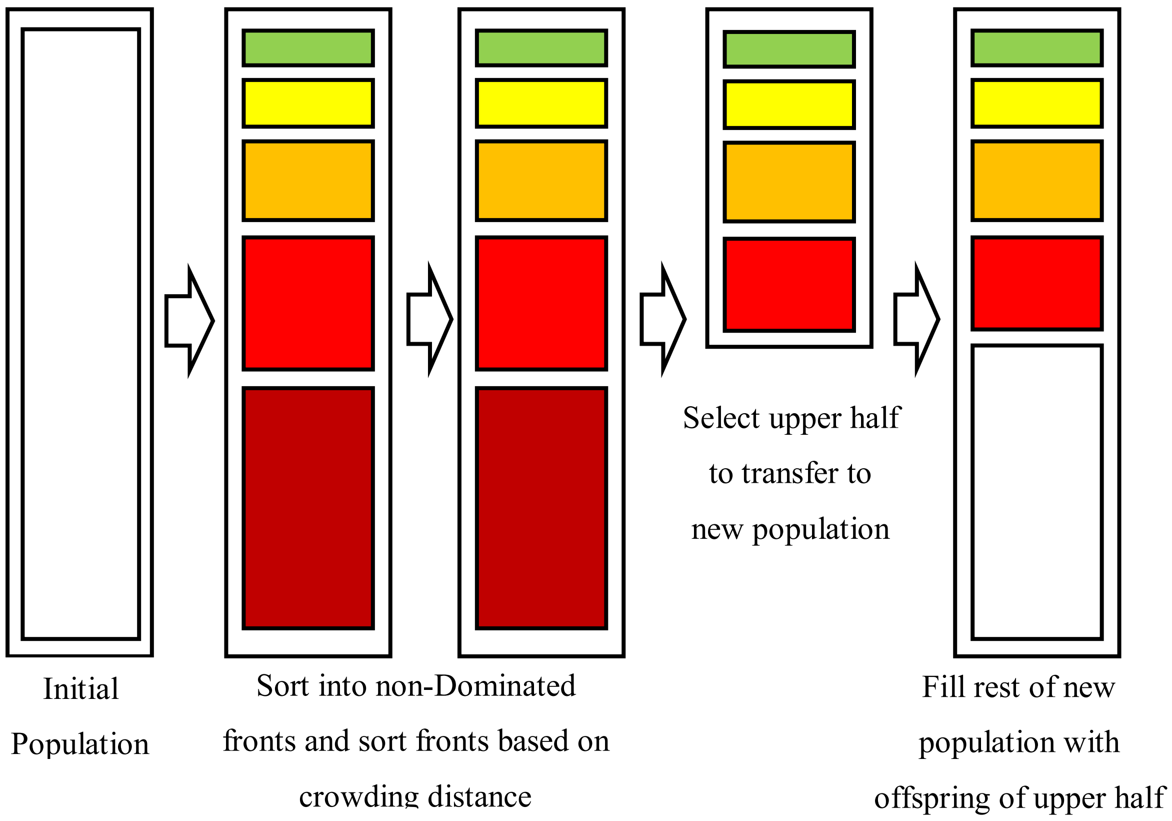


Figure 3.4.2. Illustration of the sorting, selection and offspring generation process of NSGA-II.

In the implementation used in this work, the offspring generation processes is the same as that previously described in section 3.2.

3.5 Summary

This chapter has built up a picture of genetic algorithms: starting with their inspiration in the form of biological evolution through natural selection; then moving onto an overview of their structure and functioning. This has provided context to the descriptions of the three genetic algorithms employed in this work. The simple genetic algorithm, which will be employed in initial experimentation due to its similarity to the algorithm employed in Turner's original work; The single sample generational evolution algorithm will form the basis of the robotics experiments as it is already integrated into the simulation tools; and the multi-objective non-dominated sorting genetic algorithm 2 (NSGA-II), which has a history of usage in multiobjective applications that indicates it will be suitable for the multiobjective experiments contained within this work.

The next chapter returns to the subject of Chapter 2 to look at part of what these algorithms will be used to optimise: the software implementation of the epigenetic network architecture.

4 Software Epigenetic Network

While the ultimate goal of this work is a hardware implementation of an Artificial Epigenetic Network (AEN), a software version is a necessary first step. This is for several reasons: implementing and refining Turner's original networks ensures a solid understanding of the principals involved in AENs; and the modifications needed to go from hardware to software can be prototyped and tested with greater ease. A functioning software network will also provide a metric against which any hardware networks can be compared to ensure a successful translation has been performed.

The first software network is an implementation of Turner's original AEN algorithm: 64-bit floating point numbers, with parameters taking a range between 0.0 and 1.0 (or -1.0 to 1.0 in some cases) [33]. This network will, however, not exactly copy Turner's work, due to a small number of refinements that will be detailed later in this chapter. These alterations have been made either to remove unnecessary complexity (such as the removal of gene to protein translation); make the network's architecture more "hardware like" (the changes to the I/O mechanism).

The next software network laid the ground work for the primary element of the hardware transition: a switch from floating point numbers to integers. This is an important change to make as integer based logic: takes up less silicon; utilises less power; and can be executed at greater speed than its floating point counterpart, particularly with operations like multiplication. Additionally, it is a much simpler task to alter the precision of an integer system via changing the bit width, which opens up the possibility of further reducing the needed silicon. This flows back into the justification for performing initial experiments on software networks, as altering the precision can be tested more quickly given that there is no need to synthesise a new hardware implementation, (indeed, a large range of possible bit widths could be automatically tested with relative ease).

This chapter will begin with section 4.1 detailing the general structure shared between the various implementations; section 4.2 will then describe the changes made to the original AENs; finally section 4.3 explains the differences between the floating point and integer versions.

4.1 Network Structure

Instead of modelling the exact mechanics of the epigenetic mechanisms discussed in section 2.3, the artificial networks designed by Turner employ a more abstract representation [6]. A number of network elements called epigenetic molecules take in inputs from either the inputs of the network or the genes (the term for the network's processing elements). These molecules then calculate which genes they will alter the behaviour of using these inputs, in effect abstracting the more complex chemical processes of chromatin modification. These genes and molecules are connected together via a location/proximity system, which allows varying network topologies to arise from a minimal number of parameters, something else that the artificial network shares with its biological counter parts.

4.1.1 Connections

Instead of each gene and molecule holding a complete list of its connections, a system of locations and proximities is used. Each element has a location, which corresponds to a position in a 1D space that is conceptually similar to a DNA strand; and a proximity, which determines how far on either side of its position an element will look for other units. All other elements within this proximity will be used as inputs by this first element. Note that these connections are not reciprocal, so if element A's location falls within the proximity of element B, making it one of element B's inputs, element B will not automatically be an input of element A.

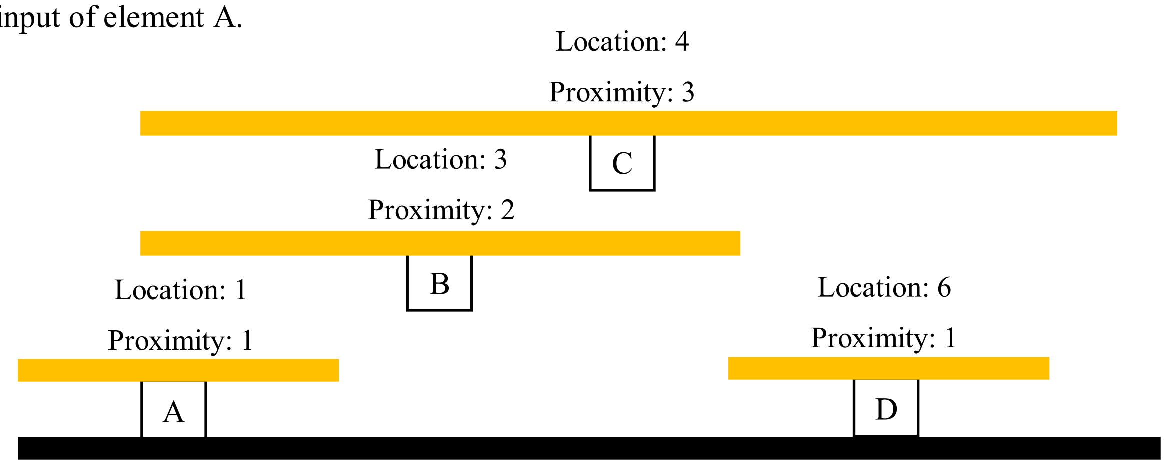


Figure 4.1.1. An example of the location/proximity system. Elements A and D have no inputs; element B's inputs are elements A and C; and element C's inputs are elements A, B and C

As elements within an AEN will not always be present (due to the action of the epigenetic molecules), these connections need to be calculated anew at each time step, a process performed by the algorithm below.

Algorithm 4.1.1 Forming connections between network elements

```
for number of elements in network (A) {
    for number of elements in network (B) {
        if element (A) is not element (B) {
            if element (B)'s location <= element (A)'s location + proximity {
                if element (B)'s location >= element (A)'s location - proximity {
                    element (B) is an input of element (A)
                }
            }
        }
    }
}
```

Note the conditional statement on line 3, which prevents self-connections. While there is nothing conceptually wrong with some elements in a network feeding back into themselves, the location/proximity system means that all elements would do this.

In terms of possible values for the location and proximity parameters, an elements location can be anywhere within the 1D space, so any value between 0 and the networks maximum is valid (e.g. 0.0 to 1.0 for a floating point network). The proximities, however, must be limited to a smaller range, in order to avoid over connection: for example, regardless of its location, an element with a proximity close to the networks maximum value would connect to every other element in the network. With this in mind, a range of 0 to 1/7th of the maximum was used in Turner's networks, and the author sees no reason to change this.

4.1.2 Genes

In addition to the location and proximity described above, each gene also holds a number of other parameters and an implementation of the firing function, used to determine the gene's output.

Once the connections between genes are established, the weighted sum of inputs is calculated. This of course requires weights, which are stored as an array within the gene. This brings up a difference between this version of an AEN and Turner's original: each input to a gene has its own weight value; rather than using a single weight, applied at the output of a gene. The original method meant that all connections drawn from a particular gene would have the same weight, regardless of which genes they were serving as inputs for. While this does reduce the complexity of the network, as well as the evolutionary process, it is an uncommon approach that limits the potential functionality of the network; for example, two different genes, all other things being equal, cannot react to the actions of a third in different ways. With this in mind, the new multiple weight system was implemented, although it does come with a problem of its own: as the number of connections each gene possesses changes, both during the evolutionary process and execution, the number of weights used also changes. Therefore each gene holds a number of weights equal to the maximum number of possible connections it could have (conveniently equal to the number genes plus the number of inputs). These weights are treated as being directly mapped to a particular possible connection, so when gene A uses gene B as an input, it uses weight B.

With the weighted sum of inputs acquired, the gene's firing function is used to determine its output. The function in question is a simple Sigmoid, used here and in the original AENs as it has been shown to be an effective choice for various applications [58] [14].

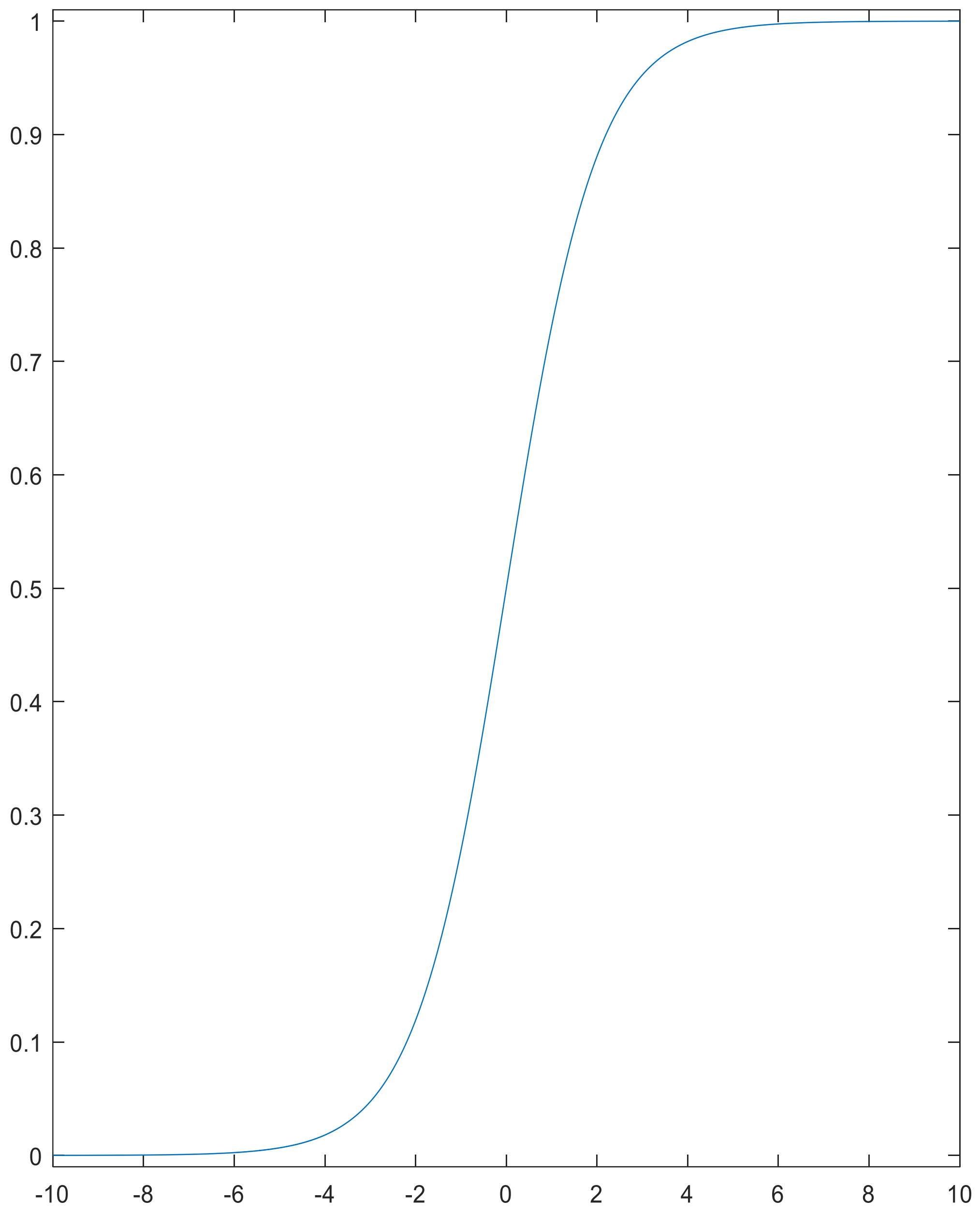


Figure 4.1.2. Plot of the sigmoid function, with trivial slope/offset, and x values in the range ±10.0.

$$\boldsymbol{y = \frac{1}{1 + e^{-sx-b}}} \qquad \textbf{(4.1.1)}$$

Figure 4.1.2 and equation 4.1.1 show this function, where x is the weighted sum of inputs, while s and b correspond to the slope and offset of the sigmoid. These two values can be altered to result in a slight variation in the firing function, and are the final two parameters stored with each gene. Once the sigmoid function calculates the gene's output, it is stored within the gene, so that it can be accessed by other genes/molecules during the next execution cycle, or read out of the network as an external output.

4.1.3 Epigenetic Molecules

Much of the functionality of the genes carries over into the functionality of the epigenetic molecules: the molecules also calculate a weighted sum of inputs, using unique weights for each possible connection; which in turn feeds a sigmoid firing function, used to generate the molecules output. The difference comes in how this output is used by the network. While gene outputs are used as inputs by other genes/molecules, or possibly network outputs; the outputs of the molecules are mapped to positions in the 1D space that all genes and molecules occupy (described above). Each molecule also possesses a second proximity which is used to determine how far from this calculated position to look for genes. All genes within this area are switched to inactive mode, and thus play no part in the next calculation cycle.

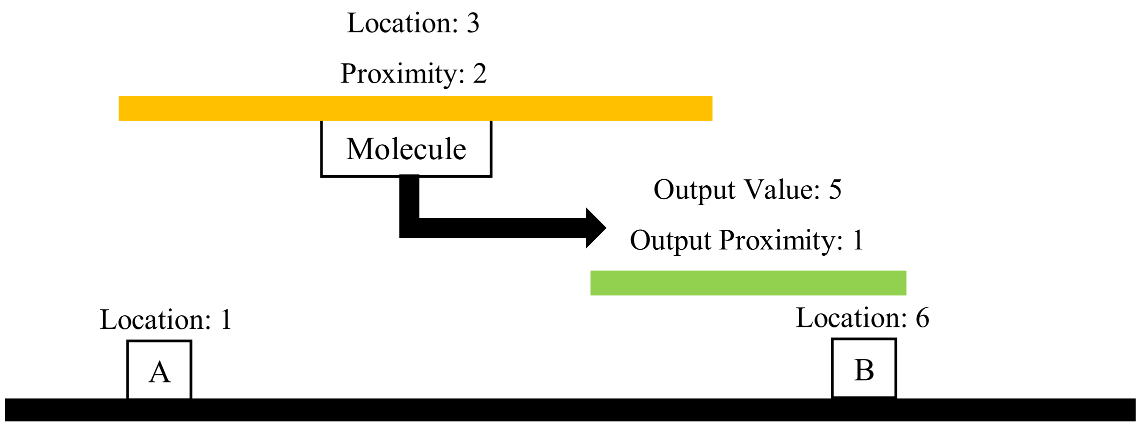


Figure 4.1.3. An example of an epigenetic molecule with the location space. The molecule's location and proximity result in gene A being used as an input, while its output and output proximity result in gene B being made inactive

4.2 Changes from Prior AENs

In addition to the change in the weights noted above, there are two other alterations that have been made from Turner’s original architecture. Both of these modifications have been made in the service of creating a more “hardware like” network structure, which will allow for a simpler translation later.

4.2.1 Removal of Gene to Protein Network Translation

As laid out in Section 2.2.2, in biological systems it is not the genes themselves that perform a function, it this the proteins they code for which do. This is replicated in Turner’s AENs, as active genes are copied into a second array called the protein network. It is the elements in the protein network that are updated during execution, before any changes are then copied back to the genes. This translation process is laid out in greater detail in algorithm 4.2.1.

Algorithm 4.2.1 Execute single iteration of AEN with gene to protein translation

```
for number of epigenetic molecules
    for number of genes in protein network
        if gene is in range of molecule's input
            add to molecule's weighted sum
    Execute epigenetic molecules
    if gene is in range of molecule's output
        mark for addition to protein network
clear the current protein network
for number of genes
    if gene is marked
        copy to the protein network
for number of genes in protein network
    for number of genes in protein network
        if gene is in range of gene's input
            add to gene's weighted sum
    Execute gene
    Record output
for number of genes
    if gene is in protein network
        store its new output
```

While this system is more biologically accurate, it can also be argued that it introduces unneeded complexity; resulting in greater memory requirements and computational overhead. Additionally, there is no evidence within Turner's work that this exact version of the mechanism is vital to the correct functioning of the epigenetic networks. With this in mind, the gene to protein network translation has been removed, replaced by a simple Boolean flag within each gene. Algorithm 4.2.2 shows how this execution differs from the original in algorithm 4.2.1. When the epigenetic molecules update the activity of the genes, these flags are altered accordingly (TRUE, active, by default; FALSE, inactive, if set by a molecule). When the genes are executed, only those whose flags are set TRUE have their expressions updated. In addition, only active genes will have their outputs used as part of the weighted sum of other active genes.

Algorithm 4.2.2 Execute single iteration of AEN with flags

```
for number of epigenetic molecules
    for number of genes
        if gene is active and in range of molecule's input
            add to molecule's weighted sum
    Execute epigenetic molecules
set all genes active
for number of genes
    if gene is in range of a molecule's output
        set gene inactive
for number of genes
    if gene is active
        for number of genes
            if gene is active and in range of gene's input
                add to gene's weighted sum
    Execute gene
    Record output
for number of genes
    update gene output
```

4.2.2 Input/output elements

The other significant change involves the handling of inputs and outputs from the network. The original AENs used a system similar to the location/proximity method, described in section 4.1.1, to map inputs and outputs to specific genes at run time. Each gene possessed an input and output number, which functioned as location values within two separate 1D regions (which were also separate from the primary one used for inter-network connections). Each of these regions was then divided up into partitions, one for each input, or output, with any space left over being ignored. In a correctly functioning network, each I/O region would therefore have at least one gene within it, although this is not always the case, as figure 4.2.1 shows.

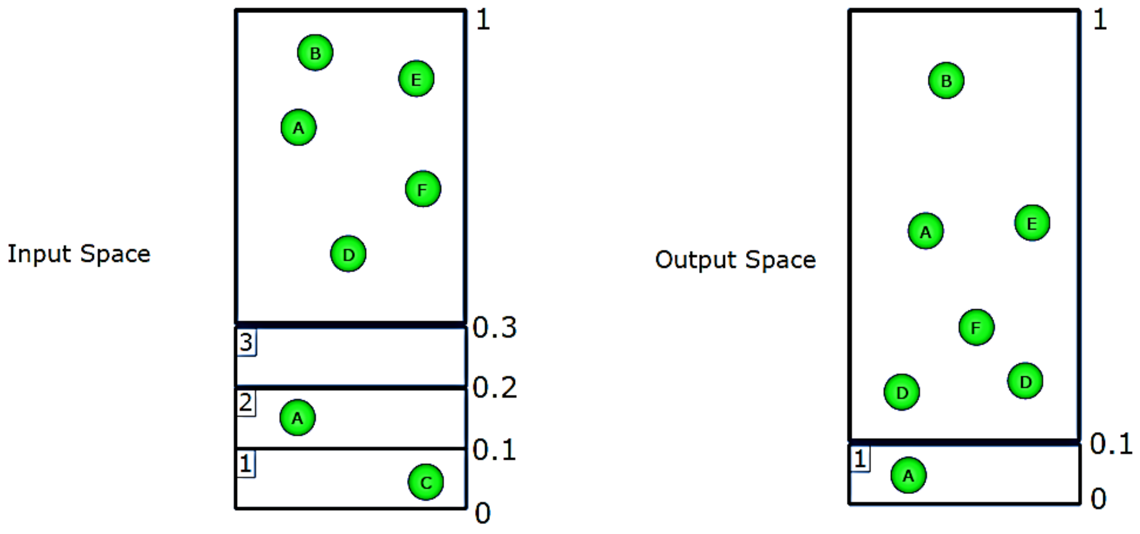


Figure 4.2.1. The input and output spaces of an example network. This network has 7 genes, 3 inputs and 1 output. Note the difference in gene position between the two regions, as well as the fact the input 3 has no gene mapped to it at this time [33].

With outputs, the expression value of the first active gene within the partition is used and fed out of the network. The method for inputs is a little more complex, as the external input value replaces the expression of the first active gene within the partition. In effect, this injects the input value into the normal network space at the location of the replaced gene.

This version of the AEN uses a similar, but more streamlined system, based on input and output elements, placed into the same 1D space as the genes and molecules, using location values of their own. Input elements hold a value, presenting it at their location for genes and molecules to use, a similar method to the one described above that uses a separate element instead of repurposing a gene. Output elements are more complex, as they also have a proximity value. Like genes and molecules, this value defines an area around themselves where they look for genes. The value they set as an output to the network is the expression of the first gene that falls within this region.

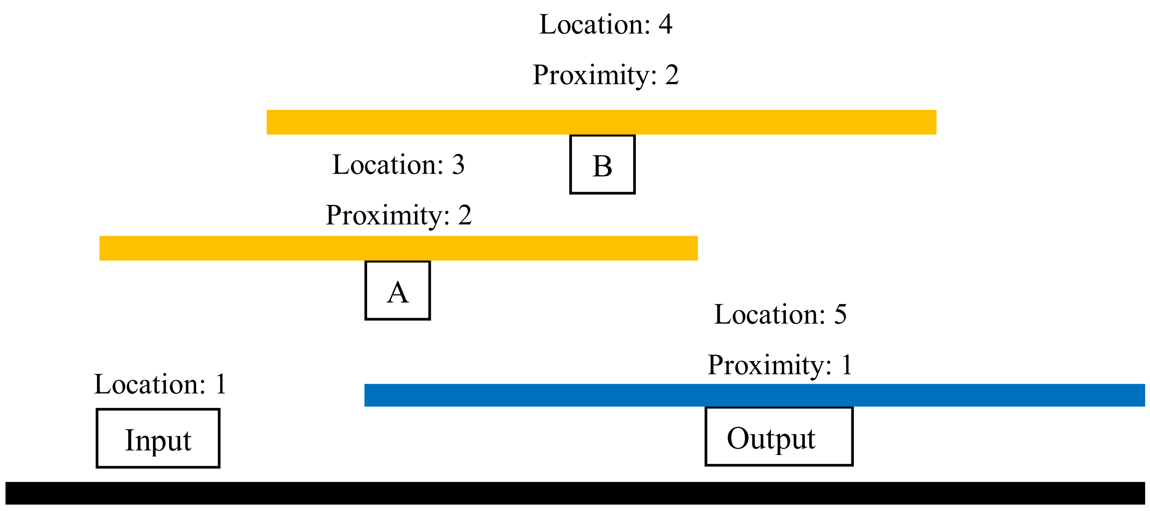


Figure 4.2.2. An example of an input and an output element within the location space. The input element is within range to be an input for gene A, while genes A and B are in range of the output. Gene A will be the first in the array, so its expression will be the network output.

This new system has a few advantages: by not replacing the expressions of genes with inputs, there is no reduction in the network's capacity; additionally, by specifying everything within a single 1D space, the resource requirements are reduced; and finally, this method allows for more of the network's functionality to be encapsulated with discreet units (the I/O elements), something that will make the transition to hardware simpler.

4.3 Floating Point to Integer Networks

The final alterations to elaborate on are those which relate to the transition from floating point to integer mathematics. The benefits of this are twofold: firstly, integer values, and their corresponding mathematical operations, are significantly easier to implement in digital hardware. Field Programmable Gate Array (FPGA) manufacturers do produce various IP cores intended to streamline the use of floating-point maths, such as the Xilinx LogiCORE Floating-Point Operator [59] however, using one would increase the hardware footprint of the network, which goes against one of the reasons for wanting a hardware implementation: reduction of resource usage, something discussed in greater detail at the start of chapter 6.

This leads to the second reason for switching to an integer network: ease of reducing the network's data width. In the same way that integer mathematics consume less resources than its floating-point counterpart, the smaller the bit width of the values used, the less resources that are required to utilise them; not just in terms of computational elements, like adders and multipliers, but also with more basic components such as registers, and even the connections between components. Consider two networks implemented on an FPGA, one with 64 bit values, the other 8 bits. Modern FPGAs are comprised of units called Configurable Logic Blocks (CLBs), which in turn contain a number of elements that can be configured in various ways in order to implement any arbitrary digital circuit. To take as an example the CLBs of a Xilinx series 7 FPGA [60], these elements are:

- 8 6-input Look Up Tables (LUTs), for implementing logic gates.
- 16 1-bit Flip-Flops, for temporary data storage.
- 2 Carry Chains, for improving the implementation of arithmetic elements such as adders.
- 256 bits of Distributed RAM, for data storage.
- And 128 bits of Shift Registers, for data storage and multiplication.

If the networks required something as simple as two signals to undergo a bit-wise AND operation, the 8-bit network would fit the required hardware with a single CLB. The 64-bit network on the other hand would not only need multiple CLBs, but also the additional complexity of the routing elements that connect the CLBs together. The same problem exists with memory: if the two networks each have, say, 4 inputs and 4 genes; then each gene requires 12 parameter values (identification, proximity, slope, offset and 8 weights). In this 8-bit network, this is a total of 96-bits of memory, once again able to fit within the resources provided by a single CLB. The 64-bit network requires 768-bits of memory, equal to the RAM of 3 CLBs.

4.3.1 Overflow Prevention

Having established the motivation behind the switch from floating-point to integer mathematics as the basis of this work's networks, the actual changes made to support this can be detailed. The network's connection mechanisms can remain unchanged, as the location/proximity system requires nothing more than a 1D space and values for an elements location within that space; the mechanism can be implemented with integers just the same as with floating point values. However, the weighted sum of inputs, as well as the inclusion of the slope/offset present an issue. With the floating-point networks, all the parameters, with the exception of the slope, are in the range -1.0 to +1.0 (the slope range is ±20.0) [6]. Recalling equation 4.1.1, this means that for any given gene:

(i) A weighted input can never exceed ±1.0.
(ii) The sum of weighted inputs can never exceed ±n, where n = number of inputs.
(iii) The sigmoid exponent can never be exceed ±40.0n.

Therefore, if a network was implemented with 64-bit floating-point values, which have a range of $\pm 1.7 * 10^{308}$, a gene would require more than $4.25 * 10^{306}$ inputs for an overflow to occur. However, if the parameters of this floating-point network were directly mapped to a 64-bit integer network, then overflow could potentially occur when an input is multiplied by its weight. In order to prevent this, the integer range will be used to fix the ranges of all parameters, while increased bit widths will be calculated for the weighted inputs, the weighted sum and the sigmoid exponent.

$$\boldsymbol{Param_{range} = \pm(2^{n-1}) - 1} \tag{4.3.1}$$

$$\boldsymbol{Weighted\ Input_{range} = \pm Param_{range}{}^{2}} \tag{4.3.2}$$

$$\boldsymbol{Sum_{range} = \pm NumInputs\left(Weighted\ Input_{range}\right)} \tag{4.3.3}$$

$$\boldsymbol{Sigmoid\ Power_{range} = \pm\left(20\left(Param_{range}\right)\right)\left(Sum_{range} \pm Param_{range}\right)} \tag{4.3.4}$$

Equations 4.3.1 through 4.3.4 detail the range calculation process. Starting with equation 4.3.1 which restates that the initial range is defied by the range of values that can be represented by the bit width of the network (n), when using the two's complement representation (Strictly, this range should be -2^{n-1} to $+(2^{n-1} - 1)$, but the version above is accurate enough and simpler). Equation 4.3.2, states the range for a weighted input is the parameter range squared, and equation 4.3.3 that the weighted sum range is that multiplied by the number of possible inputs. This is because the largest absolute value of the weighted sum of inputs would arise if the largest possible input and largest possible weight were present at all possible gene inputs, and as weights and inputs have the same range, this maximum weighted input range is the square of their range. Lastly, equation 4.3.4 details the range of the sigmoid power, which is the range of the weighted sum combined with the ranges for the slope and offset values of the sigmoid. Taking the example of a 4-bit network with 4 possible inputs for each gene: the parameters take the range ±7; the weighted inputs, ±49 (7^2); the weighted sum of inputs, ±196 ($4*7^2$); and the sigmoid exponent, ±28420 ($140*((4*7^2) \pm 7)$).

With these ranges specified, the data widths needed can be deduced. Equations 4.3.5 to 4.3.8 give them in the general form.

$$\boldsymbol{Param_{width} = n} \tag{4.3.5}$$

$$\boldsymbol{Weighted\ Input_{width} = 2n} \tag{4.3.6}$$

$$\boldsymbol{Sum_{width} = 3n} \tag{4.3.7}$$

$$\boldsymbol{Sigmoid\ Power_{width} = 6n} \tag{4.3.8}$$

4.3.5 restates that the network parameters are of the base data width n; 4.3.6 that the weighted inputs will need 2n bits, as a multiplication operation occurs which doubles to width requirement. Equation 4.3.7 states that the sum of inputs needs 3n bits, to allow for the range increase caused by the cumulative addition; and finally 4.3.8 states that the sigmoid power needs 6n bits, or 2(3n), as another multiplication occurs.

4.3.2 Sigmoid Function

The last alteration required is the sigmoid activation function, as any exponent of *e* outside the range ±1.0 will not work, and translating the integer to an acceptable value beforehand would still require the *e* exponent element to be computed, which brings with it significant hardware requirements (Xilinx's recommendation is to employ the Floating-Point Operator that was mentioned above [59], which would very much defeat the purpose of the switch to integer). It is possible however, to construct an integer version of the sigmoid by using a Look Up Table (LUT). A LUT, like its name suggests, replaces the actual calculation of a function with a simple array of possible inputs and their corresponding output values.

There is a design consideration with LUTs, which is the number of entries they hold. While a trivial function, like a two input AND gate, can be fully described with only 4 entries; something like the sigmoid function has the potential to be much too large to use this simplistic approach. For example, the 4-bit network used as an example in section 4.3.1 would need at 56841 entries in its lookup table in order to fully map each possible input to its outputs. Recalling the FPGA CLB specifications in section 4.3 shows that while this wouldn't fit into the 256 bits of distributed RAM available, let alone accommodate the other parameters required.

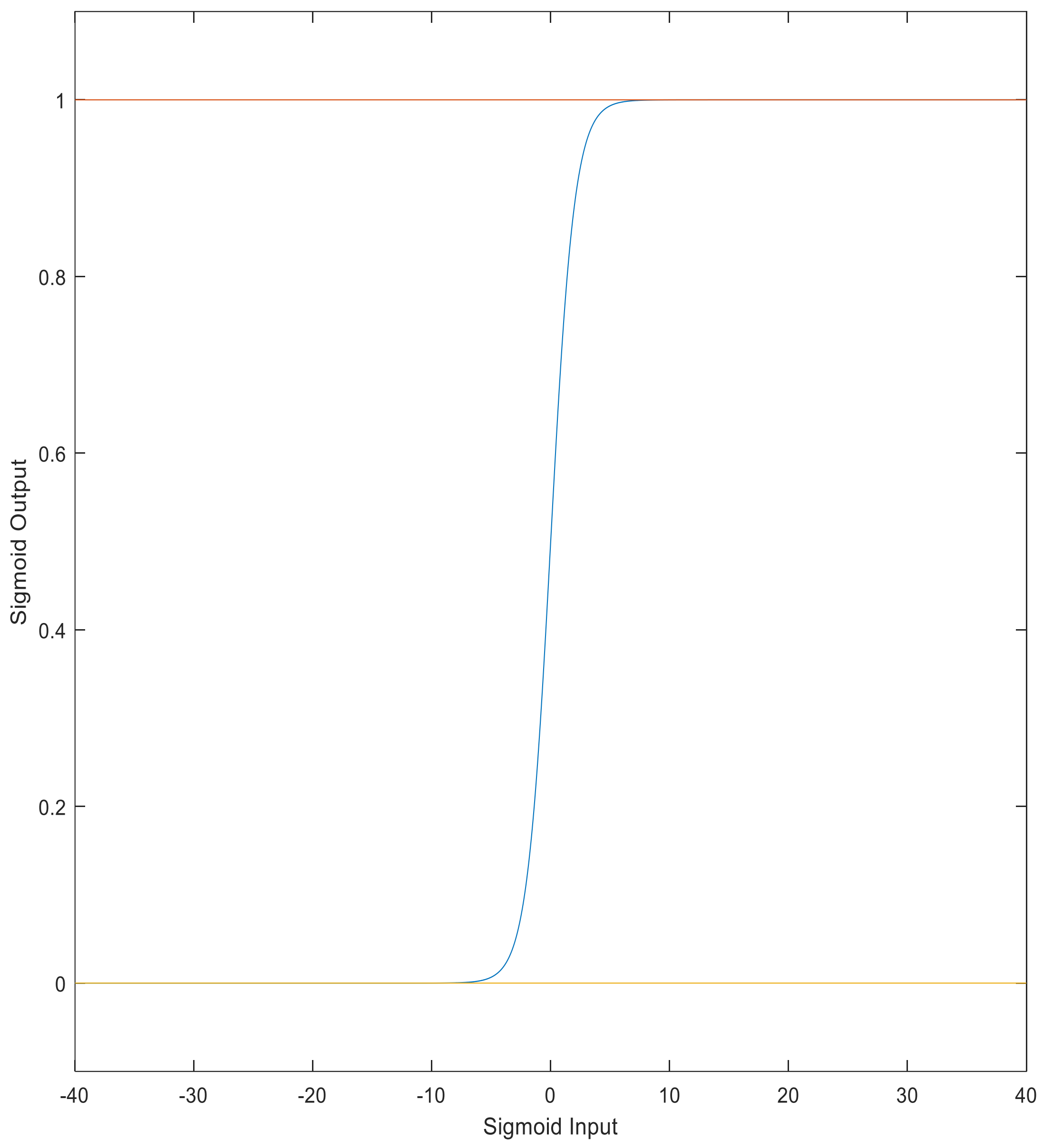


Figure 4.3.1. Plot of the sigmoid activation function for a floating point network, with an input range of ±40. The upper and lower lines show where the sigmoid output levels off at 1.0 and 0.0 respectively .

Fortunately, the sigmoid function itself provides a simple method of reducing the LUT requirements significantly. Figure 4.3.1 is a plot of the sigmoid activation function for a floating point network, where the possible inputs to the sigmoid fall into the range of at least ±40 (recall the specification of this in Section 4.3.1). Looking at this figure it is clear to see that a significant number of possible inputs result in outputs of ether 0.0 or 1.0, with the region of interest actually only being between -10.0 and +10.0. This means that when creating the integer LUTs, possible entries that correspond to outputs of 0 or $+2^{n-1}$ can be removed from the table can handled with trivial if statements. Returning again to the 4-bit network example, this method reduces the 56841 entry lookup table to only 4202 entries.

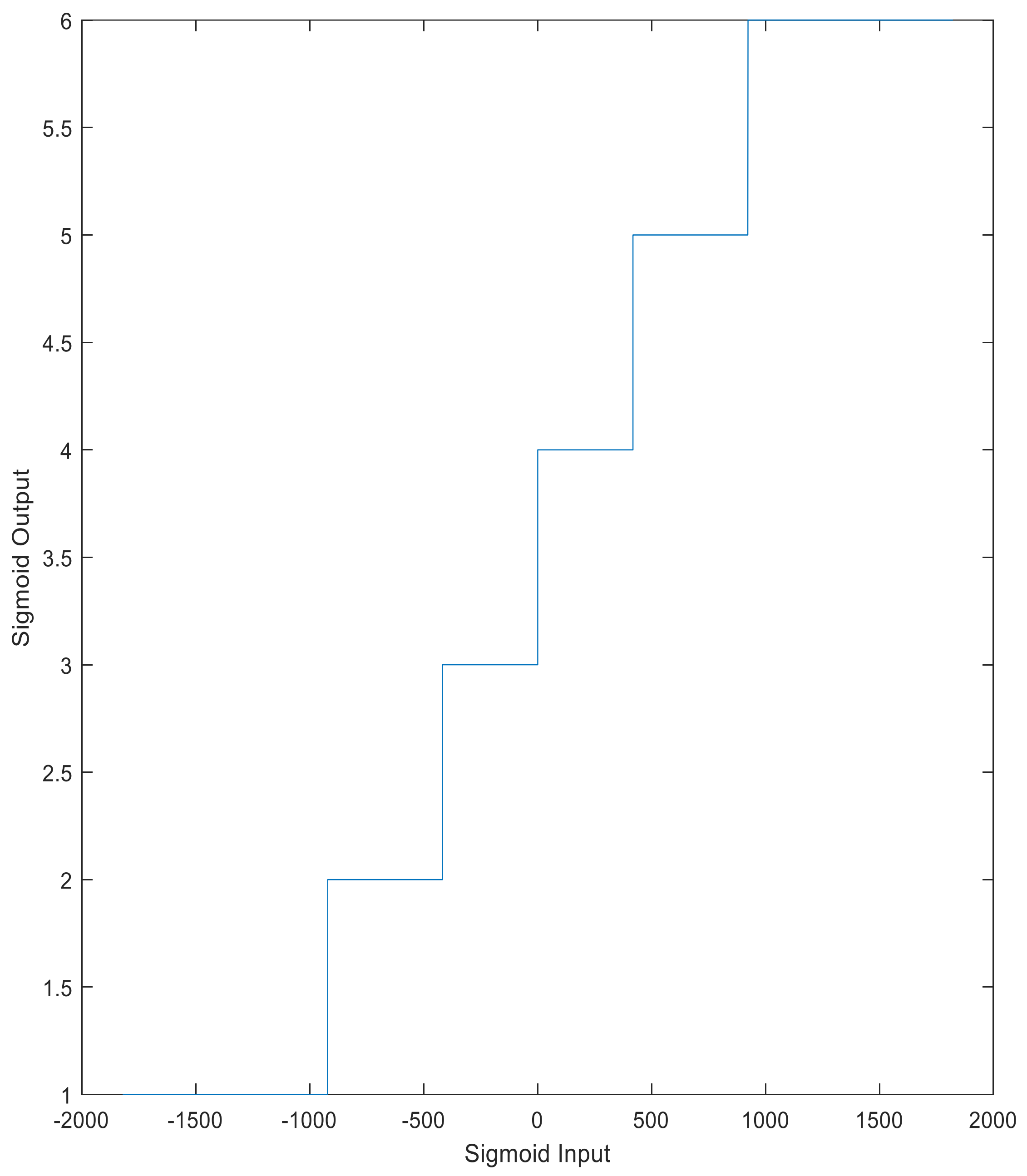


Figure 4.3.2. Plot of the sigmoid activation function for a 4-bit network with maximum and minimum output values removed.

Figure 4.3.2 illustrates the "sigmoid" described by such a 4202 entry LUT, which shows another possible route to further reducing its size. Each possible output of the LUT corresponds to multiple input values, so instead of having one entry per input value, the LUT can be reduced to one per possible output value, with external logic to determine which entry an input corresponds to (input < A then output = entry A). Switching to this implementation allows the 4-bit network previously described to have a lookup table of only 6 elements. This equates to a LUT with a total of 24 bits, meaning it fits into the 256 bits of distributed RAM that the CLBs detailed in section 4.3.

4.4 Summary

Bringing this chapter to a close, the architecture of a software based epigenetic network has been elaborated in detail, with a look at the various elements that comprise one: including the genes and molecules; and the location/proximity based interconnection system.

Additionally, the changes made from the original network architecture have been covered. The gene to protein network translation has been removed in favour of a more streamlined system that employs Boolean flags within each gene, which are set by the epigenetic elements. Also, the input/output system has been changed from its direct integration with the genes to a separate set of elements, which will make the translation to hardware easier. The alterations needed to accommodate the switch to from floating-point to integer maths. This included the replacement of the sigmoid activation function with a Look Up Table (LUT); and considerations to prevent overflows that are now possible.

The validation of the performance of the software network is covered in section 1 of chapter 7, while chapter 5 details the creation of the FPGA based hardware network architecture that forms the core of this thesis, expanding upon the design of the various network elements and how they fit together.

5 Hardware Epigenetic Network

With an integer and lookup table based version of an Artificial Epigenetic Network (AEN) already created in software, the translation to hardware requires only that the existing architecture be rebuilt. The platform used for this implementation is a Field Programmable Gate Array (FPGA), a class of chips that can be configured to implement a huge variety of digital logic circuits: from simple state machines to complete multi-processor systems. This configurability not only avoids the need to develop specialised hardware, but it is another element that allows for the easy investigation of the consequences of altering the data width of the network. FPGA configuration necessitates the use of a hardware description language, which in this case will be the Very high-speed integrated circuit Hardware Description Language (VHDL). There are two important properties that the completed hardware network must possess: paramarisability, meaning that properties such as the network's data width, or the number of network elements and be changed easily; and parallelisation, meaning that, unlike the sequential execution of the software networks, the hardware network will execute the molecules, genes and outputs of the network all at once, greatly improving time performance.

This chapter will detail the hardware versions of the various network units (i.e. genes, molecules and inputs/outputs); the means by which they are connected together to form a network; as well as the means by which the network will be configured. Finally, some additional elements will also be needed to allow for data to be sent to/from a PC to the complete network, which will be based on off the shelf components.

5.1 Input Units

The input units are the simplest network elements and the beginning of the network from a conceptual point of view. Referring back to their software counterpart in section 4.2.2 for a specification, each input element needs to hold two values: their identification, or location within the 1D space the network elements occupy; and an input value. Each of these values will need a storage element of paramatrisable width, to allow for the bit width of the network to be adjusted as needed. Additionally, while the network's sequential elements will be synchronous with the clock, a reasonable precaution will be the implementation of the ability for an element to signal when a value has been successfully stored, in order to avoid the network prematurely initiating the calculation of outputs. To facilitate this, each storage element will be paired with a comparator that will check if the input and output of the storage is the same; if it is, then the elements enable signal will be carried through. An illustration of this complete assembly is shown in figure 5.1.1.

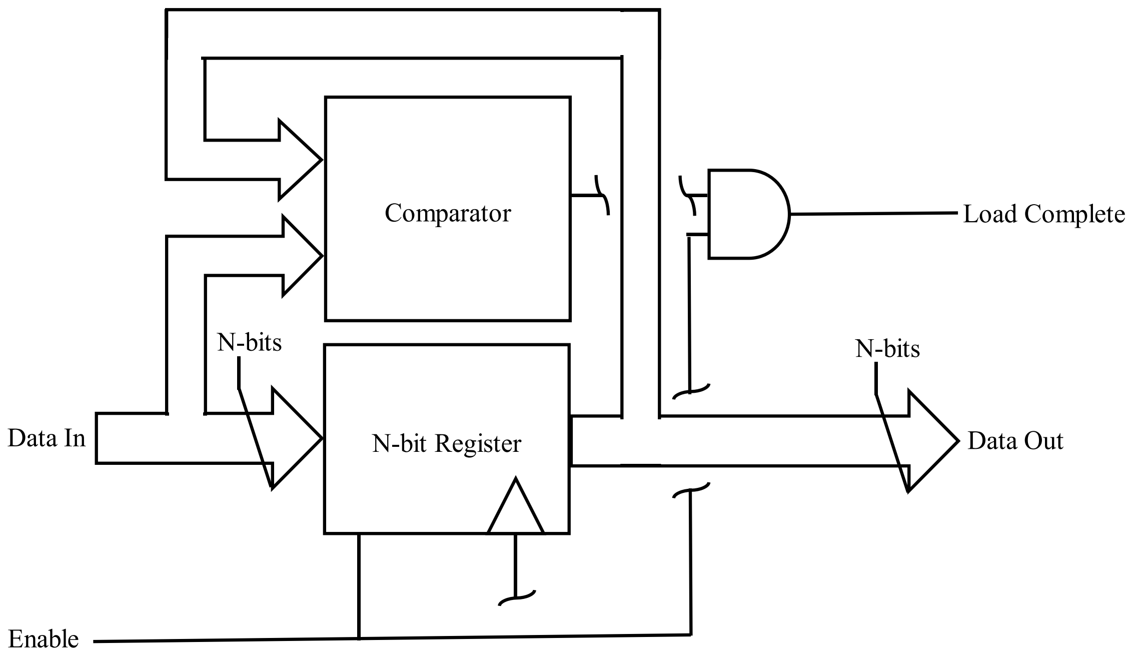


Figure 5.1.1. Single n-bit storage element with load complete signalling mechanism. Note the presence of a CLK input on the register in addition to its enable signal.

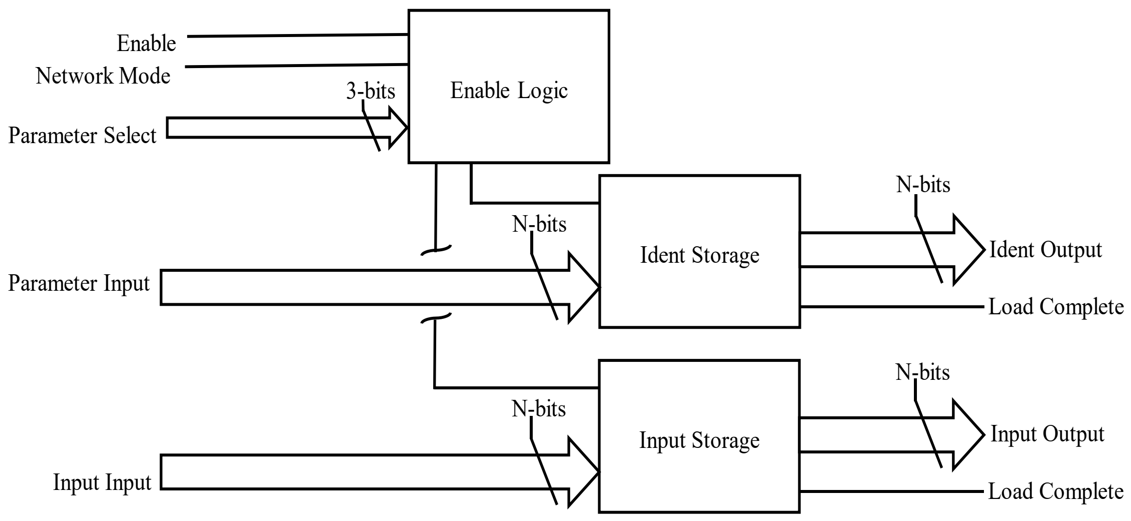


Figure 5.1.2. Complete n-bit input unit, with identification and input storage elements, enable signal generation logic.

Figure 5.1.2 shows a complete input, which as mentioned contains two of the storage elements detailed in figure 5.1.1. There is also an additional component in the form of the enable logic, which takes the external enable signal and carries it through to one of the two storage elements depending on the mode of the network and the value of the parameter selection bus.

The network mode is a simple signal, with a value of 0 specifying that the network is to execute normal behaviour (taking in inputs and computing outputs); while a value of 1 corresponds to the network being in configuration mode, during which parameters, like the input elements identification, are loaded. The parameter selection bus on the other hand is more akin to an address, as it specifies which parameter storage element a particular value is to be loaded into. This is largely irrelevant in the input units, as they have only one parameter, and so will be explained in greater detail in sections 5.2 and 5.4. Thus, in the case of the input units: a network mode of 0 allows the input storage to be enabled; and 1 allows the same for the ident storage.

5.2 Genes and Molecules

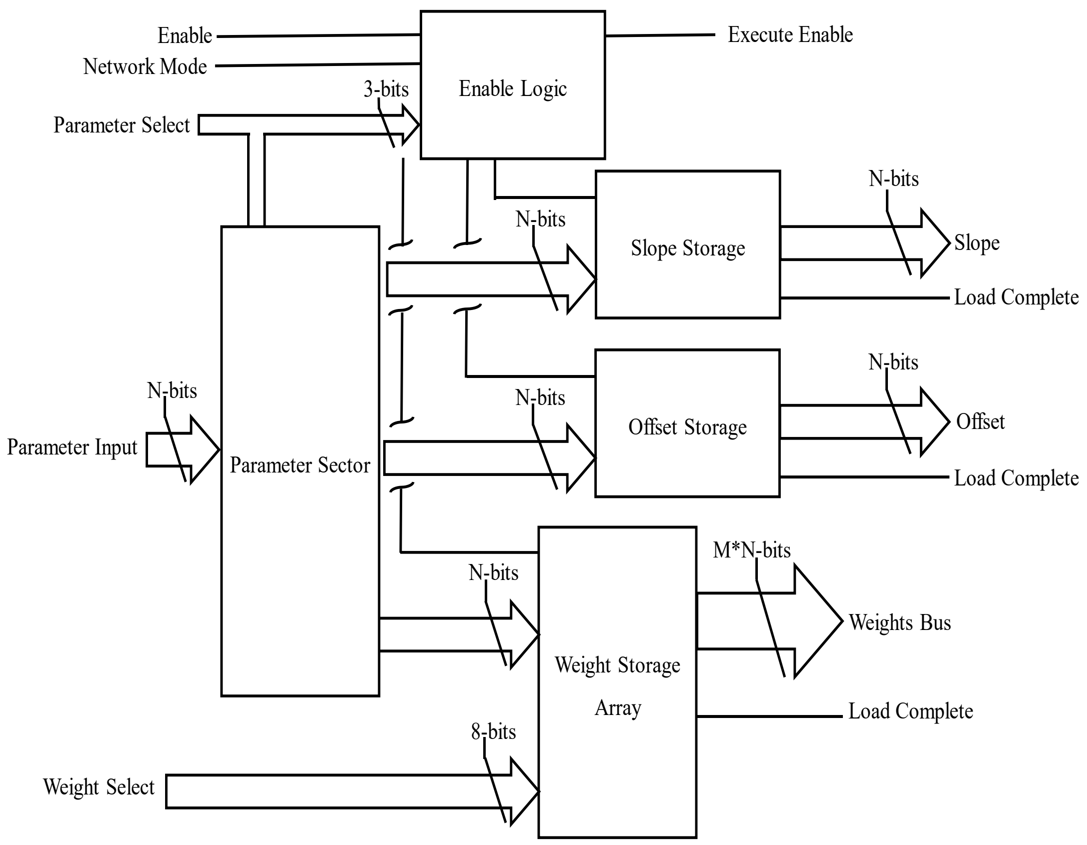


Figure 5.2.1. Parameter storage section of an n-bit gene/molecule unit with m possible inputs.

The initial section of the gene/molecule units, shown in figure 5.2.1 is quite similar to the input units, but with a few key differences. First, the presence of more than one parameter storage element necessitates the use of a selection unit, which routes the parameter input through to the input of one of the storage elements, depending on the value of the parameter select input. This signal, which also now has an effect on the enable logic due to the multiple storage array enables, acts akin to an address value, specifying which of the parameters is to be loaded. The exact values that correspond to each parameter will be detailed in section 5.5, where the network configuration mechanism will be elaborated upon in its entirety. A similar function is performed by the weight select signal, which leads into the other significant change, the weight storage array. Given that each gene requires multiple weights, one for each possible connection it can form with both other genes and the input units, the parameter select alone is insufficient to address each storage element. Instead, the parameter select routes the parameter input and enable signals through to an array of storage elements, which have their own additional layer of logic mirroring those above. This layer uses the weight select signal to pass the parameter input and enable signals through to one of the storage elements. The weight select bus uses a simple numerical value, much like an addressable memory. Unlike most other elements of the network, which are paramatrisable, the weight select bus is a fixed 8 bits, due to its requirement to connect into the parameter address system discussed in section 5.5. However, 8 bits allows for 256 possible inputs, more than sufficient. The last point to note with respect to figure 5.2.1 is the absence of two parameters. Looking back at section 4.1.1 highlights the absence of the identification and proximity values that are needed to facilitate connectivity. This is because these elements have been placed within a wrapper that handles connections, allowing the units themselves to be solely dedicated to processing. This also removes any differences between the genes and molecules, allowing the same component to be reused as their core. These wrappers will be discussed in more detail in section 5.4.

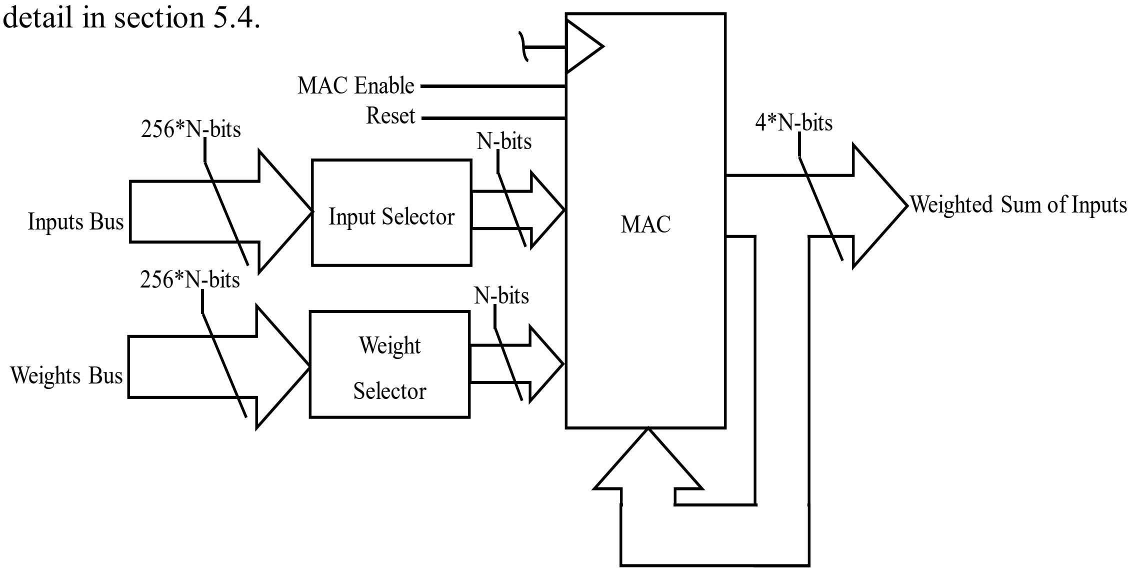


Figure 5.2.2. Weighted sum of inputs calculation section of an N-bit gene/molecule with 256 possible inputs.

The next section of the gene/molecule units is the start of the actual processing mechanism: the calculation of the weighted sum of inputs, which is show in figure 5.2.2. The weights bus, also shown in figure 5.2.1, and the inputs bus, which is an input to the gene/molecule unit, are inputs to a pair of selector elements, which route them one at a time into the Multiply Accumulate Unit (MAC). When enabled, the MAC multiplies the input and weight together, adds them to its existing output, and then replaces that output with the new value. The reset signal zeros the MACs output, allowing a summing to be started. Control of the MAC, as well as the two selectors is performed by a Finite State Machine (FSM).

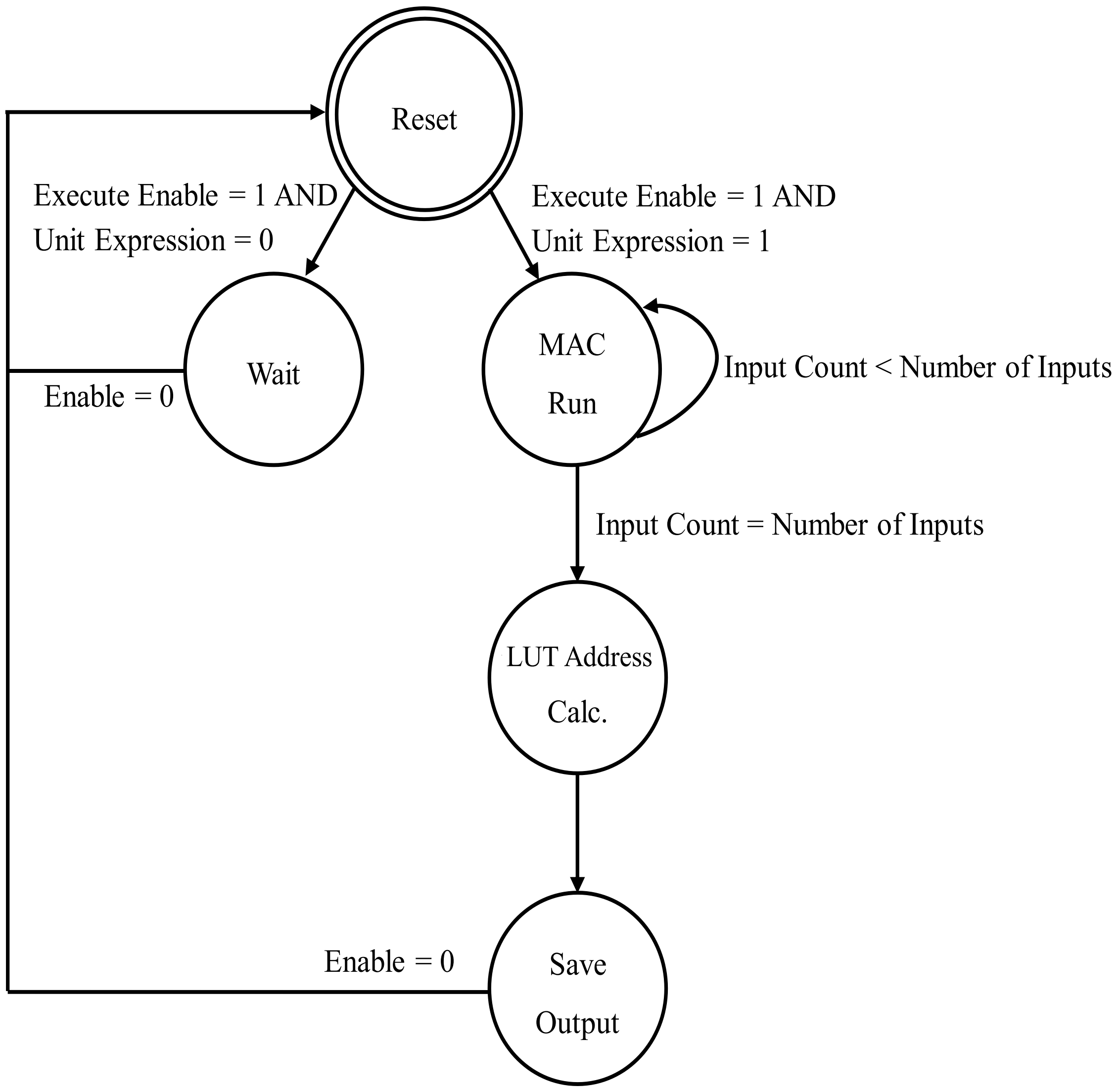


Figure 5.2.3. State transition diagram for the finite state machine that controls the calculation elements of the gene/molecule units.

