An Exploration of Spiking Neural Networks and their use on Reinforcement Learning Tasks

by Andrew William Rafe

Thesis submitted in fulfilment of the requirements for the degree of

Master of Science (Research) in Computing Sciences

under the supervision of Dr. William Raffe and Dr. Jaime Garcia

University of Technology Sydney
Faculty of Engineering and Information Technology
School of Computer Science

May 2021

Certificate of Original Authorship

I, Andrew William Rafe declare that this thesis, is submitted in fulfilment of the requirements for the award of MSc (Res) in Computing Sciences, in the School of Computer Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:
Signature removed prior to publication.

11th May 2021

This thesis is dedicated to my mum who, in the opening weeks of this research, lost her long battle with breast cancer. You always taught me to pursue my interests in life and your immense commitment to your children has given me the opportunities to do just that.

Acknowledgements

I would like to take this opportunity to thank the people in my life for whom without them, the completion of this thesis would not have been possible.

Firstly, I would like to show my gratitude to my supervisors who have supported me throughout and have always been available for me to rack their brains, get their opinions on the direction of the research and provide me with every needed support to get this thesis completed. Dr. William Raffe, my primary supervisor, would always be happy to engage in the technical discussions I needed to be able to articulate my thought process, and to keep me on track. To Dr. Jaime Garcia, my co-supervisor, you would always be there to help me through my communication of the research and has therefore allowed me to excel at presenting my ideas to a wider skill-based audience. I would also like to thank you both for creating a wonderful environment in the Game Studio Research Lab and providing me with the opportunity to teach at the university during the conduct of this research, greatly widening the skill set that this degree has provided me.

Secondly, I would like to thank all the members, new and old, of the Game Studio Research Lab which I have had the benefit of watching grow during my time with them. They would always be available to provide support during stressful assessment periods and would drop everything to attend the various candidature assessment stages to provide their moral support. Although we were not able to be face to face for a large portion of my degree, the online and in person meetups that we had along the way were some of the highlights of my time during these years. So, thank you all and I look forward to my continued engagement with the Game Studio Research Lab in future.

Lastly, but certainly not least, I would like to thank my family. To my dad, Barry, my sister, Christine, and my brother, Michael, thank you for the continued support through these difficult years. Through our family gatherings, zoom catchups and constant family chats, you have all really pushed me to keep going and to produce the best possible work. You would always be interested in what I was currently working on and would happily endure my far too detailed recounts of what I was doing and what I had found.

For all of you, I would like to say thank you and, without you, I would not have been able to get through this period of my life and I will be forever grateful.

Publications

Rafe, A.W., Garcia, J.A. & Raffe, W.L., 2021, June. Exploration Of Encoding And Decoding Methods For Spiking Neural Networks On The Cart Pole And Lunar Lander Problems Using Evolutionary Training. *In 2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 498-505. IEEE. doi: 10.1109/CEC45853.2021.9504921.

Contents

List of Figures

List of Tables

Abstract

Artificial neural networks have recently been the prominent architecture for reinforcement learning tasks. However, there is emerging evidence that spiking neural networks can perform just as well and can retain this performance across similar environments. Spiking neural networks are experiencing a surge in popularity due to their potential for large efficiency gains when compared to their traditional artificial neural network counterparts. Though, when attempting to replicate the successes of artificial neural networks, challenges are faced due to their vastly different architectures and therefore differing methods for training and optimisation. As spiking neural networks are considered more biologically plausible, methods of training inspired by natural learning have been proposed. These methods have been minimally applied to complex reinforcement learning domains, instead typically focusing on supervised learning problems. This thesis aims to explore the use of spiking neural networks in reinforcement learning domains. Methods of evolutionary and spike timing based training will be explored. Additionally, an in-depth analysis of different encoding and decoding methods is conducted. This research also addresses the trends in the effect of the time period that a state is exposed to a spiking neural network on the performance of the networks.

Chapter 1

Introduction

Artificial neural networks (ANN) have been used to far exceed the performance of humans in a range of virtual control tasks. However, they are proving to be inefficient to train (Strubell et al. 2019), and fail at generalizing across similar tasks (Markowska-Kaczmar and Koldowski 2015). There is emerging evidence that spiking neural network (SNN) architectures can be more efficient (Neil et al. 2016) and retain performance across varied tasks (Markowska-Kaczmar and Koldowski 2015), though there is minimal focus on utilizing these networks for reinforcement learning in the literature. More work is needed on developing algorithms to train networks of spiking neurons for reinforcement learning tasks and compare their performance and generalisability to state of the art training methods for ANNs.

SNNs have recently seen a surge in popularity due to their apparent benefits in efficiency over ANN methods (Zambrano and Bohte 2016). However, SNNs have been historically difficult to train on complex reinforcement learning (RL) problems due to their inability to utilise gradient based optimisation. RL is a technique whereby an artificial agent learns based off of reward signals awarded in the environment (Sutton and Barto 2018). It is important that training methods for SNNs in RL environments are developed and tested on domains of equal complexity to problems solved by state-of-the-art ANN methods so that these efficiency benefits can be more widely utilised.

The inception of a new field of computer science in neuromorphic computing

(NC) is recognition of the possible benefits of using SNNs. Davies et al. (2018) identified that traditional computational architectures of modern computers do not allow for the efficient modelling of SNN architectures. Being able to utilise abundant parallel processes is beneficial as only very few variable updates per neuron are required in order to both propagate messages through the network as well as undergoing training. Improving the viability of training algorithms utilising SNNs when dealing with incredibly complex network structures and sizes will lead to greatly increased efficiency when transitioning to NC for RL tasks (Wunderlich et al. 2019).

1.1 Motivation

Researchers have been drawn to using video games to test their machine learning algorithms for decades. Techniques for machine learning have been demonstrated with some ground-breaking achievements in recent history. Researchers were able to conquer the ancient games of Chess and Go after Deep Blue beat world chess champion Garry Kasparov in 1997 (Campbell et al. 2002) and AlphaGo beat the European Go champion Fan Hui in 2015 (Borowiec 2016). Recently Alpha Zero has beaten the best Chess and Go bots in 2018 (Silver et al. 2018), and even more recently the defeat of two world champion StarCraft II players 5 to 0 by the agent Alpha Star (Vinyals et al. 2019). Variations of a wide range of algorithms were used to make these machine learning achievements, however, most relied on the use of artificial neural networks for decision making. Alpha Zero utilised pure RL with self-play from scratch and Alpha Star used a combination of supervised learning for early development and adversarial RL techniques for later stage development. What they both have in common is the use of deep ANNs and only the ability to solve their very specific limited versions of the games that they were applied to. Additionally, both required immense computing power and time to master their specific tasks. Apply these systems to slightly varied rule sets or scenarios, and they are unable to even achieve regular human level play. This lack of flexibility

to similar tasks and these large computational requirements makes these methods unable to be realistically adapted into real world robotics technologies and everyday applications (Strubell et al. 2019).