Table 5.2.1. Signal mapping table for the finite state machine that controls the calculation elements of the gene/molecule units.

Signal	**Reset**	**Wait**	**MAC Run**	**LUT Address Calc.**	**Save Output**
Input Count	‘0’	‘0’	Count + 1	Count	Count
MAC Enable	‘0’	‘0’	‘1’	‘0’	‘0’
MAC Reset	‘1’	‘0’	‘0’	‘0’	‘1’
Output Register Enable	‘0’	‘0’	‘0’	‘0’	‘1’
Execution Complete	‘0’	‘1’	‘0’	‘0’	‘1’

Figure 5.2.3 and table 5.2.1 detail this FSM, which also produces the enable signal for the register that stores the computed output, and the execution complete signal (similar to the load complete signals the storage elements possess).

- Reset State: the default state of the FSM. In this state, the enable and complete signals are set LOW; the MAC unit is reset; and the count that tracks which input/weight combination is to be fed into the MAC is set to zero. When the gene/molecule’s enable signal is set HIGH, the FSM switches to the Wait or MAC Run states, depending on the value of the unit expression signal. The unit expression is the signal that indicated whether or not a unit should be active and involved in the calculation. For genes, this value is set as a result out the outputs of the molecules, while molecules have it permanently set HIGH.
- Wait State: when the unit’s expression signal is LOW, the unit’s activity is being suppressed by one of the epigenetic molecule units (as such, this can only happen within a gene). In this case, the unit’s execution complete signal is set HIGH, and it waits until the unit enable goes LOW, which returns it to the reset state.
- MAC Run State: when the unit’s expression signal is HIGH, then the gene/molecule is able to execute normally. In the MAC Run State, the MAC unit is enabled, and the count increments by 1 every clock cycle. As long as the count value is less than the number of input/weight parings, then the two selector units feed the appropriate ones through to the MAC. When the count exceeds this, the FSM progresses to the Power Calculation State.

- LUT Address Calculation State: in this state, the FSM does nothing for a clock cycle. This is to allow the logic that calculates the LUT address to run. The FSM then proceeds to the Save Output State.
- Save Output State: in this state, the MAC unit is reset and the enable for the output storage register is set HIGH, allowing the value produced during the calculation state to be stored. The enable complete signal is also set HIGH, to signal the completion of processing. Like the wait state, the FSM then waits for the unit's enable signal to be set LOW, whereupon it then returns to the reset state.

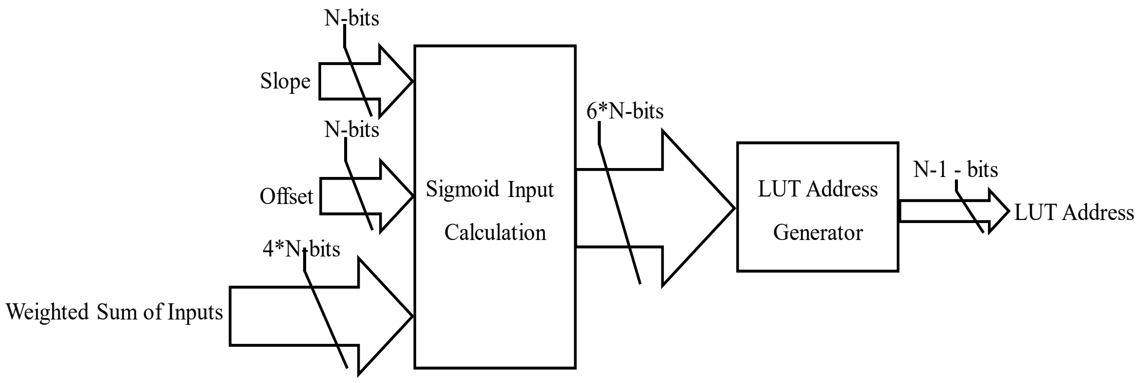


Figure 5.2.4. Sigmoid input and LUT address calculation elements of an n-bit gene/molecule unit.

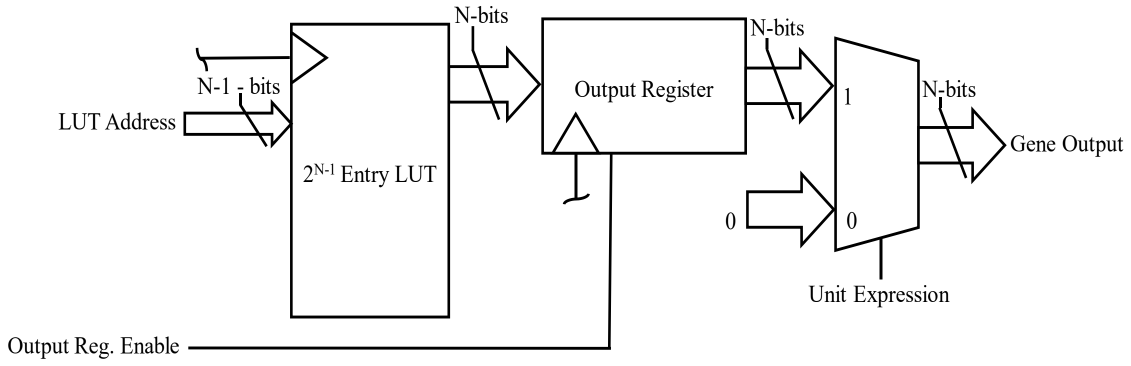


Figure 5.2.5. LUT and output storage register for an n-bit gene/molecule unit.

Figures 5.2.4 and 5.2.5 show the final stages of the gene/molecule units, which takes the weighted sum of inputs and uses it, along with the slope and offset parameters, to produce the output. The first element calculates the input for the sigmoid, as shown in equation 4.1.1 in section 4.1. There is one change to note however, which is that the slope value is first multiplied by 20. This is because, recalling section 4.3.1, the slope has a range of ±20*normal range. Rather than store it as a larger value, the hardware gene/molecules store it as if it were no different from all the other parameters, only factoring in its increased size at the last possible moment. The Look Up Table (LUT) is exactly as described in section 4.3.2, and as such the 6*N-bit output of the sigmoid input calculator needs to be transformed to a value that can function as an address. Fortunately, this can be done by taking only the top n-1 bits of the sigmoid input, which the LUT then maps to the appropriate output value. In order to allow the network to be fully paramatrisable, the content of the LUT needs to be replaced depending on the bit width used, a task that is accomplished using a simple external program that generates the LUT for a given bit-width. The output of the LUT is stored in the register, when enabled by the FSM. The final element is a multiplexer, which is controlled by the unit expression signal. When a gene unit is supressed by an epigenetic molecule, it needs to produce an output of 0.0. To ensure this, the multiplexer allows the value stored in the register to be substituted as the output to the unit under this circumstance.

5.3 Output Units

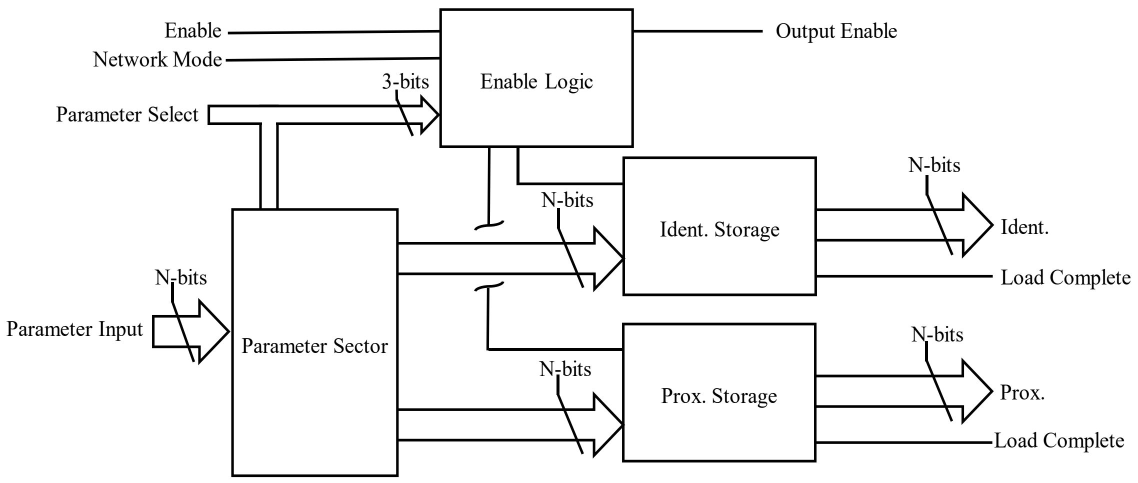


Figure 5.3.1. Parameter storage elements for an n-bit output unit.

Like all previous units, the output units start with the parameter storage elements, shown in figure 5.3.1. Like the input units, the output units hold their own connection parameters, which now includes a proximity value. This is internal to the output unit, rather than being in a wrapper like the gene/molecule units, because they're needed by the unit's internal logic.

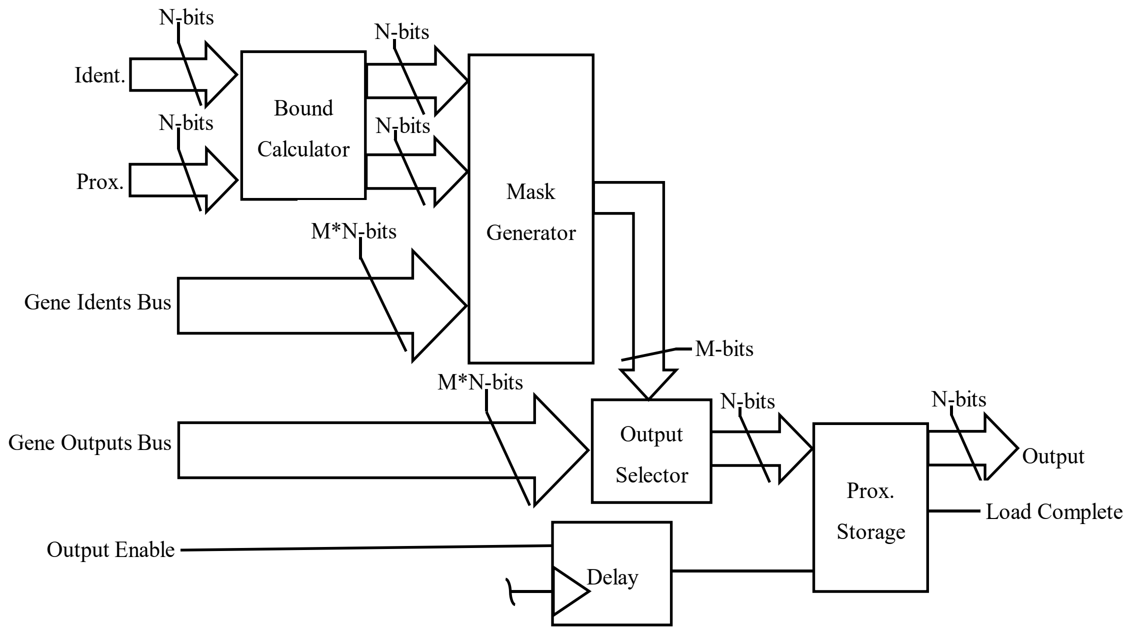


Figure 5.3.2. Output selector logic of an n-bit output unit with m inputs.

Figure 5.3.2 shows this logic, which selects and stores the gene output value that corresponds to this particular output unit's value. This process begins by using the ident. and prox. values to calculate the upper and lower bounds: the two points that define the range that the output unit will choose values from, illustrated in figure 4.2.2 in section 4.2. These two values then go to the mask generator, where together with the ident. values of the network's gene units, they produce an m-bit mask, which identifies which gene output is to be the output of this unit, and thus a network output. This is done in two stages:

1. An initial mask specifies which genes are within the output unit's range, a simple case of seeing which idents are between the two bounds. Valid genes are masked HIGH, invalid LOW
2. The final mask then specifies which of the valid genes is to be the output, which it does by taking the initial mask and carrying through only the first HIGH value.

The completed mask is then used to control the output selector, which passes through the mask chosen value to the output storage register. The last element to note is the delay, which holds back the register's enable signal to ensure that the output selection process is complete.

5.4 Network Interconnections

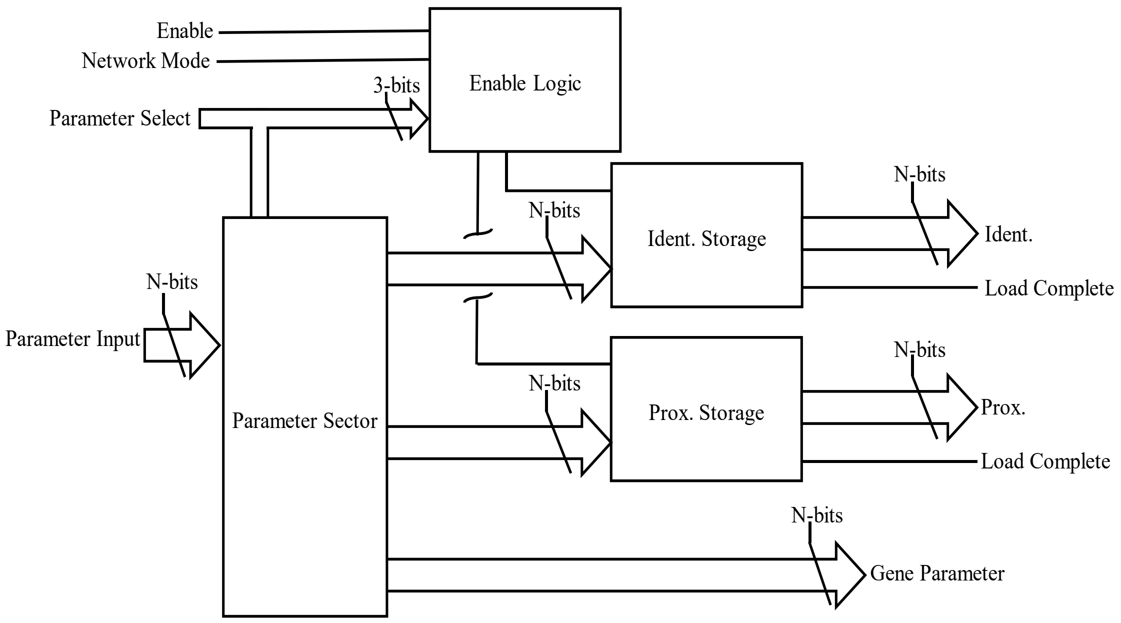


Figure 5.4.1. Parameter storage elements of an n-bit gene wrapper.

In section 5.2 it was mentioned that the gene and molecule units outsource their interconnection mechanism to wrapper units. Figures 5.4.1 and 5.4.2 show the parameter storage elements of these wrappers, which are much like their counterparts in the output units, shown in figure 5.3.1. Note, however, the addition of the gene/molecule parameter signal, which routes parameters such as slope, offset and weights through to the gene/molecule unit where they will be handled by the elements shown in figure 5.2.1. There is also the difference between the gene wrapper and the molecule version, in the form of the extra proximity storage element. Recalling section 4.1.3, this second proximity is for establishing connections with the genes that the molecule will control.

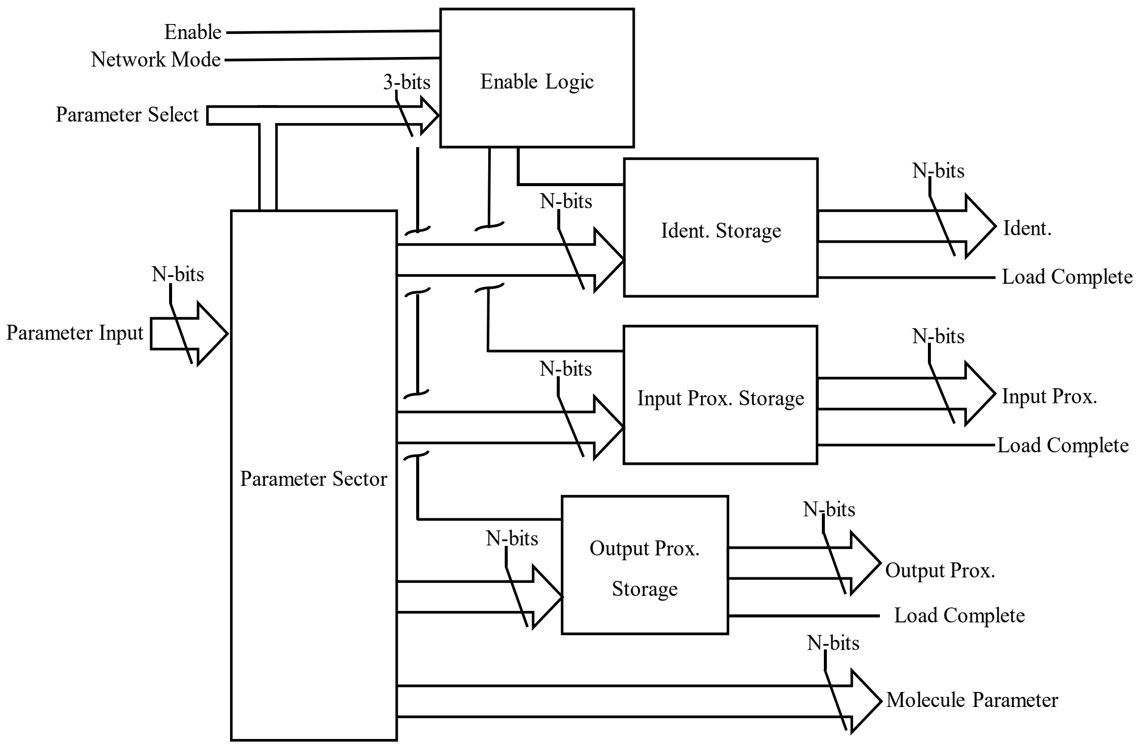


Figure 5.4.2. Parameter storage elements of an n-bit molecule wrapper.

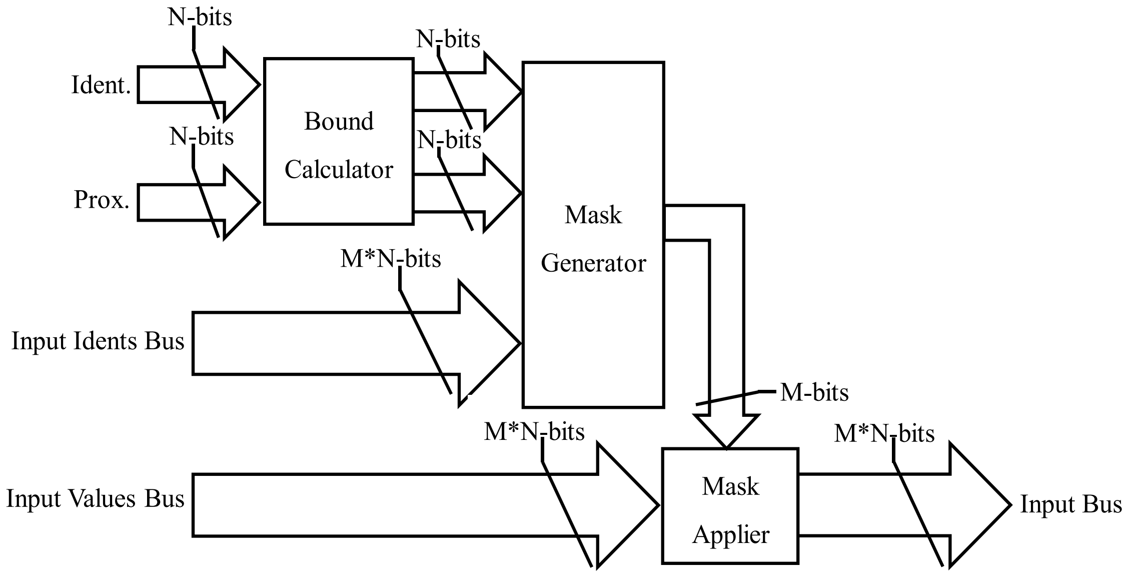


Figure 5.4.3. Input selection elements of an n-bit gene or molecule wrapper with m inputs

The other half of the gene and molecule wrappers are the same, and very similar to part of the output elements, specifically those shown in figure 5.3.2. These elements, shown in figure 5.4.3: take the ident. and prox. (input proximity in the case of a molecule wrapper); calculate the upper and lower bounds; and utilise these along with the identification values of the possible inputs to produce a mask. The first difference is that, while the output unit only considered gene units, both the gene and molecule wrappers also consider the network's input units. In addition, the mask specifies which units are within the wrapper's range rather than selecting only one. The bus of input values, with any from units outside the wrapper's range being zeroed, is passed through to the gene/molecule unit.

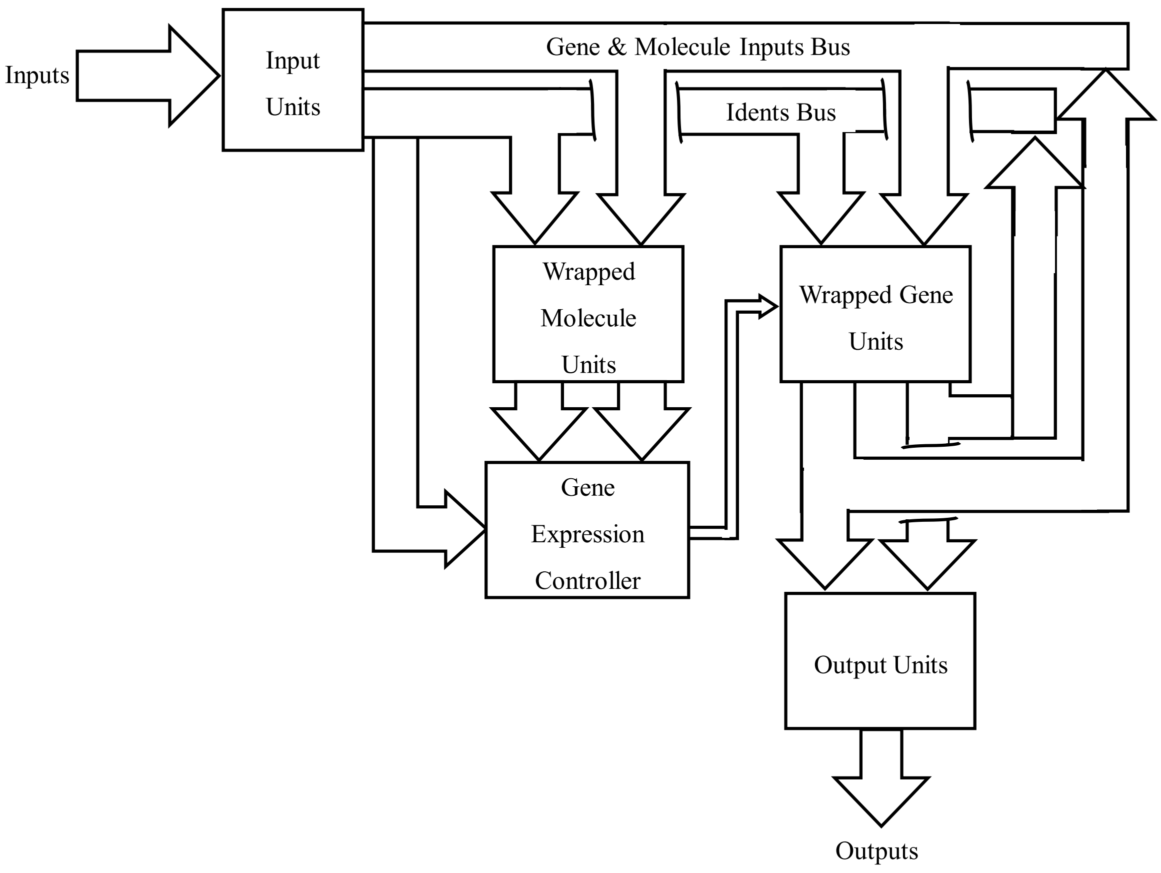


Figure 5.4.4. Network interconnection system, excluding control signals.

All the components discussed previously come together in the network interconnection system, shown in figure 5.4.4. Input values are loaded into the input units, which causes their load complete signals to go HIGH. By ANDing together all the input's load completes, a signal is created that indicates when all the input values are present. This signal then serves as the enable for the molecule units, which pull their inputs from a bus that gathers together both the values from the input units and the outputs of the gene units. A second bus gathers together all the ident. values for the input and gene units, which allows the wrappers to determine the values which go through to the gene/molecule units. The ident. values of the molecules however are fed to the gene expression logic, along with their outputs and the idents. of the genes, provided by the bus. This logic determines which genes are to have their activity supressed, following the process described in sections 4.1.3 and 4.2.1. The results of this logic are fed to the gene units in the form of the unit expression signals. Just like with the input units, the completion of the molecule unit's execution sends their complete signals HIGH, these are again ANDed together to produce the enable signal for the gene units, which in turn perform their calculations, pulling their inputs from the same buses as the molecules. When they complete, they too send their complete signals HIGH, the ANDing of which produces the enable signal for the output units. Unlike the genes and molecules, the output units do not take their inputs from the two buses, but instead directly from the gene units, as they have no need for the values of the input units. Once the output units have completed the selection and storage of the network outputs, their complete signals will go HIGH, the ANDing of which produces an execution complete signal for the entire network.

5.5 Network Configuration

The last part of the network proper is the configuration system, which allows the various parameters of the network to be changed on the fly. Parts of this mechanism have already been mentioned, in the form of the parameter storage elements within the network units; the network mode signal; and the parameter and weight select signals shown in figures 5.1.2, 5.2.1, 5.3.1, 5.4.1 and 5.4.2. These last two signals, however, are part of the larger parameter load address signal, the various sections of which control all aspects of the parameter loading process.

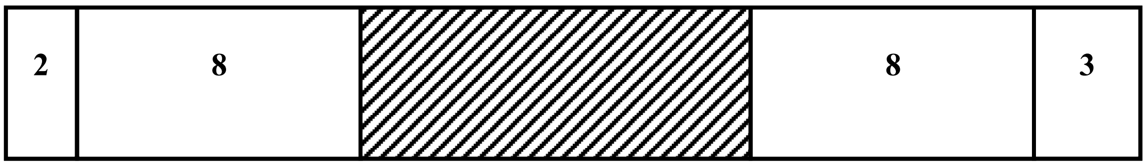


Figure 5.5.1. Structure of the 32-bit parameter load address signal, showing the width of the various sections. Note that the hashed out section in the middle is an unused 11-bits.

Figure 5.5.1 shows the complete parameter address signal, which is comprised of 5 parts. From left to right they are as follows:

- 2-bit unit type. These two bits specify which of the four types of network unit the parameter is for:
    - 00 – input
    - 01 – gene
    - 10 – molecule
    - 11 – output
- 8-bit unit address. This is a numerical identifier for a specific individual network unit.
- 11-bit unused region. These bits are unused but exist as a necessity of the address signal being 32-bits in size. The reason for this will be given in section 5.6. This region is in the signal's centre because it also acts as a clear visual divide between the sections of the address used by routing logic external to the genes/molecules, and those used by the internal logic.
- 8-bit weight address. Already discussed in section 5.2, this is only used when a weight value is being loaded and identifies which of the gene or molecule's inputs the weight corresponds to.
- 3-bit parameter type. Lastly, these three bits indicate which of the various possible parameter storage elements is to be loaded:
    - 001 – identity
    - 010 – input proximity
    - 011 – slope
    - 101 – offset
    - 110 – output proximity
    - 111 – weight

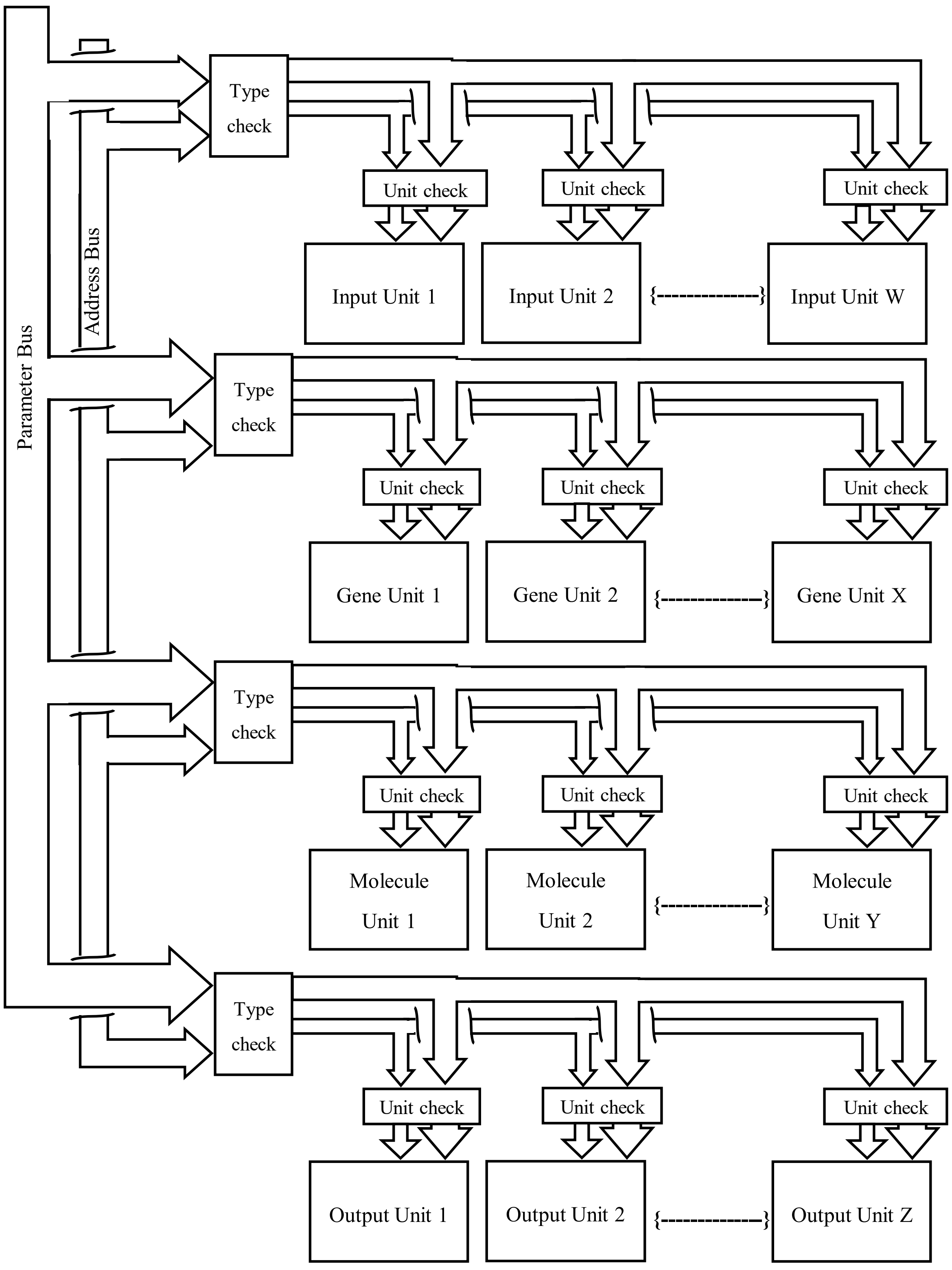


Figure 5.5.2. Parameter loading mechanism for a network with W inputs, X genes, Y molecules and Z outputs. Note that the enable signals are excluded.

Figure 5.5.2 illustrates how the parameter address, and the signal carrying the value of the parameter, connect together with the units of the network to form the complete loading mechanism. A parameter value and the complete address are provided to the two signals. The type checker looks at the first two bits of the address and roots the value and the rest of the address through if it matches the type of unit that it is connected to. The same process is repeated with the unit checkers, which consult the 8-bit unit address. This results in the parameter value, the parameter type and the weight number (if needed) being presented to the network unit, where the internal elements shown in figures 5.1.2, 5.2.1, 5.3.1, 5.4.1 and 5.4.2 take over.

5.6 AXI-Lite Interface, ARM Core and PC Connection

If the network was to have its parameters fixed, and then be directly connected to some system that it was to control, then the components detailed thus far would be sufficient. However, to facilitate the various experiments that will be carried out using this architecture, it needs to be connected to a PC, which will not only perform the configuration of the network, but also provide its inputs via the various simulations which will be detailed in chapter 6. The start of this network to PC connection is an AXI-Lite interface unit.

The Advanced eXtensible Interface (AXI) is a microcontroller bus protocol created by ARM in 2003, with the current version, AXI4, being released in 2010. It is designed to allow easy implementation of memory mapped interfaces, with AXI-Lite being a version that has a small logic footprint and a very simple interface [61]. Xilinx development tools have built in support for the AXI protocol, including the ability to generate AXI peripherals through its IP core packager [62]. Using this tool, an AXI-Lite interface has been created that allows a microcontroller to send and receive data from the epigenetic network. The AXI-Lite Breakout, as it is called, contains 64 32-bit registers: 32 read, 32 write; the size of the registers is a property of the AXI-Lite protocol, which utilises 32-bit data. A microcontroller connected to the breakout via the AXI-Lite bus will be able to read/write to the appropriate registers, while the epigenetic network does the same. The function of the various registers is detailed in table 5.6.1.

Table 5.6.1. Mapping of the 64 registers within the AXI-Lite breakout to the various inputs/outputs of the epigenetic network. Note that read/write is from the processor's perspective.

Register Number	**Read/Write**	**Network I/O**
0	Write	Control Signals (Network Mode and Enable)
1	Write	Parameter Load Address
2	Write	Parameter Value
3 to 31	Write	Network Inputs (maximum of 29, if a network has less inputs, extra registers are ignored)
32	Read	Parameter Load Complete
33	Read	Network Execution Complete
34 to 63	Read	Network Outputs (maximum of 30, if a network has less output, extra registers are ignored)

Taking the connection mapping in table 5.6.1 together with the network architecture described in sections 5.4 and 5.5, the process by which a microcontroller is able to control the network becomes clear.

- To load a set of parameters into the network:
    1. Set the network to parameter load mode via register 0.
    2. Put a parameter address and parameter value into registers 1 and 2.
    3. Set the network enable HIGH via register 0.
    4. Wait for register 32 to report the successful loading of the parameter.
    5. Set the network enable LOW via register 0.
    6. Repeat steps 2 through 5 until all parameters are loaded.
- To execute the network:
    1. Set the network mode to execute via register 0.
    2. Put the inputs for this cycle into registers 3 through 31.
    3. Set the network enable HIGH via register 0.
    4. Wait for register 33 to report the successful completion of an execution cycle.
    5. Set the network enable LOW via register 0.
    6. Read the outputs of this cycle from registers 34 through 63.

The next step in the connection process is the previously alluded to microcontroller, which reads/writes data to the AXI-Lite Breakout. There is very little to say on this matter, as the FPGA made available for this work was a Xilinx Zynq-7000 System on Chip (SoC), which includes an ARM Cortex-A9 processor on the die, which requires only the inclusion of an AXI-Interconnect peripheral to allow it to talk to the breakout [63]. In addition, given the support provided by the Xilinx tool chain, the Xilinx Software Development Kit (SDK), that is used to program the ARM core, is able to generate a driver for the AXI-Lite Breakout, which reduces the reading/writing of data to a single line of C.

The final stage is to allow communication between the ARM core and a PC, such that the epigenetic network can be configured and executed from software. This would allow the work already done to support the software network described in Chapter 4 to be carried over. Once again, the capabilities of the Xilinx tools provide a simple solution, in the form of the Lightweight TCP/IP stack, or LwIP. LwIP is an implementation of the TCP/IP networking protocol, designed for minimal resource usage, making it ideal for embedded systems applications [64]. As such, the Xilinx SDK is able to automatically generate a simple LwIP server, which would allow data to be sent to and from a PC via Ethernet. It is then a case of modifying this file such that the incoming data triggers the correct interactions with the epigenetic network via the AXI bus, and that the data is then sent pack to the PC. Thus, the control process for the epigenetic network is as follows:

1. PC and ARM core establish a network connection.
2. PC sends signal indicating subsequent data will be network parameters.
3. ARM core acknowledges this signal.
4. PC sends a network parameter, ARM core stores it in memory and acknowledges receipt. This is repeated until all parameters are sent.
5. PC sends signal indicating all parameters have been sent, and that subsequent data will be instructions.
6. ARM core acknowledges.
7. PC sends instruction to load the saved parameters into the epigenetic network.
8. ARM core executes the parameter loading sequence described previously in this section, determining addresses based on order sent.
9. Once all parameters are loaded, the ARM core sends back a completion signal.

10. The PC sends a signal indicating that subsequent data will be input values.
11. ARM core acknowledges this signal.
12. PC sends a network input, ARM core stores it in memory and acknowledges receipt. This is repeated until all input values are sent.
13. PC sends signal indicating all parameters have been sent, and that subsequent data will be instructions.
14. ARM core acknowledges.
15. PC sends instruction to execute a cycle of the epigenetic network.
16. ARM core performs the network execution sequence described previously in this section, storing the outputs of the network in memory.
17. Once all outputs are stored, the ARM core sends back a completion signal.
18. The PC sends a signal requesting an output value, the ARM core sends back the first one in memory, which is then removed.
19. This is repeated until the PC requests a non-existent value, at which point the ARM core will send an all outputs send signal.
20. The PC then possess all output values for a given set of inputs.

The source of both the parameters values and the inputs to the network will be discussed in chapters 6 and 7.

5.7 Summary

This chapter has detailed the architecture of the hardware epigenetic network which forms the heart of this work. It has shown the structure of the various individual elements; how those elements connect together to replicate the software based networks discussed in Chapter 4; and how those elements are able to be reconfigured either manually (to create networks of varying size and data-width) or automatically (to create networks with varying internal parameter values).

This chapter has also explained the hardware and software processes that allow the network to be connected to a PC, such that a program can automatically perform experiments using the network in a similar manner to a software implementation. This is a suitable point upon which to progress to Chapter 6, which begins to detail such experiments.

6 Experiment Overview

With both software and hardware networks now available, discussion of the experiments to be carried out can now begin. This chapter will detail the two varieties of experiment that will be carried out, as well as the tools used for each of them.

Section 6.1 details the general experimental process that will be employed throughout this work, in the form of the optimisation loop that will be used in all experiments. Section 6.2 will cover the Inverted Pendulum; a well understood and widely used control problem that has been employed as an evaluation tool since 1975 [65]. Additionally, it has was one of the three control problems used as the original benchmarks by Dr. Turner [66].

The second, covered in section 6.3, is Robot Foraging, a highly regarded benchmark as it encapsulates a variety of important problems [67], including exploration and navigation of environments; as well as object identification, manipulation and transport. Additionally, the use of a robotics benchmark provides an example of a resource constrained system, and a system operating in a dynamic environment, as a genuinely autonomous robot is limited in its on-board power supply and computational capacity. For example, the now wide spread ePuck robot is built around a Microchip Technology Inc. dsPIC30F6014A digital signal controller [68]. This chip consists of a RISC (Reduced Instruction Set Computer) processor, with 16-bit data and 24-bit instructions [69]. Its maximum clock speed is 40MHz, allowing for a maximum of 30 MIPS (Million Instructions per Second), although the ePuck's 3.3V power supply limits this to 10 MIPS. This draws an average of 46mA of current, up to maximum of 68mA, equating to a power consumption of 0.15W to 0.22W. While the ePuck's battery has a 5Wh capacity, it also powers the robot's motors and other systems, which results in 2-3 hours of operation on one charge. In contrast, the original epigenetic experiments were run on a standard desktop PC. Taking this author's machine as an example, this makes available an Intel Core i7-4790 CPU, a 64-bit processor (data and instructions) with average clock speeds of 3.6GHz [70]. Power consumption is also higher, averaging 80W during normal operations.

Finally, section 6.4 discusses the idea of a dynamic system vs a system (of any type) in a dynamic environment; a distinction that is important to the intended goals of this thesis and relevant to Chapters 7, 8 and 9, as they detail the experiments themselves.

6.1 General Experimental Process

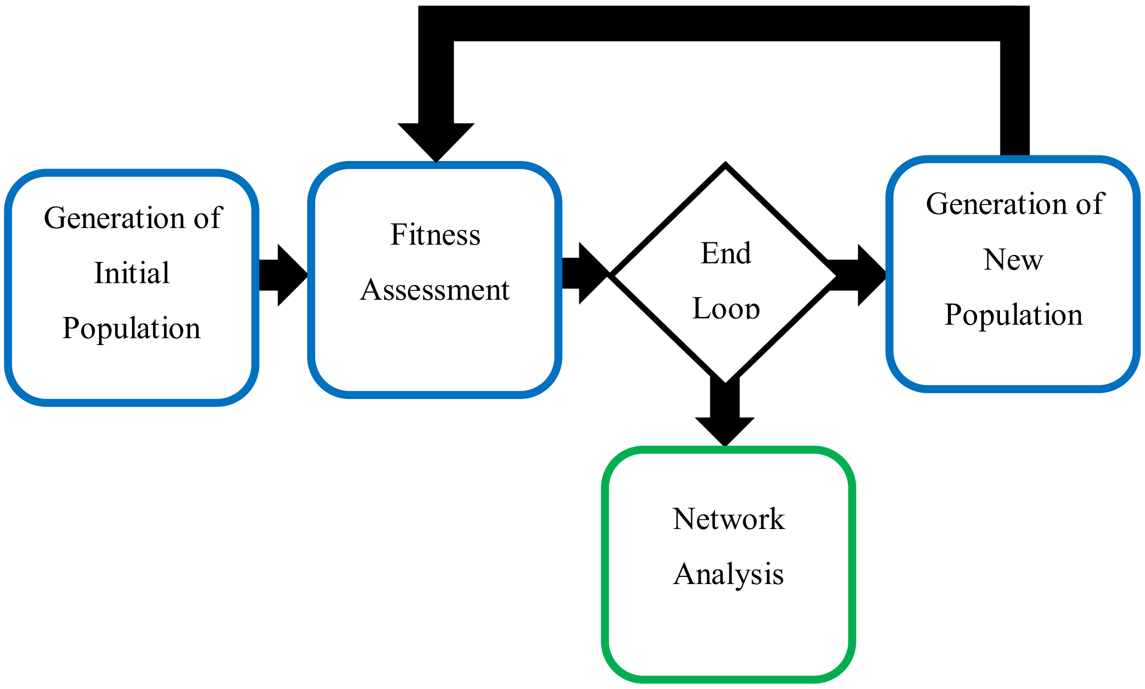

Generation of Initial Population
Fitness Assessment
End Loop
Generation of New Population
Network Analysis

Figure 6.1.1. Illustration of the general experimental process, including the software optimisation loop (shown in blue), and the on hardware analysis (shown in green).

Figure 6.1.1 shows the general process that will be employed in the experiments included in this work. Starting in software: an initial population of networks will be generated; then have its fitness assessed, based on the metrics that will be discussed later in this chapter. If the end condition of the optimisation process has not been met, then a new population will be generated, with the exact process being determined by which of the genetic algorithms, discussed in Chapter 3, is being employed. If the ending condition has been met, usually in the form of reaching a set number of iterations around the loop, or generations, then work moves over to the hardware. On hardware, analysis of the networks will be performed, such as investigation of epigenetic activity and network structure. In the cases where networks are exposed to stimulus they have not been evolved for, something that will be discussed in greater detail in Section 6.4 of this chapter, then this will also take place with the network being implemented on hardware. It should be noted however, that this hardware translation only applies to the epigenetic networks, as there is no hardware implementation of their non-epigenetic counterparts. Instead, these will remain confined to the realm of software.

6.2 Inverted Pendulum Simulation

As the inverted pendulum is a control problem that is both very well studied and not the primary focus of this work, a pre-existing model will be used. In particular, the one designed by Hamann et al [71] was chosen, for two reasons: it has prior history as a problem for epigenetic networks [6]; and it is explicitly intended for use as a robotics benchmark: instead of providing absolute measurements of factors like pendulum angle and velocity, the Hamann implementation uses simulated sensors that only monitor part of the model. In addition, the outputs of these sensors (as well as the control signals for the model's actuators) are low resolution, mapping all values to the range [0, 127]. The exact details of these sensors are given figure 6.2.1, which shows their position on the modelled cart; and table 6.2.1 which explains the function and parameters of each sensor

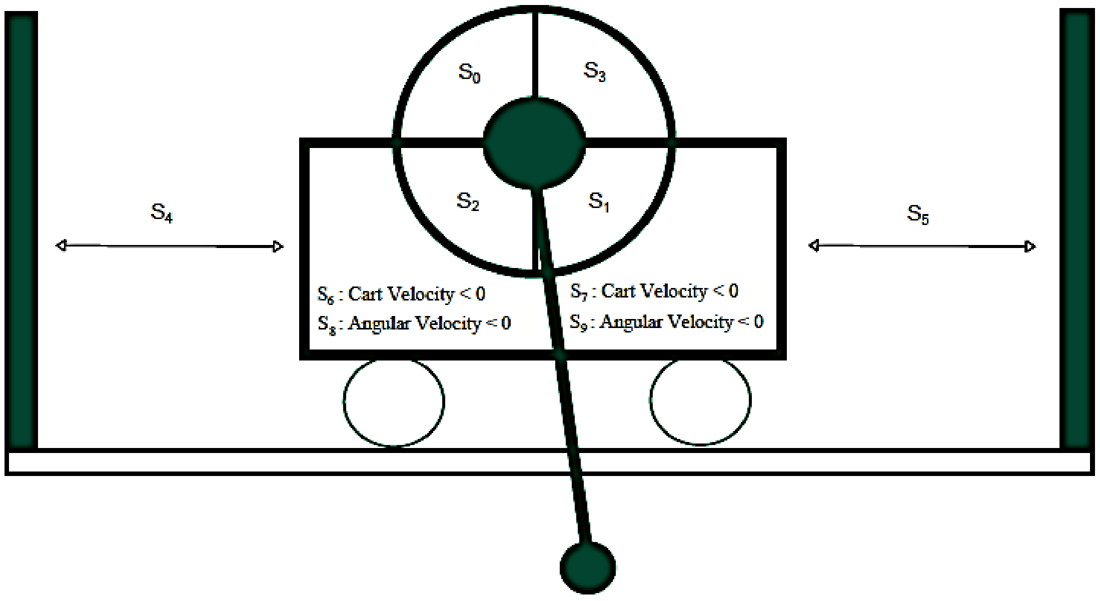

S_0
S_3
S_4
S_2
S_1
S_5
S_6 : Cart Velocity < 0
S_8 : Angular Velocity < 0
S_7 : Cart Velocity < 0
S_9 : Angular Velocity < 0

Figure 6.2.1. A cart and pendulum from the Hamann model, illustrating the "positions" of the various simulated sensors, taken from [71]. See table 6.3.1 for more detail the specific sensors.

Table 6.2.1. Details of the sensors. Note that the actual values returned by all sensors are mapped to the range [0, 127].

ID	Sensor Name	Sensor Details
S_0	Pendulum angle 1	Returns the angle of the pendulum (ϕ) when within the range [0, 0.5π], else returns 0
S_1	Pendulum angle 2	Returns the angle of the pendulum (ϕ) when within the range [π, 1.5π], else returns 0
S_2	Pendulum angle 3	Returns the angle of the pendulum (ϕ) when within the range [0.5π, π], else returns 0
S_3	Pendulum angle 4	Returns the angle of the pendulum (ϕ) when within the range [1.5π, 2π], else returns 0
S_4	Proximity 1	Returns the distance to the left hand end of the cart track, calculated from the cart position (x)
S_5	Proximity 2	Returns the distance to the right hand end of the cart track, calculated from the cart position (x)
S_6	Cart Velocity 1	Returns the velocity of the cart (v) when it is traveling leftwards, else returns 0
S_7	Cart Velocity 2	Returns the velocity of the cart (v) when it is traveling rightwards, else returns 0
S_8	Pendulum Velocity 1	Returns the angular velocity of the pendulum (ω) when it is rotating anti-clockwise, else returns 0
S_9	Pendulum Velocity 2	Returns the angular velocity of the pendulum (ω) when it is rotating clockwise, else returns 0

Looking at table 6.2.1 it is clear to see that all the sensors report some aspect of one of four factors: cart position, cart velocity, pendulum angle and pendulum angular velocity. At each time step, the value of these factors are updated, based on their previous value and a number of variables. The relationships are shown in equations 6.2.1 through 6.2.4, while the variables are expanded upon in table 6.2.2.

$$\dot{x} = x + v \tag{6.2.1}$$

$$\dot{v} = motor(u, v) \tag{6.2.2}$$

$$\dot{\phi} = \phi + \omega \tag{6.2.3}$$

$$\dot{\omega} = \begin{cases} \frac{3g}{2L}\sin\phi - \frac{3}{2L}motor(u,v)\cos\phi\,, & if\ \omega = 0 \\ \frac{3g}{2L}\sin\phi - \frac{3}{2L}motor(u,v)\cos\phi - K_p\omega|\omega| - K_l\frac{\omega}{|\omega|}, & else \end{cases} \tag{6.2.4}$$

Equations 6.2.1 and 6.2.3 are quite strait forward: the cart position at the next time step is directly based on the cart velocity; just as the new pendulum angle is directly based on its angular velocity. Equation 6.2.4, the angular velocity of the pendulum, is somewhat more complex, as the calculation depends on wither or not the pendulum is currently in motion. If not, the new angular velocity is based on the component of the cart velocity that is acting along the vector of the pendulum's current position, as well as the action of gravitational acceleration upon the pendulum, again modified by the pendulum's angle. If, however, the pendulum is currently in motion, then the equation also takes into account the inertia that results from that motion.

Finally, the velocity of the cart, equation 6.2.2 is result of the function motor(*u, v*), shown in algorithm 6.2.1. This function determines the actual velocity applied by the motor, based on current and desired cart velocity, as well as model parameter ΔUV: the smallest change in velocity the motor is capable of (this can be conceptualised as a precision value for a hypothetical motor driver circuit).

Algorithm 6.2.1 motor(*u, v*)

```
if $|u - v| > \Delta UV$
    if $u > v$
        if $v \geq 0$
            $\text{motor}(u, v) = V_{\text{motor}}$
        else
            $\text{motor}(u, v) = V_{\text{break}}$
    else
        if $v \geq 0$
            $\text{motor}(u, v) = - V_{\text{break}}$
        else
            $\text{motor}(u, v) = - V_{\text{motor}}$
else
    if $u > v$
        if $v \geq 0$
            $\text{motor}(u, v) = V_{\text{motor}} / \Delta UV(u - v)$
        else
            $\text{motor}(u, v) = V_{\text{break}} / \Delta UV(u - v)$
    else
        if $v \geq 0$
            $\text{motor}(u, v) = V_{\text{break}} / \Delta UV(u - v)$
        else
            $\text{motor}(u, v) = V_{\text{motor}} / \Delta UV(u - v)$
```

V_{motor} and V_{break} are the maximum acceleration and deceleration values of the simulated motor; their values and those of the other model parameters are shown in table 6.2.2.

Table 6.2.2. Model parameters with default values.

Symbol	Parameter	Value
g	Acceleration due to gravity	9.81 ms^{-2}
L	Pendulum length	0.5 m
V_{motor}	Maximum acceleration	7.0 ms^{-2}
V_{break}	Maximum deceleration	8.5 ms^{-2}
w	Track length	2 m
-	Cart length	0.1 m
K_p		0.005
K_L		0.05 s^{-2}
ΔUV	Minimum velocity change	0.05 ms^{-1}

Implementation of the model is done using the Runge-Kutta method of the 3rd order [72], with the discreet time step (Δt) being set to 0.01s. Equations 6.2.5 through 6.2.20 show how this method is applied to equations 6.2.1 through 6.2.4: the direct relationships of the original equations are replaced with differential equations, each term of which is the original broken down into fractional time steps.

$$\dot{x} = x + \Delta t\left(\frac{1}{6}x_{d1} + \frac{4}{6}x_{d2} + \frac{1}{6}x_{d3}\right) \tag{6.2.5}$$

$$x_{d1} = v \tag{6.2.6}$$

$$x_{d2} = v + v_{d1}\left(\frac{1}{2}\Delta t\right) \tag{6.2.7}$$

$$x_{d3} = v - v_{d1} + v_{d2}(2\Delta t) \tag{6.2.8}$$

$$\dot{v} = v + \Delta t\left(\frac{1}{6}v_{d1} + \frac{4}{6}v_{d2} + \frac{1}{6}v_{d3}\right) \tag{6.2.9}$$

$$v_{d1} = motor(u, v) \tag{6.2.10}$$

$$v_{d2} = motor(u, x_{d2}) \tag{6.2.11}$$

$$v_{d3} = motor(u, x_{d3}) \tag{6.2.12}$$

$$\dot{\phi} = \phi + \Delta t\left(\frac{1}{6}\phi_{d1} + \frac{4}{6}\phi_{d2} + \frac{1}{6}\phi_{d3}\right) \tag{6.2.13}$$

$$\phi_{d1} = \omega \tag{6.2.14}$$

$$\phi_{d2} = \omega + \omega_{d1}\left(\frac{1}{2}\Delta t\right) \tag{6.2.15}$$

$$\phi_{d3} = \omega - \omega_{d1} + \omega_{d2}(2\Delta t) \tag{6.2.16}$$

$$\dot{\omega} = \omega + \Delta t\left(\frac{1}{6}\omega_{d1} + \frac{4}{6}\omega_{d2} + \frac{1}{6}\omega_{d3}\right) \qquad (6.2.17)$$

$$\omega_{d1} = \begin{cases} \frac{3g}{2L}\sin\phi - \frac{3}{2L}motor(u,v)\cos\phi, & if\ \omega = 0 \\ \frac{3g}{2L}\sin\phi - \frac{3}{2L}motor(u,v)\cos\phi - K_p\omega|\omega| - K_l\frac{\omega}{|\omega|}, & else \end{cases} \qquad (6.2.18)$$

$$\omega_{d2} = \begin{cases} \frac{3g}{2L}sin\left(\phi + \phi_{d1}\left(\frac{1}{2}\Delta t\right)\right) \\ -\frac{3}{2L}motor(u,x_{d2})\,cos\left(\phi + \phi_{d1}\left(\frac{1}{2}\Delta t\right)\right), & if\ \phi_{d2} = 0 \\ \frac{3g}{2L}sin\left(\phi + \phi_{d1}\left(\frac{1}{2}\Delta t\right)\right) \\ -\frac{3}{2L}motor(u,x_{d2})\,cos\left(\phi + \phi_{d1}\left(\frac{1}{2}\Delta t\right)\right) \\ -K_p\phi_{d2}|\phi_{d2}| - K_l\frac{\phi_{d2}}{|\phi_{d2}|}, & else \end{cases} \qquad (6.2.19)$$

$$\omega_{d3} = \begin{cases} \frac{3g}{2L}sin(\phi - \phi_{d1} + \phi_{d2}(2\Delta t)) \\ -\frac{3}{2L}motor(u,x_{d3})\,cos(\phi - \phi_{d1} + \phi_{d2}(2\Delta t)) & if\ \phi_{d3} = 0 \\ \frac{3g}{2L}sin(\phi - \phi_{d1} + \phi_{d2}(2\Delta t)) \\ -\frac{3}{2L}motor(u,x_{d3})\,cos(\phi - \phi_{d1} + \phi_{d2}(2\Delta t)) \\ -K_p\phi_{d3}|\phi_{d3}| - K_l\frac{\phi_{d3}}{|\phi_{d3}|}, & else \end{cases} \qquad (6.2.20)$$

The final thing to note is the fitness function, equation 6.2.21, which returns a value proportional to the number of time steps that the pendulum occupies an upright position ($\phi = 0$).

$$\boldsymbol{F} = \sum_{t=0}^{t_{max}} \frac{|\boldsymbol{\phi}(\boldsymbol{t}) - \boldsymbol{\pi}|}{\boldsymbol{t}_{max}\, \boldsymbol{\pi}} \qquad (\mathbf{6.2.21})$$

t_{max} is the maximum number of time steps the model will be simulated for. The model will run for this long unless: the cart hits the end of the track; the pendulum exceeds its maximum angular velocity (5π rad s^{-1}); or the cart exceeds its maximum velocity (2 ms^{-1}). Such a premature stop will result in a reduced fitness. It should be noted that, as the pendulum starts in the lower quadrant, it is not possible to achieve a fitness of 1.0, as this requires the pendulum to occupy the upright position for all its time steps.

Chapter 7 will cover the actually experiments that this model will be utilised in, while the next section of this chapter will cover the other type of experiment: foraging robots. As discussed in the introduction, a robotics application has been chosen as it is a good example of a resource limited system that could benefit from the reconfigurable nature of an epigenetic network.

6.3 Robotic Foraging

The task of robot foraging is a widely used benchmark in robotics, inspired by the behaviour of foraging in various biological systems, especially social insects. Østergaard et al described it as a "process in which 1) robots search a designated region of space for certain objects, and 2) once found these objects are brought to a goal region using some form of navigation [73]."