There is emerging evidence that complex SNN architectures can be applied much more efficiently on neuromorphic hardware requiring less computational power for execution (Wunderlich et al. 2019). However, SNNs have not been demonstrated on tasks of the level of complexity achieved by Alpha Star and Alpha Zero. This is primarily due to the inability to use gradient based optimization algorithms which both Alpha Star and Alpha Zero utilised. Research into RL on SNN architectures is still in its infancy. In terms of generalisability, there is evidence that SNNs have some ability to transfer understanding of one scenario into a similar but different task (Markowska-Kaczmar and Koldowski 2015). This raises the question as to whether SNN architectures, which are more biologically plausible than ANNs, could be useful in overcoming problems of efficiency and generalisability in this field.

Moreover, pure RL on ANN architectures has still struggled on stochastic, noisy, maze-like, and sparse reward environments. Mnih et al. (2015) in their Deep Q Learning experiments were able to far surpass human level performance for a range of games in the Arcade Learning Environment, a machine learning research framework currently supported by OpenAI on their Gym infrastructure (Brockman et al. 2016). The most impressive being its domination of video pinball where it performed at 2539% that of human level. Interestingly, this method failed at surpassing human level performance on a number of games, including Ms Pac-Man where the artificial agent only reached 13% of human performance. Arguably these types of problems are more aligned to real world problems and therefore raises the question as to whether deep RL methods on ANNs are capable for complex, stochastic, real world applications.

Even though RL methods on ANNs have accomplished amazing feats in the past, a question emerges as to whether more computational power or slight variations to these training algorithms will be enough to shift this field into applicable

real-world scenarios. Alternatively, are the architectures that these methods are being applied to, capable of such useful real-world applications? This question sits as the motivation for research into RL algorithms for SNN architectures. Not only are they a more biologically based architecture but it has been proposed that they are more efficient and can be applied to low energy neuromorphic computers (Tavanaei et al. 2019). However, there is more need to develop algorithms and test a variety of encoding and decoding strategies applied for input and action selection.

1.2 Research Gaps and Aims

From the initial research into the field of SNNs for RL, a number of gaps in the literature were discovered.

- Although starting to be widely used for supervised and unsupervised learning, SNNs have minimally been demonstrated on complex RL domains.
- There are very few algorithms used for training SNNs in RL environments.
- There are a number of different methods for encoding the inputs and decoding the outputs in a SNN with minimal comparative analysis between the different methods in RL environments.

The gaps outlined above are the result of a comprehensive literature review into the use of SNNs in RL environments as shown in Chapter 3, as well as more widely in supervised and unsupervised learning domains. This has led to the development of a set of aims.

The aims of this thesis revolve around the experimentation of a range of RL algorithms for SNN architectures as well as investigating network structure and spike train coding methods. They can be broken into the following four main aims:

1. To test whether specific network structures including single and multi-layered networks are capable of solving RL problems with some comparison to ANN structures.

2. To investigate encoding and decoding methods for extracting useful information for action selection in virtual environments.

3. To investigate how the exposure periods affect network performance.

4. To adapt non-evolutionary and biologically plausible learning techniques to networks operating in certain RL domains.

These aims and the identified research gaps have led to the development of key research questions to be answered throughout this thesis.

1.3 Research Questions

The following research questions have been created in order to address the aims identified. These research questions will be used as identifiers throughout this thesis for a clear recognition of the specific questions being tested at each stage.

(RQ1) Are SNN structures suitable at solving RL problems?

(RQ1 - A) Using evolutionary approaches to training, are SNNs without hidden layers (single layer networks) effective at solving RL problems?

(RQ1 - B) Using evolutionary approaches to training, are SNNs with hidden layers (multi-layer networks) effective at solving RL problems?

(RQ2) What network constraining and network coding methods improve training in RL environments?

(RQ2 - A) What affect does the exposure period of the state space to the network have on network performance?

(RQ2 - B) What methods are effective at decoding spike train signals into action selection?

(RQ2 - C) What methods are effective at encoding state space information into spike train signals?

(RQ2 - D) What methods effectively constrain the strengthening and weakening of synapse weights?

(RQ3) Due to the inability to use gradient based algorithms for the training of SNNs, are non-evolutionary learning techniques effective at solving RL problems?

(RQ3 - A) Can learning methods based on spike timing (Hebbian learning methods) effectively solve RL problems?

(RQ3 - B) Can learning methods that incorporate aspects of evolutionary and spike timing training effectively solve RL problems?

1.4 Objectives

In order to answer the research questions outlined in the previous section, a number of key objectives and sub-objectives have been developed.

1. Using evolutionary methods, train SNNs to accomplish a number of RL tasks in the OpenAI gym environment (Brockman et al. 2016).
   (a) Utilising the Sutton and Barto (2018) cart pole experiment implemented on the gym environment, train and compare ANN and SNN performance using similar genetic algorithms for both (RQ1) and compare different output decoding strategies (RQ2 - B).
   (b) Utilising the Lunar Lander RL problem from the Brockman et al. (2016) gym environment, explore the performance of SNNs using evolutionary training on this more complex RL problem (RQ1).
   (c) Utilising both Cart Pole and Lunar Lander, explore the affect that exposure period has on agent training (RQ2 - A).
   (d) For both problems in the evolutionary context, investigate different encoding strategies for converting the regular state space information into SNN readable spike trains (RQ2 - C)

2. Using spike timing-based methods, train SNNs in the Cart Pole problem from the Open AI gym(Brockman et al. 2016).

   (a) Test and compare different spike timing-based training methods on the Cart Pole problem and also compare their achievements compared to the evolutionary techniques (RQ3).

   (b) Test different methods of weight constraining techniques that will not necessarily be an issue in the evolutionary training framework (RQ2 - D).

These objectives and the methodology for each of the associated experiments will be explored in more detail in Chapters 4 and 5.

1.5 Methodology Constraints

The data collected to measure the performance of the SNNs against the RL problems is quantitative. The data will be collected from the rewards awarded to the agents over the course of their learning generations. In order to draw meaningful conclusions, it is imperative that, when comparing the performance of different network structures, coding methods and training algorithms, there is a consistent number of trials or generations depending on the training methods used as well as the same number of repeated tests for each. The specific number of trials and repetitions will be dependent on the complexity of the environment that they will be training in. Additionally, this thesis will compare the use of the same set of original inputs between networks with the understanding that SNNs will need a separate encoding method to transfer the states into a temporal message, however it will undertake this encoding with those same original inputs presented by the RL environment. Finally, when comparing different encoding and decoding methods of the SNNs, the only thing differing between the networks will be the method of converting the state space into temporal information and the extraction of the information from the final layer of the network for action selection. This set of

methodology constraints is imperative for drawing meaningful conclusions to the research questions.