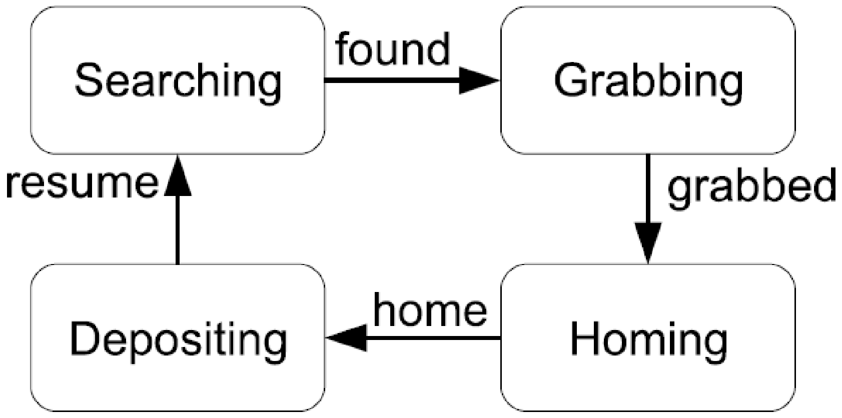
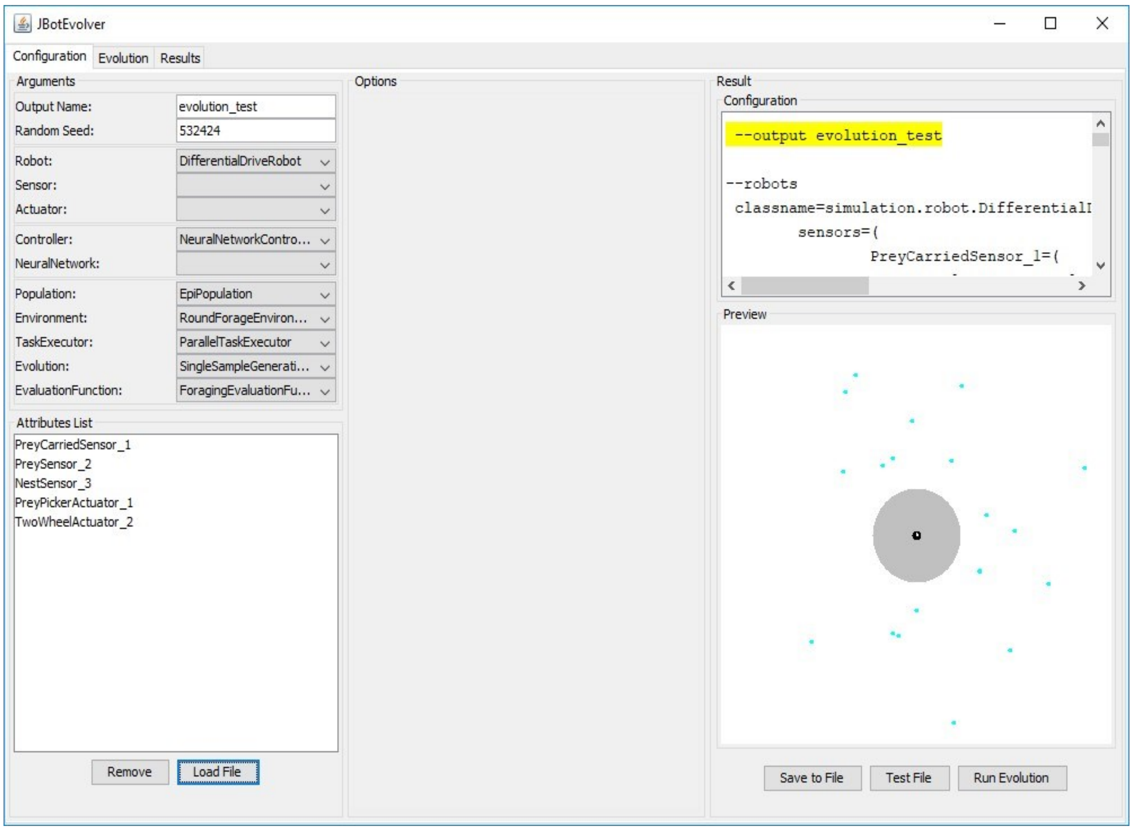


Figure 6.3.1. A Finite State Machine (FSM) representation of basic foraging, taken from [67].

Figure 6.3.1 breaks down the foraging process in more detail by conceptualising it as a Finite State Machine (FSM). Starting in the upper left, the robot searches the environment for the object they are after. Once successfully located, it is acquired, typically with some type of manipulator or "grabber", ready for transport. The robot then searches the environment again, but this time for its home location, where it deposits the acquired object before returning to the start of the process. It is this compartmentalisation that raises the possibility of performance gains through the use of epigenetic mechanisms. If an epigenetic network were evolved to control a robot carrying out this task, then it is likely that the network would also become compartmentalised. In a very simplistic example, the network could be divided between elements dedicated to prey acquisition and prey return behaviours. Additionally, a robot operating within an environment containing other objects that it has no direct insight into is an excellent example of a system in a dynamic environment, a concept that will be explored in greater detail in section 6.4 of this chapter.

As this work utilises evolutionary methods to optimise the control networks, using a real robot would be impractical, as time even a single test of a possible network configuration would be considerable, and attempting to accelerate the process by testing with many robots in parallel would prohibitively expensive. Therefore, a computer model of a robot will be employed along with a suitable simulated environment for it to interact with. The robot model will be controlled by an implementation of the epigenetic network, either as a software application or the actual hardware version. Much like the inverted pendulum, robotics simulations are widely used and not the focus of this work, so again an off the shelf solution will be employed. After investigating a few possibilities including ARGoS [74], Webots [75], and V-Rep [76]; a suitable application was found in the form of JBotEvolver, shown in figure 6.3.2, a Java program designed at ISCTE – University Institute of Lisbon specifically for experiments involving the evolution of robotic controllers [77].



Output Name:	evolution_test
Random Seed:	532424
Robot:	DifferentialDriveRobot
Sensor:	
Actuator:	
Controller:	NeuralNetworkContro...
NeuralNetwork:	
Population:	EpiPopulation
Environment:	RoundForageEnviron...
TaskExecutor:	ParallelTaskExecutor
Evolution:	SingleSampleGenerati...
EvaluationFunction:	ForagingEvaluationFu...

PreyCarriedSensor_1
PreySensor_2
NestSensor_3
PreyPickerActuator_1
TwoWheelActuator_2

```
--output evolution_test

--robots
 classname=simulation.robot.DifferentialI
       sensors=(
               PreyCarriedSensor_1=(
```

Figure 6.3.2. Screenshot of JBotEvolver's GUI, configuration view. The left hand side is the configuration arguments menu; the centre the options for a selected argument; and the right hand side and overview of the configuration and the simulator environment.

JBotEvolver is comprised of two primary components. JBotEvolver itself generates initial populations of controllers, executes fitness evaluations and the evolutionary algorithm. It is able to execute all stages of these operations in parallel thanks to Java's inherent multi-threading capabilities. The other half is JBotSim, which translates the populations from JBotEvolver into actual controllers, before handling the simulation of the robots and the environment they occupy. There is also an optional tool called EvolutionAutomator, which allows the automatic execution of multiple evolutionary runs, in parallel if able. The automator can also use different configuration options of each evolutionary experiment. JBotEvolver allows for easy design of experiments through the use of a configuration file that can be used to specify every aspect: from the parameters of its integrated genetic algorithm, which is detailed in Chapter 3; to the exact specification of the various sensors that the robot(s) can be equipped with. The creation of custom variations of all these elements can also be easily achieved, as JBotEvolver's code is open source and available through GitHub [78].

With a platform chosen, consideration should first be given to the design of the robot that will be simulated using it. Looking at an example of one that has been successfully used to carry of the foraging task would be an excellent starting point.

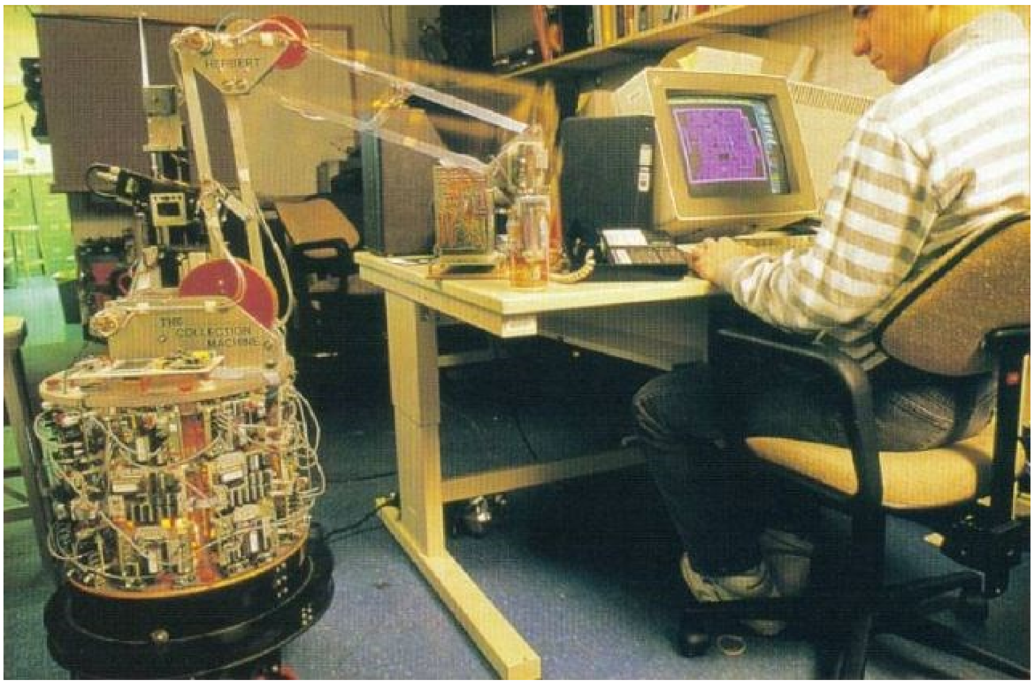


Figure 6.3.3. Herbert, the Collection Machine, one of the first examples of a foraging robot. Built in the 1980s at MIT by Rodney Brooks et al, it was able to autonomously navigate the laboratory, collect objects of interest (usually soft drinks cans) and return them to its home location [79]. Image taken from [80].

Figure 6.3.3 shows one of the first robots to successfully perform this kind of task: Herbert, or the collection machine, built at MIT. Herbert utilised a number of sensor systems, including an early LIDAR scanner, to navigate the MIT AI laboratory in search of soft drinks cans. Once located, its gripper arm picked up the can, before it transported it to a designated home location, usually a waste bin [79]. From both this and the breakdown of the foraging task already performed, it can be deduced that the simulated robot will require the following elements:

- A means of locomotion about the environment.
- A sensor for detecting obstacles or other environmental hazards.
- A sensor for detecting the prey objects that it is to acquire.
- A means of picking up or otherwise moving the prey object.
- And a method of navigating to its drop off point.

Fortunately, all of these components are built into JBotEvolver, and as such it is simply a matter of creating the configuration file that describes the robot, which is shown in figure

6.3.4. Additionally, figure 6.3.5 shows an illustration of the robot. It starts with a standard DifferentalDriveRobot, a java object that the simulator can add other components, such as the sensors and actuators that are described next. Sensor 1 is quite strait forward, as it simply indicates if the robot is currently carrying a prey object. Sensors 2, 3 and 4 on the other hand are a little more complex. They are Prey, Nest and Wall sensors respectively, each of which responds to their respective object within the environment. On a functional level, these sensors are modified versions of JBotEvolver's LightSensor class, they simply treat the objects they are designed to detect as "light sources". The parameters for all three of these sensor types are the same:

- NumberSensors: how many of this type of sensor at to be attached to the robot, in this case 2. Normally, they will be spaced equidistant around its body.
- Angle: the angle of the sensors vision cone, in this case 60°.
- Eyes: setting this parameter to 1 changes the way in which the sensors are placed on the robot. Instead of equidistant spacing, they will instead be placed at n° increments away from the front of the robot, with n being the value of the EyesAngle property. In this case, EyesAngle is 15°, which means the two sensors will be placed on either side of the robot's front, a total of 30° apart.

The two actuators allow the robot the ability to move and interact with its simulated environment. The first is a PreyPickerAcctuator, which is able to grab and hold prey objects, allowing them to be transported; while the TwoWheelActuator allows the robot itself to move by adding a simple two wheeled differential drive.

```
--robots
      classname=DifferentialDriveRobot,
      sensors=(
            PreyCarriedSensor_1=(
                  classname=PreyCarriedSensor,
                  id=1
            ),
            PreySensor_2=(
                  classname=PreySensor,
                  numbersensors=2,
                  angle=60,
                  eyes=1,
                  eyesangle=15,
                  id=2
            ),
            NestSensor_3=(
                  classname=NestSensor,
                  numbersensors=2,
                  angle=60,
                  eyes=1,
                  eyesangle=15,
                  id=3
            ),
            WallRaySensor_4=(
                  classname=WallRaySensor,
                  numbersensors=2,
                  angle=60,
                  eyes=1,
                  eyesangle=15,
                  id=4
            )
      ),
      actuators=(
            PreyPickerActuator_1=(
                  classname=PreyPickerActuator,
                  id=1
            ),
            TwoWheelActuator_2=(
                  classname=TwoWheelActuator,
                  id=2
            )
      )
```

Figure 6.3.4. The JBotEvolver configuration file for a foraging robot.

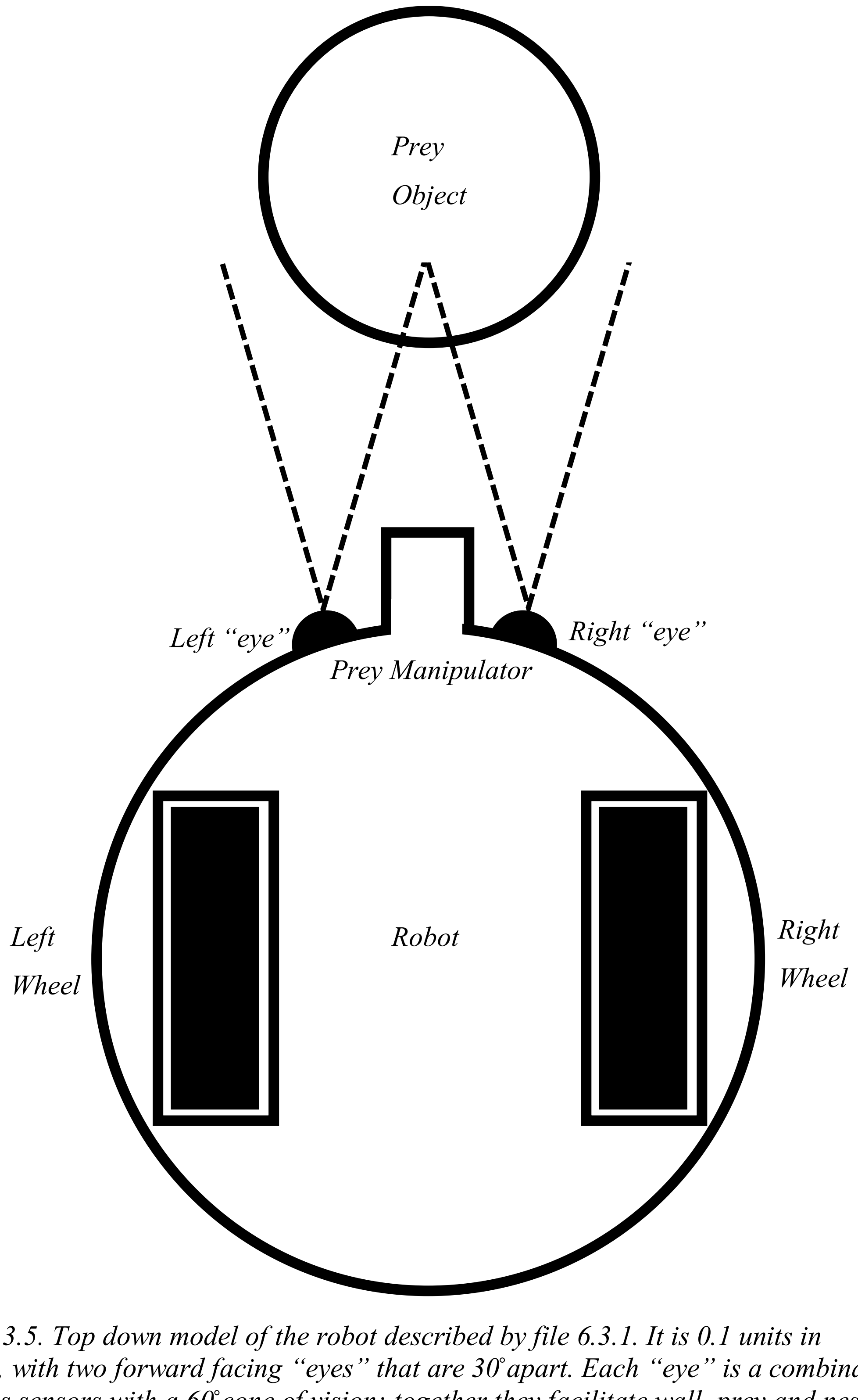


Figure 6.3.5. Top down model of the robot described by file 6.3.1. It is 0.1 units in diameter, with two forward facing "eyes" that are 30° apart. Each "eye" is a combination of various sensors with a 60° cone of vision; together they facilitate wall, prey and nest detection. Two wheels create a differential drive, with a maximum speed of 0.1 units per time step. The protrusion on the front is the prey actuator, which allows manipulation of prey objects. It incorporates a sensor that detects wither or not the actuator is holding a prey object. Also shown is a prey object, which is a simple cylinder 0.05 units in diameter.

As for the environment this robot will be tested in, two different types will be created: a simple foraging environment designed to test basic prey identification, acquisition and return; and a set of more complex, maze type environments, which will add not only navigation, but object avoidance to the foraging task.

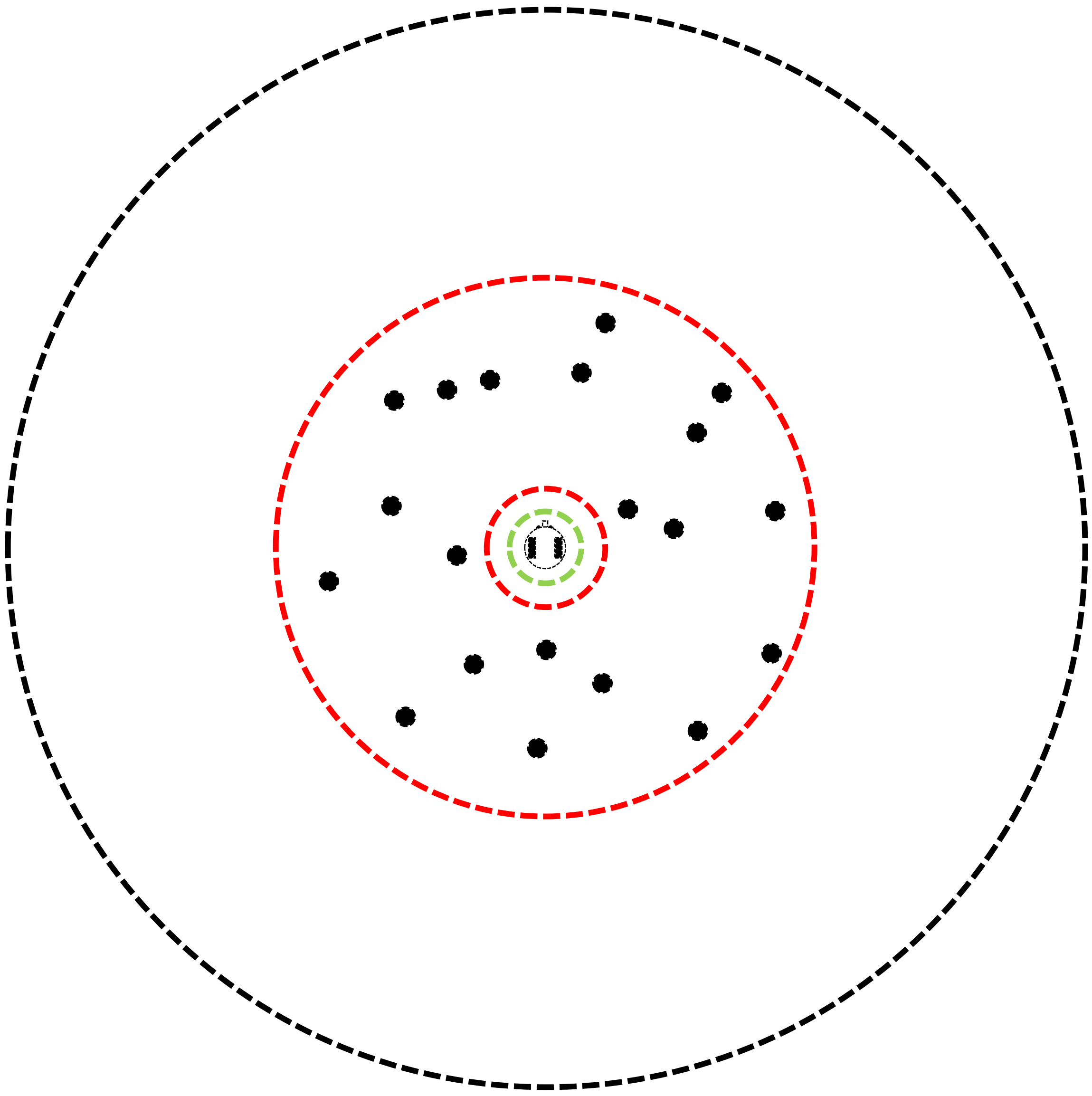


Figure 6.3.6. Illustration of the basic foraging environment. The robot starts within the nest, the green circle, at the centre of the environment. Prey objects, the black circles, are randomly spawned in the region between the two red circles, the outer of while defines the forage limit: the furthest from the nest a robot will need to move in order to gather prey. The large black circle represents the boundary of the environment, with robots being heavily penalised for traveling beyond it.

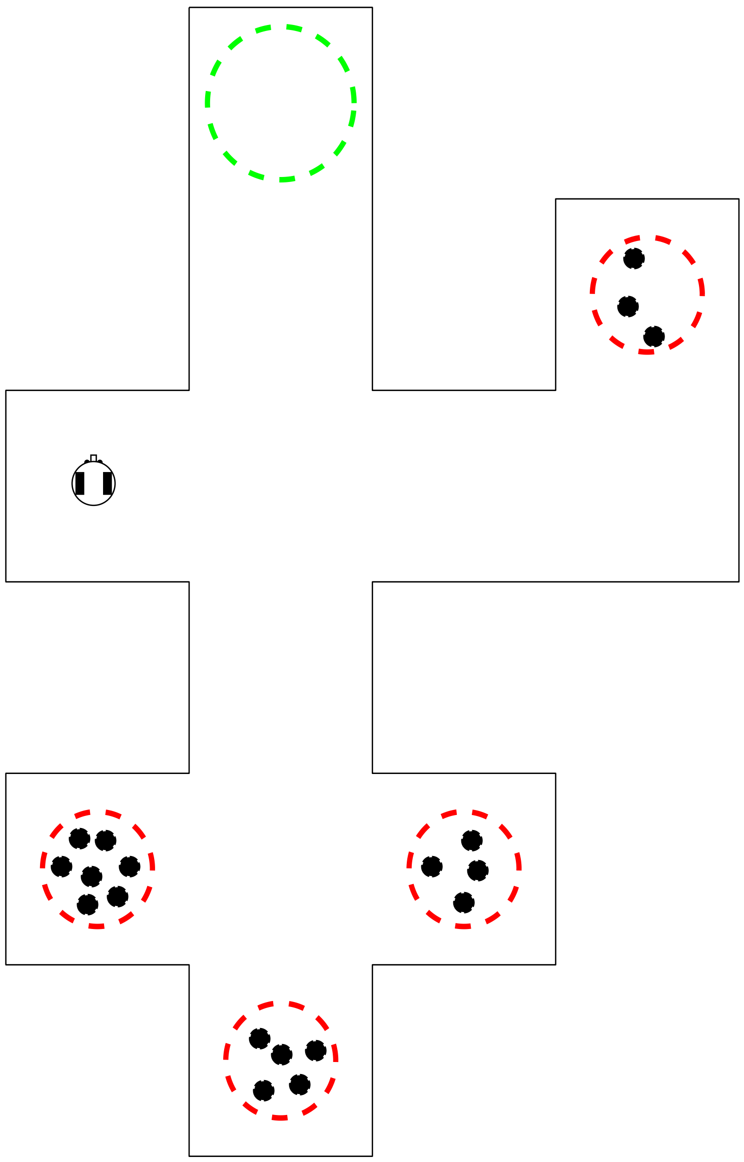


Figure 6.3.7. Illustration of the first maze foraging environment. The robot starts in the left alcove, while the nest, the green circle, is at the northern tip. Prey objects, the black circles, spawn within the four red circles, which are distributed around the maze and not immediately visible from the start point. There is no forage limit or environment boundary, as the maze is completely encapsulated by solid walls, which not only prevent travel, but also block the robot's sensors.

$$F = Prey_{nest} + \sum_{t=0}^{t_{max}} \left(\frac{1}{t_{max}} R_{carrying} - \frac{100}{t_{max}} R_{enviro\ limit} - \frac{0.1}{t_{max}} R_{forage\ limit} \right) \quad (6.3.1)$$

The fitness function for the foraging task, equation 6.3.1, is comprised of two parts; the simplest is the first term, $Prey_{nest}$, which is the number of prey objects returned to the nest. The second is the sum over the total run time of three weighted factors:

- $R_{carrying}$, which is 1.0 if the robot is currently carrying a prey object.
- $R_{enviro\ limit}$, which is 1.0 if the robot is beyond the boundary of the environment. This only really has an impact in the first environment.
- $R_{forage\ limit}$, which is 1.0 if the robot is beyond the foraging limit. Once again this only has an impact in the first environment.

In plain language, the robot will be rewarded for each time step it is carrying prey, and rewarded for each prey object it returns to the nest. While being tested in the non-maze environment, the robot can also be penalised for moving beyond the foraging limit or the boundary of the environment.

The second maze, shown in figure 6.3.8, isn't significantly different from the maze in figure 6.3.7, but the alteration to the distribution of prey objects, or more accurately: rescue targets, is part of the set up for a slightly different type of task: a rescue operation. As well as providing a real world context for the experiment, a recue task brings with it an additional factor that will be incorporated to increase the complexity: time pressure.

$$F = \sum^{total\ rescued} \left(\frac{t_{max} - t_{rescued}}{0.1 t_{max}} \right) + \sum_{t=0}^{t_{max}} \left(\frac{1}{t_{max}} R_{carrying} \right) \qquad (6.3.2)$$

Equation 6.3.2 shows the altered fitness function that incorporates this time pressure factor. The second half is the same as its equivalent in equation 6.3.1, but instead of simply adding the number of people rescued (returned to the nest), the new element instead corresponds to the point in time at which they were rescued. The closer to the start of the simulation, the greater the value. Each person recovered has such a value, which are then summed together at the end of the simulation run, producing a fitness that rewards rescuing as many people as possible, as quickly as possible.

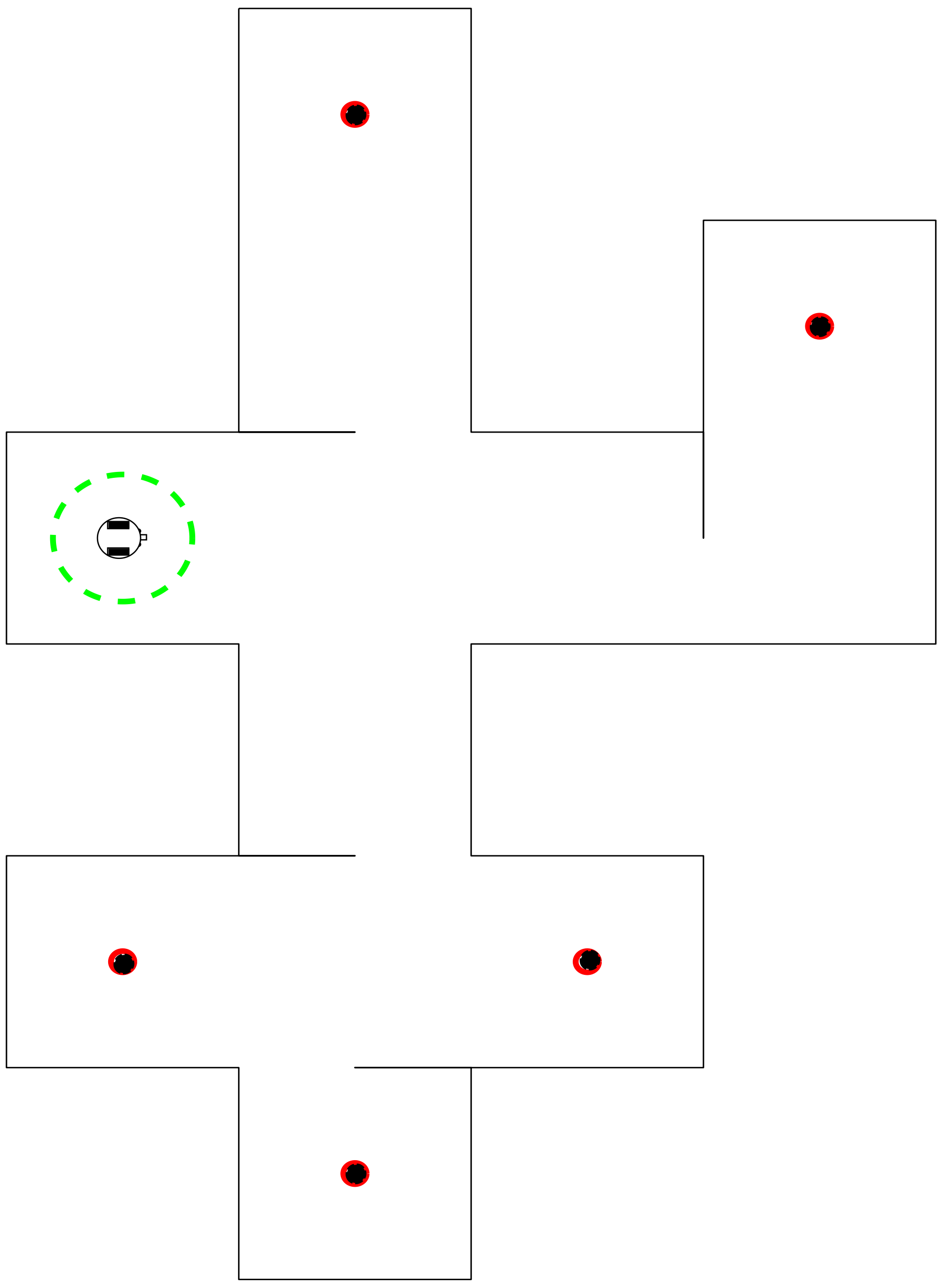


Figure 6.3.8. Illustration of the second maze foraging environment. Once again the robot starts in the left alcove, but in this instance that is also the location of the nest. There are only five prey objects, but they are distributed in similar locations around the maze. There is no forage limit or environment boundary, as the maze is completely encapsulated by solid walls, which not only prevent travel, but also block the robot's sensors.

6.4 Dynamic Environment

The ultimate goal of this project is to show that an epigenetic network is able to demonstrate improved performance over a more standard gene regulatory of neural network, even when restricted to the reduced precision of a hardware implementation, and especially when confronted with a degree of unpredictability. Such networks have already been shown to give measurable performance improvements when controlling dynamic systems; however, these systems do not demonstrate the kind of unpredictability this project wishes to investigate.

It is the opinion of the author that a distinction can be made between a dynamic system, and a system that exists within a dynamic environment. An example of the former would be an inverted pendulum: a system that possesses complex behaviours, ones that even render it unstable under certain conditions. However, these behaviours can all be considered internal to the system. A model like the one detailed in 6.2, encapsulates everything about not only the pendulum itself, but everything that might affect the, uncontrolled, pendulum's behaviour. Simply put, no matter how complex it might be, a dynamic system is usually considered in isolation. Taking this idea forward, a system within a dynamic environment can be described thus: a system of arbitrary complexity, the behaviour of which is altered due to external factors not considered part of the system itself. To return to the inverted pendulum, a simple of example of a system in a dynamic environment would be a pendulum that is subject to an externally provided impulse. This distinction between a system and the environment it exists in becomes more important when a controller is introduced. A controller for an inverted pendulum is something easily designable, the reason for its use in the teaching of control theory. If such a simple controller were connected to a pendulum, which after an arbitrary period of time were to be subject to external inference, the controller would most likely experience difficulty maintaining the stability of the system. The impulse was not encapsulated within the pendulum model, which was used to design to controller. Put simply, the controller was not designed to handle the environment surrounding the system it controls.

The great strength of epigenetic networks is their capacity to modify their own behaviour. It is the opinion of the author that this may extend to the ability to handle changes in the system under the networks control, even if such changes come from an external source.

To investigate this possibility, networks evolved under the conditions described in Sections 6.2 and 6.3 of this chapter will be evaluated in environments that incorporate these dynamic factors. In the inverted pendulum's case, the example already discussed in this section will be utilised: the injection of an impulse of force, representing something like the pendulum being subjected to a "kick". This impulse will be modelled as an increase in the pendulum's angular velocity, occurring between two time steps. The robot foraging's environmental dynamics however will be much more varied.

6.4.1 Mobile Prey Objects

In the standard foraging task, once a prey object has been spawned into the environment, it remains in place, unless moved either by the robot picking it up or by something colliding with it. It is possible to think of this scenario as being akin to the food acquisition of a herbivore: the only thing moving about is the creature/robot. In comparison, consider the more complex situation facing predatory animals, wherein their food is most often also ambulatory. It is from this line of logic that the first environmental dynamics come: the switch from static to mobile prey objects. While a more complete simulation of ambulatory prey, with its own distinct behaviours, would be interesting, it is beyond the scope of this work. So instead, a simplified model will be used, where the prey objects exhibit random motion. Algorithm 6.4.1 shows the process by which the random motion is achieved, as well as the inclusion of a function to avoid the prey wondering into the nest, as this would result in a scenario where the robot could achieve a high fitnesses while doing completely nothing. It should also be noted that the maximum distance the prey can move in a time step must be lower than the robot's maximum speed. Otherwise the possibility exists that the robot will never catch anything.

Algorithm 6.4.1 Introduction of random motion into prey objects

```
for number of prey in the environment
    Generate random movement distance
    Generate random movement direction
    Calculate new position based on current position/distance/direction
    If new position is inside the nest
        Regenerate distance and movement
    Else
        Move to new position
```

6.4.2 Falling Debris

In addition to the factors already discussed, contextualising the foraging task as a search and rescue one brings with it another benefit: an ample supply of possible environmental dynamics. For example, figure 6.3.1 shows a test environment used by the robocup rescue challenge, an international competition designed to "Develop and demonstrate advanced robotic capabilities for emergency responders [81]", include features such as: tight, winding corridors; slopes, ramps and stairs; uneven ground; and simulated fires and chemical hazards [82].

Figure 6.4.1. A robot competing in the 2017 RoboCup Rescue League. Note the presence of simulated hazardous materials as it attempts to complete its task. Image taken from [82].

In this case of this work, the hazard chosen to introduce environmental dynamics is: falling debris, as it is readily implementable via the tools provided by JBotSim. Figure 6.3.2 shows the rescue environment, previously depicted in figure 6.2.8. Note however, the addition of the 6 black regions, distributed throughout. At the start of the simulation, these squares are empty; but a pre-determined time steps, a random one of them will become filled with a wall object, simulating a fall of debris. In abstract terms, this results in the environment becoming harder to navigate as time progresses.

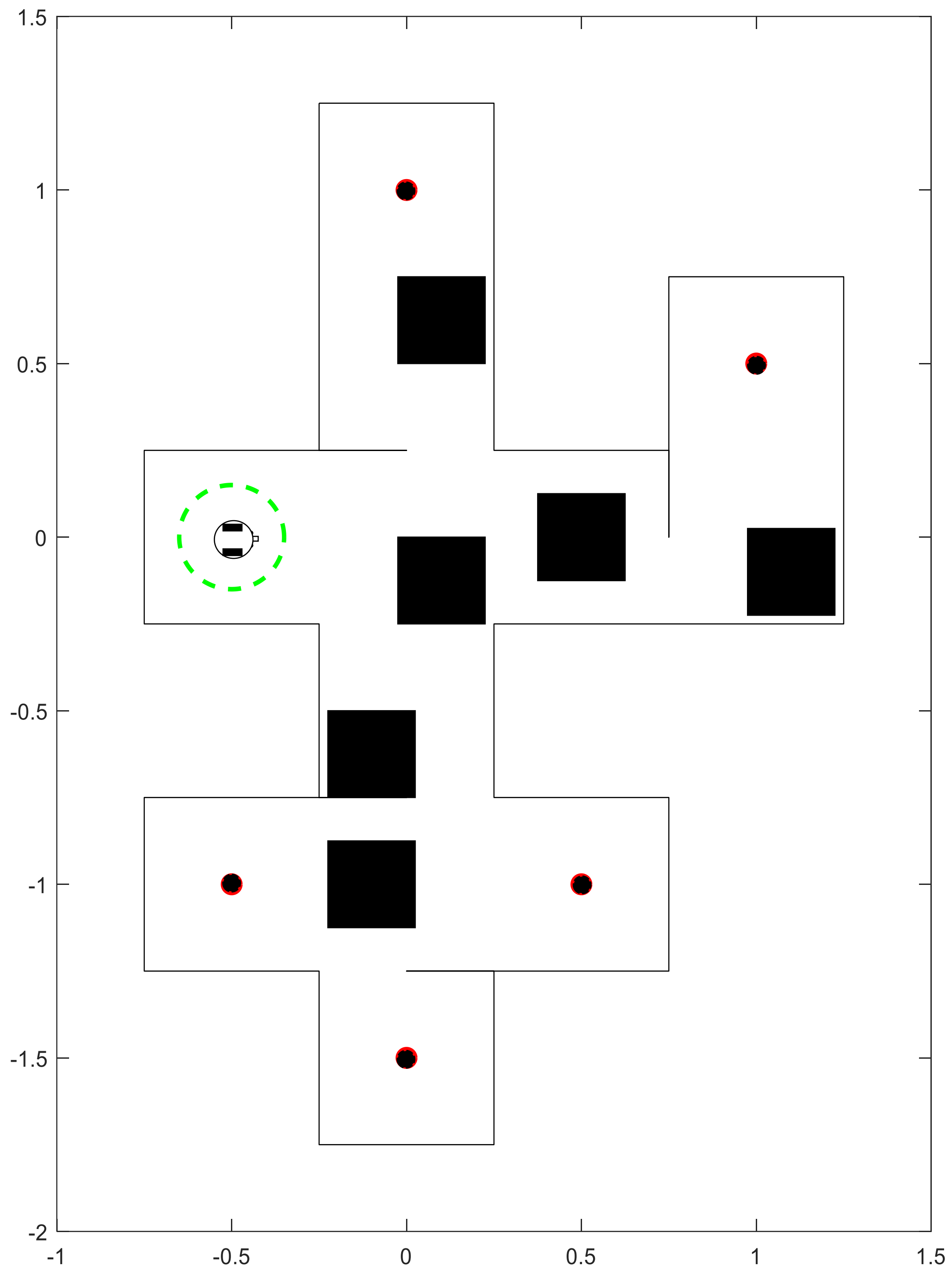


Figure 6.4.2. Illustration of the rescue environment, with the inclusion of the debris fall locations, depicted as black squares.

6.5 Summary

This chapter has covered the tools created to facilitate the experiments of this thesis: an inverted pendulum simulation and simulation of a robot, as well as the varied environments that it will occupy. The tasks that the epigenetic network will be evolved to perform have also been discussed, along with the various fitness functions that will be used to characterise their performance.

Finally, section 6.4 elaborated on one of the core hypothesis of this thesis; the ability of an epigenetic network to handle the introduction of environmental dynamics into the system that it is controlling, while retaining a high degree of fitness. This section also covered the details of how these environmental dynamics will be introduced into the various tasks: impulse injection; moving prey objects; and the simulation of an unstable environment via falling debris. If the assertion that this thesis makes is correct, then not only will the epigenetic networks be able to maintain performance better than their non-epigenetic counterparts, but analysis of the networks should show the action of the epigenetic mechanisms as the cause. This will be put to the test in Chapter 7, which contains the methods, results and analysis of the experiments performed using the simulation of the inverted pendulum.

7 Inverted Pendulum Experiments

This chapter describes the experiments performed using the Hamann inverted pendulum model [71], described in detail in Chapter 6 of the thesis. The first set of experiments will replicate those of Turner's work: evolving an epigenetic controller capable of swinging up and balancing the pendulum [66], although in the experiments outlined in this chapter, the controllers will be integer based networks, first in software and then hardware (FPGA implementation). This will also involve experiments to determine suitable data width and network size for the integer network. This validation of functionality is then be built upon with new work, demonstrating the ability of an epigenetically enhanced controller to withstand externally introduced dynamics better than a standard Artificial Gene Regulatory Network (AGRN). The external perturbation takes the form of the injection of an impulse into the pendulum, analogous to an object hitting it. Finally, experiments involving the evolution of epigenetic controllers with two different methods of applying connection weights are described, giving justification to the changes detailed in Section 4.1.2.

7.1 Evolution of Integer Based Controllers

7.1.1 Experimental Design and Parameters

A major aim of this work is to implement a form of epigenetic network model on to a hardware platform, in this case an FPGA. The first step is to validate that the translation from a software floating-point network implementation to a hardware integer network implementation and to show that this does not adversely affected the performance of the epigenetic network. This was done in several stages: first, a series of experiments were performed to identify the data width for the integer network, more specifically how small it could be made without negatively impacting the network's performance. These experiments used the simple genetic algorithm, detailed in section 3.2, to evolve a number of controllers for the Hamann inverted pendulum model [71], with the data width varying between 4 and 32 bits. Ideally, this range would go up to 52 bits, so as to be equivalent to the mantissa width of the 64 bit floating point values of Turner's original floating-point networks [66], but this limitation is imposed by the 32 bit width of the AXI-Lite bus that the hardware implementation uses for communication (see section 5.6 for details). Any data width lower than 4 is considered too small to be worth testing, as this would cause the connection mapping mechanism described in section 4.1 to connect all the network elements together in one large mass. The results of these experiments will allow the selection of a suitable data width that demonstrates a high enough fitness while also being efficient to implement in hardware. A similar process will then be repeated to determine the number of genes and epigenetic molecules the final network iteration will require, with a range of 12 to 25 genes and 3 to 5 molecules, these being the range of values used in Turner's original work [6]. Note that while Turner's work allowed network size to be an evolvable parameter, these experiments determine a fixed value. This is necessary because the hardware implementation does not facilitate a change in the number of network elements without resynthesise, so it must be of fixed dimensions.

With data width and network size determined, the final evolution of the software network with reduced size is performed, the results of which are compared to Turner's original data. Success for the integer epigenetic network is defined as the integer network demonstrating finesses equivalent to those of the floating-point networks, within the limits of variation due to the non-deterministic nature of genetic algorithms. Next, a suitable candidate from among these networks is used to generate a configuration for the hardware platform, which is then tested to see if it matches the software network's control ability. Success is defined as the hardware network performing the same as its software counterpart.

Throughout these experiments, the parameters of the inverted pendulum model are the same as those noted in table 6.2.2 in the previous chapter. The parameters for the simple genetic algorithm and integer network are specified in the tables 7.1.1, 7.1.2 and 7.1.3.

Table 7.1.1. Parameters for the data width determination evolution using the simple genetic algorithm.

Parameter	Value
Population Size	64
Number of Generations	2048
Number of Repeats	50
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	4 - 32
Number of Genes	20
Number of Epigenetic Molecules	3

Population size, number of generations and the number of repeats are chosen from experimental experience, crossover and mutation rates come from Turner's original work [6], and equate to a 50% chance of a crossover and a 5% chance of a mutation (see Chapter 3 for a definition of these operators and section 3.2 for the specifics of them within the simple GA). The number of genes and molecules for these experiments were arbitrarily chosen from the ranges that will be used later.

Table 7.1.2. Parameters for the network size determination evolution using the simple genetic algorithm.

Parameter	**Value**
Population Size	64
Number of Generations	2048
Number of Repeats	50
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	Determined by prior experiments
Number of Genes	12-25
Number of Epigenetic Molecules	3-5

Table 7.1.3. Parameters for the final evolution of the integer network using the simple genetic algorithm.

Parameter	**Value**
Population Size	500
Number of Generations	200
Number of Repeats	50
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	Determined by prior experiments
Number of Genes	Determined by prior experiments
Number of Epigenetic Molecules	Determined by prior experiments

7.1.2 Results

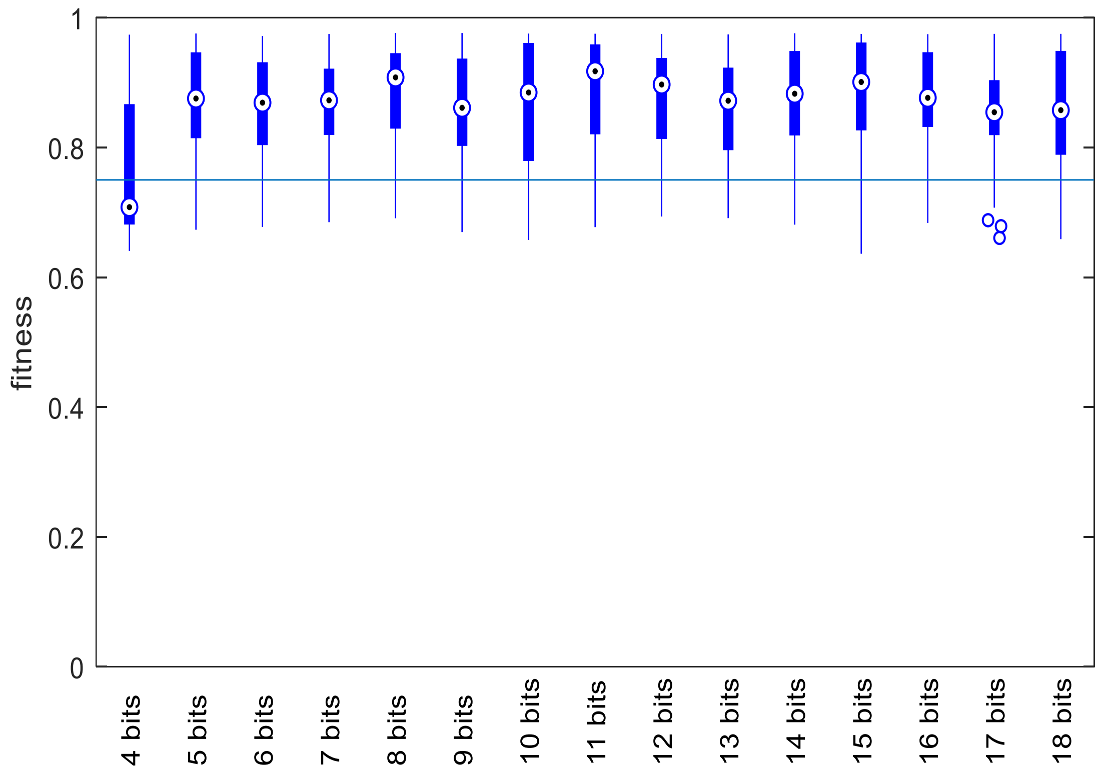


Figure 7.1.1. Fitnesses of the best integer epigenetic networks from each repeat, with data widths in the 4- to 18-bit range. The blue line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position. Note that: the dotted circle represents the median fitness; the box encapsulates values between the 25th and 75th percentiles; the whiskers encapsulate all non-outliers; and the circles are outliers. MatLab's boxplot function defines outliers as values greater than $q_3 + w(q_3 - q_1)$, or less than $q_3 - w(q_3 - q_1)$; where q_1 and q_3 are the 25th and 75th percentiles of the data, and w is the maximum whisker length, defined as $\pm 2.7\sigma$.

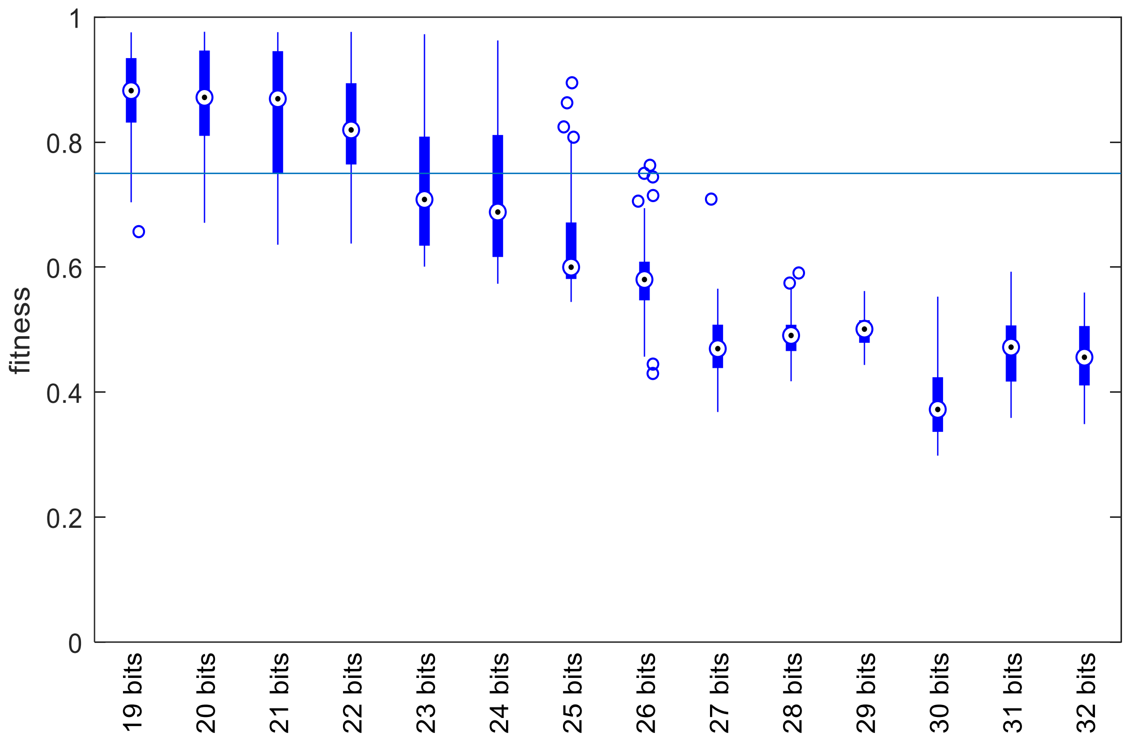


Figure 7.1.2. Fitness of integer epigenetic networks with data widths in the 19- to 32-bit range.

The results of the data width evolutions are show in the figures 7.1.1 and 7.1.2. 4-bit networks are able to exceed the 0.75 fitness threshold, but most fall short, with a median fitness of 0.71. Networks with data widths between 5 and 20-bits all exhibit similar results to each another, with most networks achieving a fitness greater than 0.75.

However, unexpected behaviour occurs with the networks with data widths beyond this point. Given that the floating-point networks have a data width of 64 bits, it should be expected that the fitness values would slowly increase as the data widths got closer to this value, but instead a decline begins at 21-bits. By 23-bits the median value is below 0.75, and by 27-bits even outliers are unable to exceed this fitness. To understand what occurred, a further experiment was performed. Two networks were evolved, an 8-bit and a 32-bit, in these cases however the number of generations was increased to 4000. The intention was to compare the change in fitness over a longer time period.

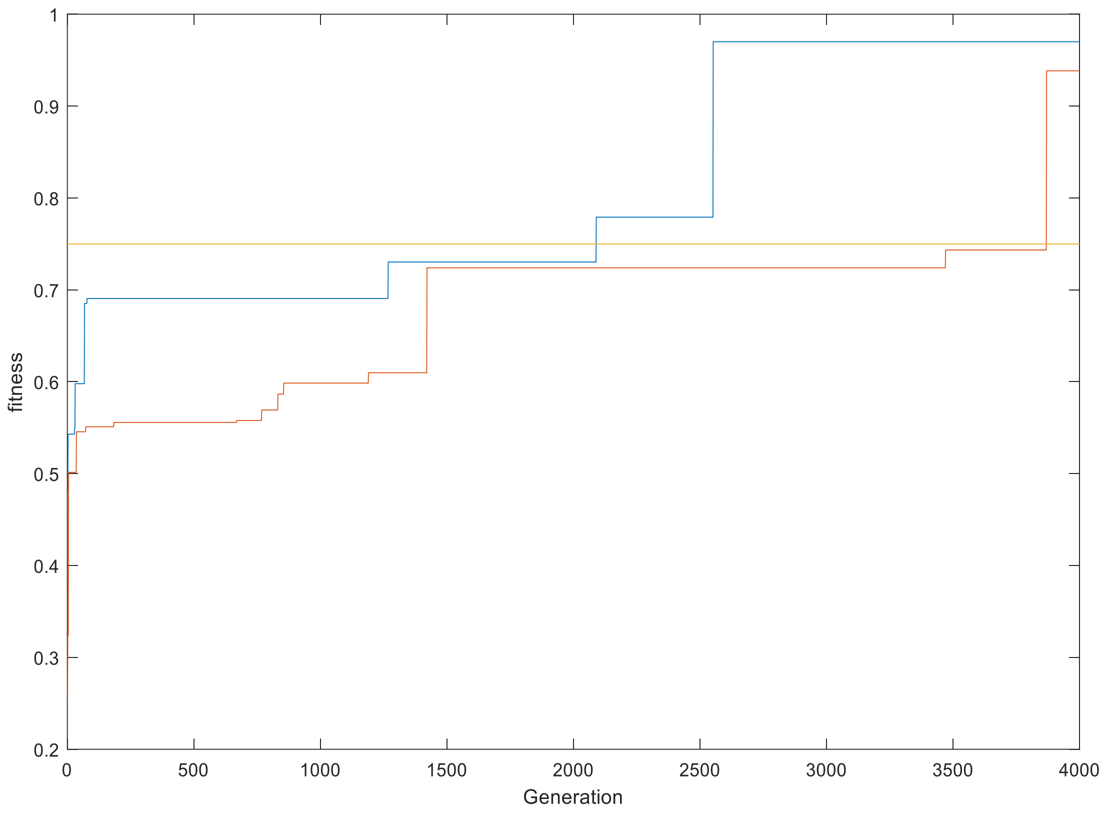


Figure 7.1.3. Time evolution plot, showing fitness over 4000 generations: the blue line is an 8-bit network; the orange a 32-bit network; and the yellow line is the 0.75 fitness threshold.

The Figure 7.1.3 shows the change in fitness as evolution proceedes, and it presents a simple conclusion: given sufficent generations, the higher bit width networks will reach fitness values eqivalent to those that the smaller width networks. This also suggests a possible cause, namely that the increase in data width comes with an increase in the search space of possible network configurations. To illustrate this conclusions, a system with 10 inputs, 20 genes, 3 epigenetic elements, and 2 outputs has a total of 799 parameters that the evolutionary process can optimise. In an 8-bit network, each of those parameters can have 256 possible values, which means a search space of 204,544 possible epigenetic networks. By comparision, in a 32-bit network, each parameter can have 4.2×10^9 possible values, leading to a serach space of over 3.4×10^{12} epigenetic networks. With a search space over 16 million times larger to search, it is easy to see why it takes longer to identify good solutions, although it is also a potent reminder of why computing architecures like epigenetic networks require optimisation methods such as evolutionary algorithms.

The result of these experiments give a clear picture: reducing the width of the integer networks even to as low as 5-bits can still produce viable controllers. Given its median fitness of over 0.90, a data width of 8-bits appears to be a suitable choice for future experiments. This is a useful as this equates to 1 Byte, which in turn is 1/8th the size of 8 Bytes a 64 floating point value occupies.

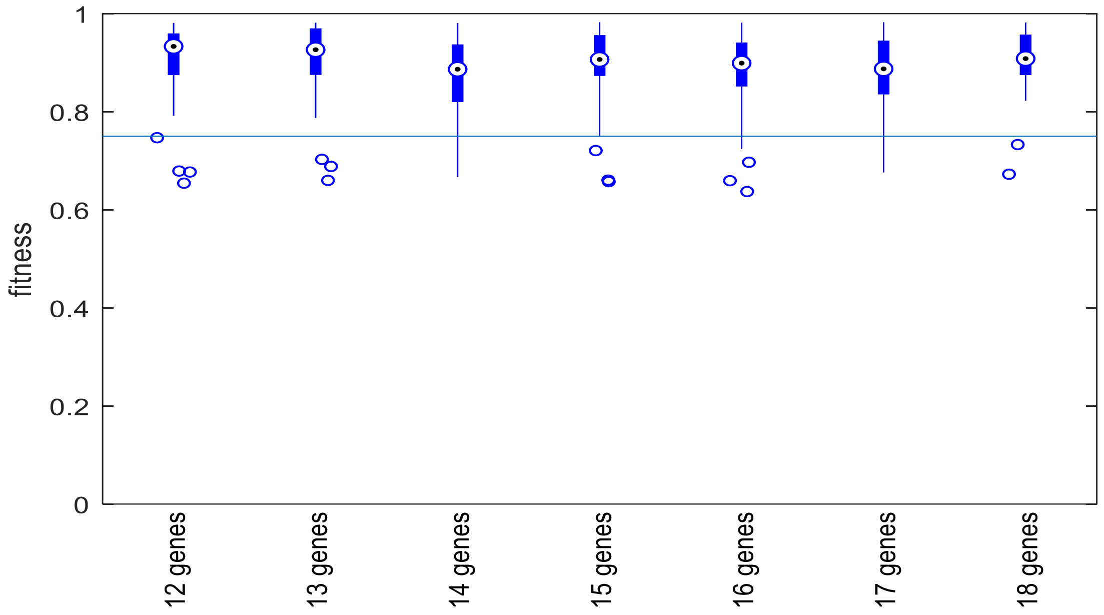


Figure 7.1.4. Fitness of integer epigenetic networks with an 8-bit data width, 3 molecules and between 12 and 18 genes. The blue line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position.

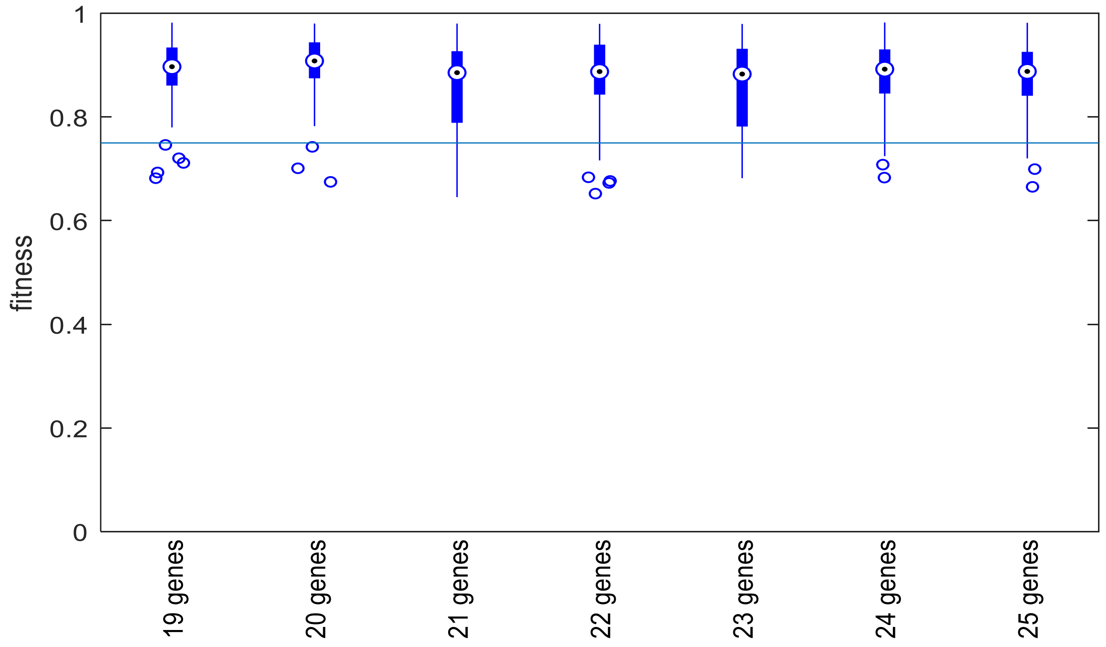


Figure 7.1.5. Fitness of integer epigenetic networks with an 8-bit data width, 3 molecules and between 19 and 25 genes.

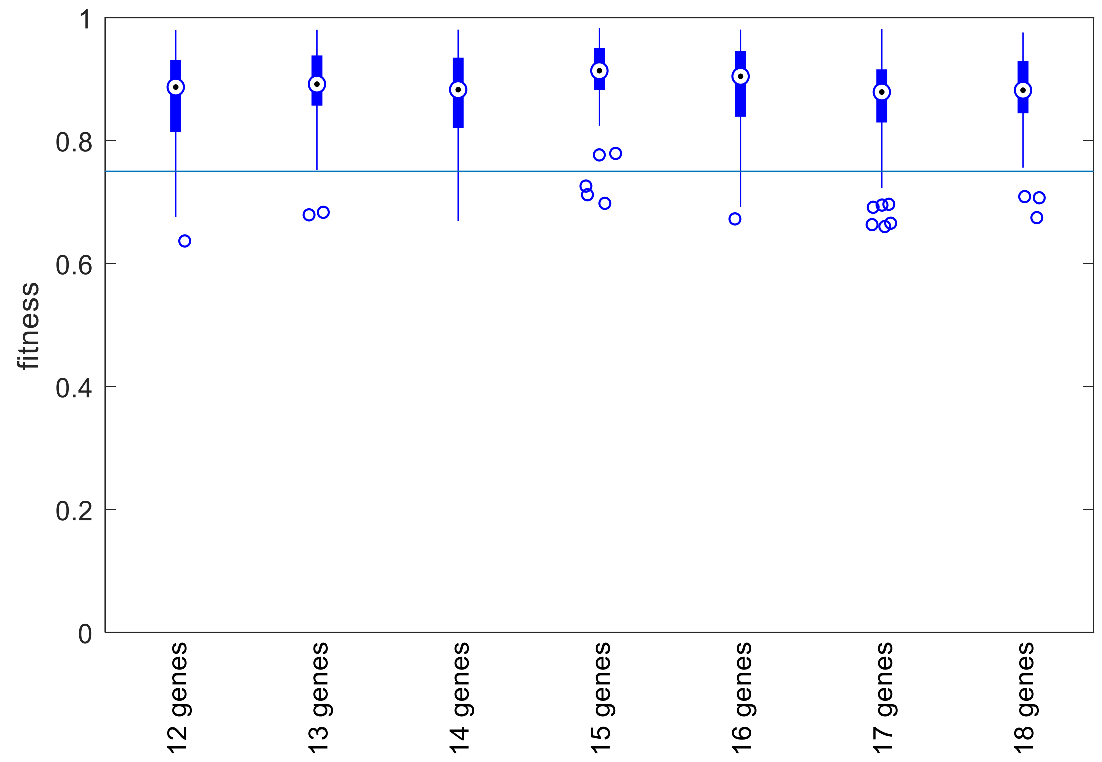


Figure 7.1.6. Fitness of integer epigenetic networks with an 8-bit data width, 4 molecules and between 12 and 18 genes.

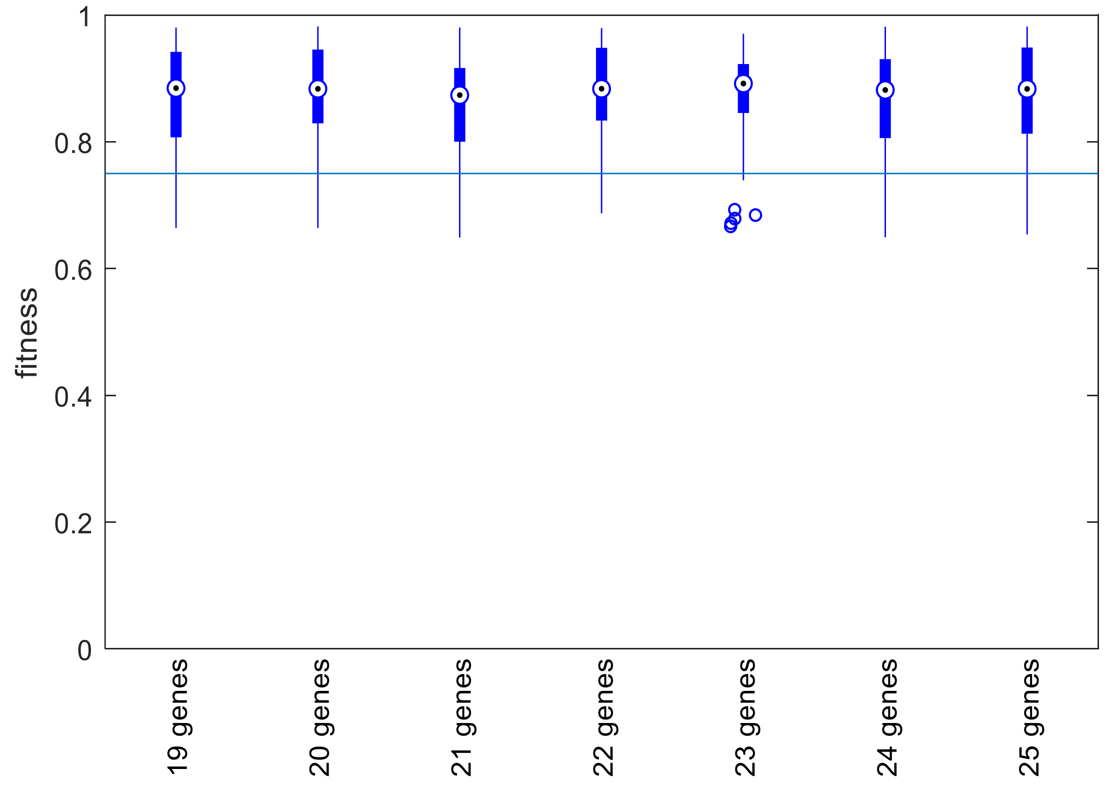


Figure 7.1.7. Fitness of integer epigenetic networks with an 8-bit data width, 4 molecules and between 19 and 25 genes.

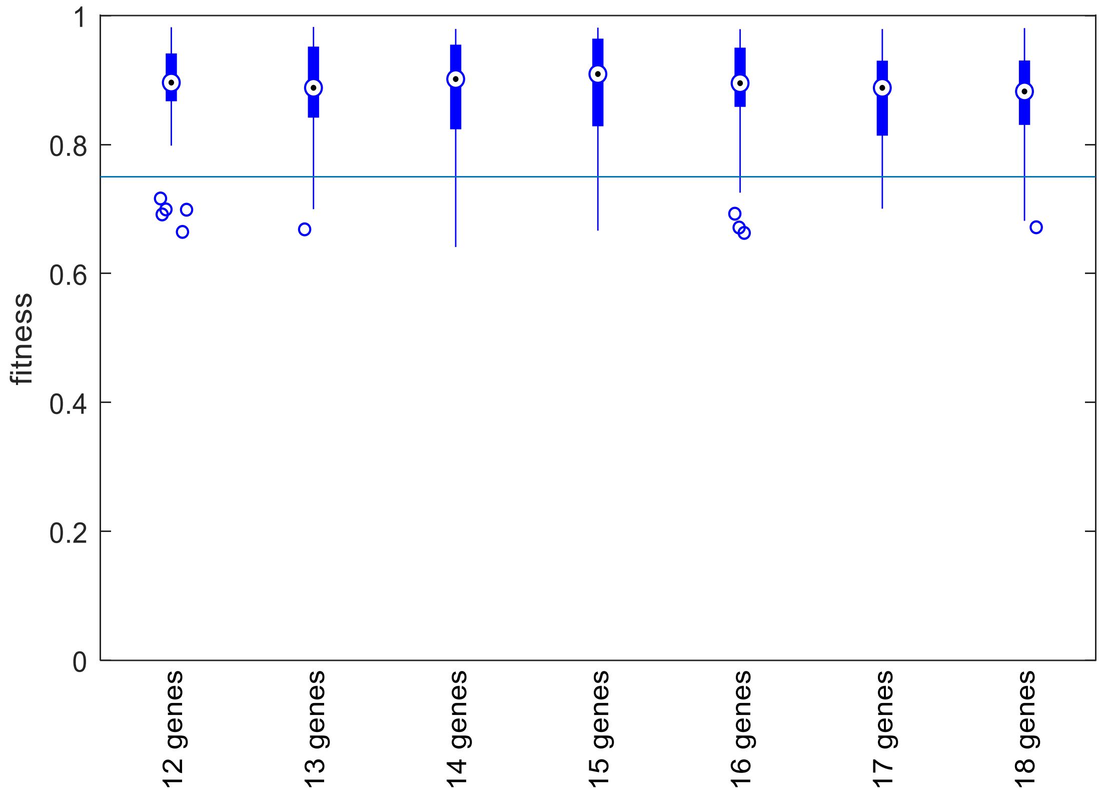


Figure 7.1.8. Fitness of integer epigenetic networks with an 8-bit data width, 5 molecules and between 12 and 18 genes.

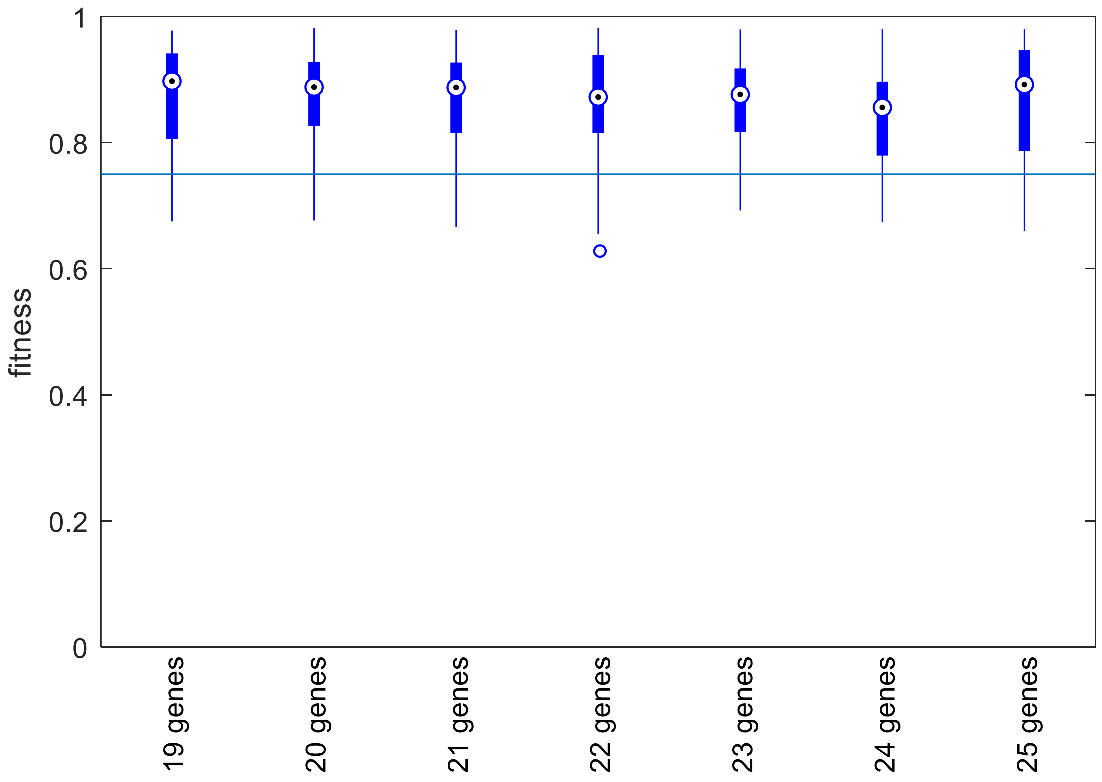


Figure 7.1.9. Fitness of integer epigenetic networks with an 8-bit data width, 5 molecules and between 19 and 25 genes.

The results of the network size evolutions are shown in figures 7.1.4 through 7.1.9. All gene and molecule combinations have median fitnesses above the 0.75 threshold, and in a few cases all non-outliers achieve at least this value also.

Table 7.1.4. Network size, number of genes/molecules and median fitness of networks where all non-outliers are above 0.75.

Network Size	**Number of Genes**	**Number of Molecules**	**Median Fitness**
15	12	3	0.93
16	13	3	0.92
18	13	5	0.88
19	15	4	0.91
21	18	3	0.90
22	18	4	0.88
22	19	3	0.89
23	20	3	0.90

Considering the results in the Table 7.1.4, a clear choice presents itself: 12 genes and 3 molecules is not only the smallest network, with a total size of 15, but it also has the highest median fitness, 0.93.

With all parameters of the network determined, evolution of the completed integer networks and hardware configuration generation can proceed.

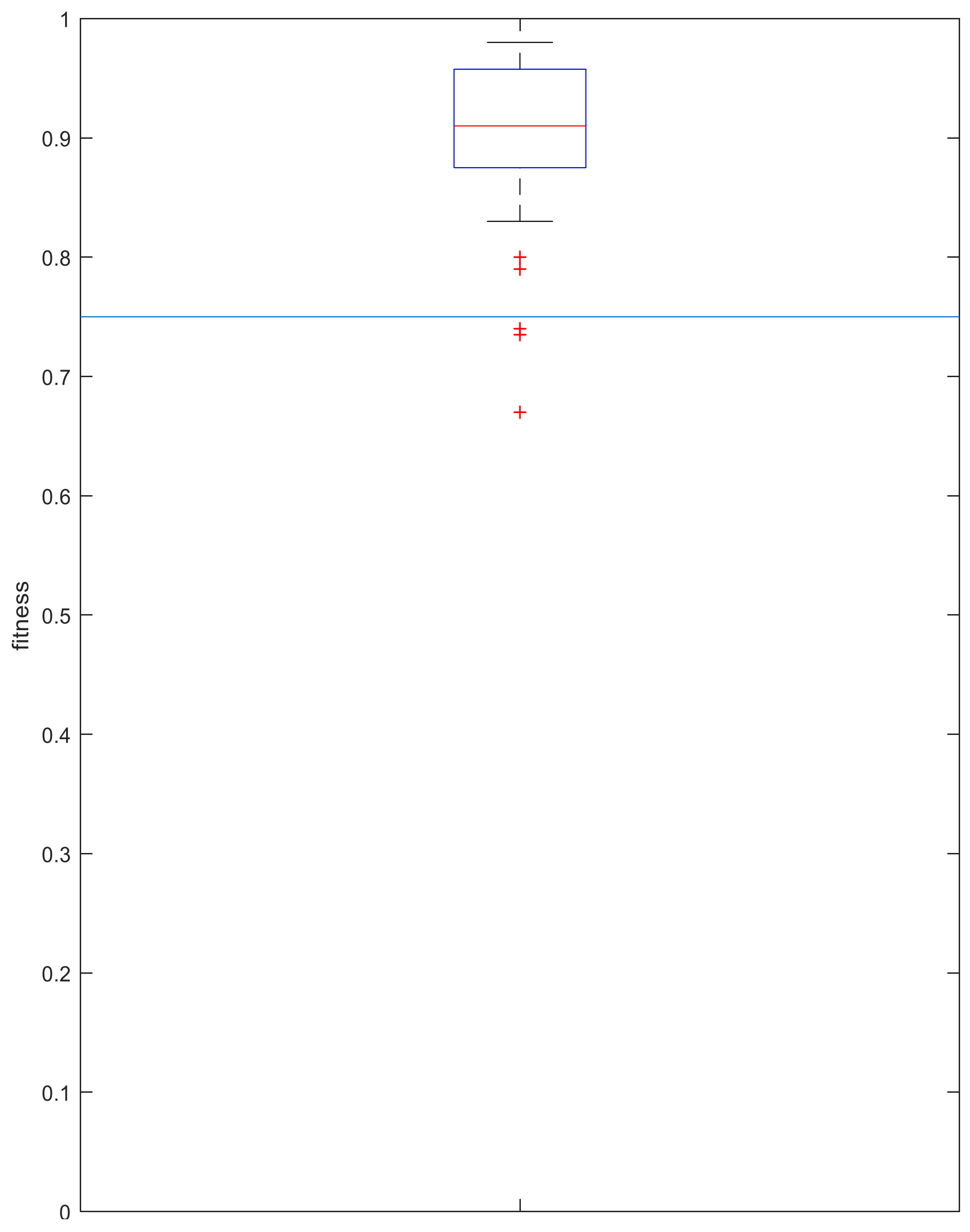


Figure 7.1.10. Fitness of integer epigenetic networks with an 8-bit data width, 12 genes and 3 molecules. The blue line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position.

The results from the evolution of the complete integer networks are shown in Figure 7.1.10. With a median fitness of 0.91 and only a few outliers below the 0.75 threshold, the software integer network is clearly able to complete the inverted pendulum task. The final task is now to transfer these integer networks from software on to the hardware platform detailed in Section 5. To do this, the software networks have the parameters of their various input/output nodes, genes and molecules extracted to produce configuration files for the hardware, as detailed in section 5.5.

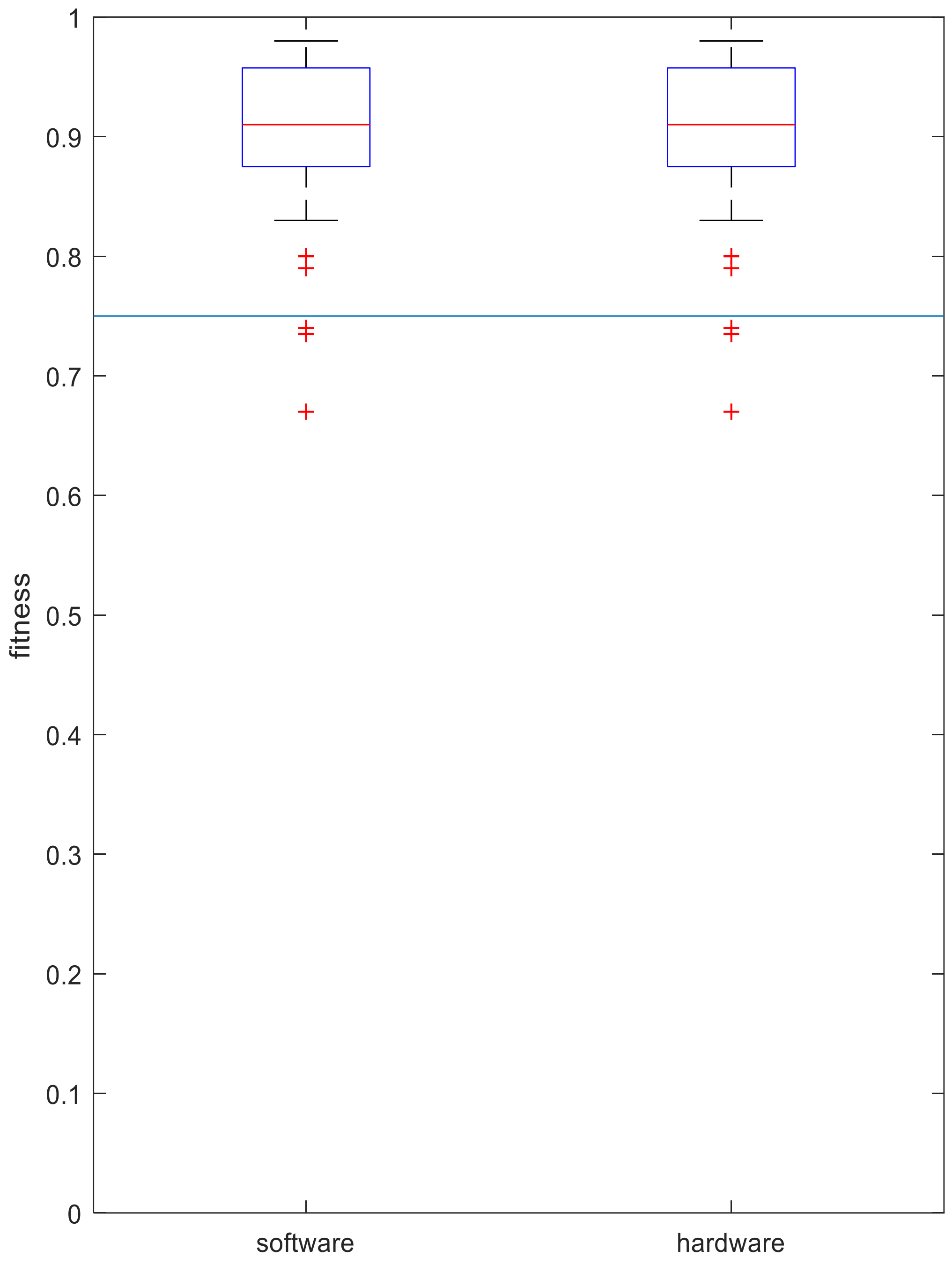


Figure 7.1.11. Fitness of integer epigenetic networks in software and hardware. The blue line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position.

The software to hardware translation results, shown in Figure 7.1.11, show excellent comparison, giving confidence that the translation of various parameters has been achieved with appropriate accuracy. When translated on to the hardware, all networks were able to achieve the same fitness as their software counterparts

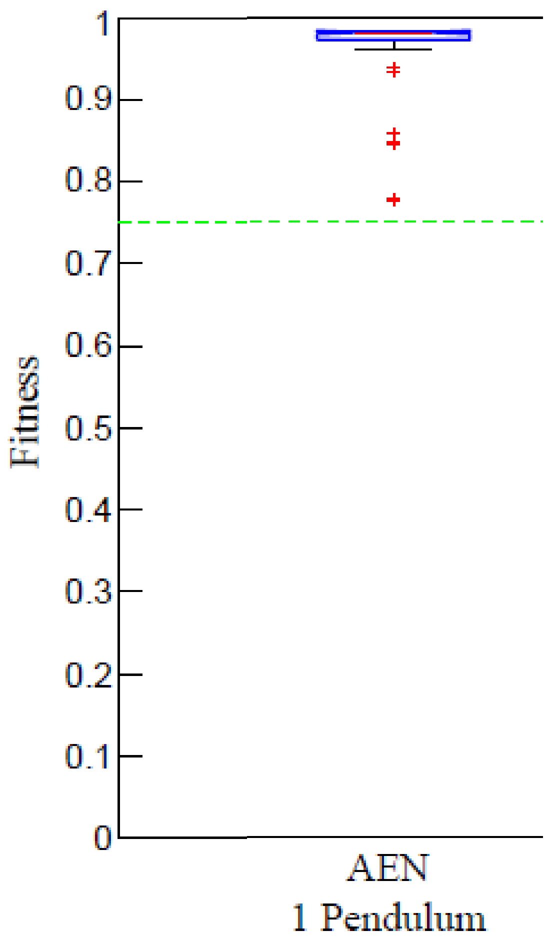


Figure 7.1.12. The results of Turner's original evolutions to produce a floating point network able to balance the inverted pendulum, taken from [6]. The green line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position.

The final comparison is now made between Turner's original 64-bit, floating point software epigenetic networks and this work's 8-bit, integer hardware networks. Firstly, both versions are able to produce a number of individuals able to cross the 0.75 fitness threshold. Although three of the hardware networks fall below this limit. The maximum, median and minimum fitness of both network types, as well as the 25th and 75th percentiles are detailed in Table 7.1.5 for ease of comparison.

Table 7.1.5. Maximum, Median and Minimum fitness, as well as 25th and 75th percentiles of Turner's 64-bit, floating point software networks compared with those for this work's 8-bit, integer hardware networks. Note that outlier values are not included.

Property	**64-bit, Floating Point Software Networks**	**8-bit, Integer Hardware Networks**
Minimum Fitness	0.95	0.83
25th Percentile Fitness	0.96	0.89
Median Fitness	0.97	0.91
75th Percentile Fitness	0.98	0.95
Interquartile Range	0.02	0.06
Maximum Fitness	0.98	0.98

Comparing the two sets of results using the A measure [83] returns a value of 0.2, which indicates a signficant divergence in the two populations. In this case, it corrisponds to a statistical drop in fitness between the origninal 64-bit floating point software networks and the new 8-bit integer hardware networks. However, they are able to match them in maxium fitness, 0.98. Given this fact it is reasonable to assert that, although statistically speaking there is a performance reduction, integer hardware epigenetic networks are able to acheve a similar peformance as a 64-bit, floating point software version.

7.1.3 Hardware Analysis

The reduced hardware network architecture is able to perform comparably to its predecessor, but an important argument for making the transition is a reduction in the use of resources. Taking each step of the transition in turn, starting with the reduction in the data width of the network, comparisons can be made to see if the resource usage reduction has been successful.

Table 7.1.6. Resources available on a Xilinx ZynQ-7000 XC7Z020 FPGA, as utilised in this work [74].

Resource	**Description**	**Number Available**
LUTs	Lookup table, for implementing logical/mathematical operations or storage. Configurable as 6 or 5 input.	53200
Registers	Short term storage.	106400
DSP Slices	Digital Signal Processing elements, for implementing mathematical operations, specifically available: multiplication and accumulation.	220
F7 Multiplexers	Selecting between multiple signals.	26600
F8 Multiplexers	Selecting between multiple signals.	13300
BRAM Tiles	Block Random Access Memory, longer term storage. 36 Kbits up to 36 bits in width.	140

Table 7.1.7. Comparison of FPGA resource utilisation for 32-bit and 8-bit integer networks. Both networks have: 10 inputs; 12 genes; 3 molecules; and 2 outputs. Table includes percentage utilisation relative to the values in table 7.1.6.

Resource	**32-bit network**		**8-bit network**	
	Number Used	**Percentage**	**Number Used**	**Percentage**
LUTs	34501	64.48%	18073	33.97%
Registers	6160	5.79%	4418	4.15%
DSP Slices	262	119.09%	30	13.64%
F7 Multiplexers	1648	6.20%	804	3.02%
F8 Multiplexers	464	3.49%	216	1.62%
BRAM	480	342.86%	7	5.00%

Table 7.1.8. Comparison of FPGA resource utilisation for 32-bit and 8-bit integer network components: input units; wrapped genes; wrapped molecules; and output units. See chapter 5 for descriptions. Table includes percentage utilisation relative to the values in table 7.1.6.

Network Element	**Resource**	**32-bit network**		**8-bit network**	
		Number Used	**Percentage**	**Number Used**	**Percentage**
Input Unit	LUTs	607	1.14%	272	0.51%
	Registers	40	0.04%	18	0.02%
	DSP Slices	0	0.00%	0	0.00%
	F7 Multiplexers	0	0.00%	0	0.00%
	F8 Multiplexers	0	0.00%	0	0.00%
	BRAM	0	0.00%	0	0.00%
Wrapped Gene	LUTs	2643	4.97%	1105	2.08%
	Registers	379	0.36%	278	0.26%
	DSP Slices	16	7.27%	2	0.91%
	F7 Multiplexers	368	1.38%	62	0.23%
	F8 Multiplexers	116	0.87%	18	0.14%
	BRAM	32	22.86%	0.5	0.36%
Wrapped Mole.	LUTs	2485	4.67%	965	1.81%
	Registers	386	0.36%	284	0.27%
	DSP Slices	16	7.27%	2	0.91%
	F7 Multiplexers	368	1.38%	20	0.08%
	F8 Multiplexers	116	0.87%	0	0.00%
	BRAM	32	22.86%	0.5	0.36%
Output Unit	LUTs	377	0.71%	145	0.27%
	Registers	48	0.05%	24	0.02%
	DSP Slices	0	0.00%	0	0.00%
	F7 Multiplexers	0	0.00%	0	0.00%
	F8 Multiplexers	0	0.00%	0	0.00%
	BRAM	0	0.00%	0	0.00%

Table 7.1.7 shows the resource utilization of two complete networks, one 8-bit and one 32-bit; each with the same number of genes and molecules; while table 7.1.8 shows the utilization of the various components of those networks. The two largest differences are in the DSP and BRAM utilization; although this is not surprising, as the DSP resources are partly responsible for mathematical operations, along with the LUTs; while the BRAM resources hold the sigmoid lookup tables. In both these cases, increasing the data width brings increased demands. In fact, looking at the utilization percentages shows that the 32-bit network couldn't actually be implemented onto the hardware utilised in this project. Outside these two stark instances, the other resources are also used to a greater extent by the 32-bit network, with the LUT and Multiplexer utilisations almost doubling. This clearly illustrates the advantage of reducing the bit width of the network when it comes to reducing silicon resources. And while there isn't a hardware floating point network to compare with as well, the fact that one would need an even greater data width, as well as specialised mathematical elements, shows the benefits of making the switch.

Another axis of comparison, and one where a floating-point network can be looked at, is execution time. The hardware network is designed to allow for parallel execution, whereas the both the integer and floating-point software versions are forced to execute each network element in turn.

Table 7.1.9. Average execution times of: an 8-bit integer hardware network; an 8-bit integer software network; a 64-bit integer software network; and a 64-bit floating point software network. Times encapsulate: molecule execution; gene activation state updates; gene execution; and output execution. Input execution is not included as their execution timings are primarily dependent on factors external to the network.

Network Type	**8-bit integer hardware network**	**8-bit integer software network**	**64-bit integer software network**	**64-bit floating-point software network**
Execution Time	3 µS	26 µS	29 µS	542 µS

Table 7.1.9 shows these averaged execution time mesurments, measured using the system clock for software and a dedicated clock unit for hardware. The mesurments for the software networks were from executions performed on the standard desktop PC described in the introduction of Chapter 6. The hardware network is an order of magnitude faster than its software counterparts, which is almost certanly down to the parallelisation. The two integer software networks have comparable execution times, indicating that the bit width has little impact on execution times. However, the floating point network is dramatically slower than even the 64-bit integer network, something that demonstrates the additional computational complexity brought on by floating point maths. Taking this together with the hardware utilisation numbers, and the performance results in section 7.1.2, it is resonable to make the following statement: **By transitioning to a dedicated, integer-based hardware architecture, it is possible to significantly reduce the resource requirements of an epigenetic network without sacrificing performance.**

With confermation that the transition has been succesful, the next stage is the investigation of the response of epigenetic networks to unevolved for external factors, as described in section 6.4. Section 7.2 leverages the existing inverted pendulum model to look at a simple instance of this: Impulse Injection.

7.2 Unexpected Impulse Injection Response

Recall from Section 6.4, that a system in a dynamic environment is considered different from a dynamic system, in that all the properties that might affect the behaviour of a dynamic system are encapsulated with its model; where as a system in a dynamic environment is one subject to unanticipated external influences, such as an object impacting the pendulum while it is in the upright position. Such an occurrence would require the pendulum's controller to alter its behaviour to compensate for the sudden change, and as such behavioural changes are an ostensible strength of epigenetic networks, this should be a worthwhile line of investigation.

7.2.1 Experimental Design and Parameters

The first stage of these experiments will be to generate two suitable populations of networks, one which includes epigenetic elements and the other which does not. To ensure valid comparisons, it is important that the total number of network elements are the same across both populations.

Table 7.2.1. Parameters for the evolution of epigenetic networks.

Parameter	Value
Population Size	64
Minimum Target Fitness	0.90
Number of Repeats	48
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 7.2.2. Parameters for the evolution of non-epigenetic networks.

Parameter	**Value**
Population Size	64
Minimum Target Fitness	0.90
Number of Repeats	48
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	8
Number of Genes	16

The parameters for these networks are listed in the tables 7.2.1 and 7.2.2, most of the evolutionary algorithm and network parameters remain the same as those detailed in Section 7.1.1, although there are two changes. The most obvious change is that instead of a number of generations, there is a minimum target fitness specified. This is because the goal of this series of experiments is to see how a network reacts when confronted with stimulus that is outside the parameters it was designed for. In order to make clear any significant differences, it is important that networks are as capable as reasonably possible. Thus, instead of looking to see what networks can be produced within certain time constraints, the evolutionary algorithm will instead continue to optimise the networks until they reach a fitness value of at least 0.9. Additionally, the number of epigenetic molecules has been increased to 4. This is because it results in the total number of network elements equalling 16, or 2^4, meaning that the hardware version of these networks will be able to address all the network elements with only 4-bits.

Once both populations have been produced, they will be used to control the inverted pendulum model for an extended period of time. During these extended runs, once the pendulum is stable in the upright position, external dynamics will be introduced. This will take the form of a simple impulse injection, used to represent something striking the top of the pendulum, causing it to change angular velocity. Considering Section 6.2 and equations 6.2.1 through 6.2.4, which describe the properties of the pendulum model at each time step, it can be noted that a term for angular velocity exists (ω). Thus, the impulse injection can be easily modelled simply by adding to the angular velocity term at the appropriate time step. The initial impulse will be limited to 1 rad s^{-1}, injected at time step 1500, as previous results indicate that a network of good fitness should have balanced the pendulum by this point.

Comparison between the epigenetic and non-epigenetic networks will take two forms:

(i) The inverted pendulum fitness function is still a valid metric, as it measures time spent in the upper equilibrium position. Therefore, the difference between the evolved fitness of a network, and the fitness measured during the impulse injection experiment will directly equate to the time the pendulum spent off vertical due to the impulse. Thus, a smaller fitness difference means a network that recovered from the impulse faster.

(ii) Directly measuring the pendulum angle and its angular velocity, then plotting over time, allows for a more qualitative performance metric.

7.2.2 Results

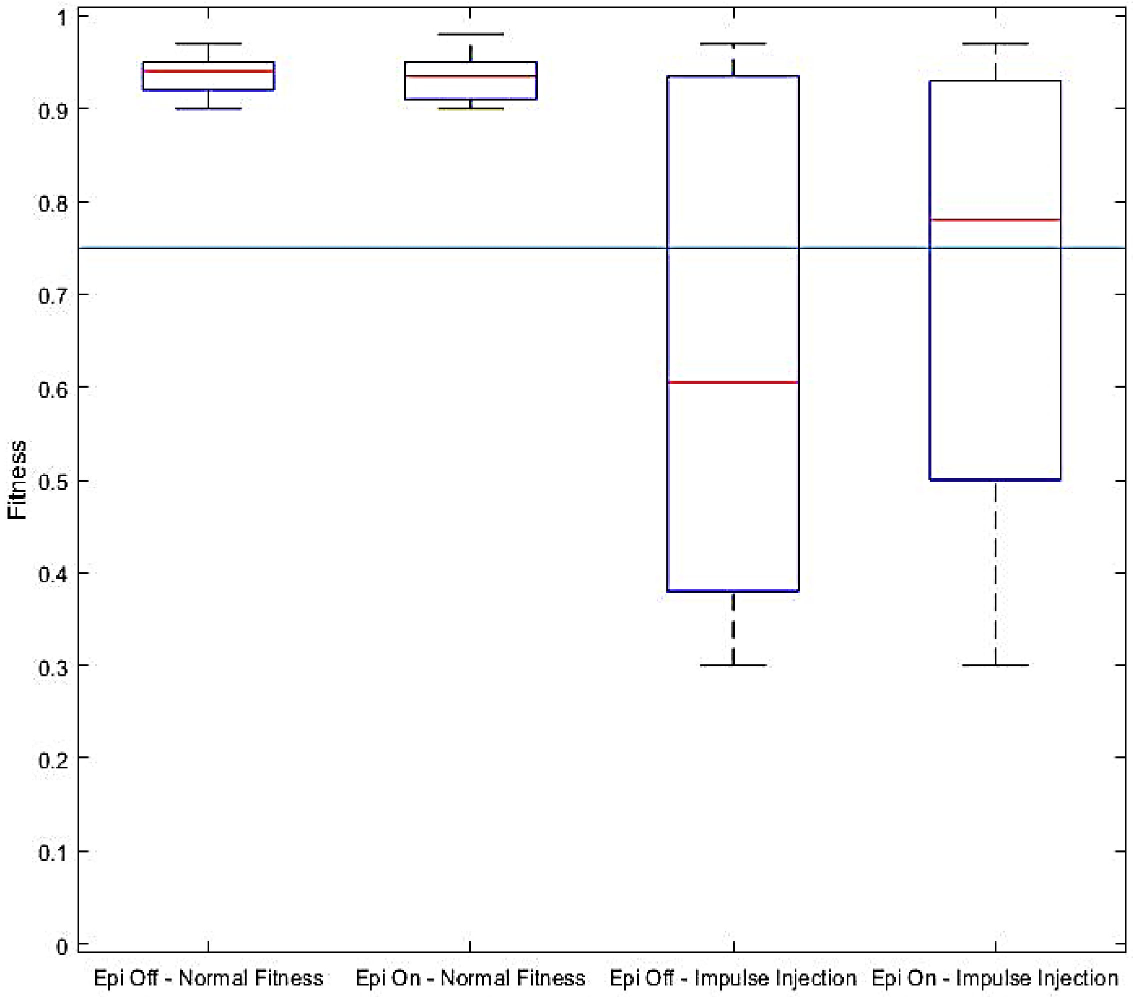


Figure 7.2.1. The results of the evolution of epigenetic and non-epigenetic network, as well as their fitness values during impulse injection. The blue line at 0.75 denotes the fitness at which the networks are able to maintain the pendulum in the upright equilibrium position.

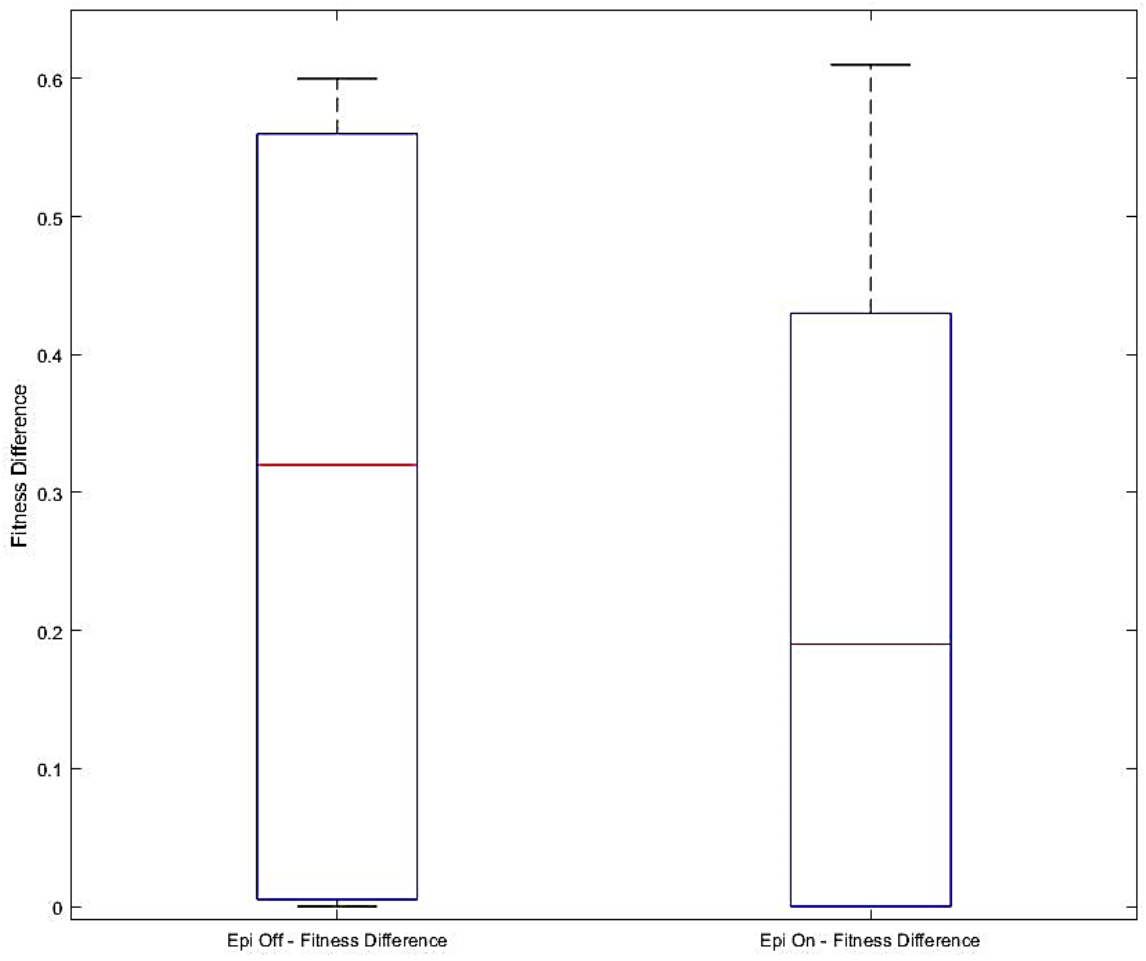


Figure 7.2.2. Difference between the evolved fitness and impulse injection fitness for both epigenetic and non-epigenetic networks. Note that in this case, a lower value is better.

Considering figure 7.2.1 it is obvious that both the epigenetic and non-epigenetic network populations appear to include individuals that are able to handle the impulse injection without any measurable reduction in fitness. However, from figures 7.2.1 and 7.2.2 additional observations can be made: First, the median post-impulse fitness of the epigenetic population is 0.78, above the 0.75 threshold, while the non-epigenetic networks have a median of 0.60. Combined with the epigenetic network's superior fitness difference median, 0.19 against 0.32, it appears that the epigenetic networks do demonstrate a resilience to the impulse injection in this instance.

Table 7.2.3. Maximum, Median and Minimum fitness, as well as 25th and 75th Percentiles and mean of the epigenetic networks' basic fitness, fitness with the impulse injection and fitness difference. Fitness difference is reversed, as in its case, smaller values are better. Note that outlier values are only included in the calculation of mean fitness.

Property	**Basic Fitness**	**Impulse Fitness**	**Fitness Difference**
Minimum	0.90	0.30	0.61
25th Percentile	0.91	0.50	0.43
Median	0.94	0.78	0.19
75th Percentile	0.95	0.93	0.00
Interquartile Range	0.04	0.43	0.43
Maximum	0.98	0.97	0.00
Mean	0.93	0.72	0.21

Table 7.2.4. Maximum, Median and Minimum fitness, as well as 25th and 75th Percentiles and mean of the non-epigenetic networks' basic fitness, fitness with the impulse injection and fitness difference. Fitness difference is reversed, as in its case, smaller values are better. Note that outlier values are only included in the calculation of mean fitness.

Property	**Basic Fitness**	**Impulse Fitness**	**Fitness Difference**
Minimum	0.90	0.30	0.60
25th Percentile	0.92	0.38	0.56
Median	0.94	0.61	0.32
75th Percentile	0.95	0.93	0.01
Interquartile	0.03	0.45	0.55
Maximum Fitness	0.97	0.97	0.00
Mean	0.94	0.63	0.31

Tables 7.2.3 and 7.2.4 summarise these data points, as well as providing the means, which allows for confidence limits to be placed on the data, and thus the calculation of p-values.

- Impulse Fitness:
  - Confidence Limits: $0.09 \pm 2.16(0.05)$
  - p-value: $p < 0.05$
- Fitness Difference:
  - Confidence Limits: $0.1 \pm 2.16(0.05)$
  - p-value: $p < 0.025$

In both cases $p < 0.05$, which indicates that these results are significant, reinforcing the assertion that the epigenetic elements of the networks are positively contributing to their recovery from the impulse injection. To confirm this, an example epigenetic network was analysed in greater detail.

7.2.3 Network Analysis

As going over all the networks and looking at their angle and vecloity plots would be impractical, a few specific examples will be selected, these are chosen because they exemplify specific points of interest. These points are:

(i) An epigenetic network with a fitness difference of 0.0;

(ii) An epigenetic network with a post impulse fitness as close to the population median as possible.

(iii) And a non-epigenetic network with a post impulse fitness as close to the population median as possible.

In addtion, each of these networks will be investigated in greater detail, to see if correlations can be identified between the behaviour of the pendulum under their control; and their internal states.

The first network under analysis is the sample of a network with a a fitness difference of 0.0. In this case it has both a pre and post impulse injection fitness of 0.97.

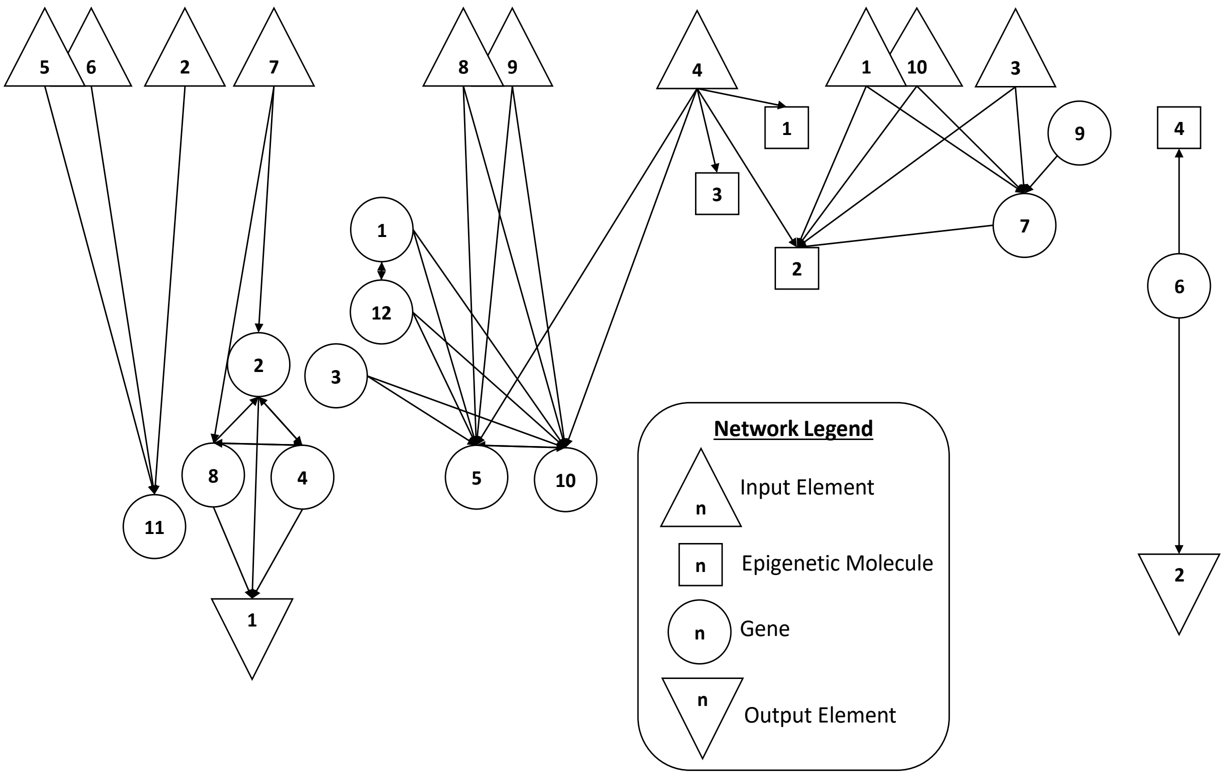


Figure 7.2.3. Structure of an epignetic network that suffered no fitness drop when exposed to impulse injection.

Figure 7.2.3 details the stucture of the network, which is derived from the parameter tables in appendix A, section 11.1. Starting at the begining, the network uses all of its inputs, with each one being connected to at least one gene or epigeneitc molecule. This is particually noteworthy as Turner's networks often used as few as two of their inputs, tipically the equivalnets of inputs 1 and 10 [6]. Moving on to the epigenetic molecules, all but molecule 4 take input 4 into consideration, not unsupprising given that it is one of the two sensors that would indicate that the pendulum was in the upright possition. With the genes, there are three densely intraconnected groups: genes 2, 4 and 8; genes 7 and 9; and genes 1, 3, 5, 10 and 12. Genes 6 and 11 are not connected to any other genes.

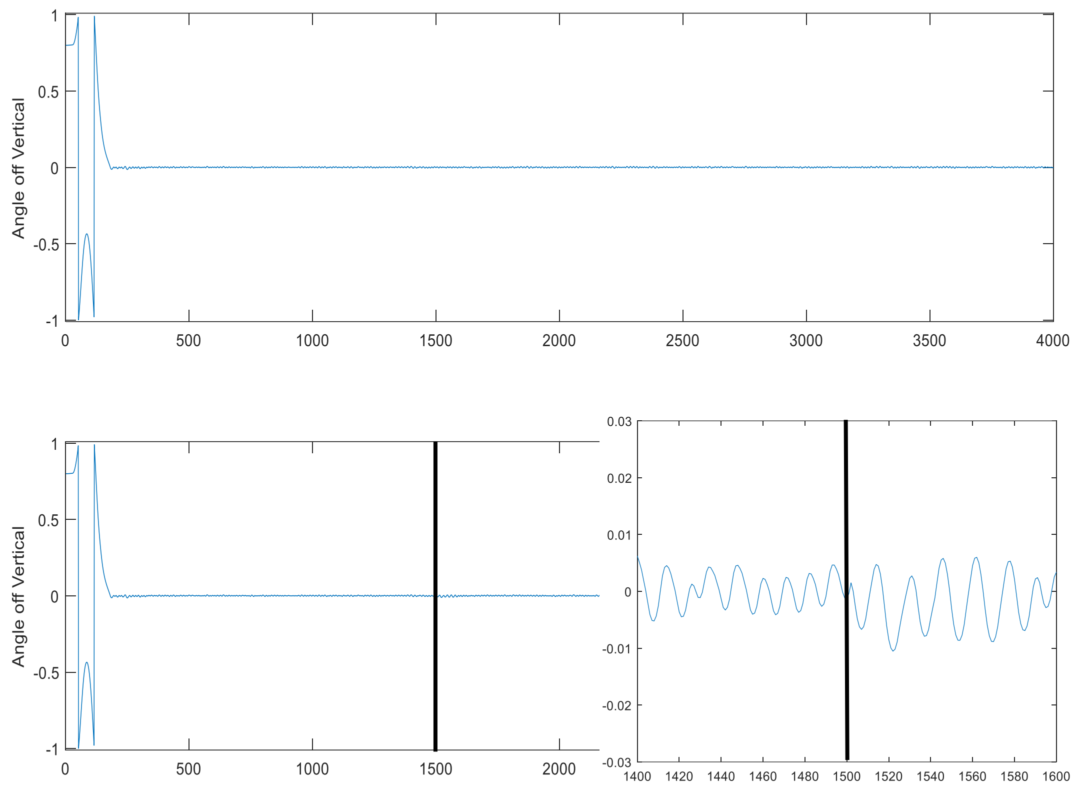


Figure 7.2.4. Plots of the angle off vertical of the pendulum when controlled by network 20401. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line. The inset shows a close-up of the time either side of the impulse injection.

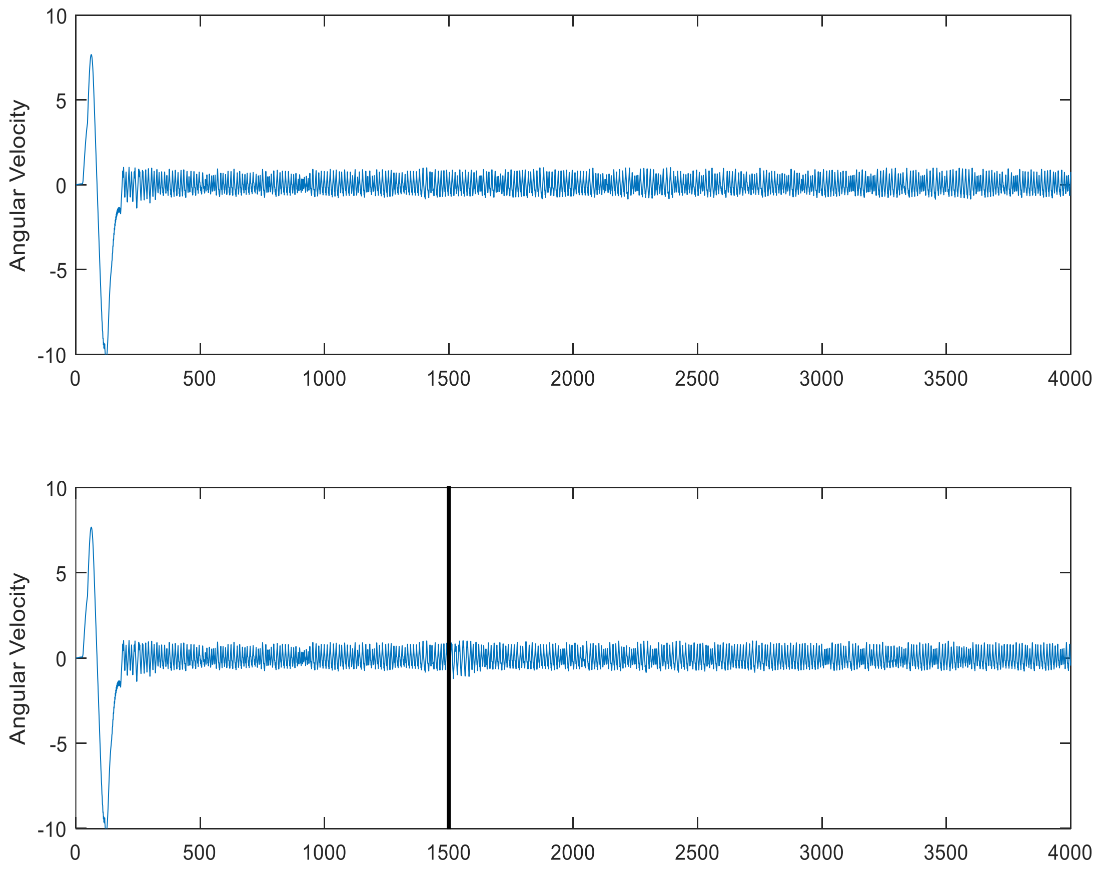


Figure 7.2.5. Plots of the angular velocity of the pendulum when controlled by network 20401. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line.

Looking at the angular velocity plots, figure 7.2.5, there is no discernible difference in the pendulum's behaviour between the two. This reiterates the fact that the fitness difference of 0.0, as any measurable deviation would have increased this value. The angle of vertical plots, figure 7.2.4, do show a small perturbation following the impulse injection, but it is quickly damped down.

The next series of plots detail the action of the epigenetic molecules and genes during execution of the network. Figure 7.2.6 shows which genes have their activity altered by the epigenetic mechanisms, while figure 7.2.7 shows which specific molecules are responsible. Plots 7.2.8 and 7.2.9 then compares the epigenetic behaviour of the network under normal conditions (7.2.8) and during the injection of the impulse (7.2.9).

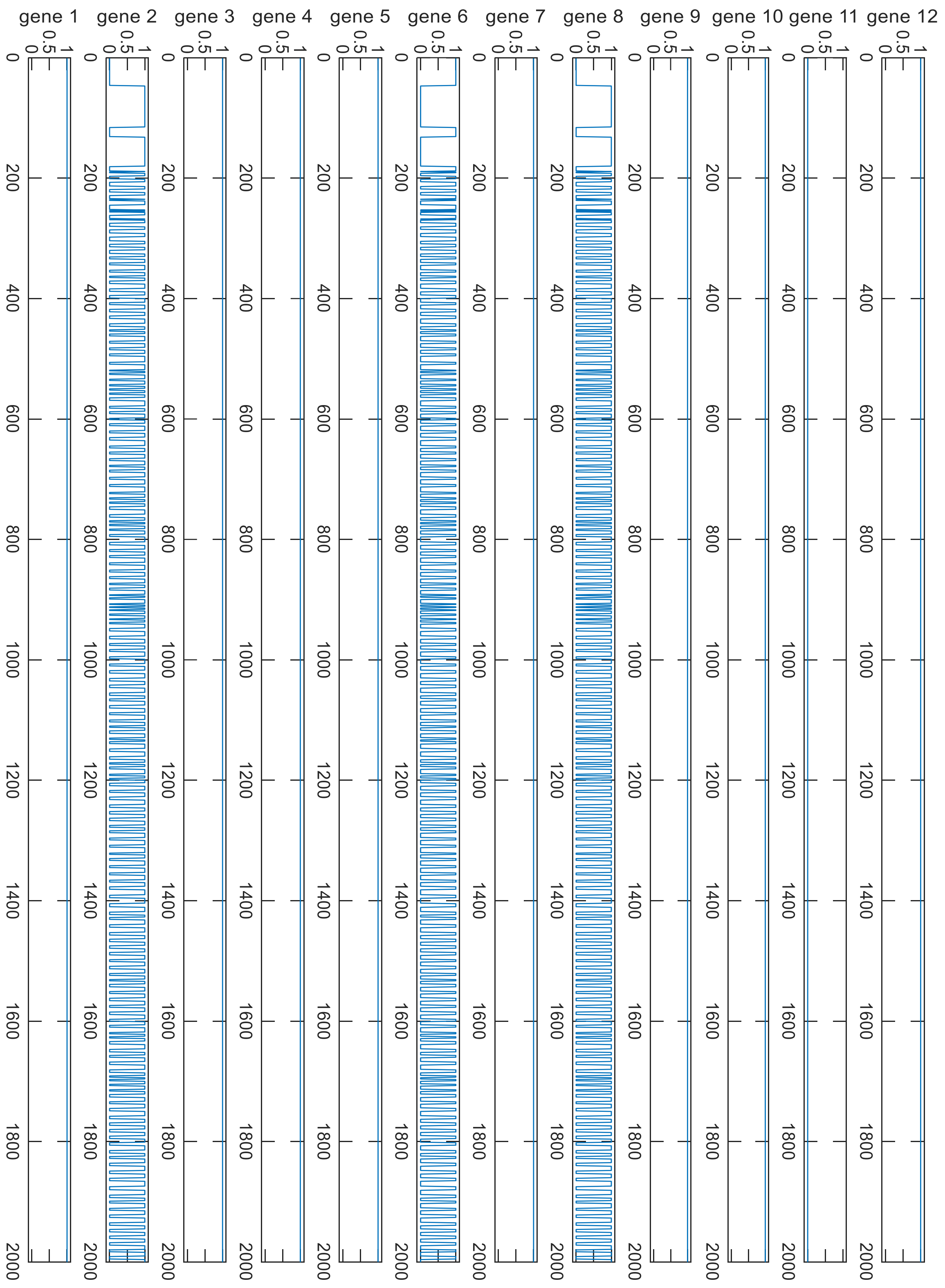


Figure 7.2.6. Activation state of genes within network 20401 between time steps 0 and 2000 (20 simulated seconds). A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

Figure 7.2.6 shows the epigenetic behaviour of the network's genes during the first 2000 time steps under normal conditions (no impulse at step 1500). From this, it can be seen that genes 2, 6 and 8 are the only ones that have their activation state affected by the epigenetic molecules, with all other genes being permanently active, except for gene 11 which is permanently inactive.

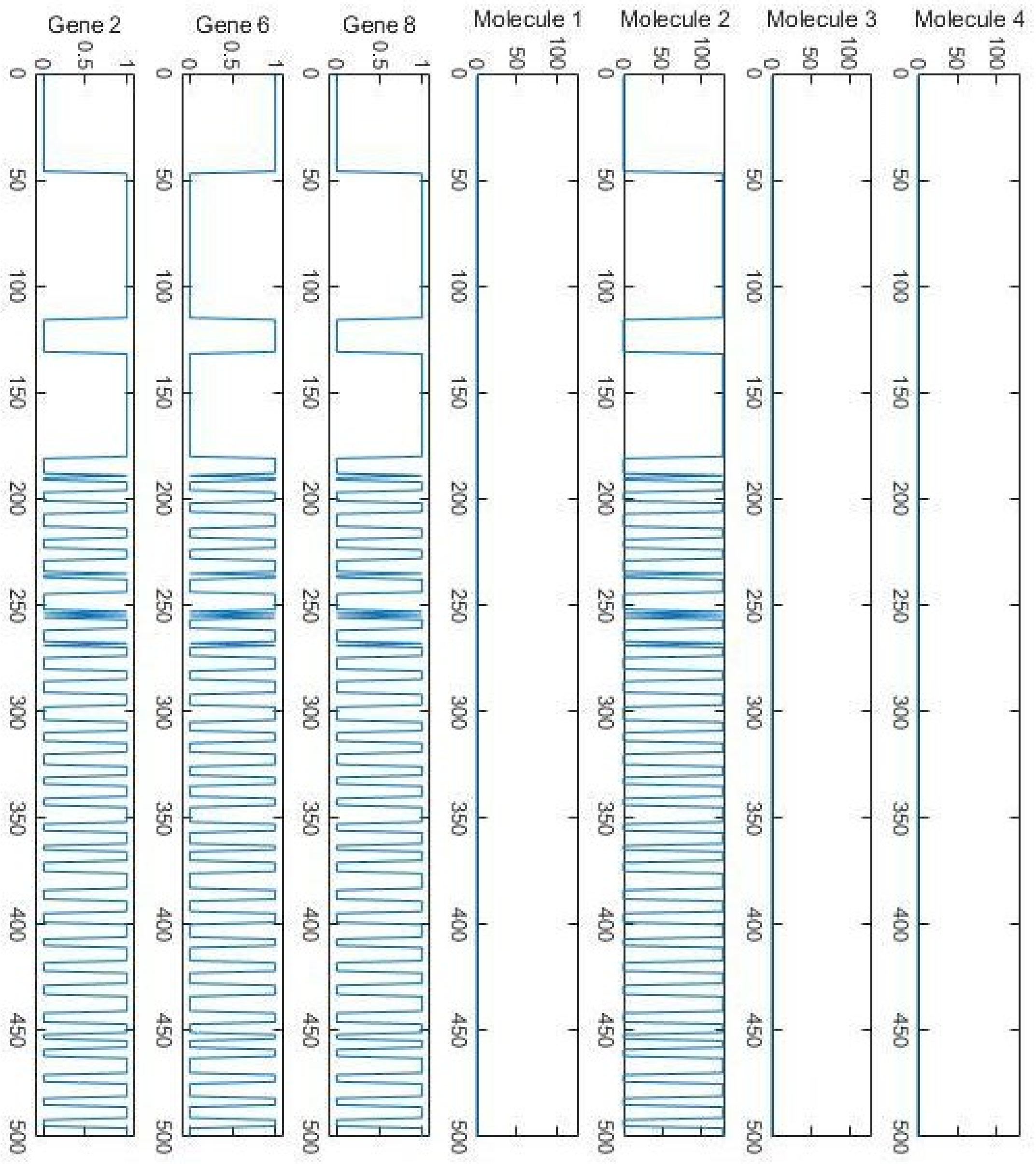


Figure 7.2.7. Activation state of the epigeneticaly active genes within network 20401, along with the outputs of the network's epigenetic molecules. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output. Molecule outputs are between 0 and 127, with the value relating to the location in the network space where it looks for genes to deactivate. See section 4.1.3 for more details.