1.6 Out of Scope

Although the computational efficiency of execution and training is paramount in the success of SNN architectures for RL tasks, the analysis of this will be ruled out of scope. There is evidence that traditional computing architectures are not efficient at modelling SNNs (Davies et al. 2018). Neuromorphic hardware is still in its infancy of development making gaining access to these technologies difficult. As a result, comparing efficiencies between ANNs and SNNs can only be achieved on traditional computing hardware. It appears that the benefits in efficiency will emerge through the application of effective RL algorithms on SNNs on neuromorphic based hardware and so this comparison on traditional computing will be irrelevant to the overall task of demonstrating whether the algorithms perform as efficiently when compared to common methods. Utilising neuromorphic computing for creation of SNN agents would be difficult due to the lack of availability of the technology as well as greatly increase the scope of the problem. For these reasons, testing the computational efficiency of SNN architectures will be ruled out of scope.

1.7 Significance

There are four main areas that this thesis will contribute to knowledge in the field of SNNs for RL problems. Firstly, the development and testing of a variety of training methods for SNNs in RL problems. Secondly, a comparison in performance between these set of algorithms and the state-of-the-art ANN training methods. Thirdly, the identification and development of the best methods for encoding and decoding information in a SNN. Finally, the exploration of the affect that the length of exposure of the state to the network has on network performance. These

will be discussed in this section.

1. Effective training algorithms for SNNs in RL domains are still in their infancy with many in the field identifying a lack of direct training methods (Bing et al. 2018) (Kaiser et al. 2016). Therefore, the development and testing of these algorithms is a valuable contribution to the field adding to the growing interest in finding effective methods for training.

2. A common criticism in the supervised and unsupervised learning domains for SNN architectures is their inability to match the performance of state-of-the-art ANN methods (Neil et al. 2016). This comparison has minimally been done in the field of RL. This thesis will add a minor contribution to the comparison of ANNs and SNNs and their performance on the Cart Pole RL problem.

3. Unlike traditional ANN architectures where the decision for action selection is based on a simple activation function in the final layer of the network, SNNs have to deal with their information being encoded in a temporal way. There are a number of different methods to firstly convert the state information into a temporal fashion and secondly, to turn this temporal information back into a useful form to be used for action selection. There is conflicting evidence to which of these methods is the most effective so this thesis will act as a comparison of these methods. This will be significant as it will unify results among different methods and different environments.

4. Rarely discussed in literature is the effect that exposure periods have on the learning performance of SNNs. This research will attempt to contribute to this area by identifying trends over a range of different exposure periods.

1.8 Summary

This chapter introduced the problem, suggested the aims and research questions, limited the scope and set out the significance for this research. Fundamentally, the research and experimentation in this thesis is an exploration of spiking neural networks specifically in the comparative analysis of different components of the network architecture, as well as an investigation into both evolutionary and spike based training methods on two reinforcement learning problems. The comparative analysis will look at multiple methods of encoding and decoding as well as investigating the effect that different state exposure periods have on the performance of the networks in solving the reinforcement learning problems. It will additionally look at the effect of different network shapes and training methods. In order to better understand the terminology and techniques that will be used in order to answer the research questions, the following chapter will cover the background.

Chapter 2

Background

The previouse chapter introduced the problem and set out the bounds of this research. It laid out a range of terminologies and methods that must be described in detail in order to understand the research questions and the method of solving these questions. This chapter will introduce the basic concepts necessary for an understanding of those research questions. Firstly, an introduction to the concepts of reinforcement learning will be made. Secondly, an explanation of the current state of the art methods using artificial neural networks with Deep Q Learning, with an identification of the challenges that these methods face. Thirdly, an explanation of networks of spiking neurons and lastly, an introduction to the various methods of handling SNN signals. Within this chapter, justifications for selecting certain network mechanisms will be made as there are a variety of different methods for modelling these types of networks.

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a field of machine learning whereby an agent learns from reward signals awarded during interaction with an environment. This process can be modelled in its simplest form as a Markov Decision Process (MDP) (Sutton and Barto 2018).

The MDP describes the interaction that an agent has with an environment

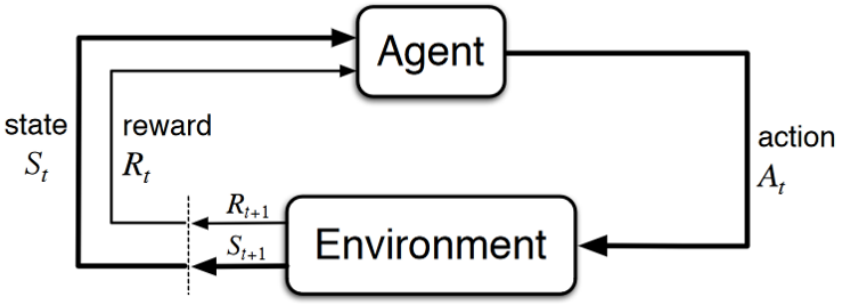

Figure 2.1: *A visual description of a Markov Decision Process (Sutton and Barto 2018)*

during a reinforcement learning task. The agent receives some input on the state of the environment as well of the reward signal it received from the environment. This input allows the agent to make a decision on the action that should be taken. This action is fed into the environment which will update based on that action thus generating a new state representation and a new reward signal. The challenge of reinforcement learning is teaching the agent which actions can net the most reward in an environment (Sutton and Barto 2018). These rewards can be both positive, in order to encourage certain behaviour, or negative, in order to punish the agent for taking certain actions or ending up in certain damaging states. Central to the challenge in RL is the trade-off between exploration and exploitation (Sutton and Barto 2018). The agent does not know the actions that it can take to net the high rewards until it tries them at some point during development, the exploration side of things. But during exploration the agent may need to exploit actions of high reward in order to progress and explore actions further in the environment. Yanguas-Gil (2018) suggests that there is a growing consensus in the fact that our central neural systems are affected by external reward signals in the world that reinforce our trends of behaviour. They demonstrate this in a simple insect brain but suggest that wider ranges of animals also undergo these processes of learning. This is supported by Mozafari et al. (2018) that for humans, the connections between neurons are strengthened and weakened according to the reward signals that our environment give off which in turn alters our behaviour. The similarities between

real world learning and reinforcement learning tasks make this an important area of research.