Figure 7.2.7 shows that molecule 2 is the element responsible for affecting the activation state of all three of the epigenetically active genes, with genes 2 and 8 being inhibited when its output is low and gene 6 when its output is high; a fact that makes sense given the positions of these genes within the network space, see figure 7.2.3. Figure 7.2.3 also shows that molecule 2 takes inputs 1, 3, 4 and 10 and gene 7 as its inputs, gene 7 in turn is connected to the same inputs and gene 9. Gene 9 has no inputs. From this it can be ascertained which network elements attention needs to be payed to when investigating if any notable epigenetic change occurs when the network is confronted with the impulse injection.

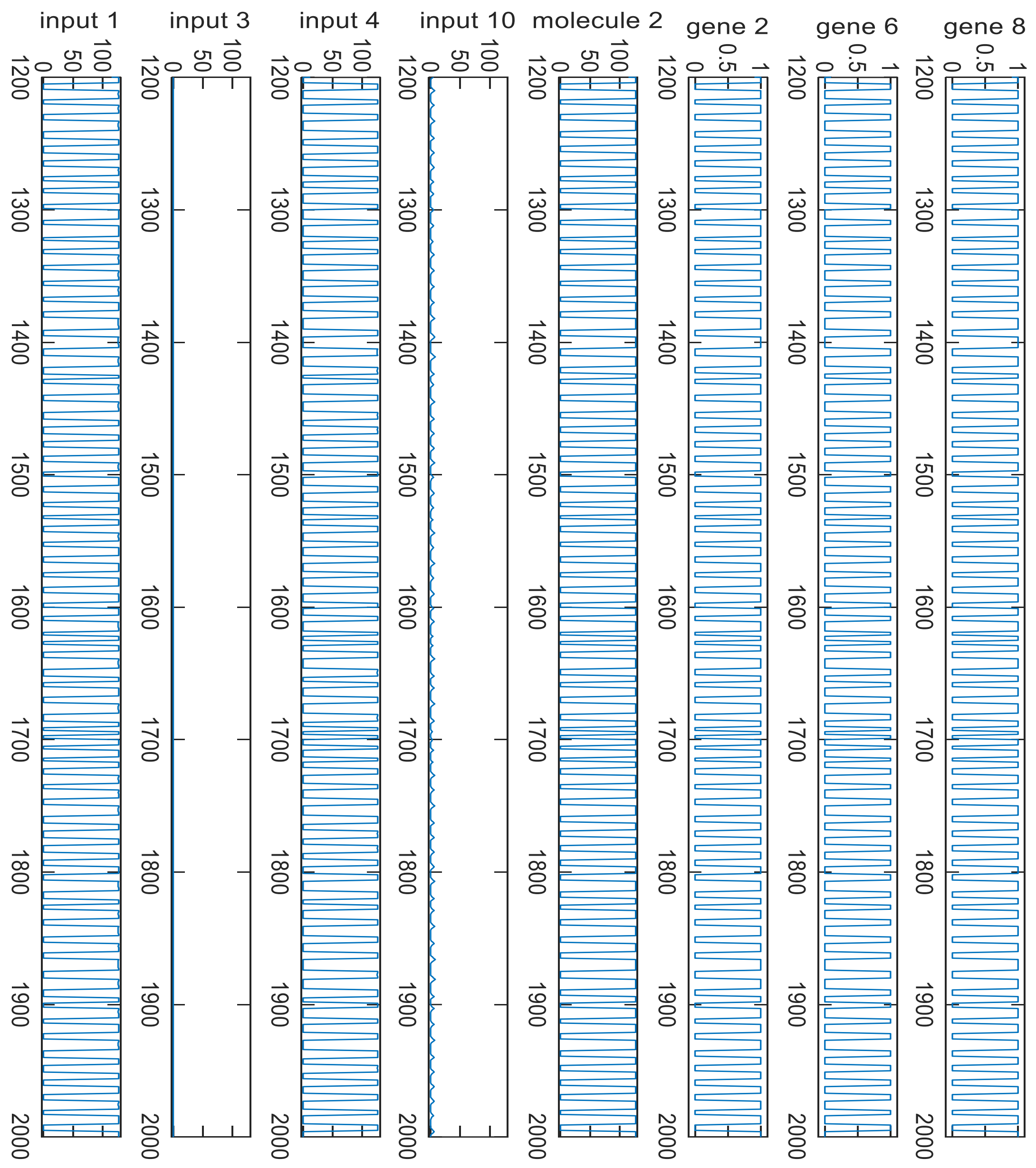


Figure 7.2.8. Values of inputs 1, 3, 4 and 10; Output of molecule 2; and activation states of genes 2, 6 and 8 of network 20401 between time steps 1200 and 2000 with no impulse injection.

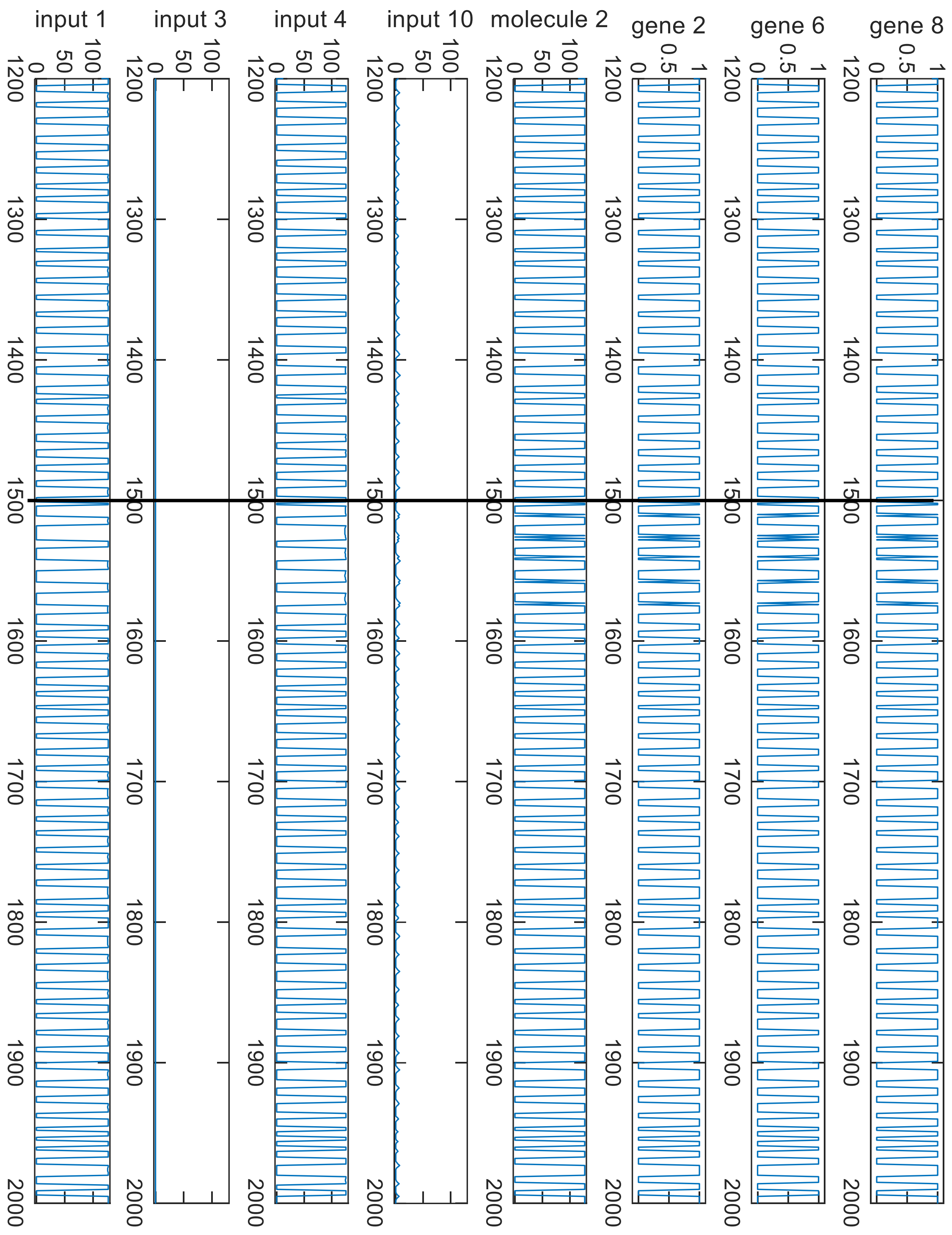


Figure 7.2.9. Values of inputs 1, 3, 4 and 10; Output of molecule 2; and activation states of genes 2, 6 and 8 of network 20401 between time steps 1200 and 2000 with an injection of a 1 rad s^{-1} impulse at time step 1500, marked by a black line.

Directly comparing figures 7.2.8 and 7.2.9, it is clear to see not only the difference in behaviour triggered by the impulse injection, but how exactly this behavioural change comes about. Referring back to table 6.2.1: inputs 1 and 4, the two which show a clear change following the impulse injection, are the two sensors that measure the pendulum angle either side of the vertical position. The disturbance of the pendulum from vertical alters the output of molecule 2, which in turn changes the activate states of the three genes under its purview. Finally, table figure 7.2.3 confirms that these genes are connected to the outputs of the network, with genes 2 and 8 connecting to output 1 and gene 6 connecting to output 2.

Considering the second network, which has a post impulse injection fitness of 0.79, the closest to the population median of 0.78.

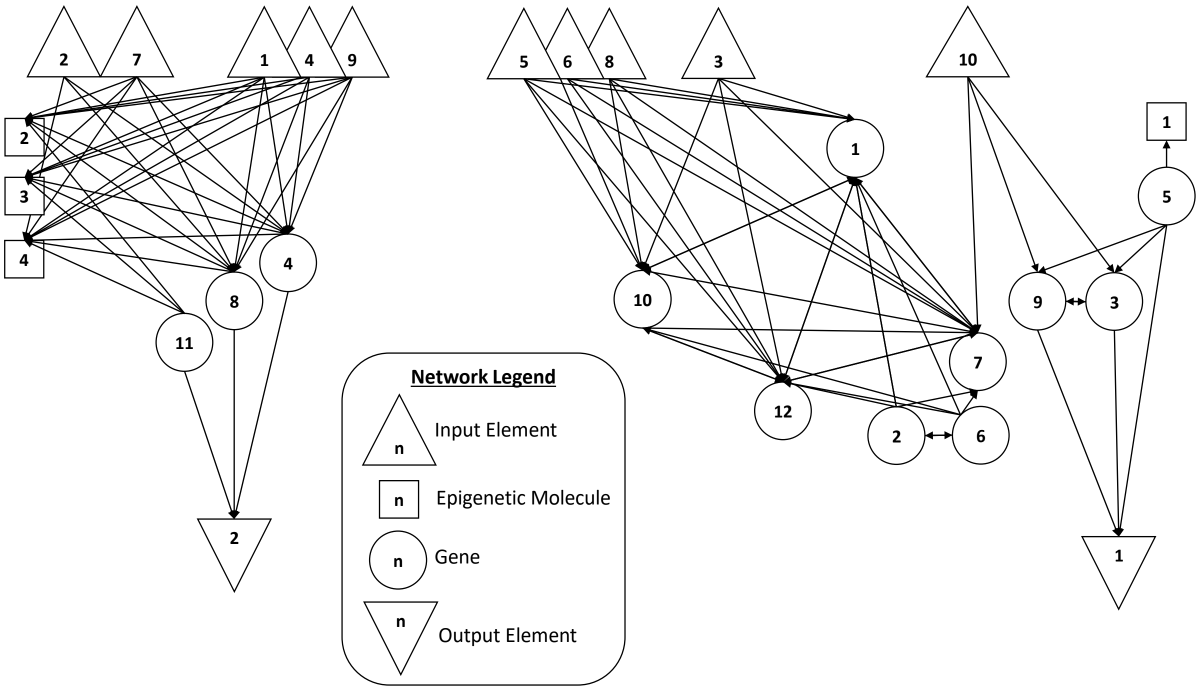


Figure 7.2.10. Structure of an epigenetic network of median fitness.

There are no significant revelations from the structure of the network, which is shown in figure 7.2.10 and derived from the parameter tables in appendix A, section 11.2. As with previous networks the intra-connected clusters are present, although there are no outliers for this network.

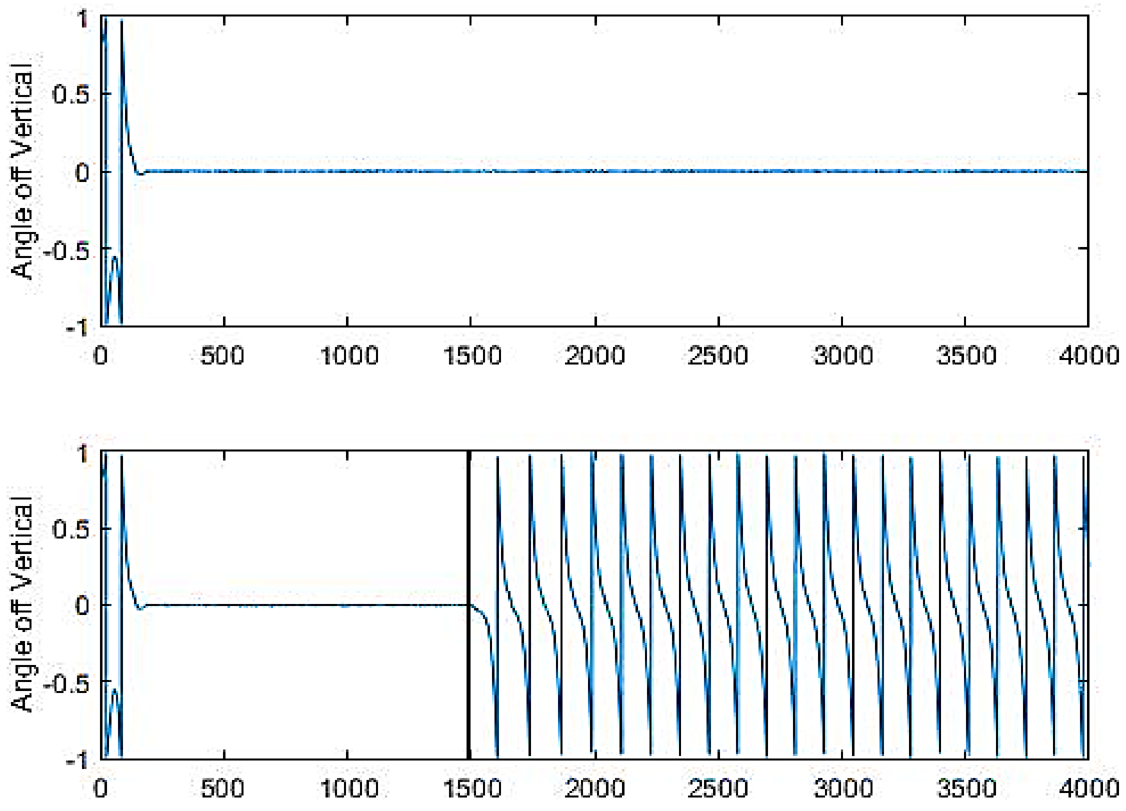


Figure 7.2.11. Plots of the angle off vertical of the pendulum when controlled by network 8578. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line.

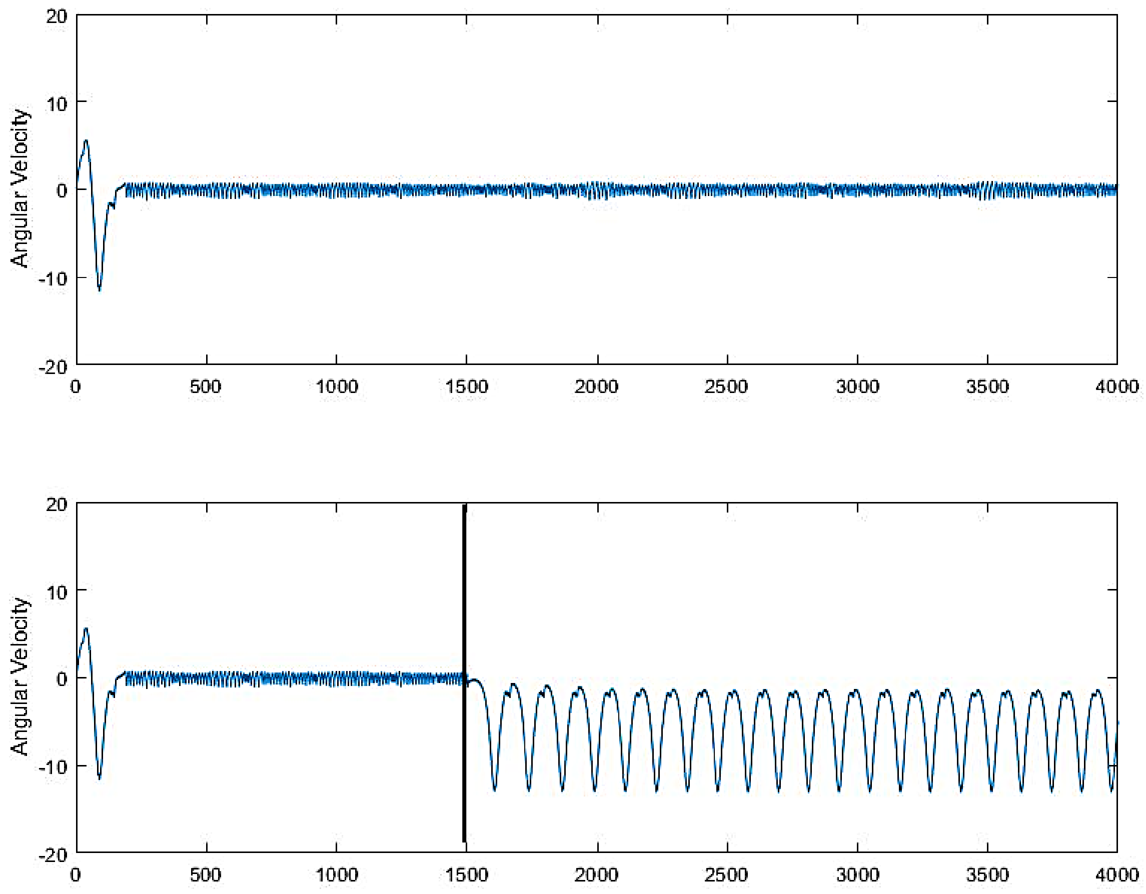


Figure 7.2.12. Plots of the angular velocity of the pendulum when controlled by network 8578. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line.

Figures 7.2.11 and 7.2.12 illustrate the angle and angular velocity for this network, which displays the closest fitness difference (0.18) to the median of the epigenetic network population (0.19). Unlike the previous two networks, the impact of the impulse injection is very clearly seen. The angle plot indicates that following the impulse, the pendulum starts rotating. However, the speed of rotation is not consistent, nor does it steadily decrease as it would if the cart was not moving. The angular velocity plot indicates that when the pendulum approaches the vertical position, its velocity decreases. This suggests the network is attempting to bring the rotation under control.

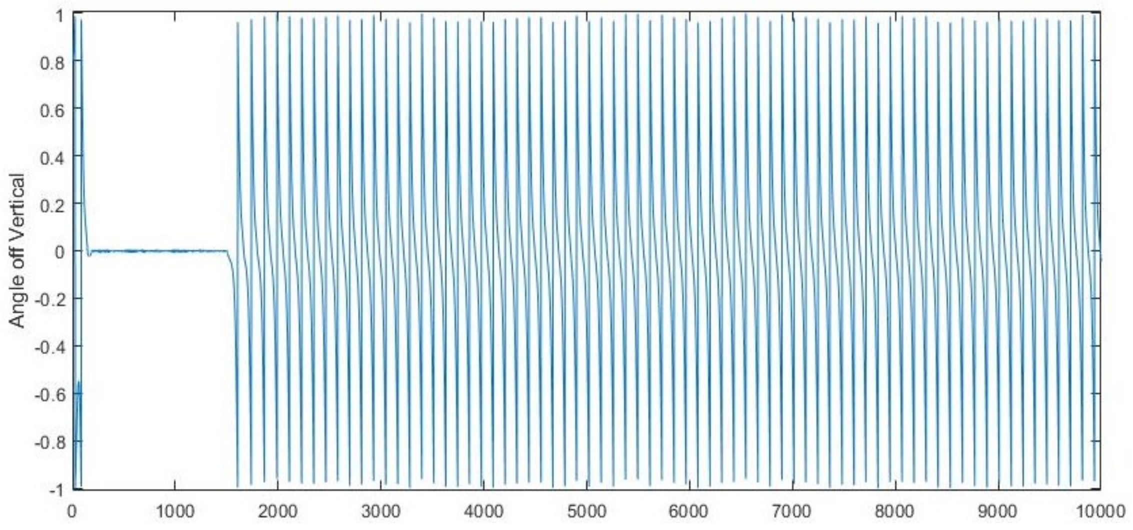


Figure 7.2.13. An extended plot of the angle off vertical of the pendulum when controlled by network 8578, with an impulse injection at 1500.

However, running the model for an extended period (10,000 cycles, equating to 100 seconds real time), it would appear that this rotational is something the network is not able to handle. This indicates that the boundary between a network able to recover from impulse injection, and a network that cannot is very small. Despite the difference between their fitness being only 0.18, the previous network almost ignores the impulse, while this network becomes unstable.

The next series of plots detail the action of the epigenetic molecules and genes during execution of the network. Figures 7.2.14 and 7.2.15 compare the epigenetic behaviour of the network under normal conditions (7.2.14) and during the injection of the impulse (7.2.15) by showing the difference in the activation states of the genes. The plot in figure 7.2.16 then illustrates which specific epigenetic molecules are affecting the activation state of which genes.

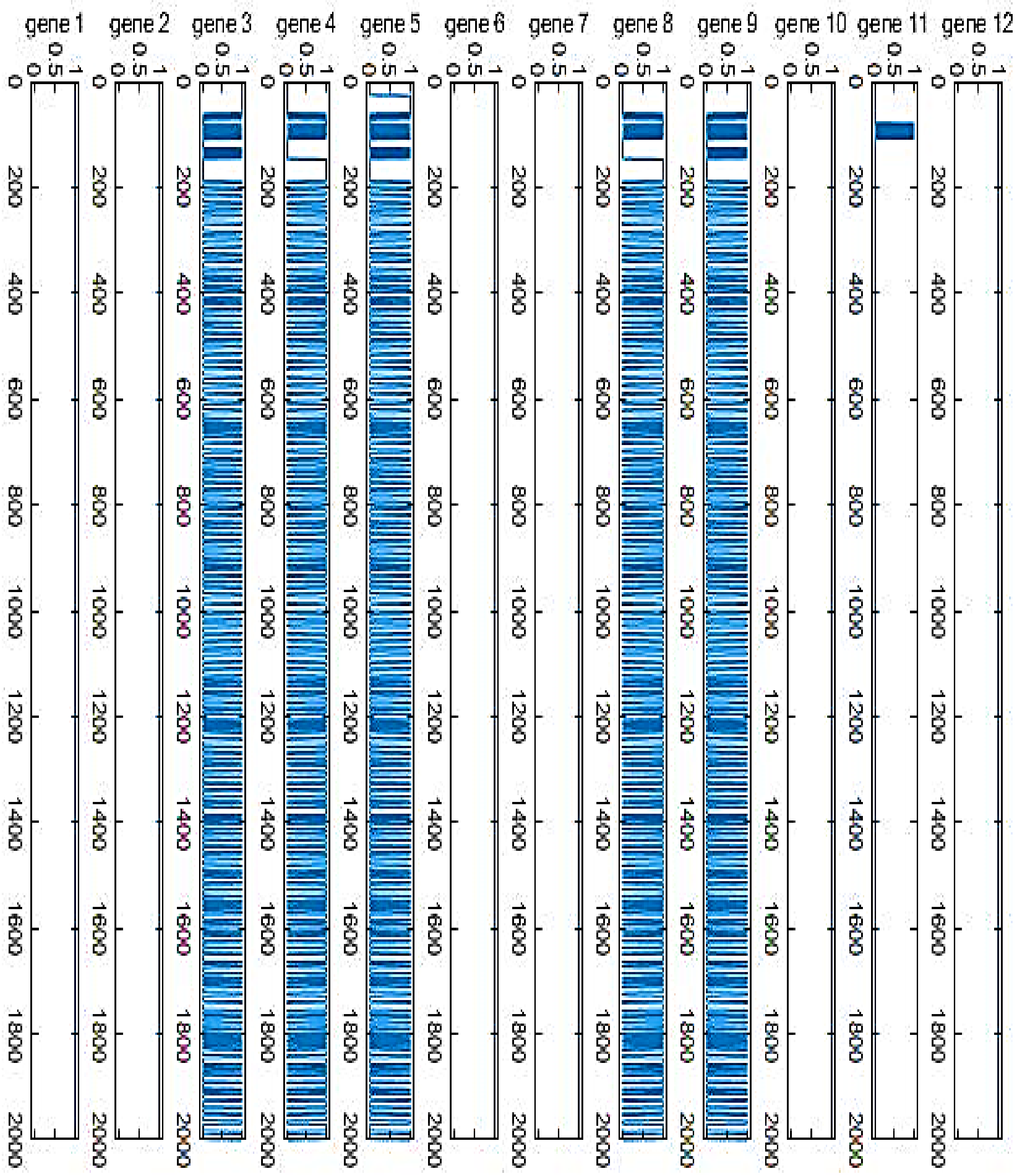


Figure 7.2.14. Activation state of genes within network 8578 between time steps 0 and 2000 (20 simulated seconds). A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

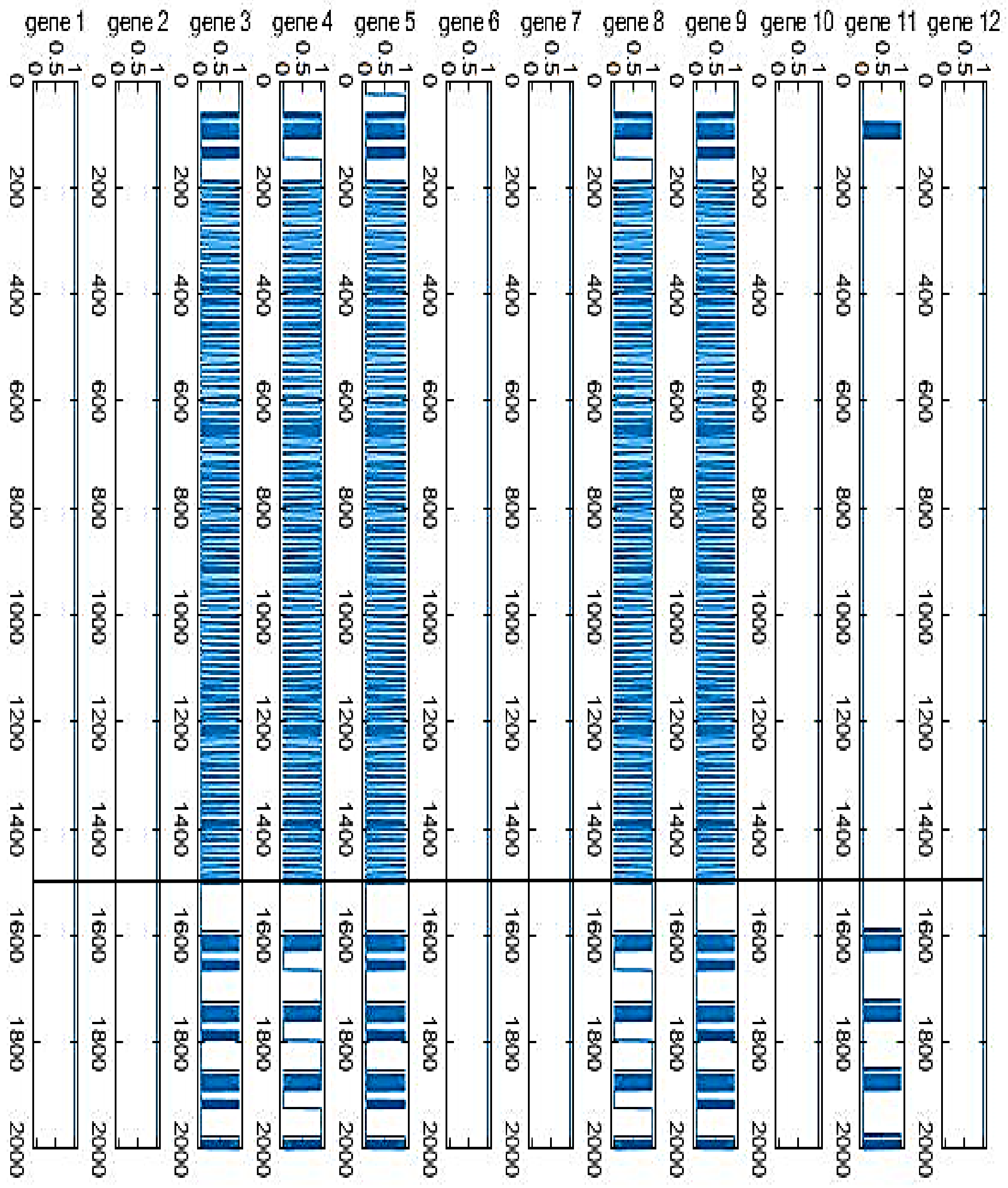


Figure 7.2.15. Activation state of genes within network 8578 between time steps 0 and 2000 (20 simulated seconds), with impulse injection at time step 1500, denoted by the black line. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

As with the previous network, only some of network 8578's genes are epigenetically active, and they fall into two of the three gene clusters (as indicated in figures 7.2.14 and 7.2.15). Genes 3, 5 and 9 all display the same pattern of epigenetic behaviour, as do genes 4 and 8. Gene 11 is a slight anomaly, in that while it is part of the same intra-connected cluster as genes 4 and 8, its epigenetic pattern is different. It does however share portions of its behaviour with the other genes in its cluster, suggesting that multiple epigenetic elements are acting on it.

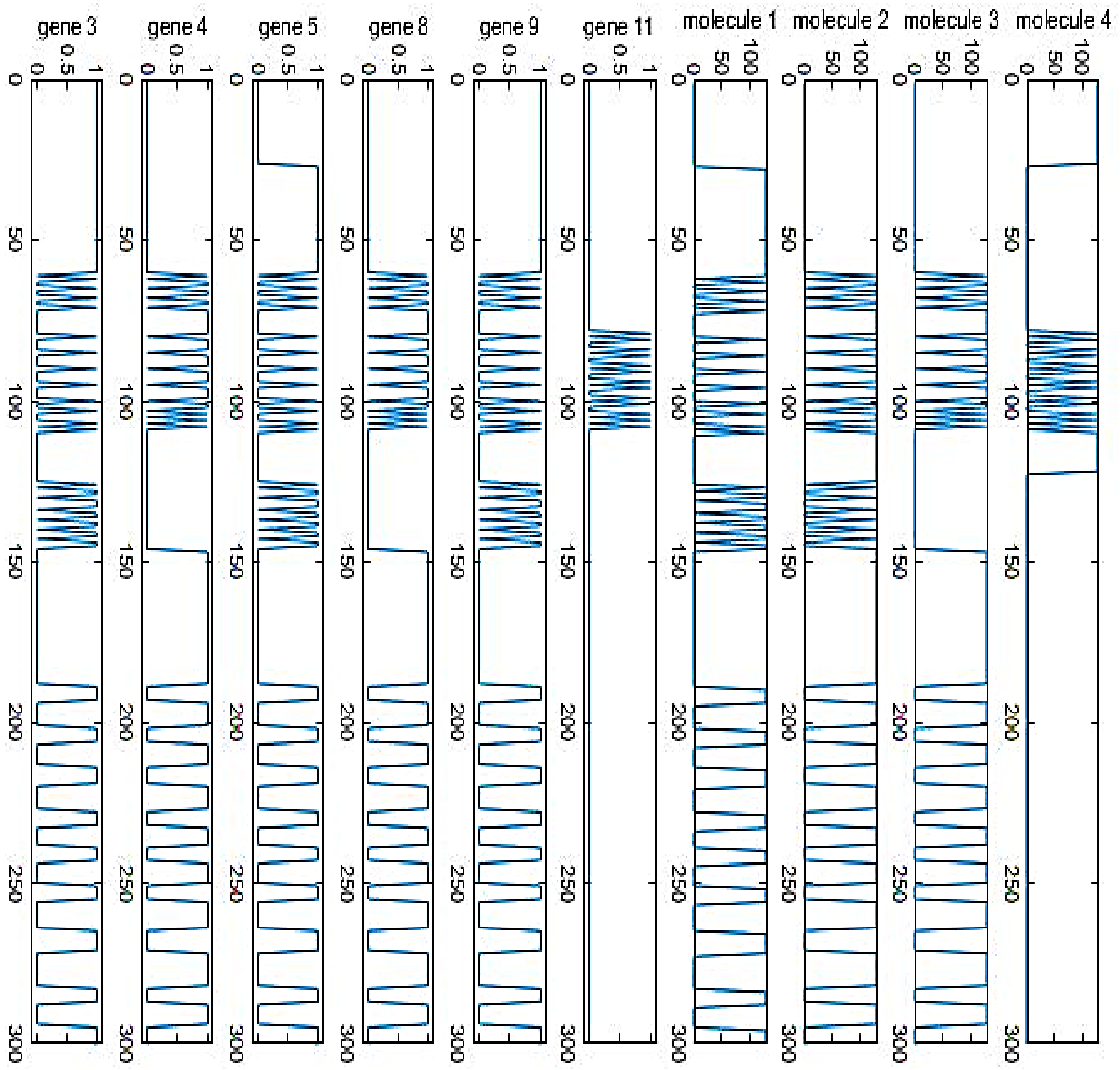


Figure 7.2.16. Activation state of genes exibiting epigenetic behaviour between time steps 0 and 300 (3 simulated seconds) and the outputs of the epigenetic molecules. With genes: a value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output. With molecules, their output is between 0 and 127, with the value relating to the location in the network space where it looks for genes to deactivate. See section 4.1.3 for more details.

Looking at the gene activation states and epigenetic molecule outputs together, Figure 7.2.16, confirms that this is indeed the case, and not just for gene 11. This shows that gene 5 also has slightly different behaviour to the other two genes in its cluster, in this case genes 3 and 9. Comparing the gene patterns with the outputs of the molecules indicates the following:

- Genes 3 and 9 are controlled by molecule 2;
- Gene 5 is also controlled by molecule 2, but can also be affected by molecule 4;
- Genes 4 and 8 are controlled by molecule 3; and
- Gene 11 is controlled by molecules 3 and 4.

The final network to undergo analysis is a non-epigenetic network with a post impulse injection fitness of 0.55, the closest to median impulse fitness value of 0.60.

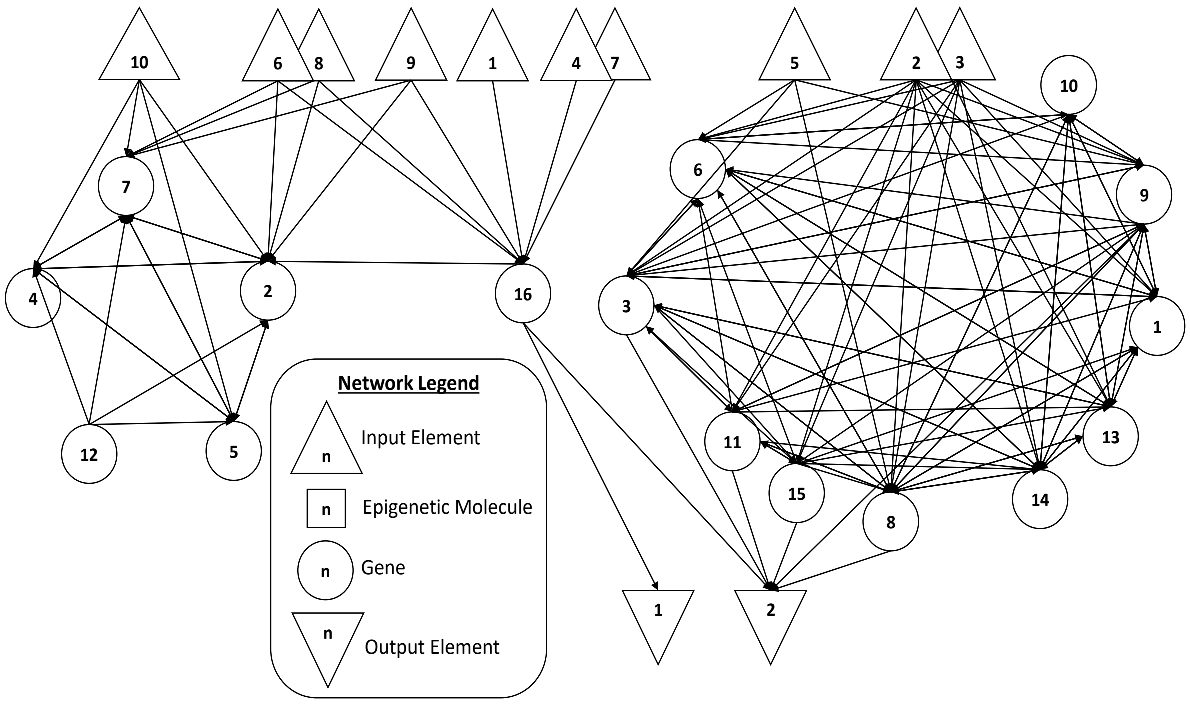


Figure 7.2.17. Structure of a non-epigenetic network of median fitness.

Figure 7.2.17 details the structure of this network, which is derived from the parameter tables in appendix A, section 11.3. As with previous networks, there is a clear clustering, with two intra-connected groupings developing:

- Genes 2, 4, 5, 7 and 12, which connect to Inputs 6, 8, 9 and 10.
- Genes 1, 3, 6, 8, 9, 10, 11, 13, 14 and 15, which connect to inputs 2, 3 and 5.

Gene 16 is on its own, connected to Inputs 1, 4, 6, 7, 8 and 9, an observation made noteworthy by the fact that gene 16 is the only one connected to output 1. Additionally, no other genes act as inputs to gene 16, making it solely responsible for one of the network outputs. The other output is connected to several of the genes from the second cluster, as well as gene 16, indicating that genes of the first cluster have no impact on the behaviour of the network.

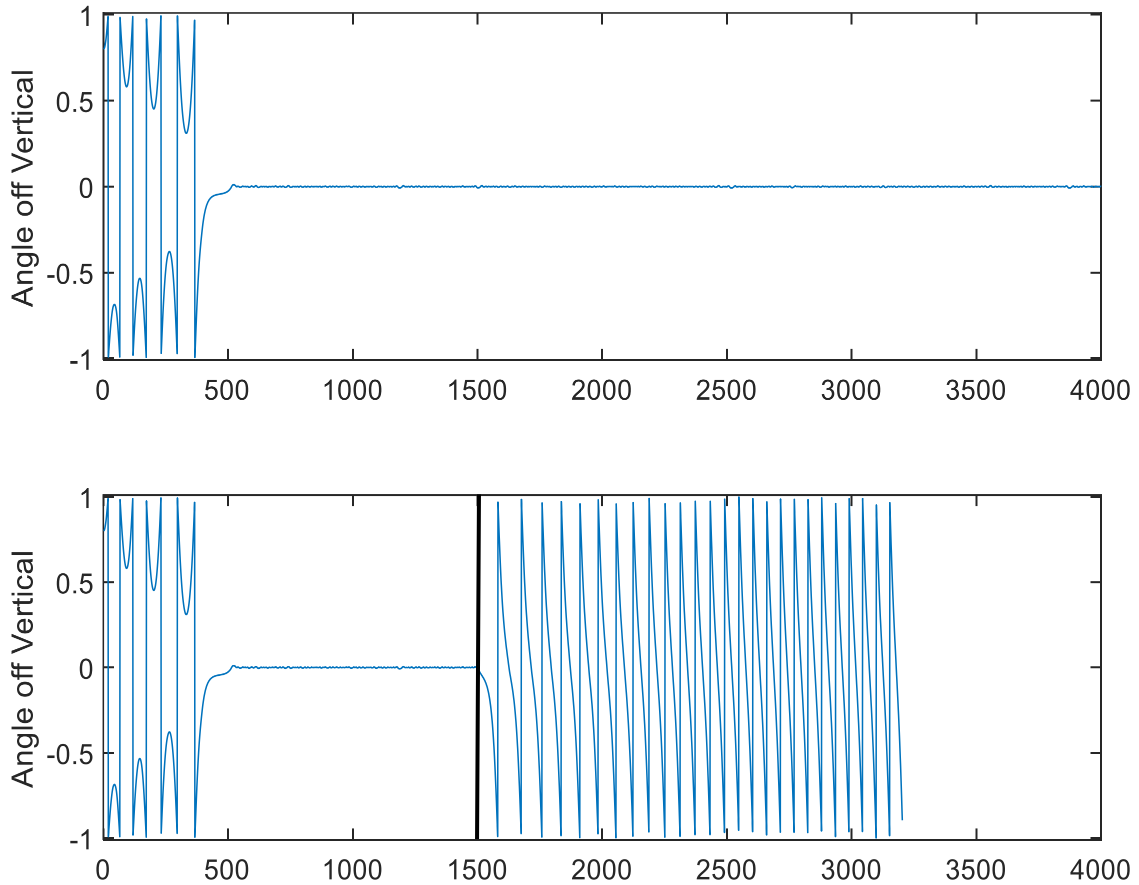


Figure 7.2.18. Plots of the angle off vertical of the pendulum when controlled by network 28314. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line.

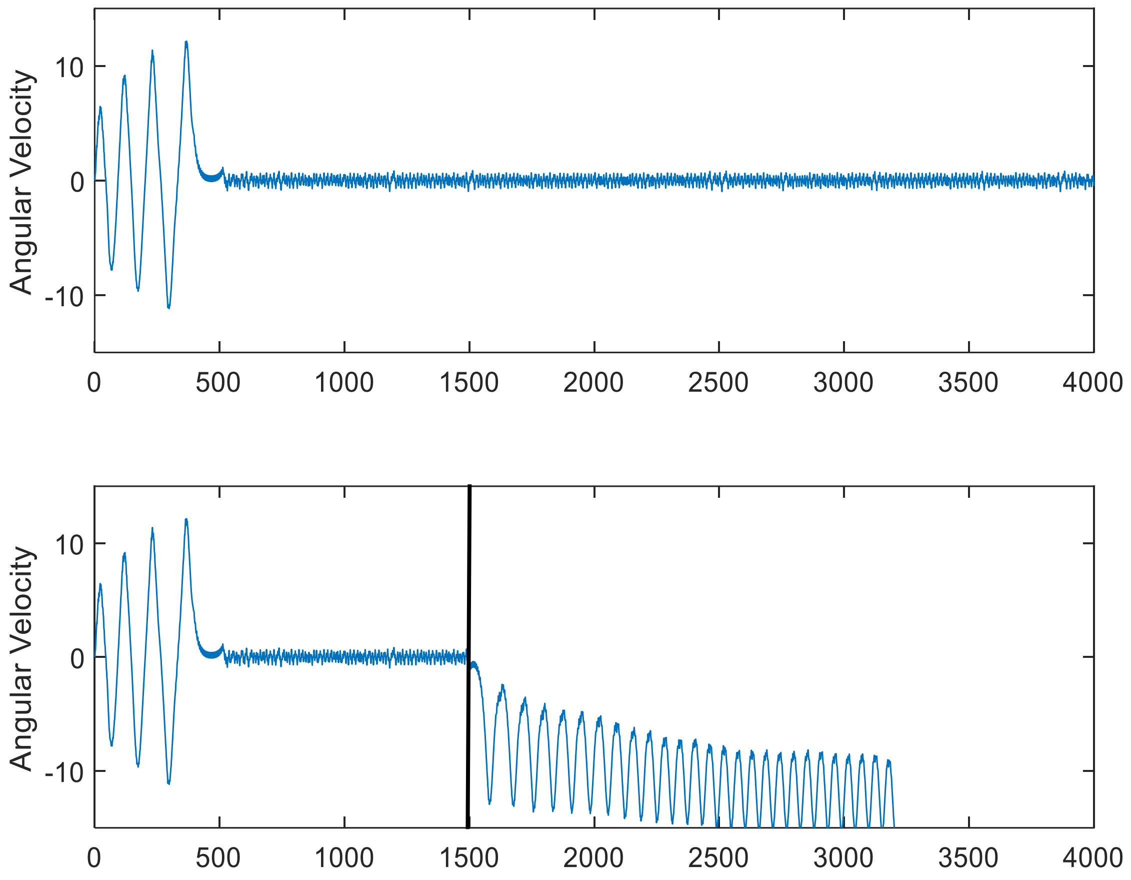


Figure 7.2.19. Plots of the angular velocity of the pendulum when controlled by the network. The top plot shows normal balancing; the bottom shows the response to impulse injection at time step 1500, denoted by the black line.

The Network's initial response to the impulse injection, shown in figures 7.2.18 and 7.2.19, is similar to that of the previous shown in figures 7.2.11 and 7.2.12. However, rather than developing into the steady rotational behaviour, the pendulum controlled by this network instead accelerates, eventually reaching over 15 rad S^{-1}, at which point the plots cut off. This is a consequence of one of the properties of the pendulum model, detailed in section 6.2, where it was noted that the inverted pendulum simulation can be terminated prematurely under certain conditions. One such condition was if the pendulum exceeded an angular velocity of 5π rad S^{-1}, which is clearly the case here.

Taking this together with the statistical results from section 7.2.2, it is reasonable to make the following statement: **Within the conditions of this experiment, an epigenetic network is capable of a superior response to an unevolved for external impulse, when compared to a counterpart network without epigenetic behaviour.** While this is only an initial experiment, it suggests that the core thesis of this work may be valid. Further investigation will be detailed in Chapter 8, where the switch to the more crucial robotics experiments are made. Section 7.3 of this chapter covers a brief aside into validating a structural change made to the epigenetic networks during their translation to hardware.

7.3 Connection Weights: Single Output vs Multiple Input

7.3.1 Experimental Design and Parameters

Separate from the mainline experiments, outlined in this section is an experiment to validate one of the design choices made when creating the epigenetic network outlined in this thesis. As detailed in Section 4.1.2, and illustrated in figure 7.3.1, Turner's original epigenetic networks applied connection weights at the output of each gene, effectively resulting in inhibitory or excitatory genes. This means that if Gene A and Gene B need to react differently to the output of Gene C, the network requires a version of Gene C for each of Genes A and B.

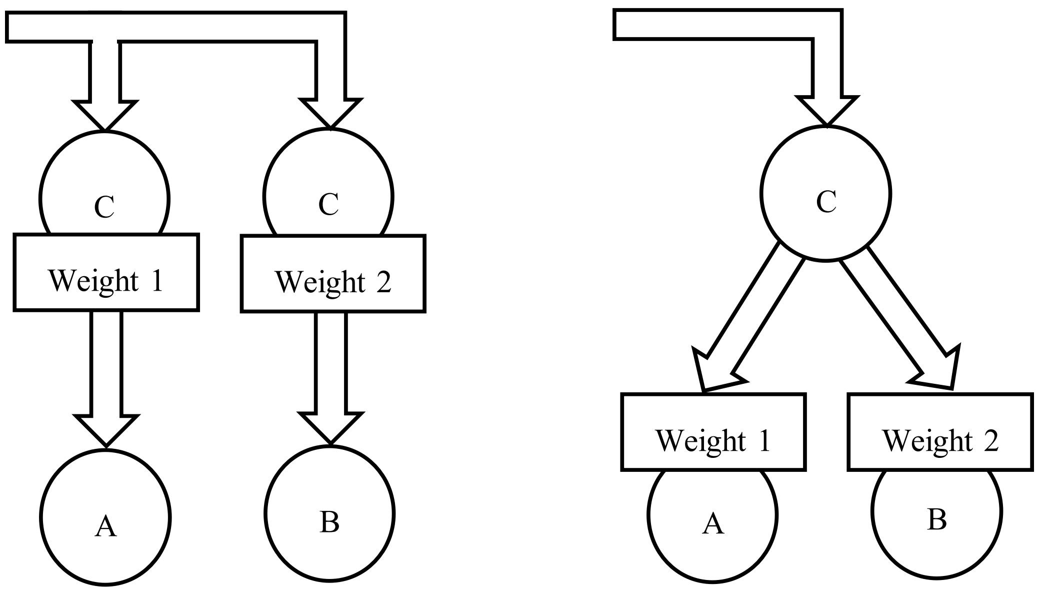


Figure 7.3.1. Single output applied weights (left) versus the more conventional input applied weights (right).

Given network compactness is a goal of this work, a more standard weight application method was implemented, with weights being applied to each input of each gene and molecule individually. This experiment is intended to confirm that this alteration does indeed permit smaller networks with higher fitness to be evolved than the original. The process to do this is straightforward, a series of networks with both types of weight structures will be evolved, optimising not only for fitness, but compactness (e.g. smaller networks will be better). The results for both network types will then be compared.

In order to evolve a network for both fitness and compactness, the NSGA-II genetic algorithm detailed in Section 3.4 will be employed. Additionally, a metric for network compactness will be required, equation 7.3.1.

$$\boldsymbol{F = 1.0 - \left(\frac{Size}{Size_{max}}\right)} \qquad \mathbf{(7.3.1)}$$

The compactness metric is straightforward, simply producing a value inversely proportional to the size of the network.

Table 7.3.1. Parameters for the NSGA-II evolution of fitness and compactness.

Parameter	**Value**
Population Size	128
Number of Generations	4096
Number of Repeats	50
Crossover Rate	0.5
Mutation Rate	0.05
Data Width	8
Maximum Number of Genes	25
Maximum Number of Epigenetic Molecules	5

Two important differences should be noted with this experiment's parameters, compared to the others outlined in this chapter:

(i) Both population size and the number of generations have been increased significantly from the experiments previously detailed in this chapter. This is because this experiment is intended to push the networks to their maximum possible limits, with no premature cut off due to insufficient time to evolve.

(ii) There is only a maximum number of genes and molecules, meaning that a network could potentially have only one of each. Again, this is so that the limits can be found, if the evolutionary algorithm is able to produce a high fitness network with so few elements, then it should be allowed to.

7.3.2 Results

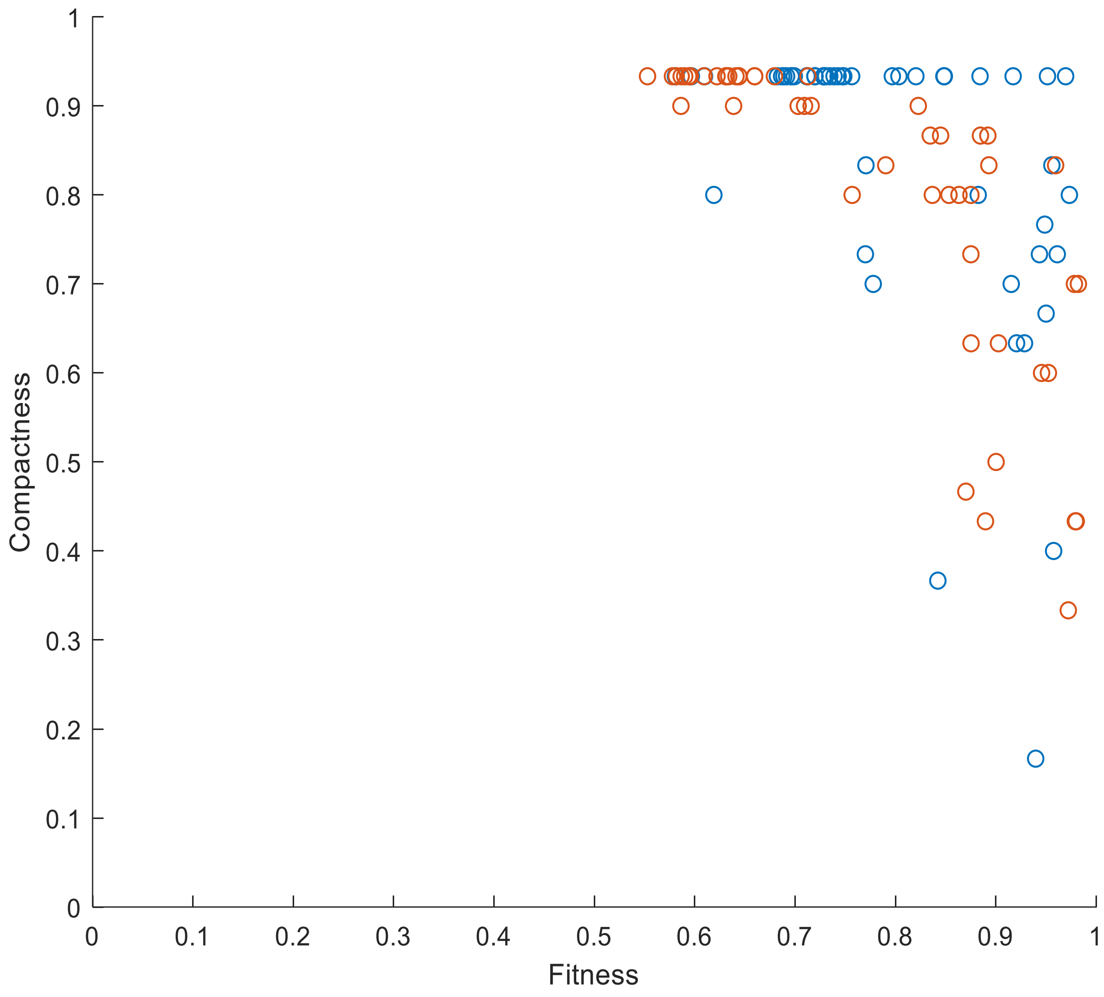


Figure 7.3.2. Functional fitness and compactness of normal, input applied weights (blue) and single, output applied weights (orange).

Figure 7.3.2 shows the results of NSGA-II evolution, with the two objectives, fitness and compactness, plotted against each other. Both the input and output applied weight networks are able to produce individuals with high compactness or high fitness, but it is clear that the normal, input applied weights are able to better achieve both together.

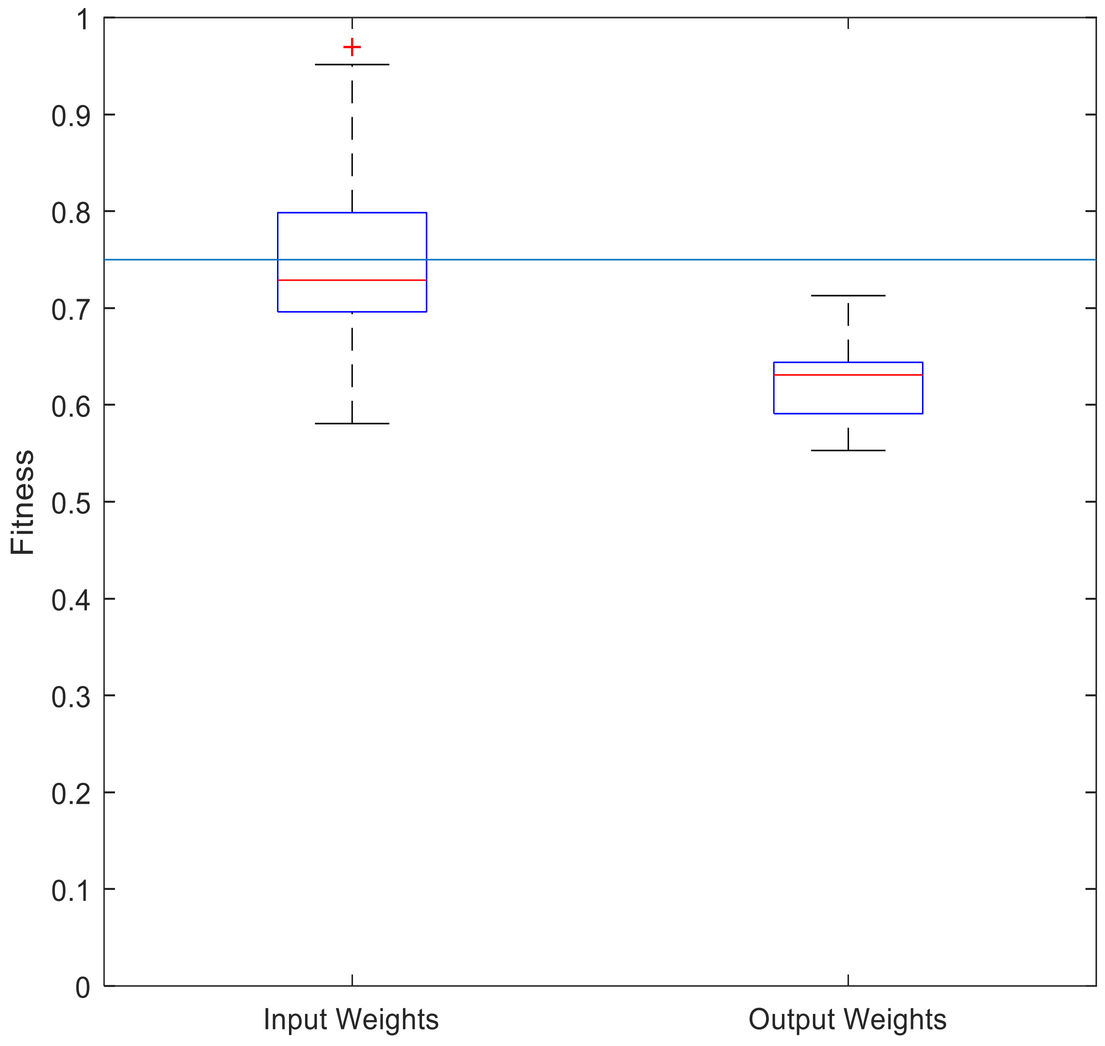


Figure 7.3.3. Fitness of all networks with a compactness of 0.93.

Figure 7.3.3 shows only the fitness of networks, of both types, that have a compactness of 0.93: 1 gene and 1 molecule. This shows even more than Figure 7.3.2 the difference, as it can easily be seen that while the median fitness of the normal, input weight networks is below 0.75, there are no single, output weight networks with a fitness above 0.71. It is therefore reasonable to state that the purpose of this experiment has been fulfilled, as these results show that the switch to a more conventional method of connection weighting allows for greater network compactness while retaining performance.