2.2 Artificial Neural Network (ANN)

Artificial neural networks are widely used as function approximators in machine learning tasks. ANNs are made up of a series of interconnected neurons. A neuron takes in a set of inputs, does some processing on those inputs, and produces a single output value. These neurons are connected to one another in a layered format. A neurons output will be sent to all of the next layer neurons down weighted connections. Traditionally a neuron will sum all of the incoming neuron values multiplied by the respective weighted connections that they travelled down. They will then pass this summed value into what is called an activation function which will do some processing on that value usually clamping or restricting the types of output that it can give out.

For internal network neurons these activation functions usually consist of what are called rectified linear units (ReLU) (Zeiler et al. 2013). ReLUs will produce zero output if their sum of incoming inputs is less than zero and normal positive output otherwise. Functions like the ReLU are continuous activation functions and pass on the data received and processed onto all of its connected neurons. ANNs are traditionally made up of an input layer followed by zero to many hidden layers with each layer connected to the next followed by a final output layer. These layers can be fully connected with each neuron in the previous layer connected to every other neuron in the next layer. This is not always the case and can either have a subset of fully connected connections or even have connections to previous or future layers.

A deep artificial neural network is a network made up of two or more hidden layers. These ANNs are becoming troublesome for approximating complex functions and machine learning problems as they are growing in size. They are hindered by computational efficiency in the propagation of data as well as plagued by several

problems of optimisation of the weighted connections using Deep Q Learning in both efficiency and performance.

2.3 Deep Q Learning

Q-Learning is a process in RL whereby a network or policy learns values associated with particular state action pairs. These values should, if trained correctly, represent the predicted reward value of the next state if the agent were to take a specific action. The decision on which action to take is based off which of the state action pairs has the highest predicted value (Sutton and Barto 2018). The method of Q Learning utilising deep ANNs, often referred to as Deep Q Learning is a perfect method for comparing the ability of SNNs utilising natural learning algorithms. This method is the current state of the art for the Arcade Learning Environment (ALE) (Mnih et al. 2015). Variations of the Deep Q Learning method such as Double Q Learning (van Hasselt et al. 2016) have been able to surpass the original Deep Q Learning methods in the Arcade Learning Environment but has not been widely researched against other problem sets. Deep Q Learning on the other hand is widely used in other reinforcement learning tasks and is therefore more suitable to use as a comparison due to its wider reach and research base.

2.3.1 Back-Propagation

Generally, networks that utilise Deep Q Learning for training, as a part of the training, also use back-propagation for the altering of the weighted connections in the neural network. It works on some loss function or error function calculated at the output layer of the network. The partial derivatives are then calculated for the loss function with respect to every one of the weights in the network. This then alters the weights in the direction of the negative gradient of this partial derivative in order to minimise this loss function (Lee et al. 2016). In large networks, back-propagation can be incredibly time consuming and is therefore usually conducted on the average of these gradients over a large number of training cycles.

2.4 Spiking Neural Network (SNN)

Spiking neural networks (SNNs), although similar in structure to artificial neural networks (ANN), work in a fundamentally different way. Both their structures are made up of several layers of neurons connected to one another through weighted connections or synapses. The difference comes from how the signals are sent down these connections (Mozafari et al. 2018). ANNs will propagate data through a continuous iteration of neuron values down each of their connections every iteration of execution. SNNs differ in that their propagation of data is done through discrete binary spikes. Neurons will build in some potential until such a point that a threshold is surpassed where a spike will be passed down its connections which will work to build or diminish the potentials of those connected neurons (Burkitt 2006). Once a neuron has spiked it will enter a period of refractory where it is unable to spike and ignores any incoming input. Mapping a neuron spiking over time produces a spike train. This temporal information on a neuron's activity can be used to extract information on action selection or classification when they occur in some output layer.

The interaction of neurons in a SNN attempt to imitate the biological processes existent in mammalian brains. The extent to which realism of cognitive neuronal level structure is replicated in these systems brings about several methods of modelling these types of networks. Generally, the more realistic the neurons and synapses in the network are, the more computation is required during processing. This trade-off seems to be a common occurrence in the literature. The most general and computationally fastest method is using the Leaky Integrate and Fire (LIF) neuron model (Stevens and Zador 1998) (Burkitt 2006). The most accurate model originates from the work by Hodgkin and Huxley (1952), known as the Hodgkin-Huxley model (H-H), through their study of membrane capacitance at the ionic level. Due to the complexity of this model, it is considered the most computationally inefficient. Izhikevich (2003) proposes a model to bridge the realism and computational efficiency gap between the LIF and H-H models.

2.4.1 Action Potential and Hodgkin-Huxley Model (H-H)

The neurons in SNNs are based off of neuronal membrane structures described by Hodgkin and Huxley (1952). It is therefore important to have an understanding of what is happening inside a membrane at the point that the membrane voltage surpasses the threshold value. The difference between the voltage inside the membrane compared to outside the membrane at its resting potential is approximately -70mV. When exposed to outside stimulus from another neuron for example, positive charge is added to this membrane making its membrane potential less negative. If this additional positive charge causes the membrane to exceed some threshold (approximately -55mV), the voltage gated sodium channel (VGSC) will open causing sodium ions outside the membrane to flood inside. This creates depolarisation as the charge of the membrane flips from negative to positive and the charge outside the membrane flips to negative. When the membrane potential exceeds the peak voltage approximately 30mV, the VGSC is inhibited preventing more sodium ions from entering the membrane. Both the VGSC and the voltage gated potassium channel (VGPC) are triggered to open at the same time when the membrane potential exceeds the threshold however the VGPC takes a longer period of time to open meaning a delay between the sodium ions flooding through and the potassium ions leaving. When the VGPC is open, as potassium ions are more concentrated inside the membrane, it causes them to flood out rapidly dropping the membrane voltage back into the negative. This process of flipping from positive back to negative is known as repolarisation.

When the membrane potential drops below its threshold both the VGSC and VGPC gates close. Similar to when it exceeds the threshold the VGPC takes longer to close than the VGSC which has already been inhibited after reaching the peak potential. Therefore, potassium ions will continue to travel out of the membrane while the VGPC takes time to close. This drops the potential to less than its resting potential causing the membrane to enter a period of hyper-polarisation. Through the use of separate channels called the sodium potassium pump (SPP)

and the potassium leak channel (PLC) the membrane will return to its equilibrium at the resting potential.

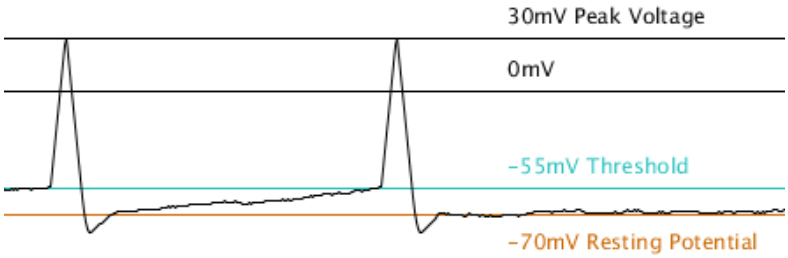

Figure 2.2: *An example of a set of action potentials occurring according to the Hodgkin and Huxley membrane capacitance model.*

Figure 2.2 shows an example of the membrane potential mapped over time in a single membrane. In this example, this membrane has undergone two action potentials. The period of time between when the potential exceeds the threshold through to it reaching the peak voltage and then dropping back below the threshold is known as the absolute refractory period. During this time, the membrane will not respond to external stimulus and cannot undergo another action potential. The period between dropping below the threshold through to the membrane being in a state of hyper-polarisation to it returning to its resting potential is known as the relative refractory period. During this time, the membrane can undergo another action potential if there is enough external stimulus however it requires more input than is otherwise necessary than when the membrane voltage is at its resting potential, so an action potential is less likely.

We can model this entire process for use in an SNN however the model would be required to deal with the movement of ions and timings of opening and closing VGSC and VGPC gates which would add to the computational cost. This model is known as the Hodgkin and Huxley (H-H) model (Hodgkin and Huxley 1952). This high cost has led to the development of simplified models that are able to process large numbers of individual membranes simultaneously as is required to

replicate a neural system.

2.4.2 Izhikevich Model

Izhikevich (2003) identifies that H-H type models are restricted by the amount of computation required to accurately model the neural system. He proposes a model that matches this level of accuracy but removes a lot of the complexity of the ionic level H-H model. The IZ model revolves around two basic piecewise equations.

$$v(t+1) = \begin{cases} v(t) + 0.04v(t)^2 + 5v(t) + 140 - u(t) + I(t) & \text{if } v(t) < 30 \\ c & \text{if } v(t) >= 30 \end{cases} \tag{2.1}$$

$$u(t+1) = \begin{cases} u(t) + a(bv(t) - u(t)) & \text{if } v(t) < 30 \\ u(t) + d & \text{if } v(t) >= 30 \end{cases} \tag{2.2}$$

I refers to the injection into the neuron, v is the membrane potential, u is the recovery variable and t is the current time-step. The other four variables, namely a, b, c and d are the parameters that affect the behaviour of the neuron. Firstly a describes the time scale of the recovery variable with smaller values resulting in slower recovery. Secondly, b describes the sensitivity of the recovery variable to "sub-threshold fluctuations of the membrane potential" (Izhikevich 2003). c determines the reset value after a spike occurs. Finally, d describes the behaviour of the reset of the recovery variable post spike.

$$\begin{aligned} a &= 0.02 \\ b &= 0.2 \\ c &= -65 \\ d &= 2 \end{aligned} \tag{2.3}$$

Equation 2.3 describes typical values found in regular spiking neurons.

When v exceeds the peak voltage of 30mV, both the membrane potential and the recovery variable, u, resets according to specified rules outlined in equations

2.1 and 2.2 respectively.

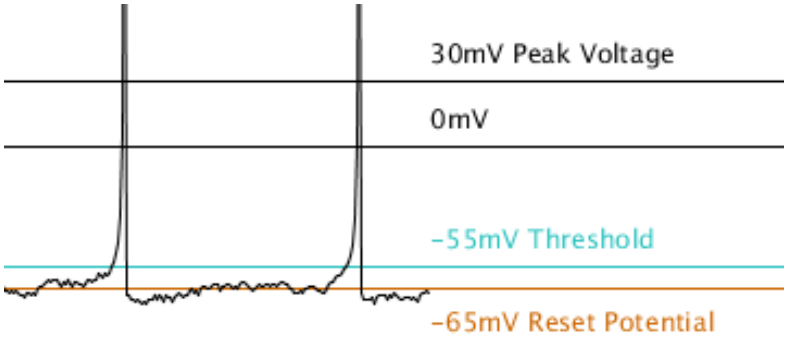

Figure 2.3: *An example of random input into an Izhikevich neuron model with the parameters from equation 2.3*

Comparing Figure 2.2 with Figure 2.3 it shows that the IZ neuron model is good at estimating the same properties of the H-H model. The action potential occurs around the -55mV threshold. Once the potential exceeds some peak voltage it enters a period of hyper polarisation before levelling back off at the resting potential. All this is done without needing to model the complexity of the ionic level movement in the neuron.

2.5 Decoding Outputs into Action Selection

In traditional ANN architectures, the final layer generally uses a soft-max activation function, σ, applied which both normalises the output vector as well as represents the possible action selections based on how likely the network determines the action taken will lead to a maximised value. An argmax function would then be applied to select this highest predicted value action.

However, in a spiking architecture, there is no ability to decode the outputs in this way due to their binary nature. This binary characteristic means that information is not encoded on particular neuron values but in the spike trains that these neurons produce (Kiselev 2016). Therefore, a method is needed to convert this spike train information into appropriate action selection. There are a number

of different ways proposed to accomplish this and this section will investigate these in detail. Fundamentally, these different types for coding can be broken into two main categories: temporal coding and rate coding.

2.5.1 Rate Coding

Rate coding methods rely on the fact that the carriage of information is based on the frequency of spikes occurring in a neuron (Li and Tsien 2017).

Temporal averaging is a method whereby the average firing rate is calculated for an individual neuron based on the time of exposure of the particular stimulus. This can be defined by Equation 2.4.

$$R = \frac{n^{sp}}{T} \tag{2.4}$$

Where R is the average firing rate of a single neuron over a single trial, n^{sp} is the number of spikes over the time of exposure of a particular stimulus and T is the total time of exposure of the stimulus. The output neuron that produces the largest of these temporal averages will be used for action selection. In order to produce more accurate temporal averaging, the time of exposure of the stimulus can be increased. It needs to be sufficiently long to be able to infer relevant information. This method can experience issues with the possibility of the same temporal average over multiple output neurons especially when the time of exposure of the stimulus is small.

This can be improved by analysing the average spike rate of a single neuron over multiple trials as the noisy nature of input to the network may make spike rates differ from one trial to the next. Taking the average spike rate over multiple trials, as described in Equation 2.5, allows for a more accurate comparison of multiple output neurons when deciding on an action to select.

$$R = \frac{\Sigma_{i=1}^{k} n_i^{sp}}{Tk} \tag{2.5}$$

The variable k in Equation 2.5 refers to the number of trials of an exposure

of a stimulus to a single neuron. This method is almost the same as Equation 2.4 however results in the consideration of multiple separate trials to come to a conclusion of which action to take. In real neural systems it is not possible for this to be the coding method for deciding action selection as a decision is made with only one exposure to an event. However, in artificial systems this can be used to ensure the correct action is taken when using rate coding. This is a slow process due to the repeated trials of multiple exposures of a stimulus. Population averaging is similar to that of the temporal averaging over trials method however it is based on the assumption that there is a pool of neurons responsible for action selection. If we find the average rate of firing over that population of neurons, we can get a more accurate reading of which pool is being the most stimulating over a certain time period. This allows for the improvement of accuracy over the temporal average method while the network only needs to be exposed to a single trial.