Table 7.3.2. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for normal, input weighted networks and single, output weighted networks with a compactness of 0.93. Note that outlier values are only included in the calculation of mean fitness.

Property	Input Weighted Networks	Output Weighted Networks
Minimum Fitness	0.58	0.55
25th Percentile Fitness	0.70	0.59
Median Fitness	0.73	0.63
75th Percentile Fitness	0.80	0.64
Interquartile Range	0.10	0.05
Maximum Fitness	0.97	0.71
Mean Fitness	0.75	0.62

Table 7.3.2 summarises the data points from figure 7.3.3, as well as the mean fitness of the two network types, thus allowing confidence limits and p-values to be calculated.

- Confidence Limits: $0.13 \pm 2.16(0.08)$
- p-value: $p < 0.10$

In this case, despite the apparent clarity of the raw data; $p > 0.05$, which indicates the results are insufficient to draw a significant conclusion.

7.3.3 Epigenetic vs Non-Epigenetic with Compactness Optimisation

Given the previous results, it is interesting to compare the epigenetic networks to their non-epigenetic counterparts when compactness is a factor under consideration. With results already available for the epigenetic networks (specifically those for the normal weights), experiments are required, with the parameters detailed in Table 7.3.1, but in this case with the epigenetic elements disabled.

7.3.4 Results

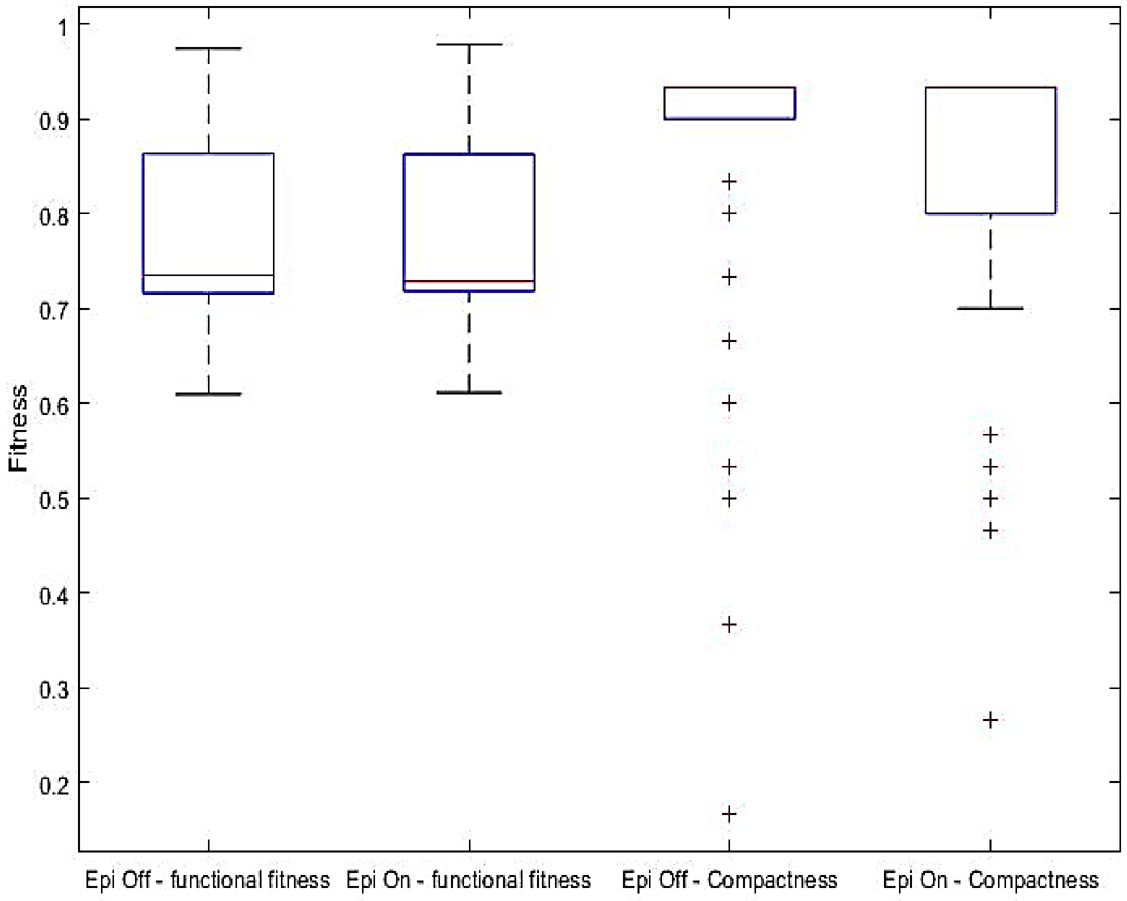


Figure 7.3.4. Functional fitness and compactness of networks with and without epigenetic elements.

Figure 7.3.4 shows the results of the new, epi off evolution compared to the original epi on evolution. The first thing to note is that in this case, the epigenetic and non-epigenetic networks have a median fitness below the 0.75 threshold, specifically 0.72. However, both were able to produce non-outlier networks with fitness values of 0.98. Regarding compactness, the vast majority of the non-epigenetic networks achieve greater compactness than their epigenetic counterparts, which on reflection is not surprising given that they have no need for the epigenetic elements.

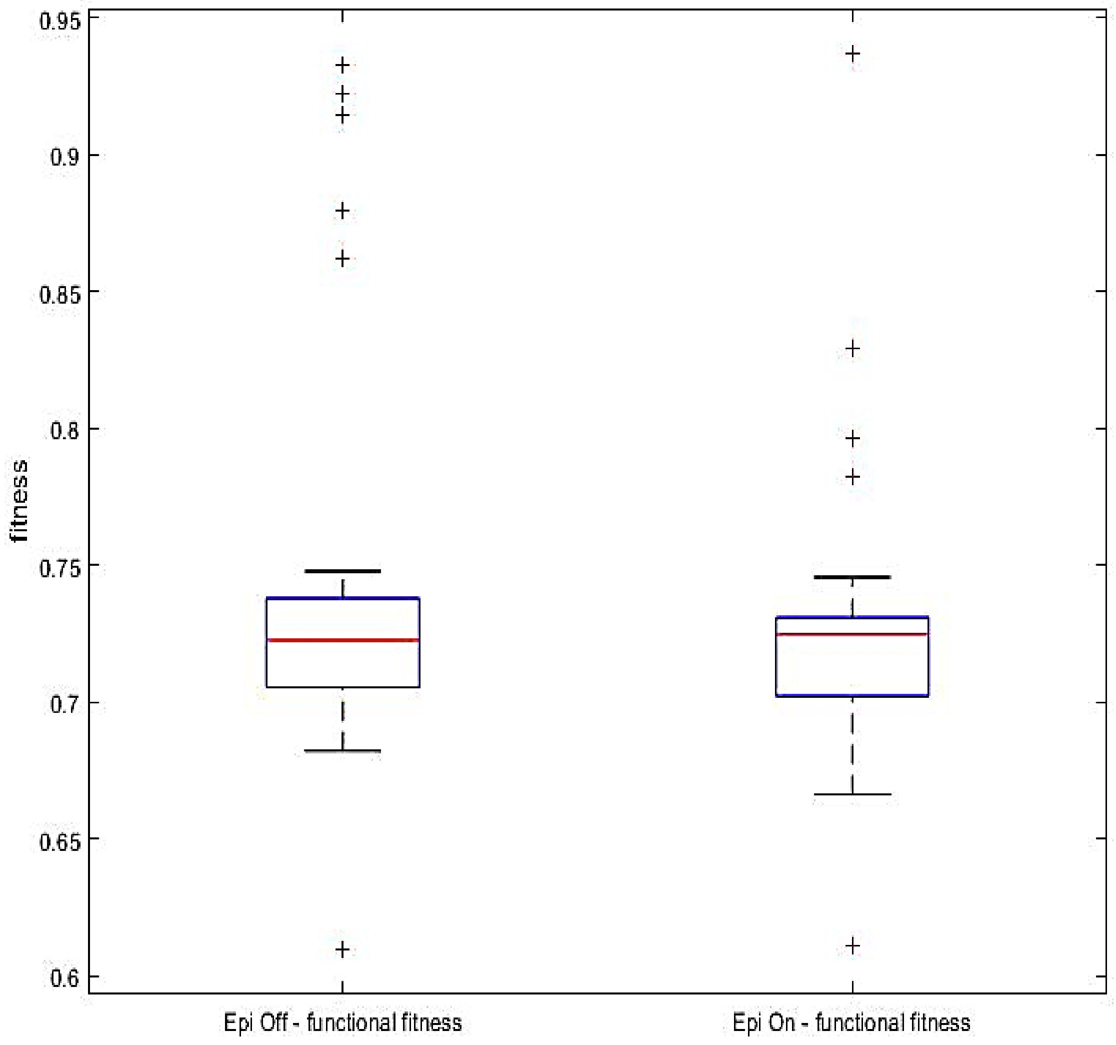


Figure 7.3.5. Functional fitness of all networks with a compactness of 0.93.

Looking solely at the maximum compactness networks, Figure 7.3.5, shows little to differentiate the two populations. In both cases, the median fitness values are 0.72, with a few outliers reaching a maximum of 0.94. The epigenetic population contains 32 individuals, while the non-epigenetic contains 36.

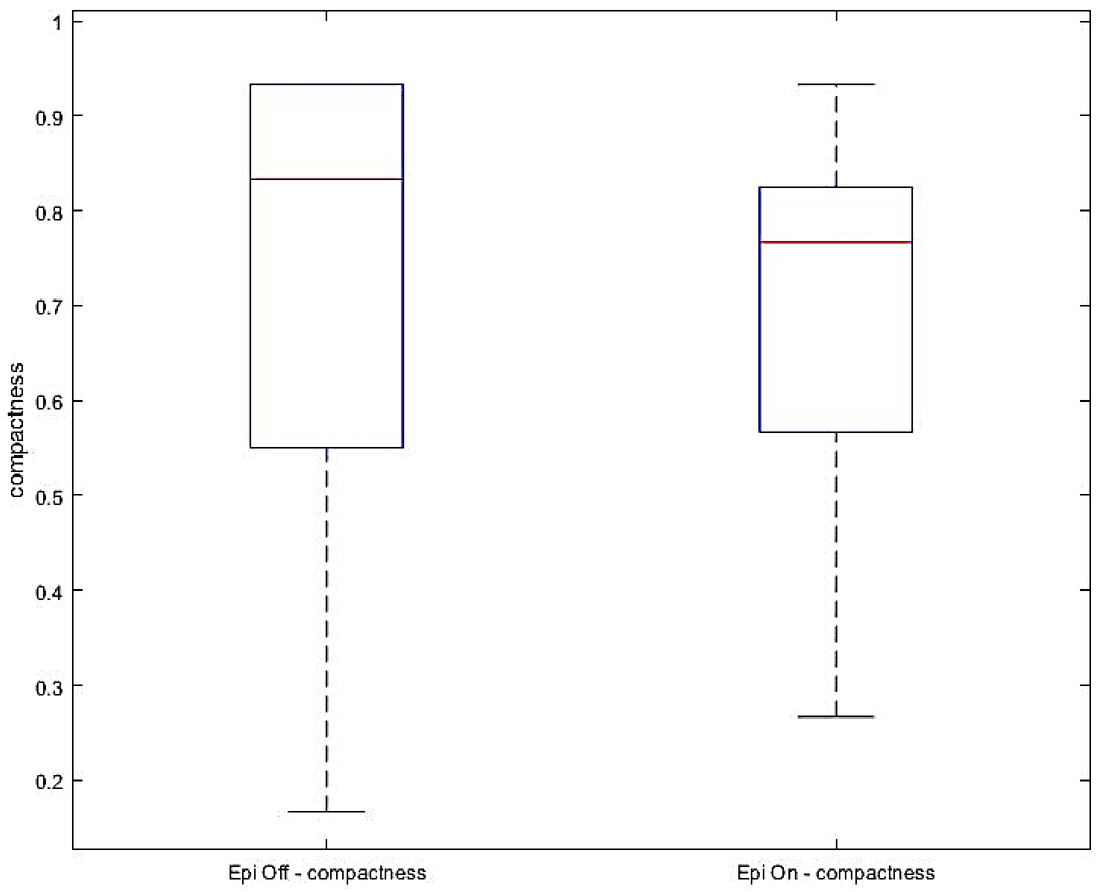


Figure 7.3.6. Compactness of all networks with fitnesses over 0.75.

Figure 7.3.6 shows the compactness of all networks above a set fitness threshold (0.75), these results also reveals little to differentiate the two populations. The total number of epigenetic networks is 19, and 20 non-epigenetic. Both populations contain individuals that achieve maximum compactness, while the epigenetic population has a median of 0.77 and the non-epigenetic and median of 0.83.

In an effort to obtain some final clarity, a version of the very first experiment is repeated: multiple epigenetic evolutions will be launched, each one with different network sizes. The crucial differences are that this time, the sizes will be from 2 to 16 network elements; and there will be a second set of evolutions of non-epigenetic networks for comparison. It is also worth noting, that the above considers network elements, not genes. This is because, in order to make a valid comparison between the epigenetic and non-epigenetic, it is important they are of the same size. It would be meaningless to consider a 2 gene epigenetic network as comparable in performance to a 4 gene non-epigenetic network, if the former also had 6 molecules.

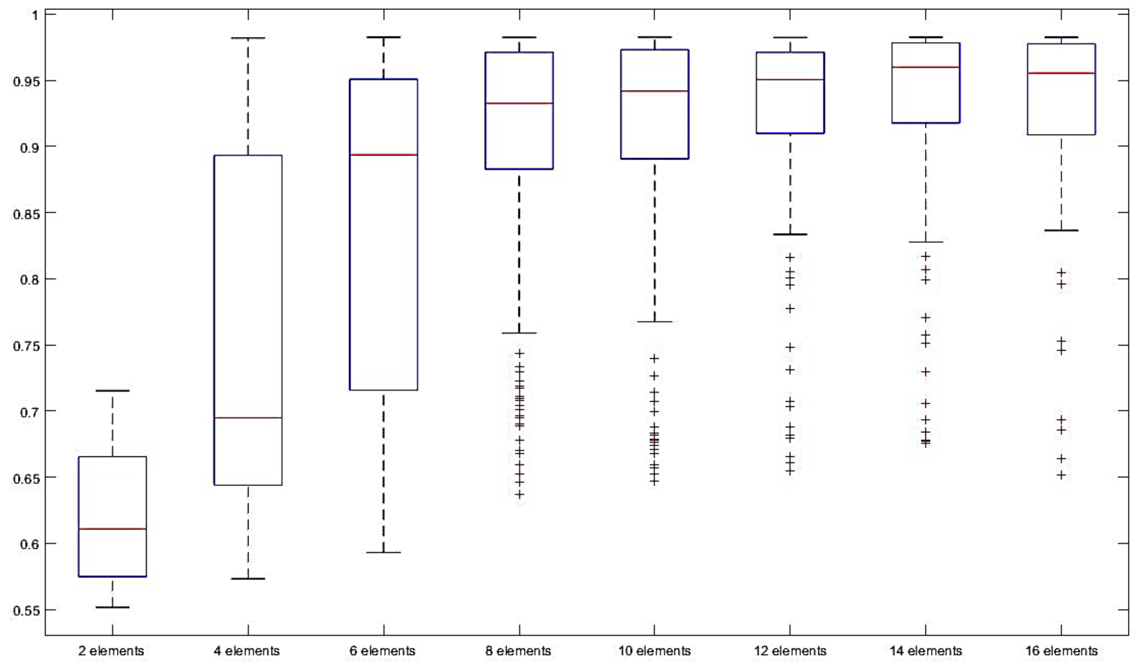


Figure 7.3.7. Fitness of epigenetic networks with between 2 and 16 network elements (genes and molecules).

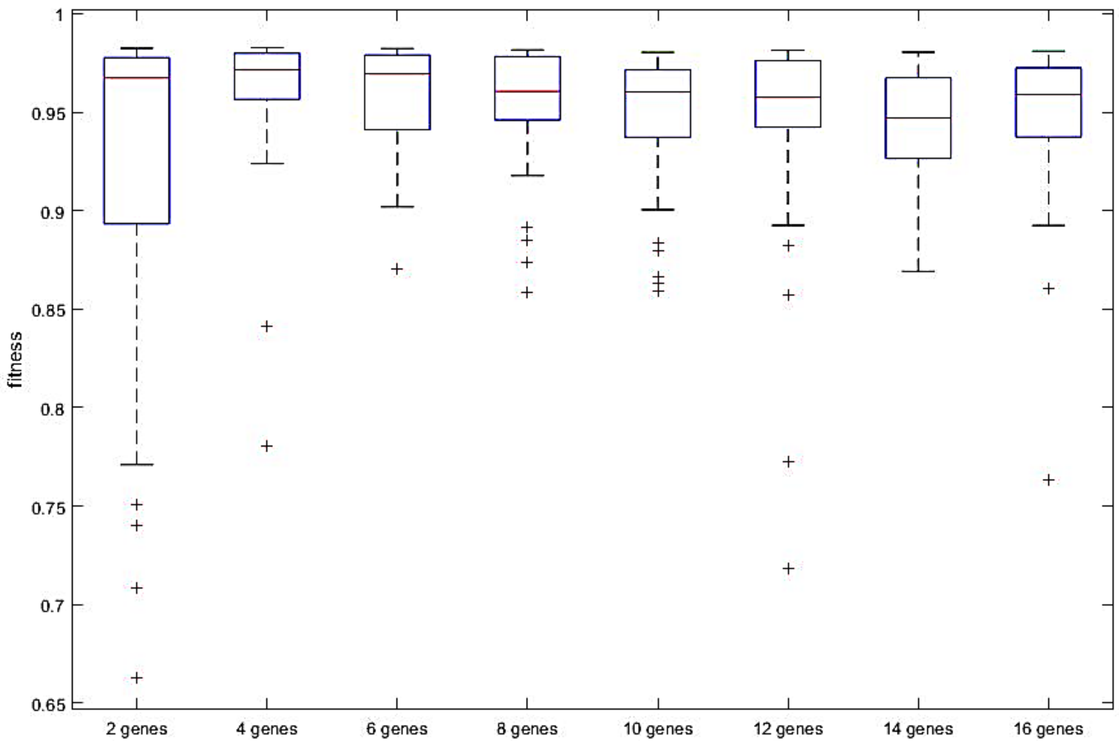


Figure 7.3.8. Fitness of non-epigenetic networks with between 2 and 16 genes.

Figures 7.3.7 and 7.3.8 show the results of these experiments. There is a very clear trend that indicates that the addition of epigenetic elements to an epigenetic network in place of some of that network's standard elements, does not provide an improvement in performance, at least with regards to the inverted pendulum. Perhaps the starkest illustration of this is with the two populations of networks with 2 elements: the epigenetic networks (1 gene, 1 molecule) were not able to exceed the 0.75 threshold, and possess a median fitness of 0.61; the non-epigenetic networks (2 genes) were able to all exceed the threshold, excepting a few outliers, but their median fitness is 0.96.

7.4 Summary

Considering all the experiments and results presented in this chapter, the following comments can be made:

The initial goal of this work has been successful, as a hardware implementation of the epigenetic network has been developed, and demonstrated to perform comparably to its software counterpart on the inverse pendulum. Furthermore, this comparable performance has survived a translation from a floating-point to integer implementation, as well as a dramatic reduction in data width. However, further analysis has revealed that the presence of epigenetic elements with a network is no guarantee of epigenetic activity, as the genetic algorithm was able to achieve high levels of fitness of the inverse pendulum without actually connecting the epigenetic elements to any of the genes involved in actual processing.

In looking to challenge the epigenetic network, with the aim of producing a clear division between it and more standard gene regulatory (or non-epigenetic) network, populations of high fitness networks were exposed to a stimulus, in the form of a dynamic environment. This took the form of an impulse injection, simulating the pendulum being impacted at high speed by another object. In this case, there was good evidence to suggest that, while further refinement would be beneficial, the epigenetic elements had a measurable positive impact on the network's performance.

Finally, in order to validate the assertions made in Section 4.1.2, the effect of the alterations made to the network's connection weighting was tested. This confirmed that by switching from the original, output applied single weights to the more conventional input applied weights, the networks can be significantly reduced in size, with some individuals even attaining high levels of functionality with only 1 gene and 1 molecule. Additionally, this prompted a comparison between epigenetic and non-epigenetic networks along similar lines, looking to see if epigenetic elements were able to allow for very compact networks. However, the results offered a very clear statement the replacement of standard network elements with epigenetic ones actually resulted in a significant reduction in overall performance.

Drawing the work of this chapter to a close, the hypothesis laid out in section 6.4 does look to have some preliminary support. More evidence is needed however for a definite conclusion, and sticking to a single type of problem would be inadvisable. Thus, the next chapter of this work will focus on repeating the type of experiment of section 7.2, but with a different problem: Foraging Robots.

8 Robot Foraging Experiments

This chapter covers the first series of experiments carried out using the JBotEvolver platform [77] and simulated robot detailed in Section 6.3 of this thesis, as well as expanding upon some of the experiments of Chapter 7.

As this is a new problem for the epigenetic network, the first stage of experiments is intended to confirm the viability of the network as a robot controller, in general, and to see if it is capable of handling a simple foraging task. In addition, the same experiment will be repeated on a comparable non-epigenetic network, to establish a baseline for later comparison. The second set of experiments introduce the added complexity of a simple maze, necessitating that the robot develop behaviours to navigate the maze while also foraging for prey. The final layer of complexity comes with the substitution of the normal, stationary prey with prey objects that move about the environment. In order to continue being a successful foraging, the robot will need to be able to compensate for this added complexity; as prey may now move out of range of its manipulator just as it is about to grab them.

8.1 Basic Foraging Environment

8.1.1 Experiment Design and Parameters

The initial validation experiments will take place in the basic foraging arena described in Section 6.3, illustrated in figure 6.3.6. The simulated robot will begin at the centre of the environment, and will have a limited amount of time to acquire as much food as possible and return it to the nest, also located in the centre of the environment. Section 6.3 also covers the characteristics of the simulated robot, which is illustrated in figure 6.3.5. Table 8.1.1 details the parameters of the environment.

Table 8.1.1. Parameters of the foraging arena.

Parameter	**Value**
Outer Boundary Radius	5.0 Units
Food Spawn Boundary Radius	2.0 Units
Nest Radius	0.5 Units
Initial Number of Food	20
Respawning Food	TRUE

The first three parameters in table 8.1.1 are self-explanatory, as they relate directly to aspects of the environment shown in figure 6.3.6. 'Initial Number of Food' is the number of food objects that will be distributed randomly throughout the environment at the start of the simulation. 'Respawning Food' relates to what occurs when a food item is deposited into the nest: TRUE means the food item will be placed in a new random location, hence keeping the total number of food items in the environment static, and giving no upper limit to the amount of food that the robot can return to the nest.

Table 8.1.2. Parameters for the evolution of epigenetic networks.

Parameter	Value
Number of Simulation Time Steps	1000
Population Size	500
Number of Generations	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 8.1.3. Parameters for the evolution of non-epigenetic networks.

Parameter	Value
Number of Simulation Time Steps	1000
Population Size	500
Minimum Target Fitness	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	16

Tables 8.1.2 and 8.1.3 contain all the experimental parameters, as in the experiments detailed in Chapter 7. Note the absence of the crossover rate parameter, as the genetic algorithm integrated into the simulator does not use crossover, see section 3.2 for more details.

8.1.2 Results

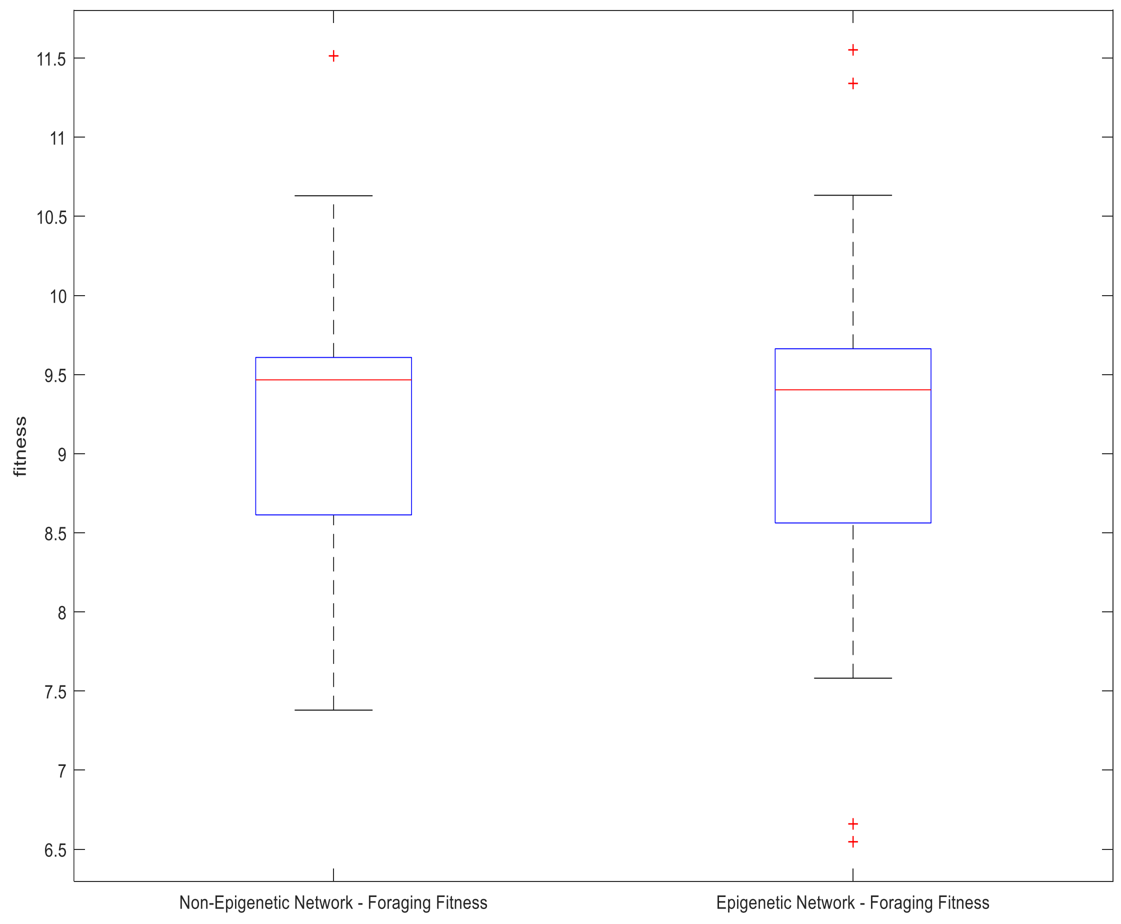


Figure 8.1.1. Fitness of the best individuals from 50 evolutionary runs of epigenetic and non-epigenetic networks. Note that any fitness value greater than 1.0 indicates the successful collection and return of at least one prey object.

Table 8.1.4. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks. Note that outlier values are only included in the calculation of mean fitness.

Property	**Non-Epigenetic Networks**	**Epigenetic Networks**
Minimum Fitness	7.37	7.51
25th Percentile Fitness	8.61	8.56
Median Fitness	9.47	9.40
75th Percentile Fitness	9.61	9.66
Interquartile Range	1.00	1.10
Maximum Fitness	10.60	10.61
Mean Fitness	9.27	9.16

Looking at figure 8.1.1, and the summary of its significant points in table 8.1.4, it can be seen that, when presented with the basic foraging environment, both the epigenetic and non-epigenetic network controlled robots are able to reach similar fitness levels. It should be noted that the integer component of the fitness value directly correlates to the amount of prey returned to the nest.

This establishes an excellent baseline for later comparison, as it means that later divergences in performance will be a result of the changing experimental parameters. This simple experiment also confirms the functionality of the experimental setup.

8.1.3 Network Analysis

As control of a foraging robot is a new task to epigenetic network, it is prudent to validate that epigenetic behaviour is occurring. As such, analysis of a sample network will be performed, specifically one with a fitness close to the median.

Starting with the structure of the network, which is shown in figure 8.1.2 and derived from the parameter tables in appendix A, section 11.4

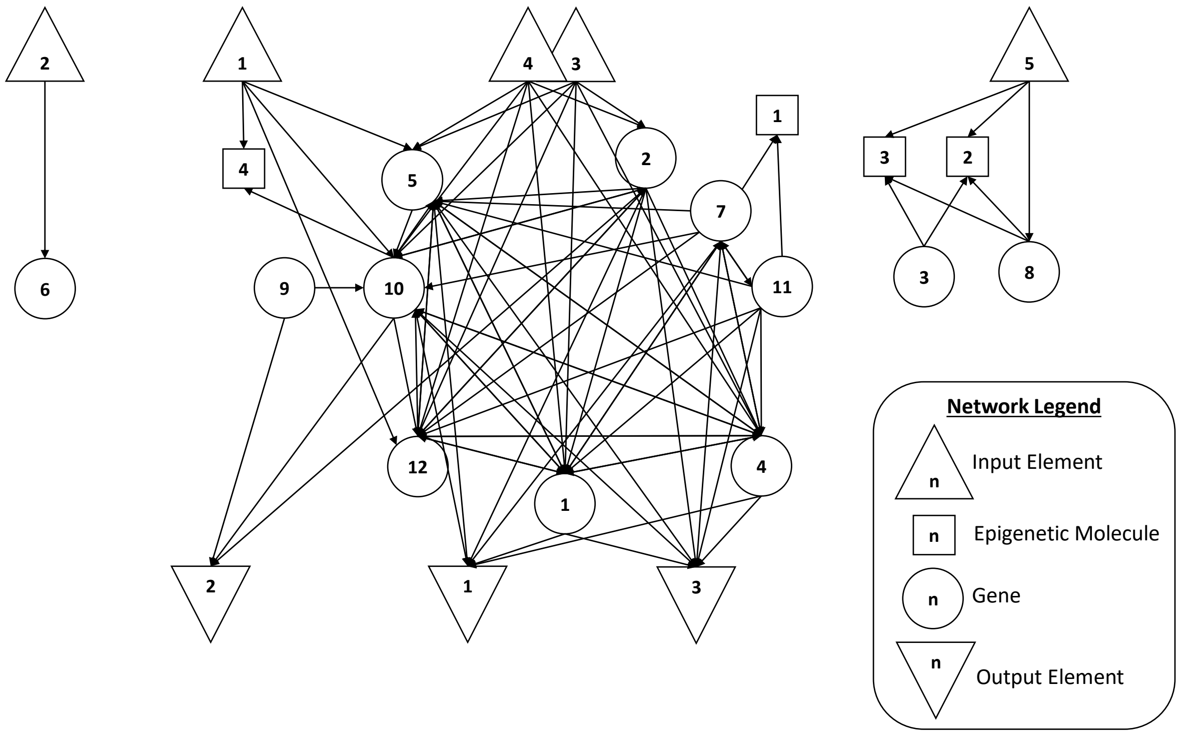


Figure 8.1.2. Structure of the network.

Considering the structure of the network, several features can be identified, the most prominent of which is an interconnected cluster of elements at the centre of the network space, consisting of genes 1, 2, 4, 5, 7, 10, 11 and 12. This cluster connects to inputs 1, 3 and 4, as well as molecule 1, and also connects to all three of the network outputs. The other 4 genes of the network have very few connections, with genes 3 and 9 not having any inputs at all, while genes 8 outputs only to molecules 2 and 3 and gene 6 has no outputs at all. Previous work, illustrated in the network analysis in Chapter 7, has shown that the elements of an epigenetic network will cluster together, but there are typically several, spread throughout the network space.

Figure 8.1.3 shows the epigenetic activity within the network during foraging, and while it clearly shows two genes (3 and 9) being affected by the epigenetic elements, which genes they are is problomatic. Looking at figure 8.1.2, it can be seen that the two epigenetically active genes are also the ones that have no inputs. Additionally gene 3, the most epigenetically active, only outputs to two of the networks epigenetic elements, and none of its outputs. This leads to the conclusion that: in this case, there is no benefit to the addition of epigenetic behaviour. One conclusion from this is that the task is not complex enough to allow the epigenetic structures to make a significant difference over the more simple non-epigenetic strcuture. It is therefore prudent to move onto the next, more complex foraging task, which will hopefully introduce the needed stimulus to trigger the development of epigenetic behaviour.

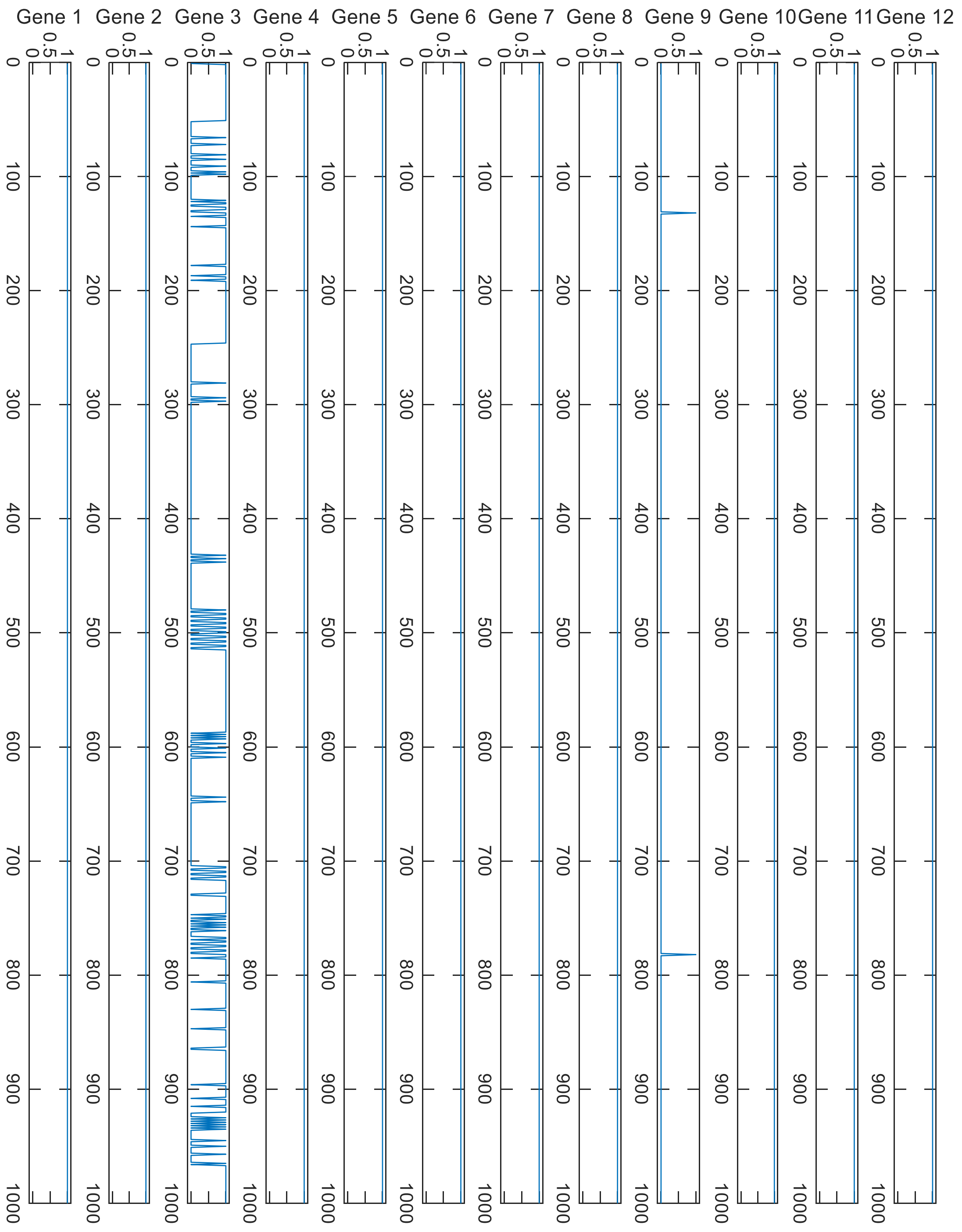


Figure 8.1.3. Activation state of genes within the network between time steps 0 and 1000. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

8.2 Maze Foraging Environment

8.2.1 Experiment Design and Parameters

As with the basic foraging experiment in Section 8.1, the simulated robot will have a set period of time to return as much food to the nest as possible.

Table 8.2.1. Parameters of the foraging arena.

Parameter	Value
Food Spawn Boundary Radius	0.1 Units
Nest Radius	0.2 Units
Initial Number of Food	20
Respawning Food	TRUE

Table 8.2.1 contains the parameters for the basic maze environment shown in figure 6.3.7. The 'Nest Radius'; 'Initial Number of Food'; and 'Respawning Food' parameters refer to the same properties as in Section 8.1, but the 'Food Spawn Boundary Radius' parameter now refers to the size of the spawning circles, as illustrated in figure 6.3.7. Food is randomly placed in these circles at the start of the simulation time, and will be respawned into one when it is returned to the nest by the robot.

Table 8.2.2. Parameters for the evolution of epigenetic networks

Parameter	Value
Number of Simulation Time Steps	10000
Population Size	500
Number of Generations	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 8.2.3. Parameters for the evolution of non-epigenetic networks

Parameter	**Value**
Number of Simulation Time Steps	10000
Population Size	500
Minimum Target Fitness	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	16

The parameters of the experiments, shown in tables 8.2.2 and 8.2.3, are largely the same as those of the pervious experiments, shown in tables 8.1.2 and 8.1.3. They only difference is that the simulation time has been increased from 1000 to 10000 time steps, to account for the increased time required for the more complex task.

8.2.2 Results

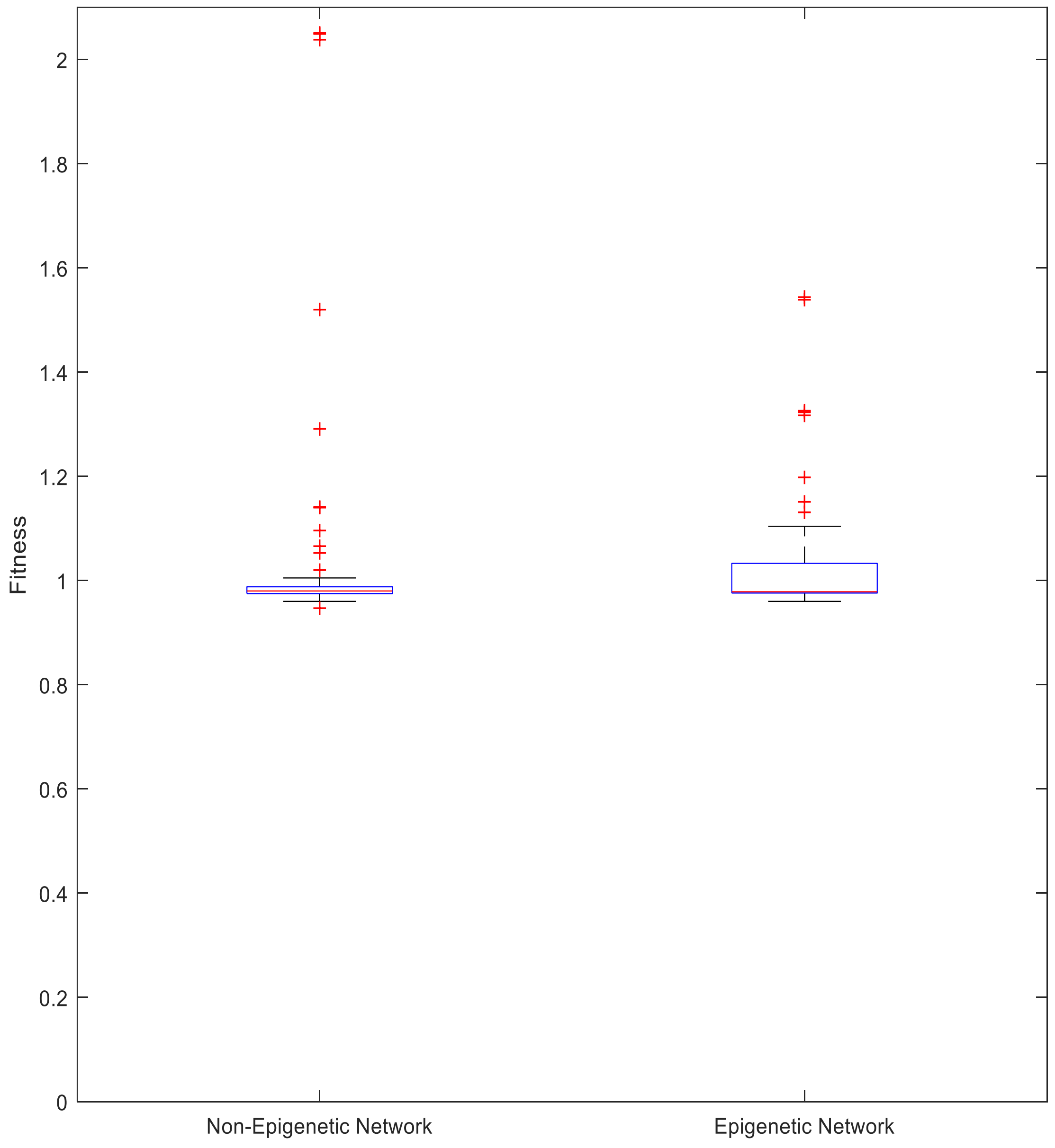


Figure 8.2.1. Fitness of the best individuals from 50 evolutionary runs of epigenetic and non-epigenetic networks. Note that any fitness value greater than 1.0 indicates the successful collection and return of at least one prey object.

Table 8.2.4. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks. Note that outlier values are only included in the calculation of mean fitness.

Property	Non-Epigenetic Networks	Epigenetic Networks
Minimum Fitness	0.960	0.960
25th Percentile Fitness	0.975	0.976
Median Fitness	0.980	0.979
75th Percentile Fitness	0.988	1.033
Interquartile Range	0.013	0.057
Maximum Fitness	1.053	1.104
Mean Fitness	1.070	1.040

Figure 8.2.1 shows the comparison between the results of the epigenetic and non-epigenetic controlled robots, with the important points summarised in table 8.2.4. The epigenetic and non-epigenetic networks perform similarly, with very close median and mean fitness values. In both cases, however, not all of networks were able to successfully return any prey to the nest, with only 14 epigenetic and 12 non-epigenetic networks completing this section of the task, with most of them being outliers. This should still be enough to progress forward, provided that, unlike in the previous case, epigenetic activity can be demonstrated.

8.2.3 Network Analysis

As an epigenetic network of median fitness would not encapsulate the completion of a successful foraging run, the network under scrutiny is instead one with a fitness as close to the mean value as possible: 1.0327. Figure 8.2.2 details the structure of this network, derived from the parameter tables in appendix A, section 11.5.

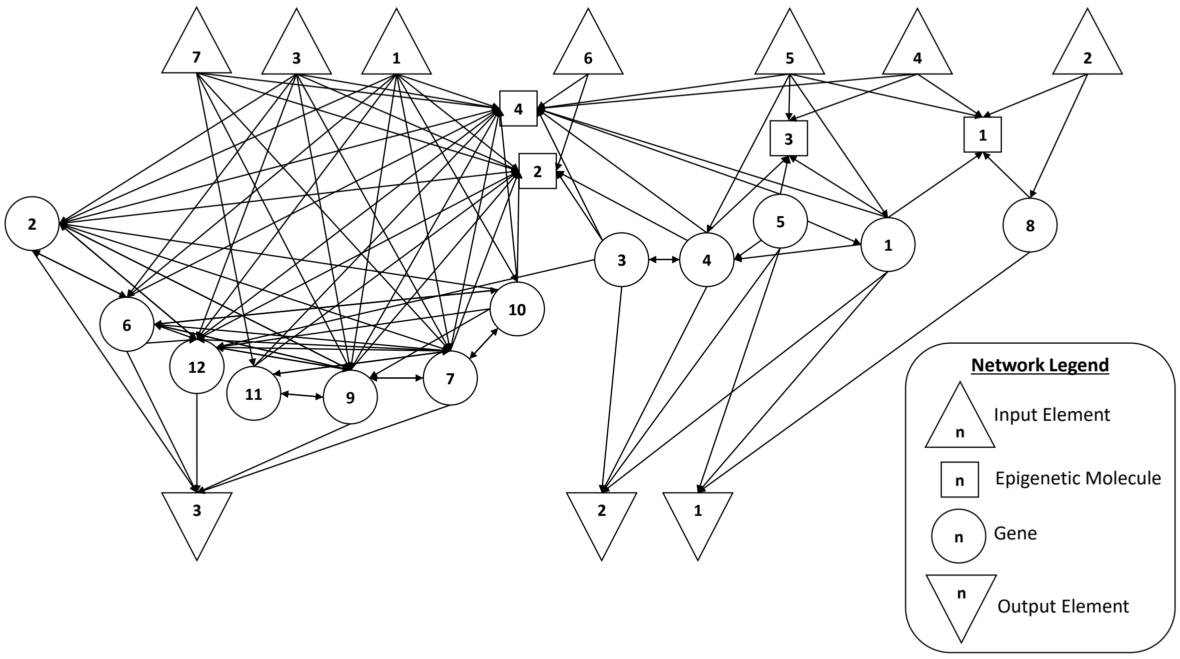


Figure 8.2.2. Illustration of the epigenetic network's structure.

Considering the network structure, the usual gene clustering can be seen, with clusters composed of: genes 2, 6, 7, 9, 11, 10, 12; and genes 3, 4, 5 and 1; with gene 8 out on its own. Gene 12 connects gene 3 into the first cluster, but other than that the cluster connections are once again primarily internal. In contrast to the networks analysed previously in this chapter and in Chapter 7 however, there is a much greater degree of connection between the genes and molecules; and the genes and the outputs: as every gene is connected to at least one of each, and most to two.

Considering now to the gene activation plots in figure 8.2.2, there is a much greater degree of epigenetic activity than in the previously analysed network, see figure 8.1.2. Genes 2, 3, 10 and 12 all display consistent epigenetic behaviour, made more promising by the roles they play not only as parts of the two gene clusters, but as inputs to two of the network's output elements. With a demonstration that the maze foraging environment is a more useful optimisation task for the development of epigenetic behaviours, it is possible to progress again to the introduction of environmental dynamics, in the form of moving prey objects.

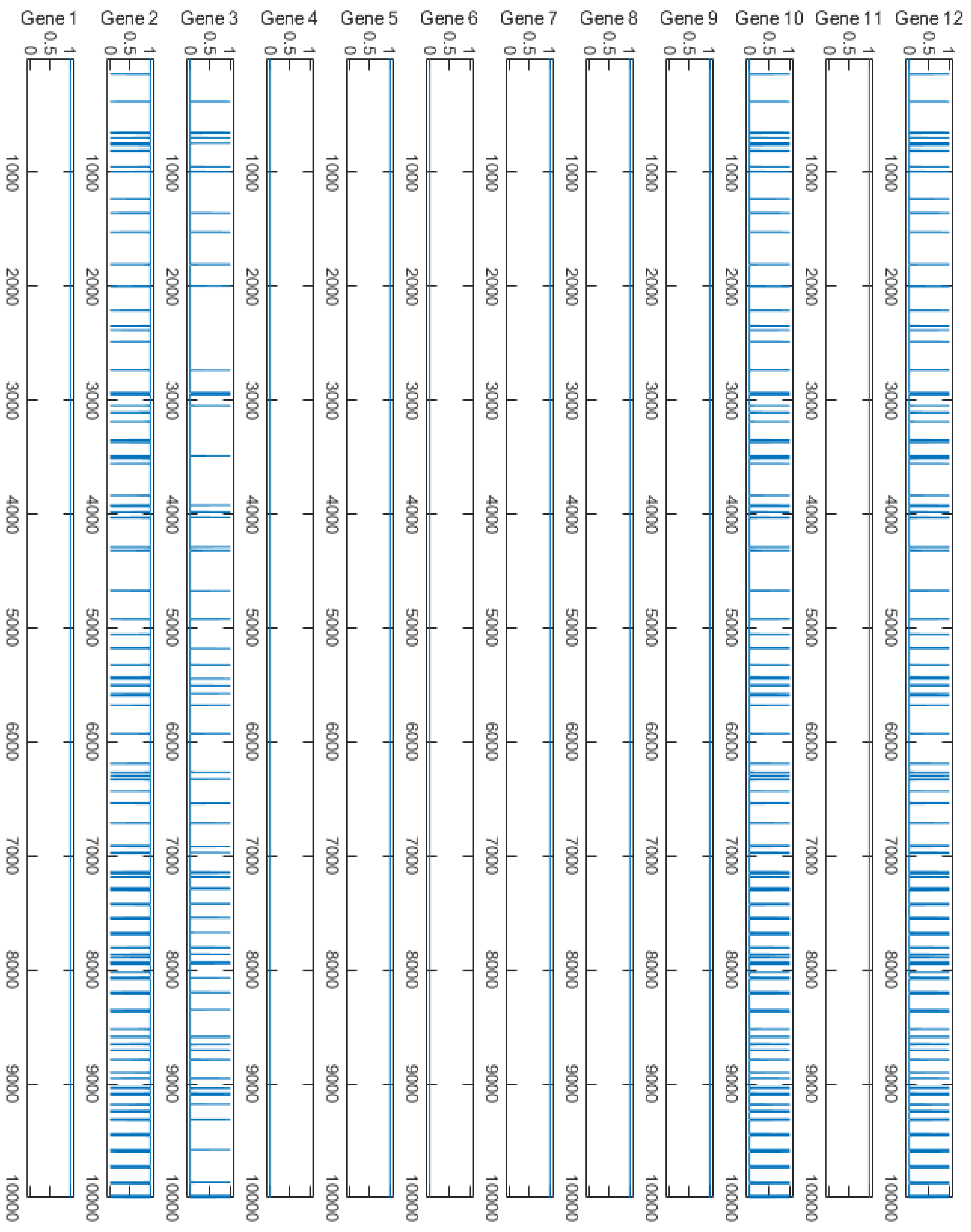


Figure 8.2.3. Activation state of genes within the network between time steps 0 and 10000. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

8.3 Foraging for Mobile Prey

As a maze based foraging task has been shown to be able to induce the evolution of epigenetic behaviour, it is now possible to move on to the first introduction of the environmental dynamics for robots: the mobile prey.

8.3.1 Experiment Design and Parameters

The foraging task will take place in the environment shown in figure 6.3.7 in Chapter 6

Table 8.3.1. Parameters of the environment.

Parameter	**Value**
Food Spawn Boundary Radius	0.1 Units
Nest Radius	0.2 Units
Initial Number of Food	20
Respawning Food	TRUE

Table 8.3.1 contains the parameters for the environment, which are the same as those from the previous experiment in section 8.2.

Table 8.3.2. Parameters for the evolution of epigenetic networks.

Parameter	**Value**
Number of Simulation Time Steps	10000
Population Size	500
Number of Generations	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 8.3.3. Parameters for the evolution of non-epigenetic networks

Parameter	**Value**
Number of Simulation Time Steps	10000
Population Size	500
Minimum Target Fitness	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	16

The parameters of the experiments, shown in tables 8.3.2 and 8.3.3, are the same as those of the previous experiments in this chapter. Once two populations of networks have been evolved, they will be reassessed in the same environment, but with the switch to mobile rather than static prey objects, as described in Chapter 6. As the experiment in Section 8.2 has shown that not every individual produced is actually able to complete the foraging task, only those that have a fitness higher than 1.0 (indicating they have foraged at least one prey object) will be used during the second stage. Similar to the impulse injection experiment from Chapter 7, the goal is to compare how the epigenetic and non-epigenetic networks react when confronted with stimulus that they have not been optimised for.

8.3.2 Results

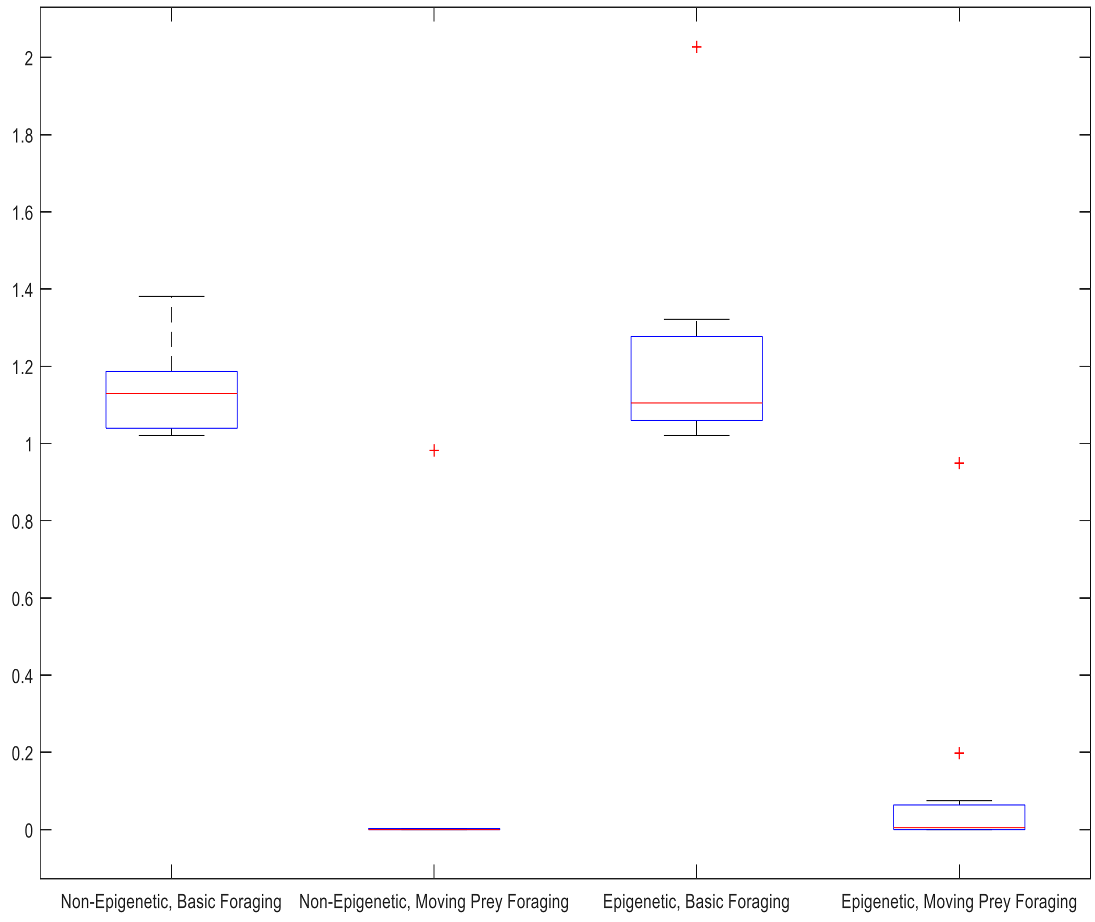


Figure 8.3.1. The results of the evolution of non-epigenetic and epigenetic networks, as well as their fitness values when attempting the foraging task in an environment with moving prey objects.

The results of the epigenetic and non-epigenetic networks are shown in figure 8.3.1, with the important points summarised in table 8.3.4. The results show clearly that none of the networks, both epigenetic and non-epigenetic, are able to return any prey objects to the nest when they are mobile. The best results in both cases are statistical outliers: non-epigenetic network 19, standard fitness 1.200, moving prey fitness 0.982; and epigenetic network 14, standard fitness 1.219, moving prey fitness 0.949. With these results, there is little reason to analyse any of the networks in greater detail.

Table 8.3.4. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks that were able to forage at least one prey object under standard conditions. Note that outliers are only included in the calculation of the mean fitness.

Property	**Non-Epigenetic Networks**		**Epigenetic Networks**	
	Standard Conditions	**Falling Debris**	**Standard Conditions**	**Falling Debris**
Minimum Fitness	1.021	0.000	1.021	0.000
25th Percentile Fitness	1.040	0.000	1.060	0.000
Median Fitness	1.129	0.000	1.105	0.005
75th Percentile Fitness	1.186	0.000	1.277	0.012
Interquartile Range	0.146	0.000	0.217	0.012
Maximum Fitness	1.381	0.000	1.322	0.075
Mean Fitness	1.141	0.141	1.214	0.115

8.4 Summary

Considering all the experiments and results presented in this chapter, the following comments can be made:

After the encouraging start provided by the results detailed in Chapter 7, the transfer to a task very different from ones previously tackled by the epigenetic network has so far provided little to support the hypothesis presented in Chapter 6. While the simulation tools have performed well, the initial failure of the basic foraging task to provide sufficient stimulus to evolve epigenetic behaviour is discouraging.

The switch to the maze environment, and thus the addition of greater complexity to the basic foraging task, did result in epigenetically active networks, but with a poorer fitness. Indeed many individuals were unable to return even a single prey object to the nest, a trend that continued with the moving prey experiments.

Drawing the work of this chapter to a close, the hypothesis laid out in section 6.4 remains unproven. Changing the nature of the foraging task may yield more encouraging results, which leads into Chapter 9, where a slightly different approach is taken in the form of the search and rescue task.

9 Robot Rescue Experiments

Taking a slightly different approach to the experiments of Chapter 8, this chapter covers the series of experiments that simulate a search and rescue scenario using the JBotEvolver platform [77] and simulated robot detailed in Chapter 6 of this thesis. As discussed in Chapter 6, there are two primary reasons for re-contextualising the foraging task into one that mimics a search and rescue scenario. The first is the introduction of time pressure as a factor in determining the fitness of a given network. If two robots both recovered 4 people, the metrics used in Chapter 8 would declare them equal, even if one did so much faster, and therefore more efficiently, than the other. The rescue fitness function, given in equation 6.3.2 in Chapter 6 allows these two robots to be differentiated, which in turn should encourage the evolution of more efficient control networks, and hopefully ones that are more epigenetically active. The second reason relates to the introduction of the environmental dynamics that are of importance to the hypothesis laid out in Section 6.3: the fact that a search and rescue task provides ample inspiration for possible dynamics to introduce.

As in Chapter 8, the first experiment will be a validation of the epigenetic networks ability to perform the task, as well as the establishment of a baseline for latter comparison by repeating the experiment with non-epigenetic networks. The introduction of environmental dynamics is described in Section 9.1, in the form of the simulated debris fall described in Section 6.3.2 for Chapter 6. This section is followed by a description of a series of experiments, results and analysis.

9.1 Basic Rescue Task

9.1.1 Experimental Design and Parameters

The basic rescue task will take place in the environment shown in figure 6.3.8 in Chapter 6.

Table 9.1.1. Parameters of the environment.

Parameter	Value
Nest Radius	0.2 Units
Initial Number of People	5
Respawning People	FALSE

Table 9.1.1 contains the parameters for the environment. The 'Nest Radius'; 'Initial Number of People' and 'Respawning People' parameters all have similar functions as their counterparts in Chapter 8, although it should be noted that: in addition to their only being five people, they will not be respawning when deposited in the nest by the robot. This is to give a more accurate representation of a search and rescue scenario, as in reality you can only find each missing person once.

Table 9.1.2. Parameters for the evolution of epigenetic networks

Parameter	Value
Number of Simulation Time Steps	10000
Population Size	500
Number of Generations	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 9.1.3. Parameters for the evolution of non-epigenetic networks.

Parameter	**Value**
Number of Simulation Time Steps	10000
Population Size	500
Minimum Target Fitness	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	16

The parameters of the experiments, shown in tables 9.1.2 and 9.1.3, are the same as those of the previous experiments from Chapter 8.