2.5.2 Temporal Coding

Temporal coding assumes that the carriage of information is dependent on the precise spike timings (Li and Tsien 2017). This means exposure of a state for a fixed time period is not needed in order to appropriately select an action to take. First-to-spike time coding selects the action based on the neuron that was first to fire since the exposure of the new state to the network (VanRullen et al. 2005). This improves on the number of exposures needed using the rate coding methods due to the action being selected as soon as a single spike occurs in the output layer. However, it is most likely not the biological method of decoding action selection as the individual neurons may not have any reference to the behaviour of other neurons tasked with action selection and therefore would not know which of them are first to spike (Li and Tsien 2017).

2.6 Summary

This chapter outlined the underlying definitions and descriptions of the concepts outlined in the research questions. This included a brief overview of the current state of the art methods of traditional ANNs for the types of RL problems that will be investigated in the following chapters, and an introduction to the SNN structures and mechanisms for modelling. There are multiple ways of modelling these types of networks with the identified IZ neuron model being selected for use in this research. This selection was justified on the improved biological accuracy of neuron behaviour when compared to the LIF neuron model, with the more efficient execution of the network on traditional hardware when compared to the H-H neuron model. Additonally, the justification to analyse both temporal and rate based decoding methods was based on their consistent and non-comparitive use in the literature. The background laid out in this chapter therefore furthers the framework of the research that will be conducted in future chapters.
The following chapter will take a more extensive look at the literature, specifically related to the applications of the processes outlined in this chapter. It will identify certain open questions in the field and relate them back to the specified research questions from Chapter 1.

Chapter 3

Literature Review

The previous chapter outlined the basic concepts needed to understand the research questions. One thing that the previous chapter lacked was a detailed description of training algorithms for SNNs and therefore requires an investigation in the form of a literature review. This chapter reflects the current state of research into the use of SNNs for a range of machine learning tasks. Although supervised and unsupervised learning problems are explored, the focus is on their use in RL. Firstly, a review was conducted into the wider use of SNNs throughout machine learning domains. Secondly, a more specific review into virtual agent control using SNNs was conducted. These reviews have been merged into the relevant headings throughout this chapter. The major findings of these reviews highlight the minimal research into complex RL domains with the common occurrence of difficulty of training of the networks. They also hint at various gains in efficiency and generalisability compared to ANNs however these also have minimal research focus in the current literature. The chapter will conclude with a discussion of the identified open questions in this field.

3.1 Computational Efficiency of SNNs

The benefits of utilising networks of spiking neurons over those of continuously activated neurons found in traditional ANNs seems like a logical computational

efficiency gain as there are less hardware operations being conducted in order to propagate the data through the network (Tavanaei et al. 2019). Experimentally, Neil et al. (2016) demonstrates how, for classification tasks, using an appropriate optimization algorithm for converted deep SNNs, far less computation is needed to maintain similar levels of accuracy of traditional ANNs. Although these SNNs did not undergo direct training but instead were converted from a trained ANN, it highlights the computational efficiency of execution but not training. They explore several different algorithms and their normalised counterparts to determine which, if any, are able to both maintain 98% accuracy on the MNIST Digit data set (Lecun et al. 1998) and are able to execute the classification in a more computationally efficient manner. Additionally, Cao et al. (2015) demonstrated that converting a trained convolutional ANN into a network of spiking neurons, they were able to maintain the accuracy in a image recognition task whilst allowing the model to run much more efficiently on specific spike based hardware. Due to the structure of a SNN, those that spike less are more computationally efficient. Comparing that to an ANN where all neurons need to send their activation values down all of their weights, a SNN only needs to send that information when a spike occurs. The most interesting question however comes from whether the networks themselves are able to be trained more efficiently than an ANN which neither of Neil et al. (2016) and Cao et al. (2015) do. There has been minimal research into this direct computational comparison of training. This has been identified as a major research gap and is most likely caused by the difficulty of training SNNs due to their vastly different architecture to ANNs.

3.2 Hand Crafted SNNs

A number of papers utilise a hand-crafted SNN with none of the following experiments undertaking parameter optimization or learning during execution. However, it is still worth investigating the range of problems that SNNs of a hand crafted nature are able to solve.

Gamez et al. (2013) attempt to craft a SNN capable of deceiving judges in a Turing test like experiment. It utilises the Unreal Tournament 2004 video game (Tournament 2004) with its objective of replicating human like behaviour in a first person shooter environment. They identify the subjectivity of using human judges to determine how human like the artificial agent is, and instead proposes human like metrics collected through observations and data from a range of human players. They then craft their SNN agent to maximise these human-like metrics. The tedious nature of hand crafting these parameters and network structure limits the optimisation ability. This was identified by Gamez et al. (2013) for possible improvements to their global workspace architecture in future by implementing forms of parameter optimization and learning.

Mitchell et al. (2016) demonstrate the use of a SNN for planning in a simulation where a robot must do a number of sub objectives in order to achieve an overall objective of sanding and spray painting an object. The objectives must be achieved in the correct sequence so the hand crafted networks is broken into modules for each sub objective. This is a similar structure to the global workspace architecture proposed by Gamez et al. (2013). They were able to craft a network capable of achieving the desired tasks however neuron and synaptic parameters were static meaning no training was undertaken. Should the objectives of the system change, the network would have to be recreated according to another set of plans.

Kamali Sarvestani et al. (2013) attempted to replicate types of instinctual behaviours in lamprey, a very basic ancient sea animal that is believed to have exhibited limited behaviours. Using a hand crafted SNN in a form similar to a Braitenberg machine (Braitenberg 1986), they were able to exhibit behaviours of escape, avoidance and approach of certain environmental stimulus. Similarly to Gamez et al. (2013) and Mitchell et al. (2016) it did this through tiers of sub modules with varying degrees of importance.

Olmsted (2011) replicated the behaviours of a sea snail foraging for food. They were able to imitate exploratory behaviours for searching for food and moving towards food sources when they were detected. They structured their network

by hand crafting neuron parameter values to imitate AND, INCLUSIVE OR and other conditional logical gates. When arranged in their network it exhibited the instinctual behaviours identified.

All of these papers successfully implement a SNN for agent control in a virtual environment, however, all had static, hand crafted parameters with no ability to improve the performance of their tasks. They demonstrate the capability of networks to bring rise to interesting behaviour, but for future progress in the use of SNNs it is imperative that natural learning algorithms can be applied and demonstrated to improve on a set of metrics related to how well they completed these tasks.

3.3 Hebbian Based Learning

As SNNs offer a more biologically plausible neural network architecture (Shim and Li 2017), many have looked to biology and psychology research to understand methods of training an architecture of this kind. Hebbian Learning (Hebb 1949) has emerged as the leading basis for a lot of training algorithms for use on SNNs due to its theory on the structure of learning in mammalian brains. Hebb posits that when a pre-synaptic neuron fires which in turn leads to the firing of a post-synaptic neuron, the connection between the two is strengthened. Inversely, if a post-synaptic neuron has fired without the assistance of a pre-synaptic neuron then the connection between those two is weakened (Hebb 1949). This simple theory is relevant to SNNs due to them being a computational representation of the biological mammalian brain. This has become known more commonly as Spike-Timing-Dependant Plasticity (STPD) (Mozafari et al. 2018) (Bing et al. 2018) and has led to several spin-off algorithms based off the original contributions by Hebb (1949). STDP itself is actually an unsupervised learning algorithm which works well at identifying trends and patterns in unlabelled data (Shim and Li 2017). However its reward-modulated STDP counterpart (R-STDP) has been proposed for reinforcement learning tasks (Mozafari et al. 2018) (Bing et al. 2018).

Both Mozafari et al. (2018) and Bing et al. (2018) demonstrate the effectiveness of using the R-STDP variations of the unsupervised STDP. Bing et al. (2018) utilise the method in a real world robotics task with the goal of training a robot to stay within its lane markings on a variety of different tracks. They demonstrate the effectiveness with a comparison to an experiment conducted by Kaiser et al. (2016) where they use a robot utilising a SNN to do a similar task. The only difference being that Kaiser et al. (2016) used hand crafted weights and feature detectors to create their network as they claimed that there were no good optimisation algorithms developed for SNNs. Bing et al. (2018) demonstrates that through the use of R-STDP for training of a spiking neural network it was eventually able to well outperform the hand crafted vehicle on the metrics of staying within its lane markings. Even when used on ever increasing complexity tracks which utilised different types of lane markings at different points in the track it was still able to maintain a closer fit to the lanes center line than Kaiser et al. (2016). Mozafari et al. (2018) demonstrates the same training algorithm as Bing et al. (2018) however does so on a natural image classification task. Their results demonstrate four fundamental properties of using the SNN with R-STDP when compared to the traditional methods for natural image classification. Firstly that it was able to match the robustness of current state of the art methods for image classification. Secondly, as the SNN utilises only one spike per image per neuron, it greatly reduces the computational cost of classification, a characteristic backed up by Neil et al. (2016). Thirdly, the classification is done in a more biologically plausible and similar way to the methods used by human brains to classify images. Finally, the use of R-STDP means the networks learn in more biologically plausible ways, as detailed by Hebb (1949), when compared to the traditional deep learning methods. R-STDP can be applied to SNNs for a variety of tasks as demonstrated through the robotic control of Bing et al. (2018) and the image classification performance of Mozafari et al. (2018).

Shim and Li (2017) builds on the research of the R-STDP algorithm with their proposition of the multiplicative reward modulated spike-timing-dependant

plasticity (M-RM-STDP). They demonstrate this algorithm on a robotic collision avoidance task whereby they place the robot in a starting location with a variety of obstacles in the environment with the task of the robot reaching a target location. Shim and Li (2017), as a comparison to their algorithm, compare the traditional q-learning methods applied to ANNs. Their results demonstrate that their robot that learnt through M-RM-STDP far outperforms the q-learning robot. Their use of a product of multiple components to control the degree of changing of the weights in their network led to shorter learning speeds than additive and base reward modulated STDP algorithms. Shim and Li (2017) demonstrates the computational efficiencies of using their proposed variation of the STDP algorithm, as well as its performance gains on traditional methods of robot collision avoidance therefore highlighting the importance of more research in this field.

Wang et al. (2016) proposes the use of a simplified Hodgkin-Huxley membrane model with comparable efficiencies to that of the Leaky Integrate and Fire neuron. Using this method they utilise a Q-Learning type training algorithm to teach a virtual agent to play the game of Flappy Bird. However the details of the precise workings of this method in the confines of a SNN is minimal. They propose an interesting use of time delayed neurons to produce contextual state input. The first of two neurons in a perception layer are fed spike inputs from the pixels of an image with the second receiving similar spike information but channelled through a time delayed neuron. This produces contextual information about general movement within the game world. This is similar to the method used in traditional ANN input pre-processing for temporal video input where several previous frames are passed into the network or some subtraction of frames occurs.

Daucé (2009) suggests utilising a local policy gradient learning mechanism with global reward. Where regular policy gradient RL algorithms use value functions to predict future rewards from some specific state and action pair, this method avoids the use of the value function by directly shaping the parameters using a local estimation of gradient at the individual neuronal level. The benefits of using localised learning means that the only information needed for learning is the

neurons firing pattern as well as the firing pattern of its pre-synaptic neuron. The single global reward can be spread across all neurons for the localised learning to take place in a Hebbian fashion. Using this method, Daucé (2009) was able to train a virtual eye to focus on some target in the environment with no information regarding the direction to the target, only a global reward signal related to how well it was looking at the target area.

Wunderlich et al. (2019) demonstrated the training of an agent on simple neuromorphic hardware to play a basic game inspired by Pong whereby the agent must bounce a ball off its paddle and into a wall and repeat trying to prevent the ball from passing its paddle. Rewards were allocated based on the centrality of the ball compared to the agents paddle. Using a R-STDP algorithm for neuronal level learning they were able to train an agent to play the game optimally. Interestingly they implemented this algorithm both on simple neuromorphic hardware and a simulated neuromorphic environment running on a traditional CPU computer. The neuromorphic hardware was able to learn the optimal strategy in just 25 seconds compared to 40 minutes in the simulation. Although this game was a simplified version of pong it highlights the extraordinary efficiency gains neuromorphic hardware could bring for problems of this nature. Future implementations on a full version of the game of Pong using pixel inputs implemented on neuromorphic hardware, we believe, would be a first.

Müller et al. (2018) used R-STDP for a navigation and collision avoidance task in a virtual field, attempting to replicate learning conditions of the honey bee. Their R-STDP algorithm only had an affect on the final hidden layer's synapses with the output layer. This method was able to replicate simple navigation of the honeybee in environments with minimal obstacles. When expanded to more complex paths in more complex environments, their network struggled to effectively navigate. They identify that their network is not suitable for those complex paths and that separate network modules would be needed to improve the navigations systems.