9.1.2 Results

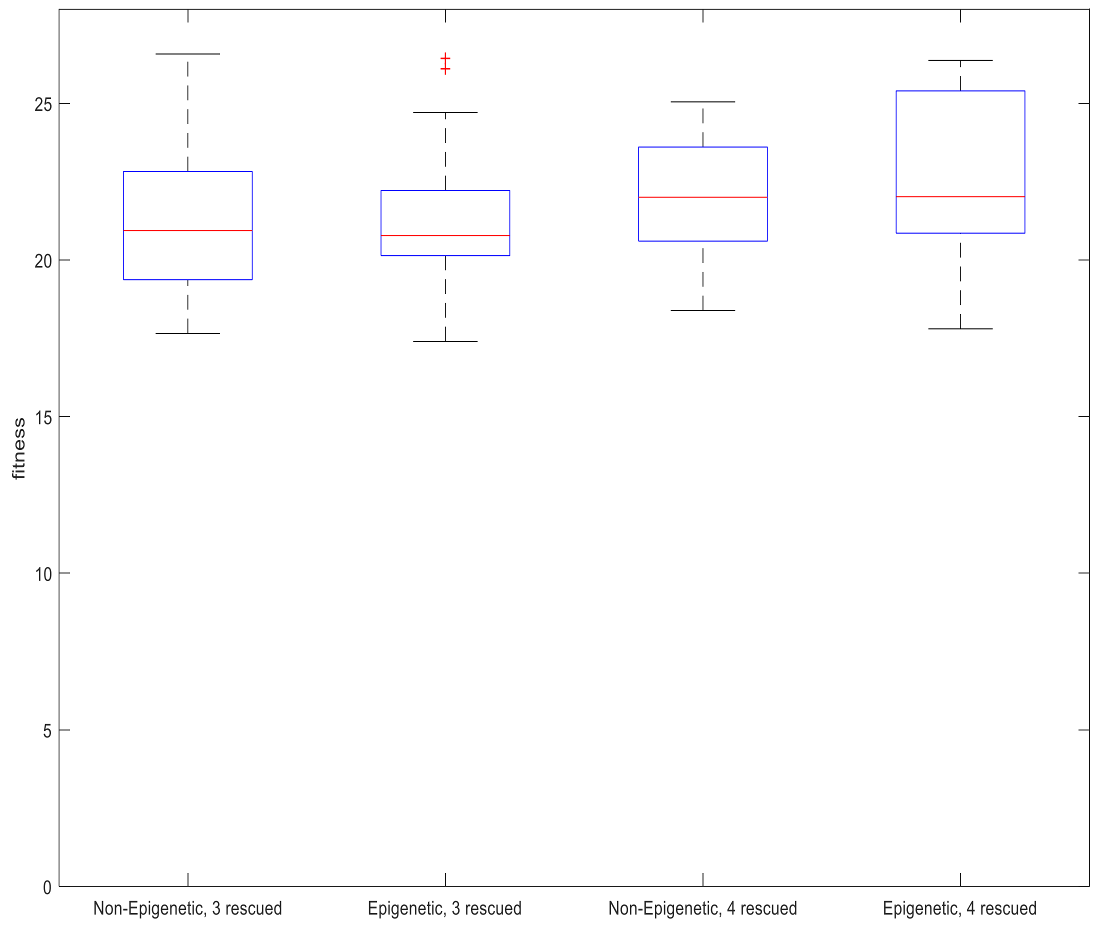


Figure 9.1.1. Fitness of the best individuals from 50 evolutionary runs of epigenetic and non-epigenetic networks. The results are divided based upon the number of people successfully rescued.

Figure 9.1.1 shows the comparison between the results of the epigenetic and non-epigenetic controlled robots, with the important points laid out in table 9.1.4. As has been the case in prior experiments, detailed in Chapter 8, the networks have comparable fitness results. Both the epigenetically and none-epigenetically controlled robots were able to rescue between and 3 and 4 people, with the finesses indicating that they were able to do so at comparable speeds.

Table 9.1.4. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks. Note that outlier values are only included in the calculation of mean fitness.

Property	**Non-Epigenetic Networks**		**Epigenetic Networks**	
	3 People Rescued	**4 People Rescued**	**3 People Rescued**	**4 People Rescued**
Minimum Fitness	17.654	18.386	17.397	17.801
25th Percentile Fitness	19.370	20.601	20.138	20.855
Median Fitness	20.937	22.002	20.777	22.019
75th Percentile Fitness	22.823	23.604	22.217	25.397
Interquartile Range	3.453	3.003	2.079	4.542
Maximum Fitness	26.575	25.047	24.707	26.371
Mean Fitness	21.373	22.024	21.234	22.701

9.1.3 Network Analysis

As the results have already been divided based on the number people rescued, it is useful to look at a sample epigenetic network from both groups. Starting with a network of fitness 20.776, the closest to the median fitness value for the recovery of 3 people, figure 9.1.2 details its structure in the usual fashion, derived from the parameter tables in appendix A, section 11.6.

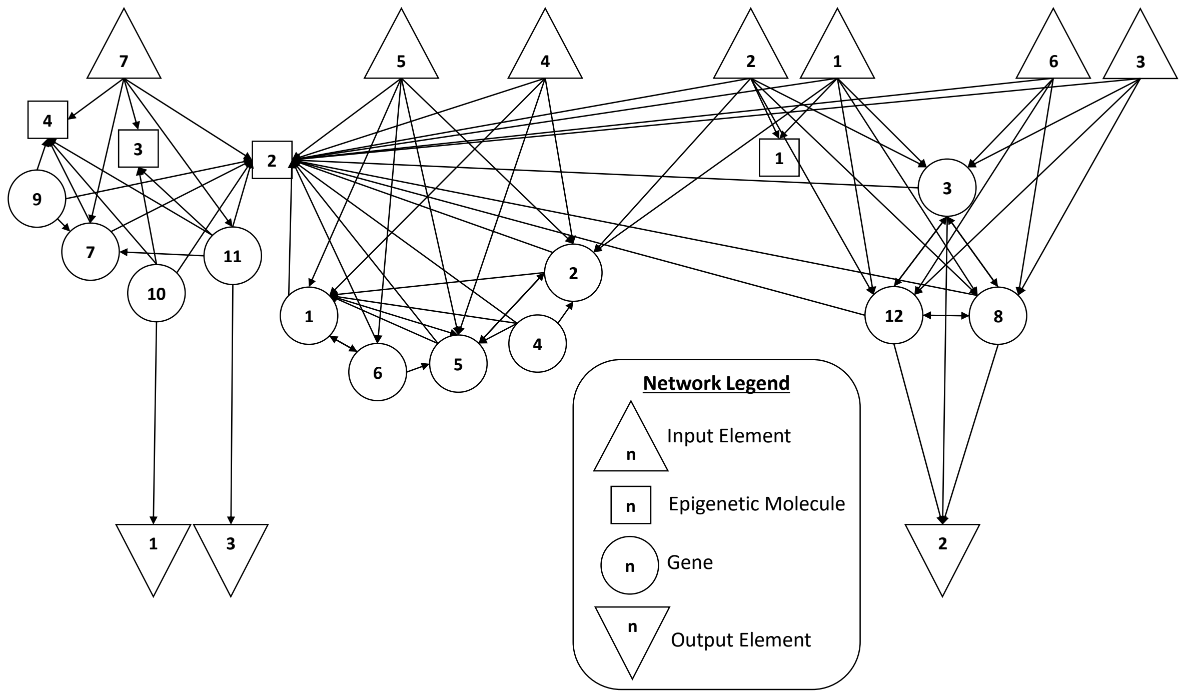


Figure 9.1.2. Illustration of the epigenetic network structure.

From the figure, there are three features of this network that are of note. Firstly, unlike previous networks analysed in this work, there are no clear gene clusters, with most genes being fairly evenly distributed throughout the network space. Second, there is one molecule, molecule 2, which is connected to all the inputs and genes in the network. Finally, gene 10, which is connected to the output that controls the grabber, has no inputs.

However, looking at the gene activation plots in figure 9.1.3 shows that gene 10 is among those that are epigenetically active, along with genes 3, 8 and 11. Furthermore, not only are all of these genes connected to the network outputs, but they are either the only ones, in the case of outputs 1 and 3, or make up the vast majority, in the case of output 2. This clearly indicates that epigenetic behaviour is playing an important role in the functioning of this network.

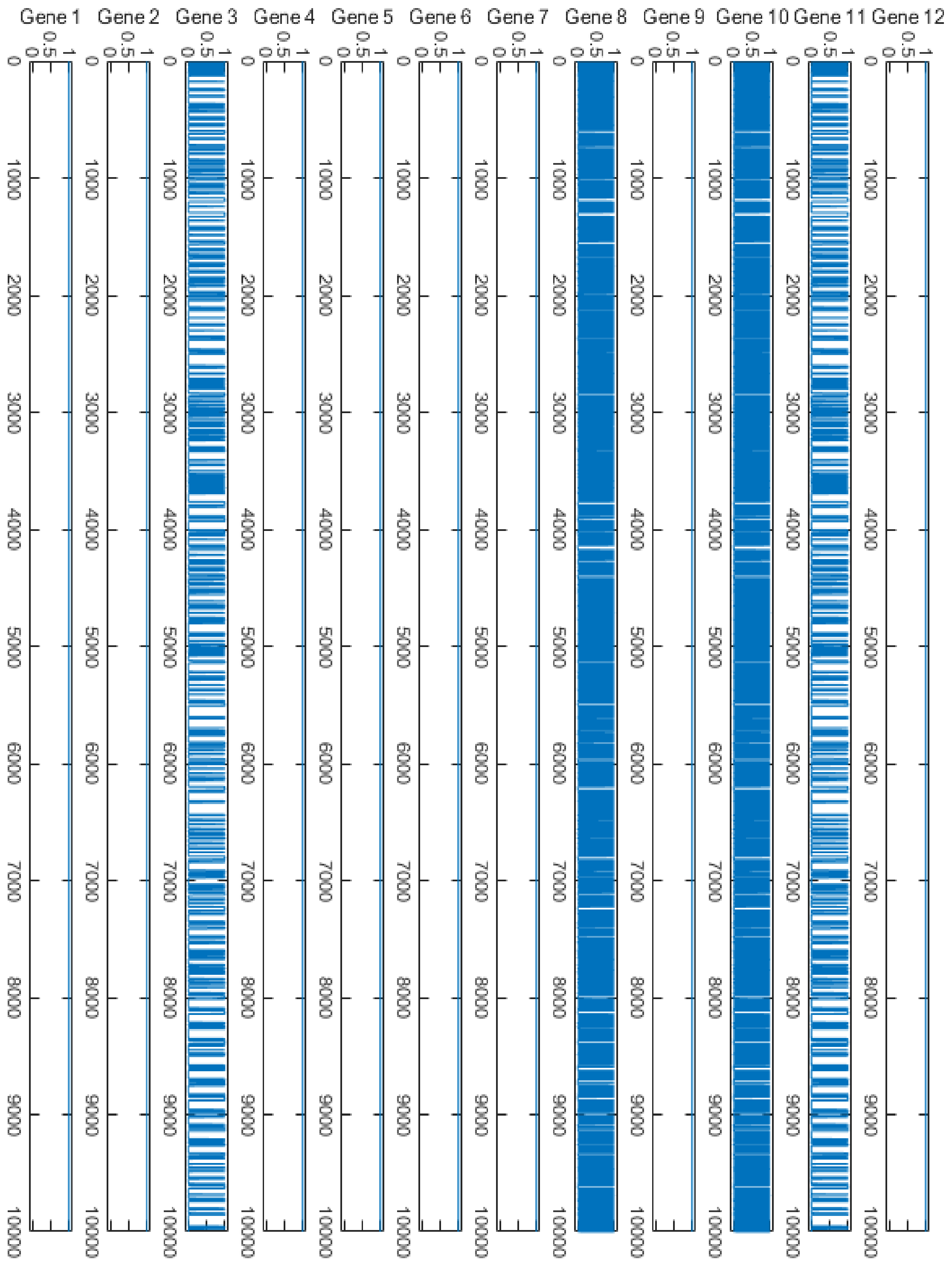


Figure 9.1.3. Activation state of genes within the network between time steps 0 and 10000. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

Moving on to the example network for the recovery of 4 people: fitness 21.475, the closest to the median fitness; the structure of which is illustrated in figure 9.1.4, derived from the parameter tables in appendix A, section 11.7.

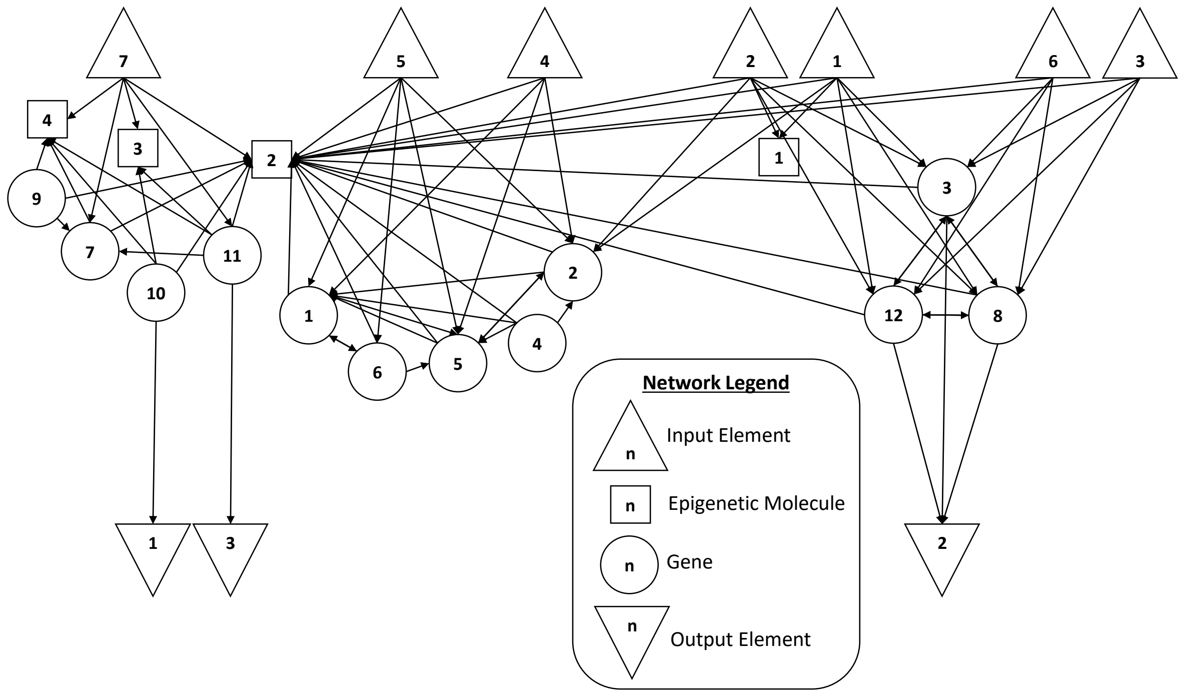


Figure 9.1.4. Illustration of the epigenetic network.

Compared to the network illustrated in figure 9.1.2, the 4 people recovered median network has some noteworthy differences. The gene clustering is present, with densely intra-connected group consisting of: genes 1, 2, 3 and 11; and genes 7, 9, 10 and 11; while the other genes are spread out through the rest of the network space. There is no molecule connected to all the network elements, but gene 7 does connect to all the inputs and all the other genes. The most unusually network feature however is output 2, which connects to the left hand motor, has no inputs. Analysis of the motor model within JBotEvolver provides the following equation for motor speed at time t:

$$\boldsymbol{Speed_t = 2Speed_{max}(-0.5 + Control\,Value)} \qquad \mathbf{(9.1.1)}$$

With the output providing a control value of 0, due to its lack of inputs, equation 9.1.1 indicates that this network was able to control the robot to complete the task while its left motor was permanently in full reverse. It would be unreasonable to attribute the ability to maintain performance to the epigenetic mechanisms of this network, so instead this stands as a testament to the ability of evolutionary and genetic algorithms to continue to produce solutions that surprise human designers. Turning to figure 9.1.5, the gene activation plots, it can be seen that there is epigenetic activity present in this network too; with genes 2, 3, 6 and 10 all displaying switching behaviour; while genes 5 and 7 are permanently deactivated by the epigenetic molecules. Genes 10 and 6 are both inputs to output elements 1 and 3 respectively; while genes 2 and 3 are part of one of the network's intra-connected clusters. It is therefore once again reasonable to deduce that the epigenetic activity plays a role in the performance of this network.

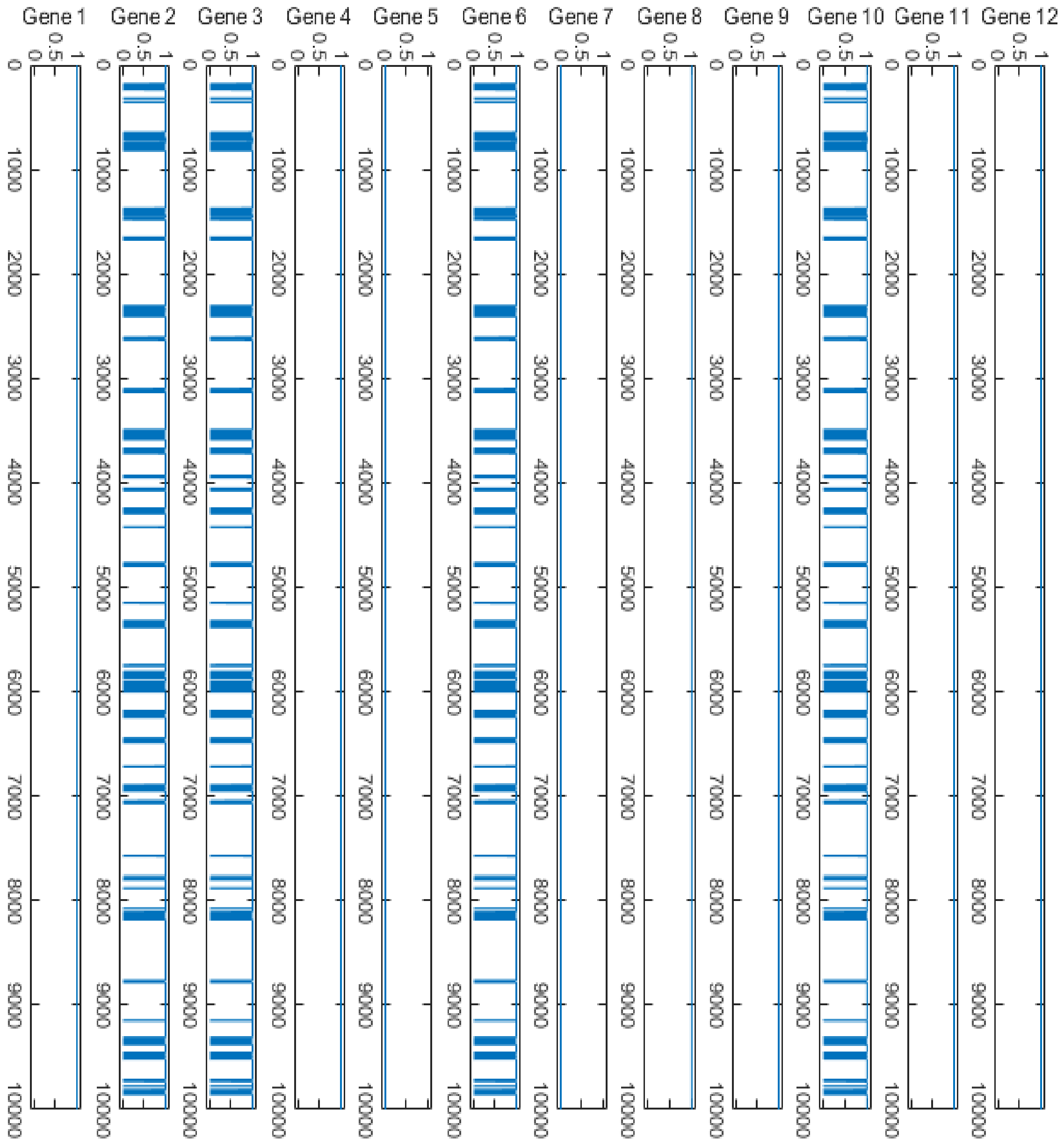


Figure 9.1.5. Activation state of genes within the network between time steps 0 and 10000. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

9.2 Rescue within an Unstable Environment

As the search and rescue task has been shown to be sufficient to induce the evolution of epigenetic behaviour, it is now possible to move on to the introduction of environmental dynamics, in the form of simulated debris falling.

9.2.1 Experimental Design and Parameters

As before, the task will take place in the environment shown in figure 6.3.8 in Chapter 6

Table 9.2.1. Parameters of the environment.

Parameter	Value
Nest Radius	0.2 Units
Initial Number of People	5
Respawning People	FALSE

Table 9.2.1 contains the parameters for the environment, which are the same as those from the previous experiment in section 9.1.

Table 9.2.2. Parameters for the evolution of epigenetic networks.

Parameter	Value
Number of Simulation Time Steps	10000
Population Size	500
Number of Generations	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	12
Number of Epigenetic Molecules	4

Table 9.2.3. Parameters for the evolution of non-epigenetic networks.

Parameter	**Value**
Number of Simulation Time Steps	10000
Population Size	500
Minimum Target Fitness	200
Number of Repeats	50
Mutation Rate	0.05
Data Width	8
Number of Genes	16

The parameters of the experiments, shown in tables 9.2.2 and 9.2.3, are the same as those of the previous experiments in this chapter. Once two populations of networks have been evolved, they will be reassessed in the same environment, but with the addition of the simulated falling debris described in Chapter 6. As with the impulse injection experiment from Chapter 7 and the mobile prey experiment from Chapter 8, the goal is to compare how the epigenetic and non-epigenetic networks react when confronted with stimulus that they have not been optimised for.

9.2.2 Results

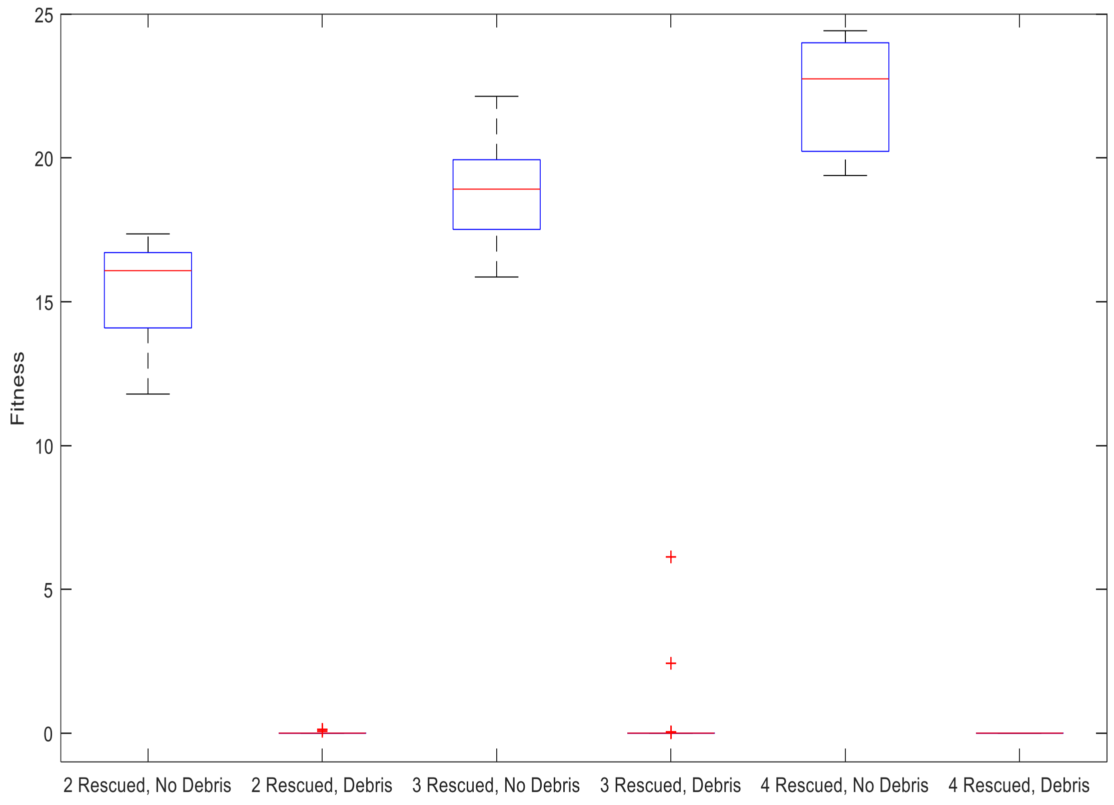


Figure 9.2.1. The results of the evolution of epigenetic networks, as well as their fitness values when attempting search and rescue in an environment with falling debris. Note that the label for the number of people rescued in the debris scenarios relates to their performance in the standard environment.

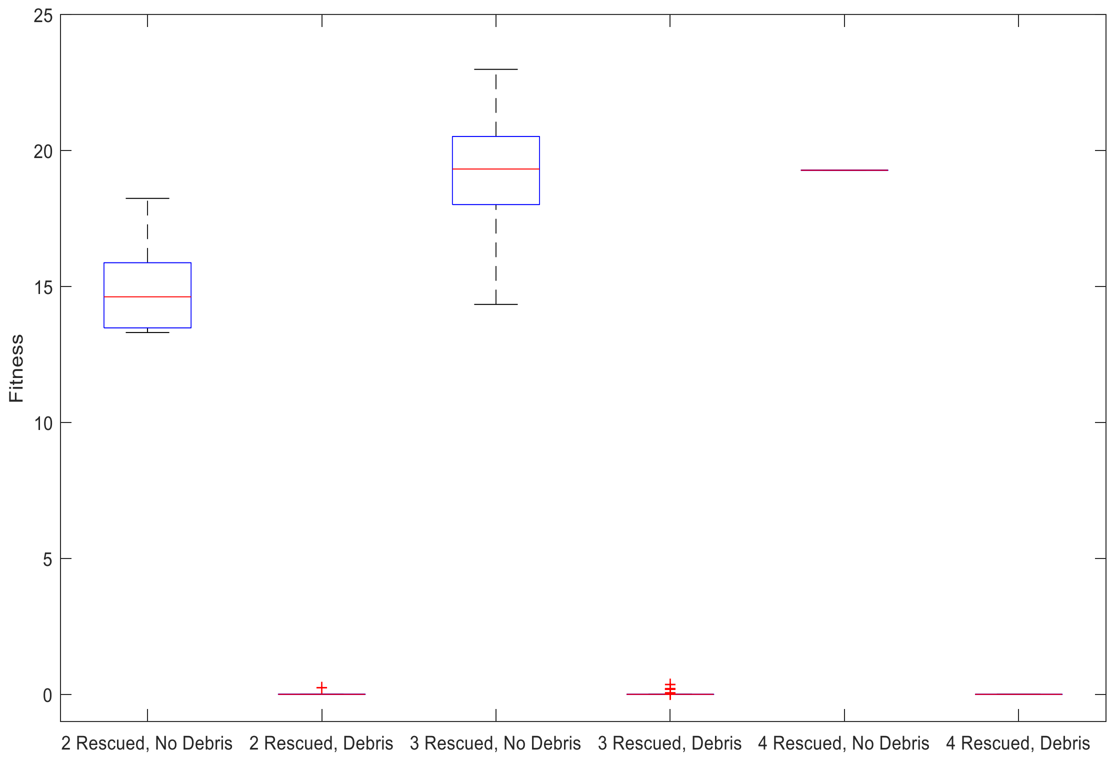


Figure 9.2.2. The results of the evolution of non-epigenetic networks, as well as their fitness values when attempting search and rescue in an environment with falling debris. Note that the label for the number of people rescued in the debris scenarios relates to their performance in the standard environment.

The results of the epigenetic and non-epigenetic networks are shown in figures 9.2.1 and 9.2.2 respectively, with the important points of equivalent groups summarised in tables 9.2.4 through 9.2.6. The results show clearly that the majority of networks, both epigenetic and non-epigenetic, fail to rescue or even locate any people when inserted into the debris environment. There is one point to mention however: there are two statistical outliers in the epigenetic network population from runs 22 and 32. Under normal conditions, both networks rescued 3 people from the environment, with fitnesses of 18.598 and 19.245 respectively. When tested in the debris environment, they were both able to rescue 1 person, with fitnesses of 6.128 and 2.429 respectively. While they are both outliers, and as such cannot be held up as confirmation of the hypothesis laid out in Section 6.4; they do indicate the potential predicted to be present within epigenetically active networks does exist. As such, further analysis of one of these networks should yield useful insights.

Table 9.2.4. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks that rescued 2 people under standard conditions. Note that outlier values are only included in the calculation of mean fitness.

Property	**Non-Epigenetic Networks**		**Epigenetic Networks**	
	Standard Conditions	**Falling Debris**	**Standard Conditions**	**Falling Debris**
Minimum Fitness	13.304	0.000	11.791	0.000
25th Percentile Fitness	13.476	0.000	14.091	0.000
Median Fitness	14.618	0.000	16.082	0.000
75th Percentile Fitness	15.871	0.000	16.711	0.000
Interquartile Range	2.395	0.000	2.620	0.000
Maximum Fitness	18.237	0.000	17.360	0.000
Mean Fitness	14.934	0.031	15.434	0.022

Table 9.2.5. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks that rescued 3 people under standard conditions. Note that outlier values are only included in the calculation of mean fitness.

Property	**Non-Epigenetic Networks**		**Epigenetic Networks**	
	Standard Conditions	**Falling Debris**	**Standard Conditions**	**Falling Debris**
Minimum Fitness	14.338	0.000	15.861	0.000
25th Percentile Fitness	18.009	0.000	17.516	0.000
Median Fitness	19.317	0.000	18.916	0.000
75th Percentile Fitness	20.514	0.000	19.940	0.000
Interquartile Range	2.505	0.000	2.424	0.000
Maximum Fitness	22.982	0.000	22.144	0.000
Mean Fitness	19.205	0.025	18.894	0.271

Table 9.2.6. Maximum, Median, mean and Minimum fitness, as well as 25th and 75th Percentiles and interquartile range for epigenetic and non-epigenetic networks that rescued 4 people under standard conditions. Note that outlier values are only included in the calculation of mean fitness.

Property	**Non-Epigenetic Networks**		**Epigenetic Networks**	
	Standard Conditions	**Falling Debris**	**Standard Conditions**	**Falling Debris**
Minimum Fitness	19.272	0.000	19.390	0.000
25th Percentile Fitness	19.272	0.000	20.230	0.000
Median Fitness	19.272	0.000	22.751	0.000
75th Percentile Fitness	19.272	0.000	24.006	0.000
Interquartile Range	0.000	0.000	3.776	0.000
Maximum Fitness	19.272	0.000	24.424	0.000
Mean Fitness	19.272	0.000	22.188	0.000

9.2.3 Network Analysis

The outlier chosen for further analysis is network 22, standard fitness 18.598 (3 people), debris fitness 6.128 (1 person). Figure 9.2.3 illistrates the network's structure, derived from the parameter tables in appendix A, section 11.8.

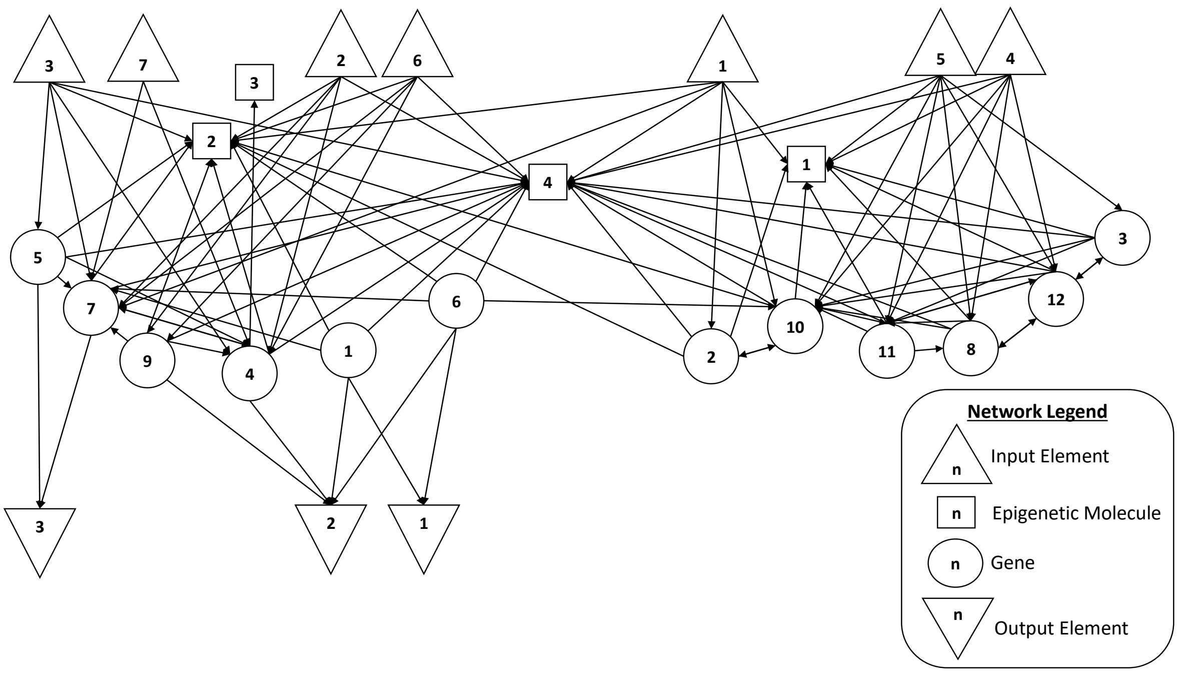


Figure 9.2.3. Illustration of network structure.

The structure of this network is possibly one of the more unsual encountered thus far in this work. Compared to the previous researched networks in this chapter and Chapter 8, as well as the pendulum networks in Chapter 7, several features are of note. Every input is connected to at least twos genes and molecules; there are multiple molecules connected to significant sections of the network; there is vertually no gene clustering; and every output has multiple inputs. The next series of plots illustrate the analysis of the network to identify the cause of the altered behaviour. Figure 9.2.4 and 9.2.5 show which genes have their activation states altered by epigenetic activity, both under normal conditions (9.2.4) and during the simulated debris run (9.2.5). Following that, figures 9.2.6 and 9.2.7 show the states of these genes in comparison to the activity of the epigenetic molecules, illustrating significantly altered behaviour during the debris fall run (9.2.7). The two figures after that, 9.2.8 and 9.2.9 show which of the various parameters the robot is sensing are having an effect on the activity of the epigenetic molecules, while figures 9.2.10 and 9.2.11 connect these effects to the sensors themselves by looking the inputs of the molecules.

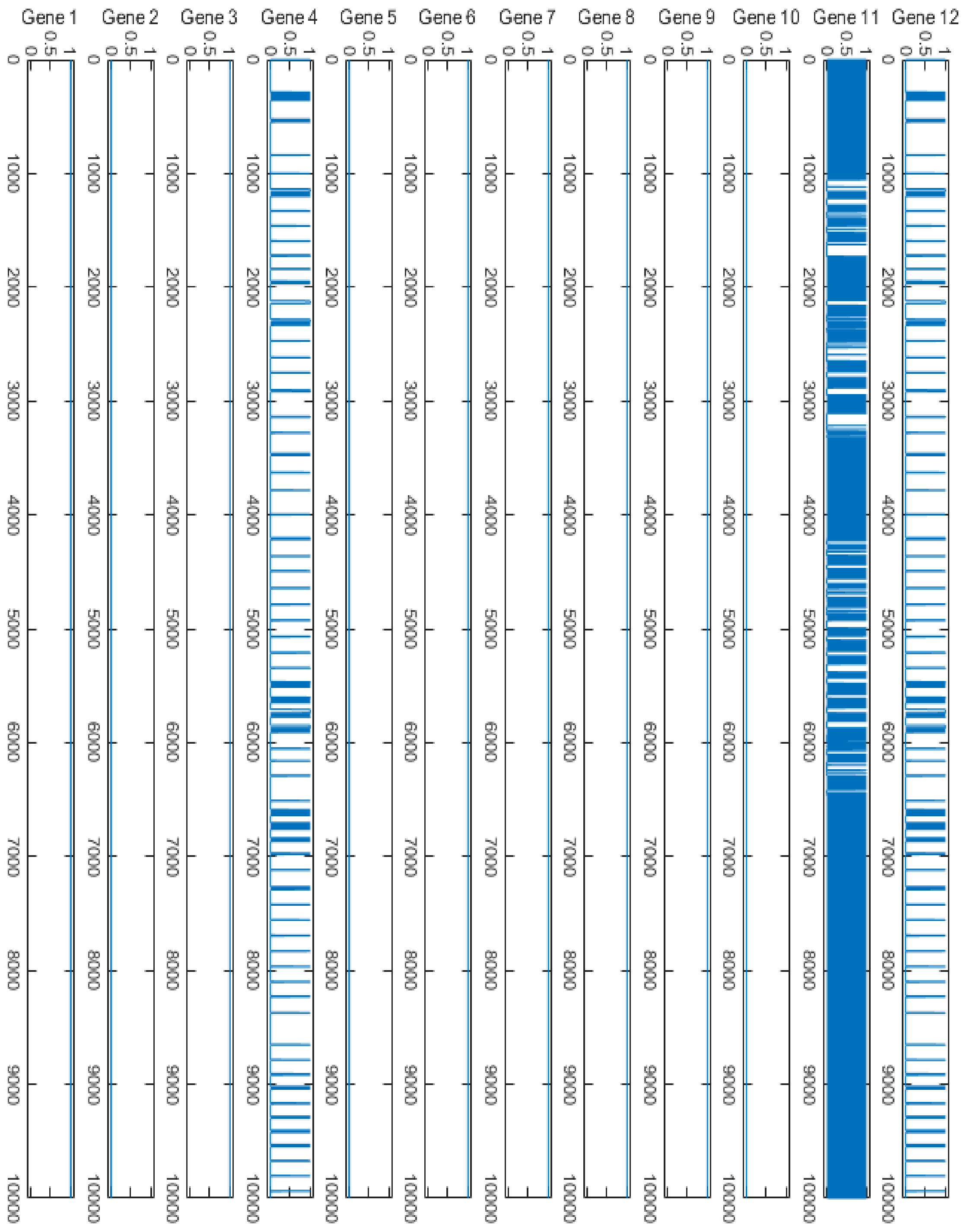


Figure 9.2.4. Activation state of genes within the network between time steps 0 and 10000 during a resuce task within a normal environment. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

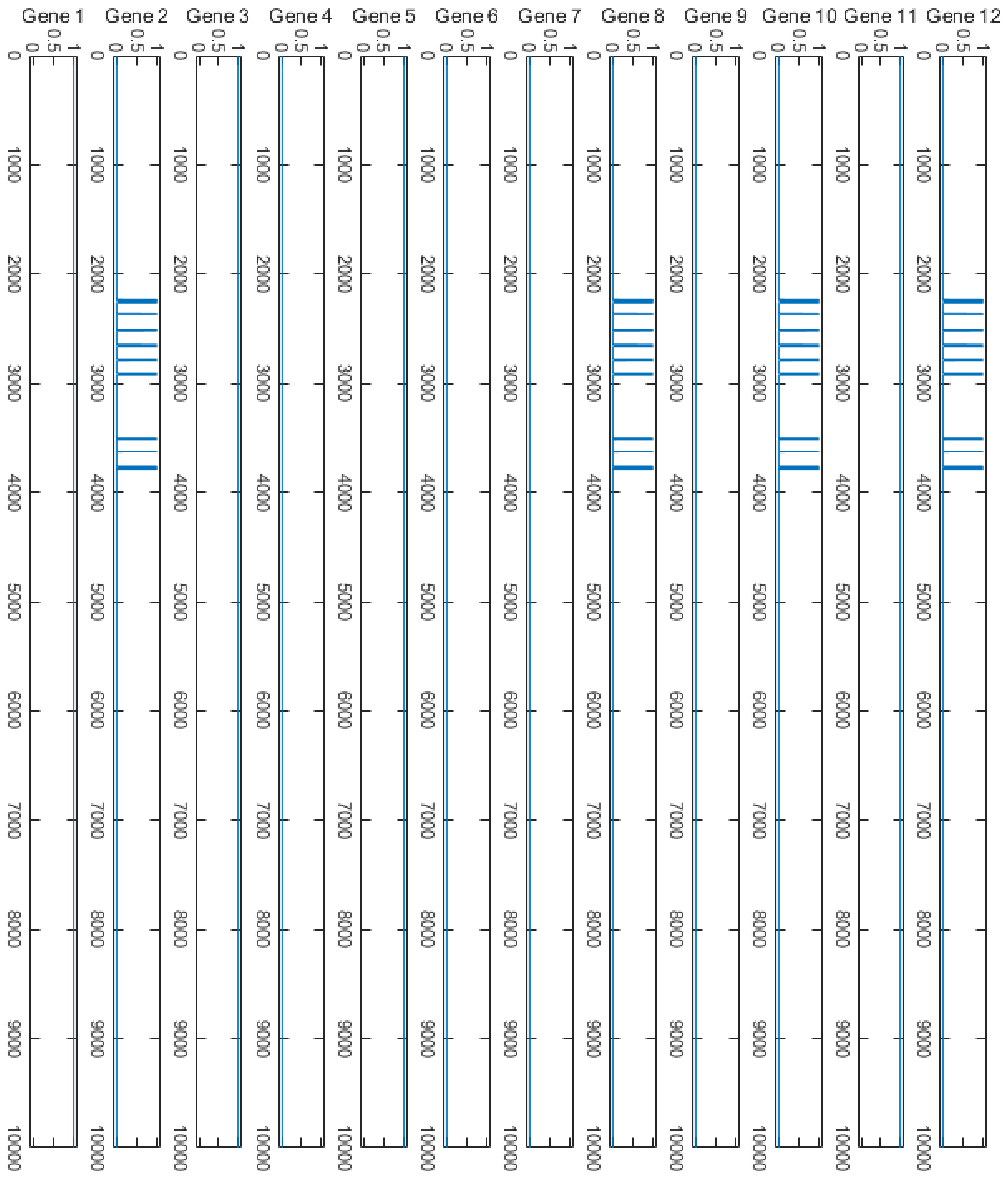


Figure 9.2.5. Activation state of genes within the network between time steps 0 and 10000 during a resuce task within an environment with falling debris. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output.

The unusual nature of the network continues with its epigenetic behaviour. During normal conditions, figure 9.2.4, genes 4, 11 and 12 are epigenetically active; with genes 2, 5 and 10 being permanently deactivated by the epigenetic elements. However, during the rescue in the debris environment, genes 2, 8, 10 and 12 are the epigenetically active ones; while genes 4, 6, 7 and 9 become the deactivated genes. It is important to restate at this time, that the behaviour exhibited in the debris environment was not evolved, it is simply the outcome of a network, optimised for a less complex task being confronted with a change in its environment.

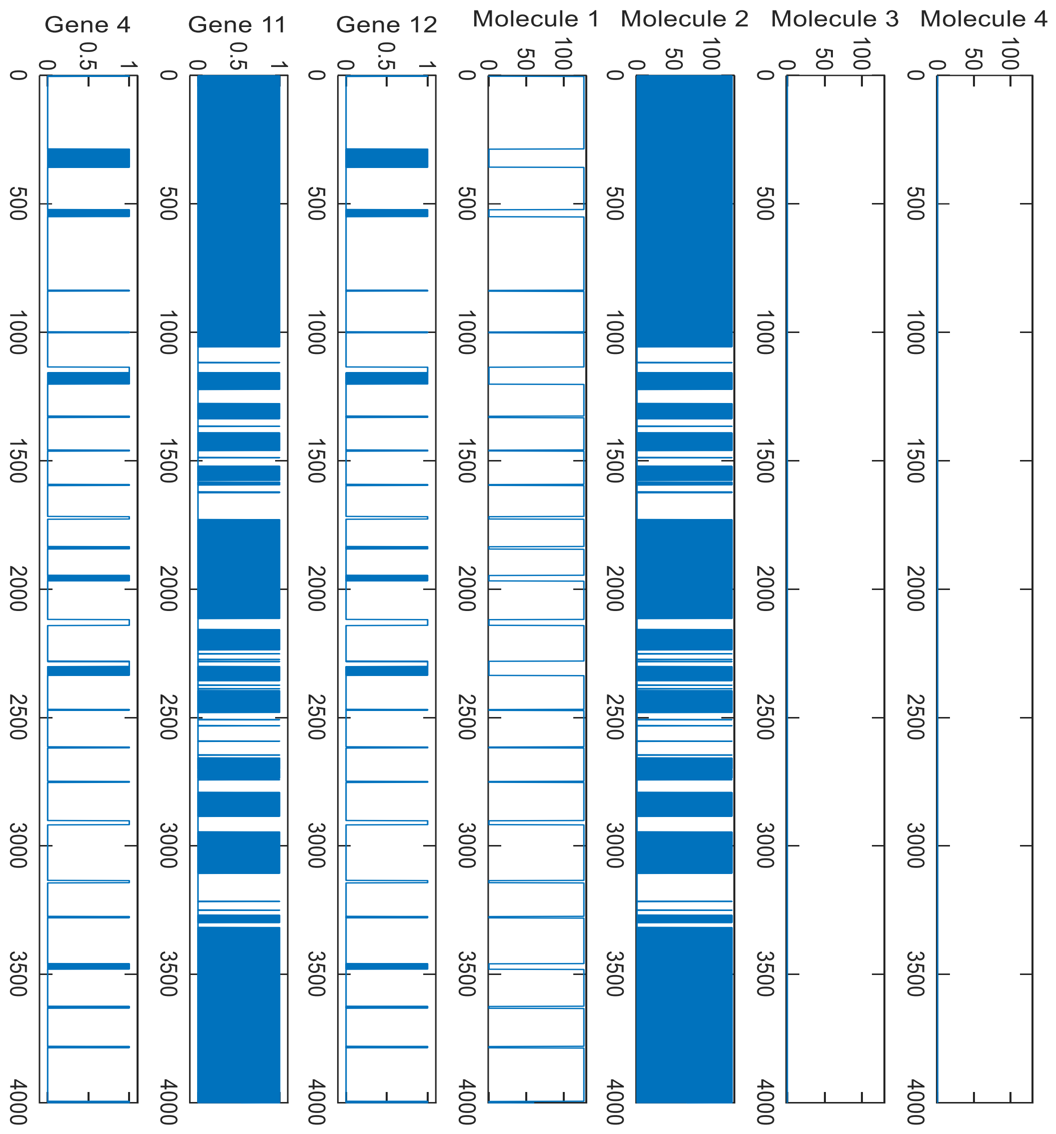


Figure 9.2.6. Activation state of epigeneticaly active genes, and the outputs of the epigenetic molcules during a resuce task within a standard environment. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity.

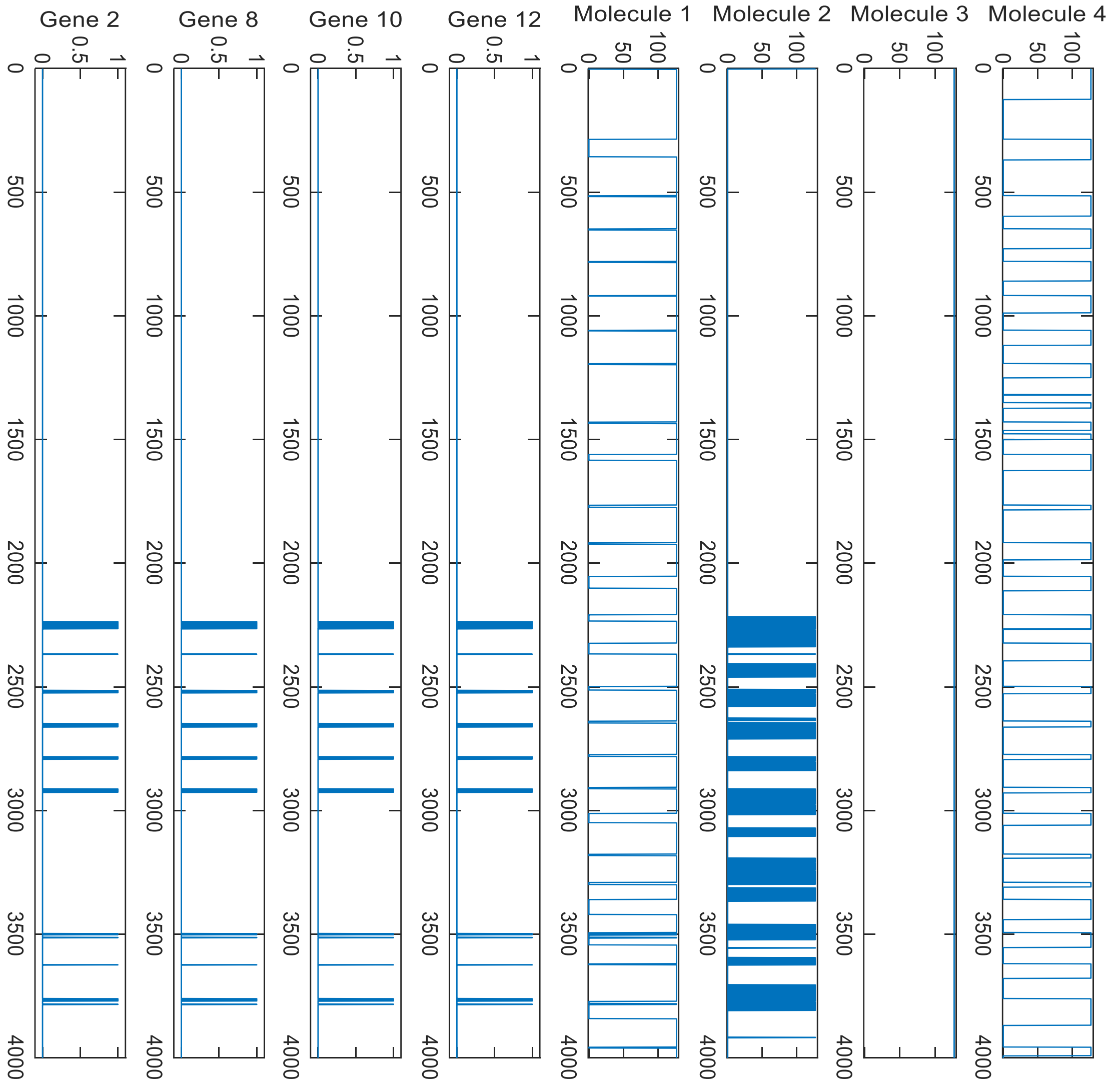


Figure 9.2.7. Activation state of epigeneticaly active genes, and the outputs of the epigenetic molcules during a resuce task within an environment with falling debris. A value of 0 indicates the gene is being deactivated by an epigenetic element, while a value of 1 indicates the gene is active and producing an output. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity.

Figures 9.2.6 and 9.2.7 show the epigenetically active genes in each instance, alongside the molecules responsible for their behaviour. This sheds some light on the changes caused by the change in the environment. In both cases, the majority of epigenetically active genes appear to be regulated by molecule 2, modulated by molecule 1. Gene 11 is directly regulated by molecule 2, but only under normal conditions. Molecule 1's behaviour is unchanged by the environmental alteration, while Molecule 2's changes significantly. In addition, molecule 4 starts producing a varying output in the debris environment, while molecule 3 switches from being inactive to active.

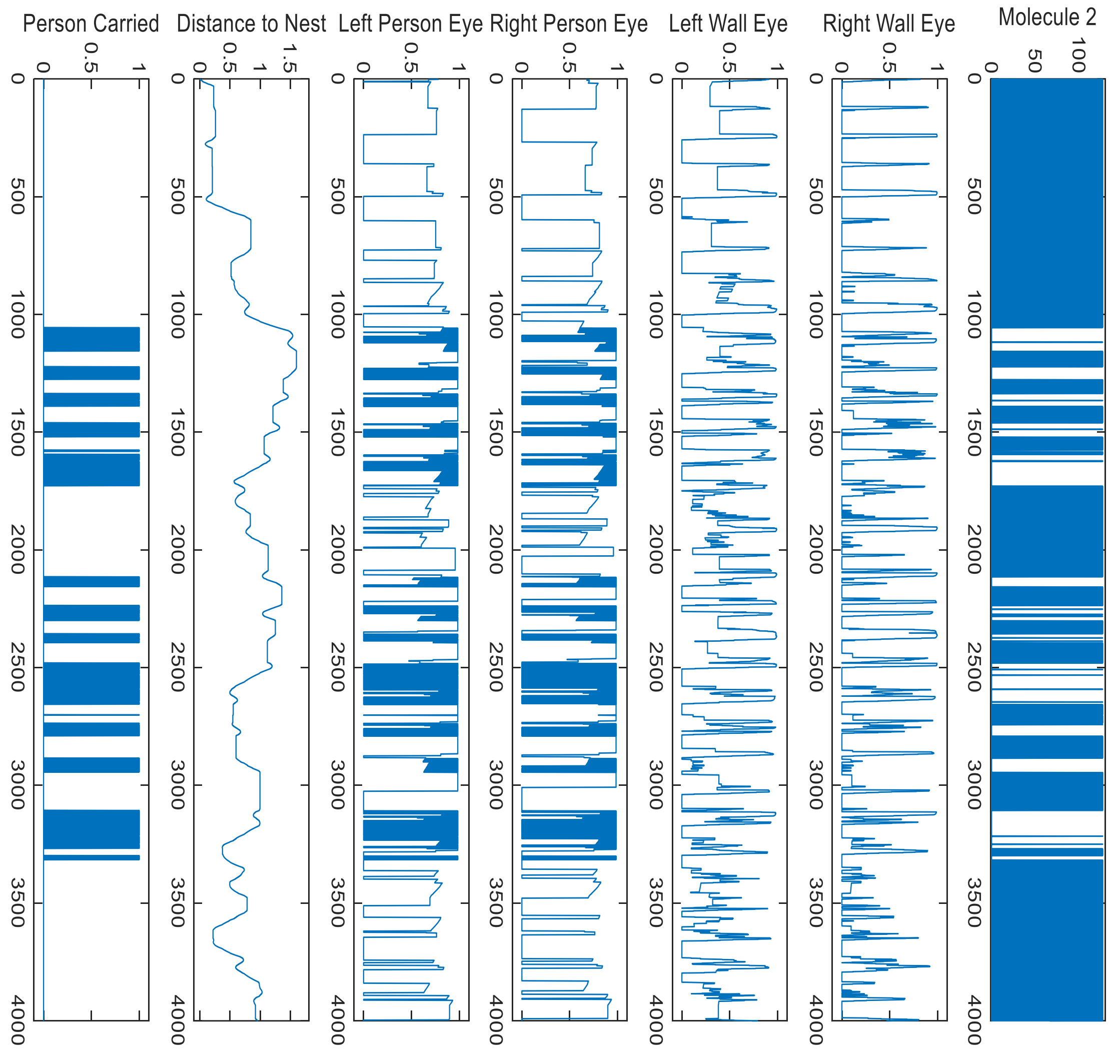


Figure 9.2.8. Output of molcule 2, and readings of various sensors during a resuce task within a standard environment. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity; values of the eye sensors relate to distance to the closest object of the relevant type, scaled between 0.0 and 1.0; distance to the nest is a measure of the strait line distance between the robot and the nest; and person carried is 1 when a person is held by the robot's manipulator.

Figure 9.2.8 shows the output of molecule 2 in relation to the values of the robots various sensors, discussed in greater detail in Chapter 6, during the standard rescue task. Considering these together, it appears that the "gaps" in molecule 2's outputs are strongly correlated to periods where the robot is carrying a person with its griper. This is also the case in figure 9.2.9, which shows the same values during the search of the debris environment. This however does not account for the initial period of inactivity on molecule 2's part in this second instance.

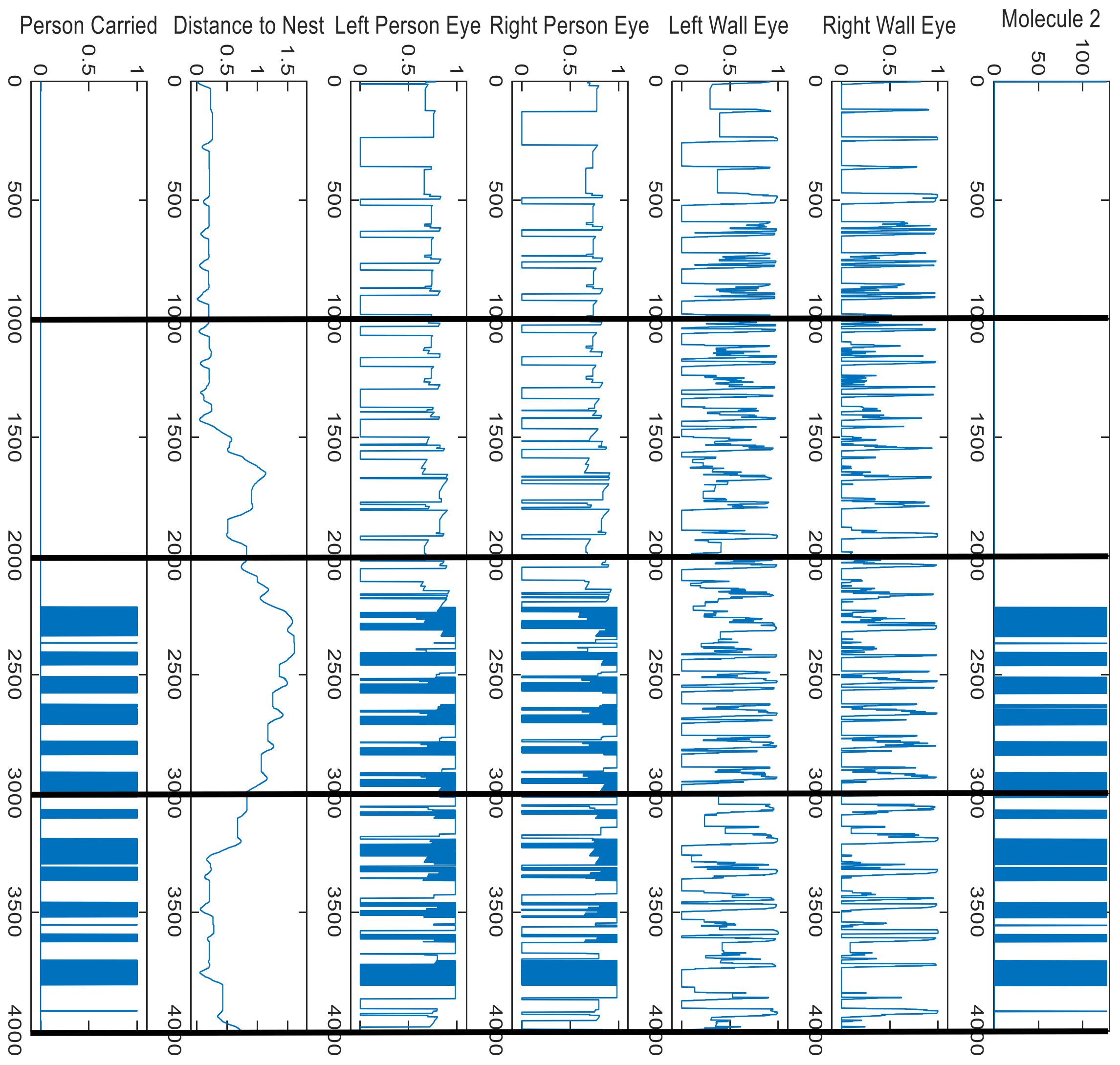


Figure 9.2.9. Output of molcule 2, and readings of various sensors during a resuce task within an environment with debris. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity; values of the eye sensors relate to distance to the closest object of the relevant type, scaled between 0.0 and 1.0; distance to the nest is a measure of the strait line distance between the robot and the nest; and person carried is 1 when a person is held by the robot's manipulator. The black lines mark the time steps at which a random piece of debris is spawned into the environment.

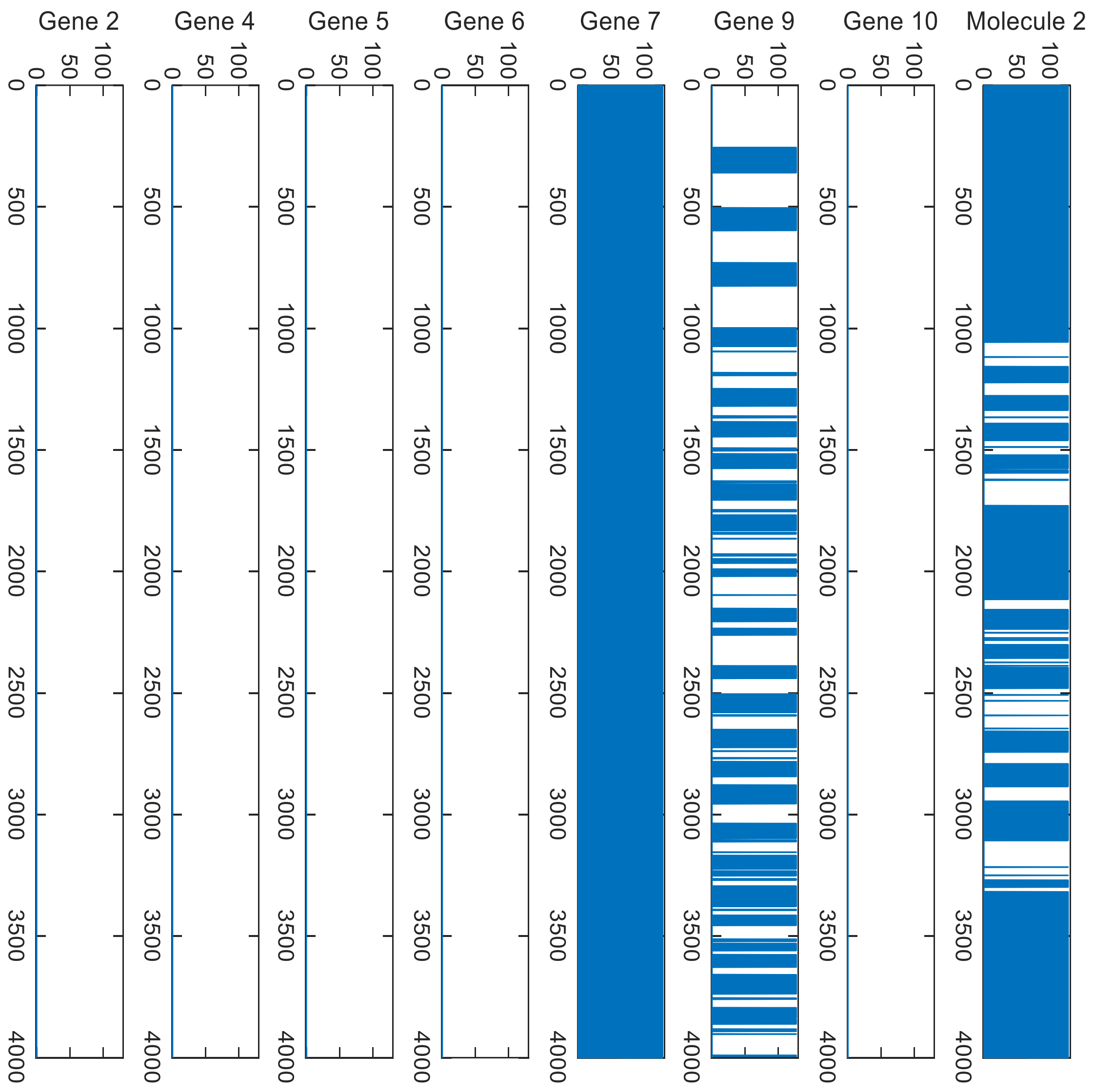


Figure 9.2.10. Output of molecule 2, and the outputs of several genes during a rescue task within a standard environment. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity.

Figures 9.2.10 and 9.2.11 show the outputs of genes 2, 4, 5, 6, 7, 9 and 10, alongside the output of molecule 2. Recall from figures 9.2.3 and 9.2.4 that these are the genes whose activation states switch between permanently off and epigenetically active depending on the environment. As figure 9.2.9 is for the normal environment, genes 2, 5 and 10 are permanently off, and as such their outputs are zero. While active, genes 4 and 6 are also producing an output of zero. Genes 7 and 9 are producing outputs, and taking this together with figure 9.2.7 it looks like molecule 2's output results from the output of gene 7, modulated by either the person held sensor or the person eye sensors.

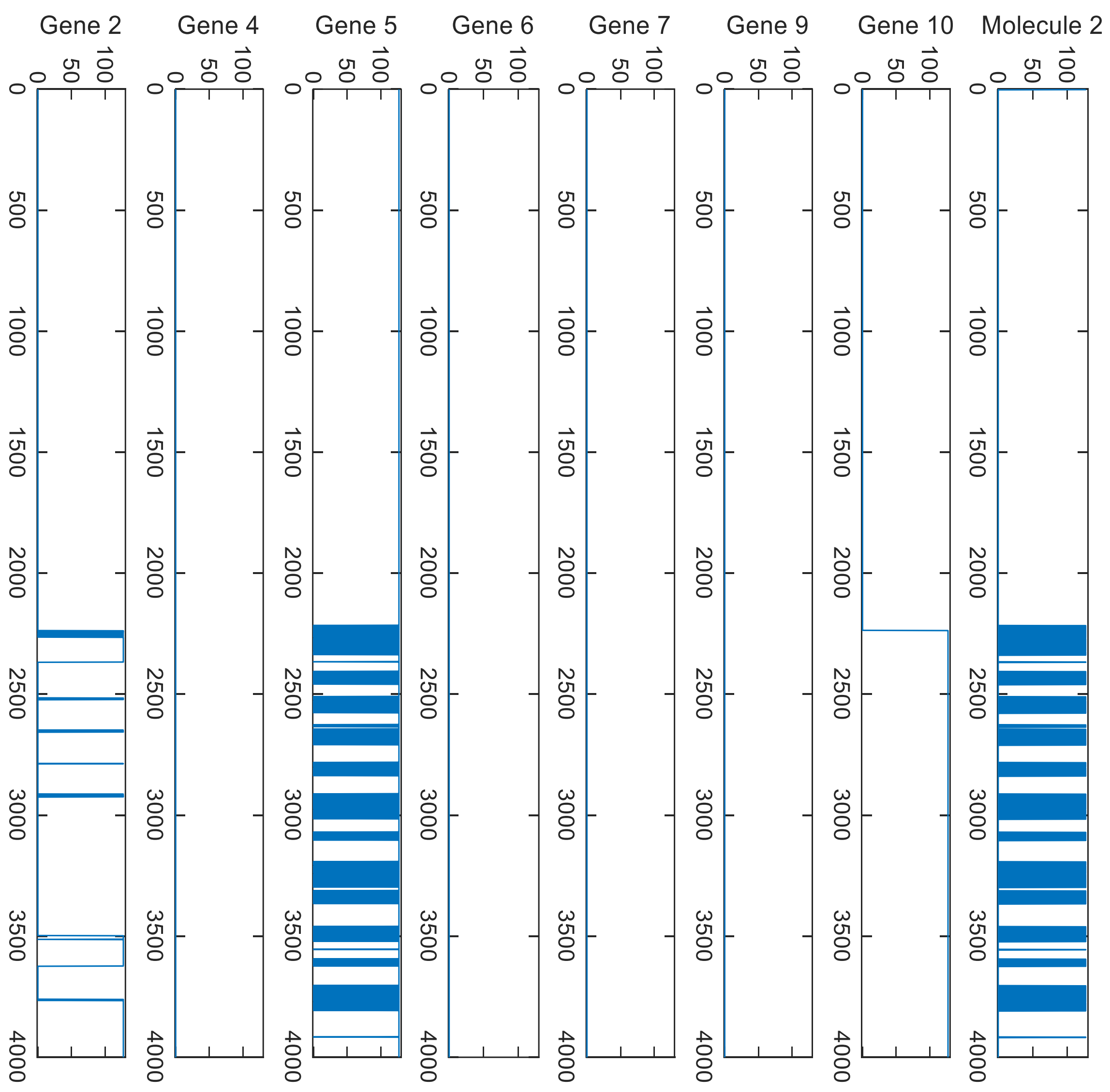


Figure 9.2.11. Output of molecule 2, and the outputs of several genes during a rescue task within an environment with debris. The output of the epigenetic molecules correlates to the location in the network space where it is supressing gene activity.

Figure 9.2.10 shows the same readings, but taken from the network during the rescue in the debris environment. Genes 4, 6, 7 and 9 are now permanently inactive, while genes 2, 5 and 10 are now active. It can immediately be seen that gene 5 now directs the outputs of molecule 2. Gene 5 has only input, in the form of the right hand person eye, which explains why its output correlates strongly to the sensor plots from figures 9.2.7 and 9.2.8. This switch in molecule inputs explains the change in network behaviour, demonstrating the capacity of the epigenetic elements to fundamentally alter the network.

9.3 Summary

Considering all the experiments and results presented in this chapter, the following comments can be made:

The shift to the search and rescue type task was a useful one, as the results in section 9.1 show that it was much more suitable for triggering the evolution of epigenetic behaviour, as well as producing networks of a much better calibre than either of the two foraging tasks in Chapter 8, due to the greater number of target objects they were able to recover (prey or people). Of the three changes that the new task encapsulated (reduced number of target objects; non-respawning target objects; and the inclusion of speed of return in the fitness function), it is most likely that the later made the most significant contribution to the performance improvement by allowing a greater degree of differentiation between networks that the less complex fitness function of the foraging task would have declared equal.

However, this improvement in network optimisation did not translate into an improvement in the network's ability to respond to the un-optimised for environmental dynamics, introduced by the addition of the falling debris. Although the analysis of network 22 did once again suggest that it is possible for an epigenetic network to display the kind of behaviour desired by this work, a single statistical outlier is not a foundation upon which to claim success.

Drawing the work of this chapter to a close, the hypothesis laid out in section 6.4 remains unproven. Further remarks upon this, as well as what other knowledge can be drawn from this completed thesis are found in Chapter 10, the conclusion of this work.

10 Conclusions and Future Work

10.1 Work Conducted

In bringing this thesis to a close, it is important to revisit the work conducted during the processes of its completion. Starting in Chapter 4, with the creation of this work's implementation of a software based epigenetic network. More than just a replication of previous work, the new epigenetic network included a number of refinements, such as the introduction of dedicated input/output elements and the removal of the gene to protein translation process. In addition, the switch from the unusual weight application method to a more conventional design also improved the compactness of the networks, something demonstrated in Section 7.3 of Chapter 7. In preparation for the eventual translation from software to hardware, Chapter 4 also discussed the switch from floating-point maths to integer, including the replacement of the sigmoid function with a look up table; and established the methods for altering the bit width of the network.

These developments came into play in Chapter 5, where the core of this work was described: the creation of the hardware epigenetic network architecture. Through the use of generic, paramatrisable units, the hardware network can have its bit width easily adjusted to allow experimentation with reduced precision; as well as permitting alteration of the number of input/output units, genes and molecules within the network. These features came into play in Chapter 7, where the experiments involving the inverted pendulum were detailed. In Section 7.1, it was shown that: the transition from floating-point to integer maths did not adversely affect the performance of the epigenetic network; that it is possible to dramatically reduce the bit width of an epigenetic network and still maintain its performance; and that not only did the hardware transition result in a reduction in resource utilisation, it also provided an improvement to certain performance metrics, most significantly execution speed, thanks to its parallelisation of the network elements.

Section 7.2 was the beginning of work on the other important through line of this thesis: the hypothesis regarding the ability of epigenetic networks to react to un-optimised for stimulus in such a way as to maintain their performance. The first test of this was the experiment where epigenetic, and non-epigenetic, networks were optimised to control the inverted pendulum, and were then tested to see how they maintained that control when the pendulum was subject to an external impulse, a.k.a. a kick. The results of this experiment provided support for this hypothesis, as the epigenetic networks were, statistically, able to maintain the pendulum in the upright position better than the non-epigenetic ones. In the case of the networks that were analysed in greater detail, the performance of the two epigenetic networks could be clearly linked to the actions of their epigenetic elements; while the non-epigenetic network actually performed so poorly that it violated one of the conditions of the task.

This strong start did not carry over into Chapter 8 however, which marked to transition from the dynamic control tasks of previous work, to the application epigenetic networks to the field of robotic control, specifically foraging robots. Section 8.1 demonstrated that, even when a network contains epigenetic elements, those elements will not necessarily be brought to bear, as efforts to optimise the networks for the basic foraging task showed. Section 8.2 proved to be more promising, as the introduction of greater complexity via a maze environment returned epigenetic behaviour to the networks. These two sections together indicate that it is not enough to include the mechanisms for epigenetic behaviour within a network, it is also important that the task itself is sufficiently complex to necessitate their utilisation, or that some other selection pressure is applied. With basic performance achieved, the application of environmental dynamics, in this case taking the form as the foraging targets being made mobile, came next. The results of this were disappointing, as all the epigenetic, and non-epigenetic, networks were unable to catch any of these new, ambulatory prey.

Chapter 9 brought with a different approach to the problem, as the foraging task was reconceptualised as a search and rescue task, which included the alteration of the fitness metric to incorporate a time pressure. Section 9.1 showed that this not only induced the evolution of epigenetic behaviour, but produced superior results overall, suggesting that this version of the task was much more suitable for the epigenetic networks. Unfortunately this performance improvement and increased epigenetic activity did not result in the same level of un-optimised stimulus response as in Chapter 7, with the introduction of environmental dynamics, in the form of simulated debris fall, resulted in almost all the epigenetic, and non-epigenetic, networks failing to recover even one person. There were, however, two statistical outliers that did, and closer analysis indicates that their epigenetic behaviour was the reason for this.

10.2 Objectives and Hypothesis Revisited

With this summation of work in mind, a re-visitation of the objectives and hypothesis laid out in Chapter 1 is in order.

10.2.1 Hardware Implementation of the Artificial Epigenetic Network

This goal was most definitely achieved successfully, with the hardware networks utilised throughout this work performing comparably to their software counterparts. Furthermore, they were able to achieve superior performance when it came to the areas of resource utilisation and execution speed, something that can be attributed to the parallelisation provided by the hardware; the improvements made to the architecture that allowed for greater network compactness; as well as the transition to reduced precision integer mathematics. These facts also answer the first three questions posed in Section 1.3:

a) Is it possible to switch the underlying logic of the artificial epigenetic network from computationally intensive floating-point maths to more simplistic integer mathematics? **Yes.**

b) Is it then possible to reduce the precision of the artificial epigenetic network's mathematics while maintain its unique functionality? **Yes.**

c) Is it possible to bring the architecture of an epigenetic network more in line with design conventions, without loss of its unique abilities? **Yes.**

10.2.2 Application of Artificial Epigenetic Networks to Robotic Control

The fourth question from Section 1.3: "does the artificial epigenetic network bring any significant benefits to the area of robotic control?". While Chapters 8 and 9 showed that an epigenetic network could perform the foraging robot task, their performance was consistently comparable to that of its non-epigenetic counterpart. However, as the results from Chapter 9 demonstrated that a notable level of epigenetic activity was present in those networks that did perform well, it is reasonable to speculate that other types of robot tasks may be more suited to the unique abilities of the epigenetic network.

10.2.3 Maintaining Performance in a Dynamic Environment

Finally, a return to the hypothesis presented in Section 1.2 of this thesis: the assertion that an epigenetic network, with its ability to alter its own structure in response to change, would be able to maintain its performance when confronted with stimulus that it had not been optimised for; with this latter part being refined in Section 6.4 into the idea of a dynamic environment. The answer here is much less clear cut. Chapter 7 did show that the epigenetic networks were able to recover from the impulse injection faster than the non-epigenetic networks; analysis showed that the results were statistically significant; and network analysis linked this improved response to the epigenetic mechanisms. However, this was in the limited case of the injection of a single, low magnitude impulse as it was intended as a proof of concept before progressing to the more substantial experiments in Chapters 8 and 9. Chapter 8 gives the least support for the hypothesis, as the epigenetic networks completely failed to maintain any degree of performance when confronted with the sudden mobility of their previously static prey. The saving grace is perhaps that the non-epigenetic networks performed equally poorly. Lastly, while not as bad as those in Chapter 8, the results in Chapter 9 also provided little support, as the supermajority of epigenetic networks failed to maintain performance within the debris fall environment. However, two outliers did recover some of the rescue targets, and network analysis showed that this limited performance was down to the epigenetic elements of the network, which dramatically altered their behaviour in response to the environmental change.

Taking all this together, it is reasonable to make the following statements:

- In the case of tasks that have already been shown to benefit from the inclusion of epigenetic elements, such as the inverted pendulum, there is evidence to support the hypothesis, although more work is needed to provide confirmation.
- In the case of the robotic control tasks, there is insufficient evidence to support the hypothesis, although it is unknown if this is due to the inherent invalidity of the hypothesis, or a consequence of the lack of benefit from epigenetic behaviour in general.

10.3 Critical Comments

Before progressing to the matter of future directions that work in this area could take, this section will serve as a repository for some general critical commentary about several aspects of this work.

The various genetic algorithms employed throughout this work performed acceptably, although all but NSGA-II were quite limited. This was especially true of the single sample generational evolution algorithm that is integrated into JBotEvolver. In an ideal world both it, and the simple evolutionary algorithm would have been replaced by something more suitable for the task, but the need to make comparisons with Turner's work and employ the simulation tool prevented that on this occation.

Despite its unusual design, the Hamann inverted pendulum model that was employed in the first set of experiments (and described in detail in Section 6.1) [71] provided no difficulties, thanks primarily to its existing history with epigenetic networks [6]. However, while it set out to act as a benchmark for robotics problems, the design decisions made in pursuit of this are unusual. The reduction of sensor/actuator precision and the replacement of abstract measurements with ones that correspond to simulated sensors positioned on the pendulum model are good, but the choice of sensor types and placements could be improved. For example, it would be more common in current practice for pendulum angle to be measured by a rotary encoder; while the speed of the cart would more likely be inferred from measurements made at the wheels or via use of accelerometers.

In a similar situation is JBotEvolver [77], the tool used for all the true robotics experiments in Chapters 9 and 8. Its status as a tool designed for evolving bio-inspired networks for robotic control made it a well suited choice, but it had a few issues. Foremost among them was the lack of high quality documentation, which resulted in a steep learning curve, as well as certain aspects, such as its sub-optimal evolutionary algorithm, only being discovered late on in its use. However, its open source nature and ease with which modifications can be made to its code mean that further work could continue to employ it.

10.4 Further Work

Now that this work is complete, it is important to consider what direction future work could take to build upon this thesis. Looking back over the work done and its results, two directions present themselves:

- Greater range of dynamic environment experiments with tasks such as the inverted pendulum. As noted, in Section 10.2, the range of dynamic environment experiments done with the inverted pendulum was very limited, intended to be a simple proof of concept. It would therefore be worthwhile to expand upon this, not only in with the inverted pendulum itself, but with other tasks such as Chirikov's standard map and the control of transfer orbits in gravitational systems. In the case of the inverted pendulum, a greater range of injected impulses would be the obvious place to begin; and the possibility has been discussed of scenarios that involve a pendulum that extends/retracts, or a pendulum subjected to changing gravitational forces, perhaps simulating an object experience the forces of acceleration atop a rocket.
- Alternate robotics experiments. Though the performance of the foraging task seams to not benefit from the inclusion of epigenetic behaviours, this doesn't mean that no robotics application can benefit. A possibility that immediately suggests itself to the author is to investigate the application of epigenetic networks to control of a robot navigating a route that consists of varied types of terrain such as: road, mud, sand, rocky ground etc. In particular, giving the network control of aspects like the robot's gearing, suspension stiffness, differential, ride height and so forth; as these would benefit from different network states developing for different terrain types, something that the epigenetic mechanisms would be ideally suited to.

- Further refinement of the hardware architecture. The hardware architecture still incorporates several features that are a legacy of the software implementation, such as: the generic sigmoid function, which necessitates the use of the slope and offset parameters; and the calculation of connectivity at runtime. Replacement of these with alternatives such as: leveraging the individual LUTs to allows each element's sigmoid to already incorporate the slope/offset alterations; and establishment of connections during configuration of the network; would be a suitable starting point from which to then investigate other refinements.
- Drawing further inspiration from biology. The biological basis for the epigenetic networks in this thesis comes second hand from Turner's work [6], which only modelled the epigenetic mechanisms discussed in section 2.3 in an abstract fashion. While this has lent itself quite well to the creation of the hardware implementation, returning to the biological roots of epigenetics might provide new inspiration for further development: for example, the alteration of gene expression with epigenetics encompasses several different mechanisms [32], thus perhaps it might be fruitful to implement a number of differently operating epigenetic molecules into a network.

In a wider sense, other avenues of investigation include: investigating alternative optimisation techniques beyond simple genetic algorithms, with the author being particularly interested in the possibility of a version of the Neuro-Evolution for Advanced Topologies (NEAT) algorithm [84]; and crossing the reality gap, as all work done this far with epigenetic networks has involved simulations.

11 Appendix A: Network Structure Tables

11.1 Network 1

Section 7.2; Epigenetic Network; Inverted Pendulum Control; Fitness 0.97

Table 11.1.1 Connection parameters of all input elements of network 1, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Pendulum Angle 1	84	Molecule 2 Gene 7
2	Pendulum Angle 2	8	Gene 11
3	Pendulum Angle 3	88	Molecule 2 Gene 7
4	Pendulum Angle 4	59	Molecules 1 and 3 Genes 5 and 10
5	Proximity 1	4	Gene 11
6	Proximity 2	4	Gene 11
7	Cart Velocity 1	10	Genes 2 and 8
8	Cart Velocity 2	39	Genes 5 and 10
9	Pendulum Velocity 1	39	Genes 5 and 10
10	Pendulum Velocity 2	84	Molecule 2 Gene 7

Table 11.1.2 Connection parameters of all molecules of network 1, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	60	4	Input 4
2	68	28	Inputs 1, 3, 4 and 10 Gene 7
3	60	4	Input 4
4	122	10	Gene 6

Table 11.1.3. Connection parameters of all genes of network 1, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
1	31	2	Gene 12	Genes 5, 10 and 12
2	13	4	Input 7 Genes 4 and 8	Genes 4 and 8 Output 1
3	19	2	None	Genes 5 and 10
4	15	2	Genes 2 and 8	Genes 2 and 8 Output 1
5	39	21	Inputs 4, 8 and 9 Genes 1, 3, 10 and 12	Gene 10
6	124	11	None	Molecule 4 Output 2
7	96	17	Inputs 1, 3, and 10 Gene 9	Molecule 2
8	13	4	Input 7 Genes 2 and 4	Genes 2 and 4 Output 1
9	111	4	None	Gene 7
10	39	21	Inputs 4, 8 and 9 Genes 1, 3, 5 and 12	Gene 5
11	6	4	Inputs 2, 5 and 6	None
12	31	2	Gene 1	Genes 1, 5 and 10

Table 11.1.4. Connection parameters of all output elements of network 1, including which genes they use as inputs.

Output Number	Actuator	Identification	Proximity	Uses as an Input
1	Motor 1	10	4	Genes 2, 4 and 8
2	Motor 2	121	4	Gene 6

11.2 Network 2

Section 7.2; Epigenetic Network; Inverted Pendulum Control; Fitness 0.79

Table 11.2.1. Connection parameters of all input elements of network 2, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Pendulum Angle 1	16	Molecules 2, 3 and 4 Genes 4 and 8
2	Pendulum Angle 2	3	Molecule 4 Genes 4 and 8
3	Pendulum Angle 3	55	Genes 1, 7, 10 and 12
4	Pendulum Angle 4	16	Molecules 2, 3 and 4 Genes 4 and 8
5	Proximity 1	52	Genes 1, 7, 10 and 12
6	Proximity 2	52	Genes 1, 7, 10 and 12
7	Cart Velocity 1	4	Molecules 2 and 3 Genes 4 and 8
8	Cart Velocity 2	52	Genes 1, 7, 10 and 12
9	Pendulum Velocity 1	16	Molecules 2, 3 and 4 Genes 4 and 8
10	Pendulum Velocity 2	97	Genes 3 and 9

Table 11.2.2. Connection parameters of all molecules of network 2, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	121	6	Gene 5
2	15	11	Inputs 1, 4, 7 and 9 Genes 4, 8 and 11
3	15	11	Inputs 1, 4, 7 and 9 Genes 4, 8 and 11
4	19	24	Inputs 1, 2, 4, 7 and 9 Genes 4, 8 and 11

Table 11.2.3. Connection parameters of all genes of network 2, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
1	66	29	Inputs 3, 5, 6 and 8 Genes 2, 6, 7, 10 and 12	Genes 7, 10 and 12
2	71	1	Gene 6	Genes 1, 6, 7, 10 and 12
3	112	23	Input 10 Genes 9 and 5	Gene 9 Output 1
4	17	25	Inputs 1, 2, 4, 7 and 9 Genes 8 and 11	Molecules 2, 3 and 4 Gene 8 Output 2
5	123	1	None	Molecule 1 Genes 3 and 9 Output 1
6	71	1	Gene 2	Genes 1, 2, 7, 10 and 12
7	73	28	Inputs 3, 5, 6, 8 and 10 Genes 1, 2, 6, 10 and 12	Genes 1, 10 and 12
8	13	18	Inputs 1, 2, 4, 7 and 9 Genes 4 and 11	Molecules 2, 3 and 4 Gene 4 Output 2
9	112	23	Input 10 Genes 3 and 5	Gene 3 Output 1
10	66	29	Inputs 3, 5, 6 and 8 Genes 1, 2, 6, 7 and 12	Genes 1, 7 and 12

11	8	1	None	Molecules 2, 3 and 4 Genes 4 and 8 Output 2
12	66	29	Inputs 3, 5, 6 and 8 Genes 1, 2, 6, 7 and 10	Genes 1, 7 and 10

Table 11.2.4. Connection parameters of all output elements of network 2, including which genes they use as inputs.

Output Number	Actuator	Identification	Proximity	Uses as an Input
1	Motor 1	121	9	Genes 3, 5 and 9
2	Motor 2	32	24	Genes 4, 8 and 11

11.3 Network 3

Section 7.2; Non-Epigenetic Network; Inverted Pendulum Control; Fitness 0.55

Table 11.3.1. Connection parameters of all input elements of network 3, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Pendulum Angle 1	61	Gene 16
2	Pendulum Angle 2	94	Genes 1, 3, 6, 8, 9, 11, 13, 14 and 15
3	Pendulum Angle 3	94	Genes 1, 3, 6, 8, 9, 11, 13, 14 and 15
4	Pendulum Angle 4	64	Gene 16
5	Proximity 1	82	Genes 3, 6, 8 and 9
6	Proximity 2	53	Genes 2, 7 and 16
7	Cart Velocity 1	64	Gene 16
8	Cart Velocity 2	53	Genes 2, 7 and 16
9	Pendulum Velocity 1	58	Genes 2, 7 and 16
10	Pendulum Velocity 2	28	Genes 2, 4, 5 and 7

Table 11.3.2. Connection parameters of all genes of network 3, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
1	111	23	Inputs 2 and 3 Genes 3, 6, 8, 9, 10, 11, 13, 14 and 15	Genes 3, 6, 8, 9, 10, 13 and 14
2	28	31	Inputs 6, 8, 9, 10 Genes 4, 5, 7, 12, 16	Genes 7, 4 and 5
3	96	23	Inputs 2, 3 and 5 Genes 1, 6, 8, 9, 10, 11, 13, 14 and 15	Genes 1, 6, 8, 9, 10, 11, 13, 14 and 15 Output 2
4	20	13	Input 10 Genes 2, 5, 7 and 12	Genes 2, 5 and 7
5	20	13	Input 10 Genes 2, 4, 7 and 12	Genes 2, 4 and 7
6	106	31	Inputs 2, 3 and 5 Genes 1, 3, 8, 9, 10, 11, 13, 14 and 15	Genes 1, 3, 8, 9, 10, 13 and 14
7	28	31	Input 10 Genes 2, 4, 5 and 12	Genes 2, 4 and 5
8	100	23	Inputs 2, 3 and 5 Genes 1, 3, 6, 9, 10, 11, 13, 14 and 15	Genes 1, 3, 6, 9, 10, 11, 13 and 14 Output 2
9	101	23	Inputs 2, 3 and 5 Genes 1, 3, 6, 8, 10, 11, 13, 14 and 15	Genes 1, 3, 6, 8, 10, 11, 13 and 14 Output 2
10	111	12	Genes 1, 6, 8, 9, 13 and 14	Genes 1, 3, 6, 8, 9, 13 and 14
11	95	6	Inputs 2 and 3 Genes 3, 8, 9 and 15	Genes 1, 3, 6, 8, 9, 10, 11, 13 and 14 Output 2

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
12	9	3	None	Genes 2, 4, 5 and 7
13	111	23	Inputs 2 and 3 Genes 1, 3, 6, 8, 9, 10, 11, 14 and 15	Genes 1, 3, 6, 8, 9 and 14
14	111	23	Inputs 2 and 3 Genes 1, 3, 6, 8, 9, 10, 11, 13 and 15	Genes 1, 3, 6, 8, 9 and 13
15	95	1	Inputs 2 and 3 Genes 3 and 11	Genes 1, 3, 6, 8, 9, 11, 13 and 14 Output 2
16	54	11	Inputs 1, 4, 6, 7, 8 and 9	Output 1

Table 11.3.3. Connection parameters of all output elements of network 3, including which genes they use as inputs.

Output Number	Actuator	Identification	Proximity	Uses as an Input
1	Motor 1	64	16	Gene 16
2	Motor 2	76	26	Genes 3, 8, 9, 11, 15 and 16

11.4 Network 4

Section 8.1; Epigenetic Network; Basic Foraging Task; Fitness 9.40

Table 11.4.1. Connection parameters of all input elements of network 4, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Prey Carried	38	Molecule 4 Genes 5, 10 and 12
2	Left Prey Eye	17	Gene 6
3	Right Prey Eye	55	Genes 1, 2, 4, 5, 10 and 12
4	Left Nest Eye	58	Genes 1, 2, 4, 5, 10 and 12
5	Right Nest Eye	121	Molecules 2 and 3 Gene 8

Table 11.4.2. Connection parameters of all molecules of network 4, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	77	8	Genes 7 and 11
2	109	27	Input 5 Genes 3 and 8
3	105	23	Input 5 Genes 3 and 8
4	38	9	Input 1 Gene 10

Table 11.4.3. Connection parameters of all genes of network 4, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
1	63	20	Inputs 3 and 4 Genes 2, 4, 5, 7, 10, 11 and 12	Genes 4, 5, 7, 10 and 11
2	49	9	Inputs 3 and 4 Genes 10 and 12	Genes 1, 4, 5, 10 and 12
3	93	0	None	Molecules 2 and 3
4	64	19	Inputs 3 and 4 Genes 1, 2, 5, 7, 10, 11 and 12	Gene 1, 5, 7, 10 and 12
5	60	27	Inputs 1, 3 and 4 Genes 1, 2, 4, 7, 10, 11 and 12	Genes 1, 4, 10 and 12
6	5	22	Input 2	None
7	71	9	Genes 1, 4 and 11	Molecule 1 Genes 1, 4, 5, 10, 11 and 12
8	121	22	Input 5	Molecules 2 and 3
9	28	1	None	Gene 10
10	47	27	Inputs 1, 3 and 4 Genes 1, 2, 4, 5, 7, 9 and 12	Molecule 4 Genes 1, 2, 4, 5 and 12
11	77	9	Gene 7	Molecule 1 Genes 1, 4, 5, 7 and 12
12	56	23	Inputs 1, 3 and 4 Genes 1, 2, 4, 5, 7, 10 and 11	Genes 1, 2, 4, 5 and 10

Table 11.4.4. Connection parameters of all output elements of network 4, including which genes they use as inputs.

Output Number	**Actuator**	**Identification**	**Proximity**	**Uses as an Input**
1	Prey Grabber	56	24	Genes 1, 2, 4, 5, 7, 10, 11 and 12
2	Left Motor	36	15	Genes 2, 9 and 10
3	Right Motor	58	19	Genes 1, 2, 4, 5, 7, 10 and 12

11.5 Network 5

Section 8.2; Epigenetic Network; Maze Foraging Task; Fitness 1.032

Table 11.5.1. Connection parameters of all input elements of the network, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Prey Carried	31	Molecules 2 and 4 Genes 2, 6, 7, 9, 10 and 12
2	Left Prey Eye	126	Molecule 1 Gene 8
3	Right Prey Eye	25	Molecules 2 and 4 Genes 2, 6, 7, 9 and 12
4	Left Nest Eye	102	Molecules 1, 3 and 4
5	Right Nest Eye	89	Molecules 1, 3 and 4 Genes 1 and 4
6	Left Obstacle Eye	51	Molecules 2 and 4 Genes 3, 4 and 12
7	Right Obstacle Eye	12	Molecules 2 and 4 Genes 7, 9, 11 and 12

Table 11.5.2. Connection parameters of all molecules of the network, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	108	20	Inputs 2, 4 and 5 Genes 1 and 8
2	22	54	Inputs 1, 3, 6 and 7 Genes 2, 3, 4, 7, 9, 10, 11 and 12
3	86	16	Inputs 4 and 5 Genes 1, 4 and 5
4	19	83	Inputs 1, 3, 4, 5, 6 and 7 Genes 1, 2, 3, 4, 5, 6, 7, 9, 10, 11 and 12

Table 11.5.3. Connection parameters of all genes of the network, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	**Identification**	**Proximity**	**Uses as an Input**	**Used as Input by**
1	88	4	Input 5 Gene 5	Molecules 1, 2 and 3 Gene 4 Outputs 1 and 2
2	30	10	Inputs 1 and 3 Genes 6, 9, 10 and 12	Molecules 2 and 4 Genes 6, 7, 9, 10 and 12 Output 3
3	58	14	Input 6 Gene 4	Molecules 2 and 4 Genes 4 and 12 Output 2
4	70	19	Inputs 5 and 6 Genes 1, 3 and 5	Molecules 2, 3 and 4 Gene 3 Output 2
5	84	0	None	Molecules 2 and 3 Genes 1 and 4 Outputs 1 and 3
6	31	10	Inputs 1 and 3 Genes 2, 9, 10 and 12	Molecules 2 and 4 Genes 2, 7, 9, 10 and 12 Output 3
7	14	36	Inputs 1, 3 and 7 Genes 2, 6, 9, 10, 11 and 12	Molecules 2 and 4 Genes 9, 11 and 12 Output 3
8	121	10	Input 2	Molecule 1 Output 1
9	22	28	Inputs 1, 3 and 7 Genes 2, 6, 7, 10, 11 and 12	Molecules 2 and 4 Genes 2, 6, 7, 11 and 12 Output 3

10	38	8	Input 1 Genes 6 and 12	Molecules 2 and 4 Genes 2, 6, 7, 9 and 12 Output 3
11	0	24	Input 7 Genes 7 and 9	Molecules 2 and 4 Genes 7 and 9
12	35	29	Inputs 1, 3, 6 and 7 Genes 2, 3, 6, 7, 9, 10 and 12	Molecules 2 and 4 Genes 2, 6, 7, 9 and 10 Output 3

Table 11.5.4. Connection parameters of all output elements of the network, including which genes they use as inputs.

Output Number	**Actuator**	**Identification**	**Proximity**	**Uses as an Input**
1	Prey Grabber	103	27	Genes 1, 5 and 8
2	Left Motor	78	31	Genes 1, 3, 4 and 5
3	Right Motor	26	19	Genes 2, 6, 7, 9 and 12

11.6 Network 6

Section 9.1; Epigenetic Network; Basic Rescue Task; Fitness 20.776

Table 11.6.1. Connection parameters of all input elements of network 6, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Person Carried	109	Molecules 1 and 2 Genes 2, 3, 8 and 12
2	Left Person Eye	106	Molecules 1 and 2 Genes 2, 3, 8 and 12
3	Right Person Eye	126	Molecule 2 Genes 3, 8 and 12
4	Left Nest Eye	87	Molecule 2 Genes 1, 2 and 5
5	Right Nest Eye	74	Molecule 2 Genes 1, 2, 5 and 6
6	Left Obstacle Eye	122	Molecule 2 Genes 3, 8 and 12
7	Right Obstacle Eye	16	Molecules 2, 3 and 4 Genes 7 and 11

Table 11.6.2. Connection parameters of all molecules of network 6, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	109	7	Inputs 1 and 2
2	23	124	Inputs 1, 2, 3, 4, 5, 6 and 7 Genes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12
3	36	24	Input 7 Genes 10 and 11
4	2	32	Input 7 Genes 7, 9, 10 and 11

Table 11.6.3. Connection parameters of all genes of network 6, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	**Identification**	**Proximity**	**Uses as an Input**	**Used as Input by**
1	63	28	Inputs 4 and 5 Genes 2, 4, 5 and 6	Molecule 2 Genes 5 and 6
2	91	20	Inputs 1, 2, 4 and 5 Genes 4 and 5	Molecule 2 Genes 1 and 5
3	120	15	Inputs 1, 2, 3 and 6 Genes 8 and 12	Molecule 2 Genes 8 and 12 Output 2
4	84	2	None	Molecule 2 Genes 1, 2 and 5
5	81	24	Inputs 4 and 5 Genes 1, 2, 4 and 6	Molecule 2 Genes 1 and 2
6	66	10	Input 5 Gene 1	Molecule 2 Genes 1 and 5
7	3	30	Input 7, Genes 9 and 11	Molecules 2 and 4
8	125	31	Inputs 1, 2, 3 and 6 Genes 3 and 12	Molecule 2 Genes 3 and 12 Output 2
9	7	0	None	Molecules 2 and 4 Gene 7
10	34	2	None	Molecules 2 and 3 Output 1
11	15	5	Input 7	Molecules 2, 3 and 4 Gene 7 Output 3
12	121	16	Inputs 1, 2, 3 and 6 Genes 3 and 8	Molecule 2 Genes 3 and 8 Output 2

Table 11.6.4. Connection parameters of all output elements of network 6, including which genes they use as inputs.

Output Number	**Actuator**	**Identification**	**Proximity**	**Uses as an Input**
1	Person Grabber	36	8	Gene 10
2	Left Motor	123	14	Genes 3, 8 and 12
3	Right Motor	19	9	Gene 11

11.7 Network 7

Section 9.1; Epigenetic Network; Basic Rescue Task; Fitness 21.475

Table 11.7.1. Connection parameters of all input elements of the network, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Person Carried	19	Molecules 2 and 4
			Genes 7, 9, 10 and 11
2	Left Person Eye	117	Genes 8 and 7
3	Right Person Eye	0	Molecule 4
			Genes 7 and 10
4	Left Nest Eye	8	Molecules 2 and 4
			Genes 7, 9 and 10
5	Right Nest Eye	95	Genes 4 and 7
6	Left Obstacle Eye	92	Genes 4 and 7
7	Right Obstacle Eye	11	Molecules 2 and 4
			Genes 7, 9 and 10

Table 11.7.2. Connection parameters of all molecules of the network, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	105	4	Gene 12
2	16	12	Inputs 1, 4 and 7
			Genes 7, 9, 10 and 11
3	51	29	Genes 1, 2, 3 and 5
4	9	64	Inputs 1, 3, 4 and 7
			Genes 1, 2, 3, 5, 7, 9, 10 and 11

Table 11.7.3. Connection parameters of all genes of the network, including what other network elements they use as inputs and what network elements use them as inputs.

Gene	Identification	Proximity	Uses as an Input	Used as Input by
1	54	29	Genes 2, 3, 5 and 11	Molecules 3 and 4 Genes 2, 3 and 7
2	43	16	Genes 1, 3 and 5	Molecules 3 and 4 Genes 1, 3 and 7
3	52	19	Genes 1, 2 and 5	Molecules 3 and 4 Genes 1, 2 and 7
4	95	22	Inputs 2, 5 and 6 Genes 6, 8 and 12	Gene 7 Output 3
5	36	0	None	Molecules 3 and 4 Genes 1, 2, 3, 7, 10 and 11 Output 1
6	85	0	None	Genes 4 and 7 Output 3
7	9	116	Inputs 1, 2, 3, 4, 5, 6, and 7 Genes 1, 2, 3, 4, 5, 6, 8, 9, 10, 11 and 12	Molecules 2 and 4 Genes 9 and 10 Output 1
8	116	17	Input 2 Gene 12	Gene 7
9	17	12	Inputs 1, 4 and 7 Genes 7, 10 and 11	Molecules 2 and 4 Genes 7, 10 and 11 Output 1

10	20	24	Inputs 1, 3, 4 and 7 Genes 5, 7, 9 and 11	Molecules 2 and 4 Genes 7, 9 and 11 Output 1
11	25	14	Inputs 1 and 7 Genes 5, 9 and 10	Molecules 2 and 4 Genes 1, 7, 9 and 10 Output 1
12	105	2	None	Molecule 1 Genes 7 and 8 Output 3

Table 11.7.4. Connection parameters of all output elements of the network, including which genes they use as inputs.

Output Number	**Actuator**	**Identification**	**Proximity**	**Uses as an Input**
1	Person Grabber	23	15	Genes 7, 9, 10 and 11
2	Left Motor	11	1	None
3	Right Motor	83	24	Genes 4, 6 and 12

11.9 Network 8

Section 9.2; Epigenetic Network; Rescue Task with Falling Debris; Fitness 18.60

Table 11.9.1. Connection parameters of all input elements of network 7, including what other network elements use them as inputs.

Input Number	Sensor	Identification	Input for
1	Person Carried	79	Molecules 1, 2 and 4 Genes 2, 7 and 10
2	Left Person Eye	30	Molecules 2 and 4 Genes 4, 7 and 9
3	Right Person Eye	0	Molecules 2 and 4 Genes 4, 5 and 7
4	Left Nest Eye	120	Molecules 1 and 4 Genes 8, 10, 11 and 12
5	Right Nest Eye	118	Molecules 1 and 4 Genes 3, 8, 10, 11 and 12
6	Left Obstacle Eye	34	Molecules 2 and 4 Genes 4, 7 and 9
7	Right Obstacle Eye	6	Molecules 2 and 4 Genes 4 and 7

Table 11.9.2. Connection parameters of all molecules of network 7, including what other network elements they use as inputs. Note that which genes are controlled by each molecule varies during runtime.

Molecule Number	Identification	Proximity	Uses as an Input
1	101	29	Inputs 1, 4 and 5 Genes 2, 3, 8, 10, 11 and 12
2	18	86	Inputs 1, 2, 3, 6 and 7 Genes 1, 2, 4, 5, 6, 7, 9 and 10
3	19	5	Gene 4
4	29	105	Inputs 1, 2, 3, 4, 5, 6 and 7 Genes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12

Table 11.9.3. Connection parameters of all genes of network 7, including what other network elements they use as inputs and what network elements use them as inputs.

Gene Number	Identification	Proximity	Uses as an Input	Used as Input by
1	59	4	None	Molecules 2 and 4 Outputs 1 and 2
2	89	11	Input 1 Gene 10	Molecules 1, 2 and 4 Gene 10
3	109	9	Input 5 Gene 12	Molecules 2 and 4 Genes 10, 11 and 12
4	20	24	Inputs 2, 3, 6 and 7 Genes 5, 7 and 9	Molecules 2, 3 and 4 Gene 7 Output 2
5	0	0	Input 3	Molecules 2 and 4 Genes 4 and 7 Output 3
6	69	9	None	Molecules 2 and 4 Genes 7 and 10 Outputs 1 and 2
7	6	80	Inputs 1, 2, 3, 6 and 7 Genes 1, 4, 5, 6 and 9	Molecules 2 and 4 Gene 4 Output 3
8	120	7	Inputs 4 and 5 Genes 11 and 12	Molecules 1 and 4 Genes 10, 11 and 12
9	31	8	Inputs 2 and 6	Molecules 2 and 4 Genes 4 and 7 Output 2

10	99	31	Inputs 1, 4 and 5 Genes 2, 3, 6, 8, 11 and 12	Molecules 1, 2 and 4 Genes 2 and 11
11	120	22	Inputs 4 and 5 Genes 3, 8, 10 and 12	Molecules 1 and 4 Genes 8, 10 and 12
12	119	18	Inputs 4 and 5 Genes 3, 8 and 11	Molecules 1 and 4 Genes 3, 8, 10 and 11

Table 11.9.4. Connection parameters of all output elements of network 7, including which genes they use as inputs.

Output Number	**Actuator**	**Identification**	**Proximity**	**Uses as an Input**
1	Person Grabber	52	19	Genes 1 and 6
2	Left Motor	49	31	Genes 1, 4, 6 and 9
3	Right Motor	3	12	Genes 5 and 7

12 Bibliography

[1] Y. Yuan, X. Yu, X. Yang, Y. Xiao, B. Xiang and Y. Wang, “Bionic building energy efficiency and bionic green architecture: A review,” *Renewable and Sustainable Energy Reviews,* vol. 74, pp. 771-787, 26th March 2017.

[2] B. Song, V. E. Johansen, O. Sigmund and J. H. Shin, “Reproducing the Hierarchy of Disorder for Morph-inspired, Broad-angle Color Reflection,” *Scientific Reports,* vol. 7, no. 46023, 2017.

[3] T. A. Brown, “Control of Gene Expression,” in *Genetics: A Molecular Approach*, Cheltenham, Stanlet Thornes Ltd, 1994, pp. 145-167.

[4] N. Carey, The Epigenetics Revolution, London: Icon Books, 2012.

[5] S. Cussat-Blanc, K. Harrington and W. Banzhaf, “Artificial Gene Regulatory Networks - A Review,” *Artificial Life,* vol. 24, no. 4, pp. 296 - 328, 2018.

[6] A. P. Turner, *The Artifical Epigenetic Network,* York: University of York, 2013.

[7] W. S. McCulloch and W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity,” *The Bulletin of Mathematical Biophysics,* vol. 5, no. 4, pp. 115-133, 1943.

[8] J. Hertz, A. Kroght and R. G. Palmer, Introduction to the Theory of Neural Computation, New York: Perseus Books Publishing, 1991.

[9] J. F. Kihlstrom, “Social Neuroscience: The Footprints of Phineas Gage,” *Social Cognition,* vol. 28, no. 6, pp. 757-783, 2010.

[10] F. A. C. Azevedo, L. R. B. Carvalho, L. T. Grinerg, J. M. Farfel, R. E. L. Ferretti, R. E. P. Leite, W. J. Filho, R. Lent and S. Herculano-Houzel, “Equal Numbers of Neuronal and Nonneuronal Cells make the Human Brain an Isometrically Scaled-Up Primate Brain,” *The Journal of Comparative Neurology,* vol. 513, no. 5, pp. 532-541, 2009.

[11] F. Jabr, “Does Thinking Really Hard Burn More Calories?,” Scientific American, 18 July 2012. [Online]. Available: https://www.scientificamerican.com/article/thinking-hard-calories/. [Accessed 20 August 2019].

[12] C. Hammond, Cellular and Molecular Neurobiology, Paris: Dion, 1996.

[13] J. B. Ahire, “The Artificial Neural Networks Handbook: Part 4,” Medium, 11 November 2018. [Online]. Available: https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e. [Accessed 2019 November 2019].

[14] M. A. Lones, A. M. Tyrell, S. Stepney and L. S. Caves, “Controlling Complex Dynamics with Artificial Biochemical Networks,” in *European Conference on Genetic Programming*, Berlin, 2010.

[15] C. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” 8 November 2018. [Online]. Available: https://arxiv.org/abs/1811.03378. [Accessed 23 August 2019].

[16] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzman Machines,” in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, 2010.

[17] A. L. Maas, A. Y. Hannum and A. Y. Ng, “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, 2013.

[18] X. Glorot, A. Bordes and B. Yoshua, “Deep Sparse Rectifier Neural Networks,” in *Proceedings of the 14th International Conference on Artifical Intelligence and Statistics*, Fort Lanerdale, 2011.

[19] F. Rosenblatt, “The Perceptron: A Probabailistic Model for Information Storage and Organisation in the Brain,” *Psychological Review,* vol. 65, no. 6, pp. 386-408, 1958.

[20] M. Minsky and S. Papert, Perceptrons: an introduction to computational geometry, Cambridge: The MIT Press, 1969.

[21] R. Rojas, "Weighted Networks - The Perceptron," in *Neural Networks - A Systematic Introduction*, Berlin, Springer-Verlang, 1996, pp. 55-76.

[22] T. A. Brown, "The Structure of DNA," in *Genetics: A Molecular Approach*, Cheltenham, Stanley Thornes Ltd, 1998, pp. 24-51.

[23] T. A. Brown, "The Genetic Code," in *Genetics: A Molecular Approach*, Cheltenham, Stanley Thornes Ltd, 1994, pp. 108-126.

[24] T. A. Brown, "Genes and Biological Information," in *Genetics: A Molecular Approach*, Cheltenham, Stanley Thornes Ltd, 1994, pp. 40-51.

[25] T. R. Robinson, Genetics for Dummies, Hoboken, New Jersey: Wiley Publishing Inc., 2005.

[26] T. A. Brown, "Transcription," in *Genetics: A Molecular Approach*, Cheltenham, Stanley Thornes Ltd, 1994, pp. 52-70.

[27] S. A. Kauffman, "Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets," *Journal of Theoretical Biology,* no. 22, pp. 437-467, 1969.

[28] A. Roli, M. Manfroni, C. Pinciroli and M. Birattari, "On the Design of Boolean Network Robots," *Evolutionary Applications,* pp. 43-52, 2011.

[29] G. Karlebach and R. Shamir, "Modelling and Analysis of Gene Regulatory Networks," *Nature Reviews: Molecular Cell Biology,* vol. 9, no. 10, pp. 770-708, 2008.

[30] C. D. Allis, T. Jenuwein and D. Reinberg, "Overview and Concepts," in *Epigenetics*, New York, Cold Spring Harbor Laboratory Press, 2007, pp. 23-61.

[31] "Chromosome Structure Proteins," Creative BioMart, [Online]. Available: https://www.creativebiomart.net/researcharea-chromosome-structure-proteins_407.htm. [Accessed 02 September 2019].

[32] T. Kouzarides and S. L. Berger, "Chromatin Modifications and their Mechanisms of Action," in *Epigenetics*, New York, Cold Spring Harbor Laboratory Press, 2007, pp. 191-209.

[33] A. P. Turner, M. A. Lones, L. A. Fuente, S. Stepney, L. S. D. Caves and A. Tyrrell, "The Artifical Epigenetic Network," in *IEEE International Conference on Evolvable Systems*, Singapore, 2013.

[34] A. Dey, "Machine Learning Algorithms: A Review," *International Journal of Computer Science and Information Technologies,* vol. 7, no. 3, pp. 1174-1179, 2016.

[35] T. Back, Evolutionary Algorithms in Theory and Practice, New York: Oxford University Press Inc., 1996.

[36] E. Mayr, The Growth of Biological Thought, Belknap Press, 1982.

[37] C. Darwin, On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life, London: John Murray, 1859.

[38] I. Rechenberg, *Evolutionary Strategy – Optimisation of Technical Systems According to the Principles of Biological Evolution,* Berlin: Technical University of Berlin, 1971.

[39] A. Oyama, S. Obayashi, K. Nakahashi and T. Nakamura, "Aerodynamic Optimization of Transonic Wing Design Based on Evolutionary Algorithms," Tohoku University, Sendai, 2000.

[40] J. A. Hilder, *Evolving Variability Tolerant Logic,* York: University of York, 2010.

[41] N. Jakobi, P. Husbands and I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," in *European Conference on Artifical Life*, Granada, Spain, 1995.

[42] K. Sims, "Evolving Virtual Creatures," in *SIGGRAPH*, Orlando, Florida, 1994.

[43] J. Lehman and et al, "The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities," 14 August 2018. [Online]. Available: https://arxiv.org/abs/1803.03453. [Accessed 02 August 2019].

[44] T. Blickle, “Tournament Selection,” in *Evolutionary Computation 1 - Basic Algorithms and Operators*, Bristol and Philadelphia, Institute of Physics Publishing, 2000, pp. 181-186.

[45] J. Grefenstette, “Rank-Based Selection,” in *Evolutionary Computation 1 - Basic Algorithms and Operators*, Bristol and Philadelphia, Institute of Physics Publishing, 2000, pp. 187-194.

[46] D. Gupta and S. Ghafir, “An Overview of Methods of Maintaining Diversity in Genetic Algorithms,” *International Journal of Emerging Technology and Advanced Engineering,* vol. 2, no. 5, pp. 56-60, 2012.

[47] T. Kuo and S.-Y. Hwang, “Using Disruptive Selection to Maintain Diversity in Genetic Algorithms,” *Applied Intelligence,* vol. 7, no. 3, pp. 257-267, 1997.

[48] L. B. Bookers, D. B. Fogel, D. Whitley, P. J. Angeline and A. E. Eiben, “Recombination,” in *Evolutionary Computation I - Basic Algorithms and Operators*, Bristol and Philadelphia, Institute of Physics Publishing, 200, pp. 256-307.

[49] H. Muhlenbein, “How genetic algorithms really work I - Mutation and hillclimbing,” in *Parrallel Problem Solving from Nature 2*, Amsterdam, Elsever, 1992, pp. 15-25.

[50] E. Zitzler and L. Thiele, “Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach,” *IEEE Transactions on Evolutionary Computation,* vol. 3, no. 4, pp. 257-271, 1999.

[51] C. M. Fonseca and P. J. Fleming, “An Overview of Evolutionary Algorithms in Multiobjective Optimisation,” *Evolutionary Computation,* vol. 3, no. 1, pp. 1-16, 1995.

[52] J. D. Schaffer, “Multiple Objective Optimization with Vector Evaluated Genetic Algorithms,” in *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, 1985.

[53] M. P. Fourman, “Compaction of Symbolic Layout using Genetic Algorithms,” in *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, 1985.

[54] A. Ben-Tal, “Characterisation of Pareto and Lexographic Optimal Solutions,” in *Lecture Notes in Economics and Mathmatical Systems - Multiple Criteria Decision Making Theory and Application*, Berlin, Springer-Verlag, 1979, pp. 1-11.

[55] F. Kursawe, “A Variant of Evolutionary Strategies for Vector Optimisation,” in *Parallel Problem Solving from Nature - 1st Workshop*, Berlin, 1991.

[56] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation,* vol. 6, no. 2, pp. 182-197, 2002.

[57] N. Srinivas and K. Deb, “Multiobjective Optimization using Nondominated Sorting in Genetic Algorithms,” *Journal of Evolutionary Computation,* vol. 2, no. 3, pp. 221-248, 1995.

[58] T. Mestla, E. Plahteb and S. W. Omholt, “A mathematical framework for describing and analysing gene regulatory networks,” *Journal of Theoretical Biology,* vol. 176, no. 2, pp. 291-300, 1995.

[59] Xilinx, “LogiCORE IP: Floating-Point Operator,” Xilinx, San Jose, California, 2014.

[60] Xilinx, “7 Series FPGAs Configurable Logic Block User Guide,” Xilinx, 2016.

[61] Xilinx, “AXI Reference Guide,” Xilinx, San Jose, California, 2011.

[62] Xilinx, “Vivado Design Suite - AXI Reference Guide,” Xilinx, San Diego, California , 2017.

[63] Xilinx, “Zynq-7000 SoC Data Sheet,” Xilinx, San Diego, California, 2018.

[64] A. Dunlels and L. Woestenberg, “LwIP Lightweight IP Stack,” [Online]. Available: http://www.nongnu.org/lwip/2_1_x/index.html. [Accessed 12 07 2019].

[65] K. H. Lundburg and T. M. Barton, “History of Inverted-Pendulum Systems,” in *8th IFAC Symposium on Advances in Control Education*, Kumamoto, Japan, 2010.

[66] A. P. Turner, M. A. Trefzer, M. A. Lones and A. M. Tyrrell, “Evolving Efficient Solutions to Complex Problems Using the Artificial Epigenetic Network,” *Information Processing in Cells and Tissues,* vol. 9303, pp. 153-165, 2015.

[67] A. F. Winfield, “Foraging Robots,” in *Encyclopedia of Complexity and Systems Science*, New York, Springer, 2009.

[68] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. -C. Zufferey, D. Floreano and A. Martinoli, “The e-puck, a Robot Designed for Education in Engineering,” in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009.

[69] Microchip Technology Inc., “dsPIC30F6011A/6012A/6013A/6014A Data Sheet,” Microchip Technology Inc., Chandler and Tempe, Arizona, 2011.

[70] Intel, “Desktop 4th Generation Intel® Core™ Processor Family, Desktop Intel® Pentium® Processor Family, and Desktop Intel® Celeron® Processor Family Datasheet – Volume 1 of 2,” Intel, Santa Clara, California, 2015.

[71] H. Hamann, T. Schmickl and K. Crailsheim, “Coupled Inverted Pendulums: A Benchmark for Evolving Decentral Controllers in Modular Robotics,” in *Genetic and Evolutionary Computation Conference*, Dublin, 2011.

[72] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, Numerical Recipes in C++, Cambridge: Cambridge University Press, 2002.

[73] E. H. Ostergaard, G. S. Sukhatme and M. J. Mataric, “Emergent bucket brigading: A simple mechanism for improving performance in multi-robot constrained-space foraging tasks,” in *Proceedings of the International Conference on Autonomous Agents*, Monteal, Canada, 2001.

[74] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatell, M. Birattari, L. M. Gambardella and M. Dorigo, “ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems,” *Swarm Intelligence,* vol. 6, no. 4, pp. 271-295, 2012.

[75] O. Michel, “Webots: Professional Mobile Robot Simulation,” *International Journal of Advanced Robotic Systems,* vol. 1, no. 1, pp. 39-42, 2004.

[76] E. Rohmer, S. Singh and M. Freese, “V-REP: a Versatile and Scalable Robot Simulation Framework,” in *Proc. of The International Conference on Intelligent Robots and Systems*, 2013.

[77] M. Duarte, F. Silva, T. Rodrigues, S. M. Oliveira and A. L. Christensen, “JBotEvolver: A Versitile Simulation Platform for Evolutionary Robotics,” in *Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, New York, 2014.

[78] M. Duarte, F. Silva, T. Rodrigues, S. M. Oliveira and A. L. Christensen, “GitHub - JBotEvolver,” BioMachinesLab, 03rd September 2016. [Online]. Available: http://github.com/biomachineslab/jbotevolver. [Accessed 07th June 2019].

[79] R. A. Brooks, J. H. Connell and P. Ning, “Herbert: A Second Generation Mobile Robot,” *A.I. Memo,* January 1988.

[80] R. Hoggett, “1986c - "Herbert" the Collection Machine,” CyberneticZoo.com, 26th June 2011. [Online]. Available: http://cyberneticzoo.com/cyberneticanimals/1986c-herbert-the-collection-machine-brooks-connell-ning-american/. [Accessed 11th June 2019].

[81] “League Overview,” RoboCup Rescue League, [Online]. Available: https://rrl.robocup.org/league-overview/. [Accessed 20 September 2019].

[82] R. League, “RoboCup Rescue Rulebook 2019 version 2.4,” 17th June 2019. [Online]. Available: https://rrl.robocup.org/wp-content/uploads/2019/06/rrl_rulebook_2019_v2.4.pdf. [Accessed 20th September 2019].

[83] A. Vargha and H. D. Delaney, “A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong,” *Journal of Educational and Behavioral Statistics,* vol. 25, no. 2, pp. 101-132, 2000.

[84] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation,* vol. 10, no. 2, pp. 99-127, 2002.