Vitanza et al. (2015) implement multiple SNN agents in an environment with

the task of reaching certain objectives before other agents. Similarly to Gamez et al. (2013) and Mitchell et al. (2016), multiple network modules are used to support different sub-objectives. Due to the competitive nature of this environment with the exposure to multiple other agents, a sub-objective of not colliding with other agents is necessary. They implement a collision avoidance block which contains sensory inputs that spike when nearby an obstacle with a separate block of inputs dealing with information about its wider environment. The agents behaviour in the environment is altered through traditional means of STDP however it makes use of a global reward signal which controls threshold adaptation. This is the only one of the reviewed papers to train neuron thresholds. The use of these techniques and the exposure to other agents in the environment form some predictive behaviour whereby agents will pick objectives that they assess other agents are not travelling to. This predictive behaviour is interesting and unique amongst the other papers reviewed.

3.4 Back-propagation of Spiking Neural Networks

As back-propagation has been such a successful method of optimising artificial neural networks in the past, being able to apply these methods to networks of spiking neurons is important. As explained, SNNs are not intrinsically differentiable and therefore in their basic form are unable to undergo the process of back-propagation. Wu et al. (2018) was able to overcome the non-differentiable nature of the Leaky IF neuron model by approximating a derivative based on spike characteristics such as neuron potentials and spike frequency. They were able to match performance of a supervised ANN on an object detection task. Lee et al. (2016) using a similar method was able to, with a convolutional spiking neural network, outperform a convolutional ANN in the MNIST digit dataset (Lecun et al. 1998). Back-propagation is still a viable option. By estimating derivatives we can still use gradient descent methods in order to optimise the weighted connections. However, if interested in replicating biological processes, it seems evident that back-propagation is in no

way a natural process used in mammalian brains (Tavanaei et al. 2019). Although back-propagation has been recently successful in supervised learning tasks when applied to SNNs it will be ruled out of scope due to the complications in selecting appropriate estimates for the derivative as well as the method being an unlikely natural process used in the human brain.

3.5 Evolutionary Algorithms applied to Spiking Neural Networks

Evolutionary and genetic algorithms on neural networks have been promisingly used to solve complex optimisation problems in the past. Generally, experiments surrounding evolutionary algorithms and their use on ANNs have been divided into two areas. Namely, the optimisation of the weights in a static neural network (Yee and Teo 2011) and the optimisation of the structure in a topologically changing network. Often, in the case of the later, both optimisation of weights and structure is used (Hausknecht et al. 2012). Although evolutionary algorithms are not specifically a reinforcement learning algorithm, they can be used to solve reinforcement learning tasks with the fitness of the individual agents within generations being modelled off of the reward signals that are awarded during play. Their use on spiking neural networks is demonstrated by Yee and Teo (2011) on their application to a driving simulator called TORCS whereby they were able to train an agent to drive three increasingly complex tracks. However their arbitrary use of reward factors influencing the agents fitness highlights the issues surrounding multi-objective optimisation in evolutionary algorithms. Slade and Zhang (2018) attempt to implement a topological and weight evolution on an SNN to implement the XOR logical gate. Although their results were promising in that they were able to create systems that could classify according to XOR, the size and complexity of the networks created where worse than ANN implementations of the XOR network.

Markowska-Kaczmar and Koldowski (2015) in their use of a SNN for a top-down racing simulator use an evolutionary algorithm to optimize the Izhikevich (IZ) neuron model parameters. Interestingly this is the only one of the reviewed papers to include a comparison to a traditional ANN method also utilising a genetic algorithm for training. They first trained both the SNN and ANN on a simple racetrack with gentle and uncomplicated curves. The average fitness of the ANN was always better than the agents using SNNs. The trained networks were then exposed to two more complex tracks where the SNNs faired much better than the ANN counterparts. This demonstrates the greater general ability of SNNs interacting with unforeseen environments where as the ANNs were simply optimising the environment that they are exposed to. Further exploration of this idea is important in the search for general AI as, like biological systems, being able to quickly adapt to new environments from past experiences is an important ability. However the nature of the fitness function is limiting in this particular experiment due to it being calculated off a predefined optimal racing line, instead of the agents being exposed to more complex learning through more need for exploration.

Eskandari et al. (2016), similarly to Markowska-Kaczmar and Koldowski (2015) evolve the IZ neuron parameters in their experiment of simple creatures exploring a 2-dimensional environment searching for food. They test both the evolution of a large colony of different more complex neuron structured creatures with colonies of multiple identical simpler creatures. They found that utilising the multiple colony approach leads to a more expansive exploration leading to better outcomes quicker. Even though the complexity of the SNNs were greater in the individual creature trials, that complexity did not assist in its ability to perform better. However, both sets of experiments were trained for the same number of trials. They did not test whether the individual agents could surpass the abilities of the simpler creatures if given more time to explore the environment and its neuron parameters.

Evolutionary algorithms work well at optimising the performance of SNNs. They demonstrate the generalisability of SNNs as well showing that more complex network structures do not give rise to better performing agents necessarily.

3.6 Coding Strategies

In order to address RQ3, research was conducted into coding strategies, both decoding and encoding, adopted by SNNs used in literature. Biologically, Vreeken (2003) suggests rate coding methods are incompatible with our understanding of neurons due to the speed that they process information. It is suggested that for a human to recognise a face for example, ten synaptic steps are made from the retina to the temporal lobe with each step taking around 10ms. Vreeken (2003) therefore suggests that "such a time-window is much too little to allow an averaging mechanism like rate-coding". Qiu et al. (2018) suggests overcoming this hurdle by introducing an underlying current throughout the entire network in order to produce more spiking activity therefore any external stimulus will change the spike rate in the output layer even in short exposure periods. Eskandari et al. (2016) overcomes the limitations of rate decoding by having very long state exposure periods with their creature only selecting actions every 600 network steps. Additionally, Bing et al. (2018) proposes and demonstrates that the spiking rates of output neurons can be used in continuous action spaces where they changed the turn speed of a motor based on the rate of firing. Rate coding seems to be a viable strategy for decoding information from a SNN in artificial systems albeit by implementing strategies to overcome the limitations proposed with short exposure periods.

Comparatively, VanRullen et al. (2005) indicates that there is evidence in certain locations in the brain that temporal coding methods such as first-to-fire (F2F) are used by animals. This suggests that in some instances, the precise spike timings may be important in deciding actions. They also discuss issues with this method when a continuous sensory input is being fed into the network that the networks that use F2F may need to be reset or deactivated in between action selections to prevent the incorrect firing of other neurons. If this is the case then there needs to be some internal signal that is connect to an entire neural system to trigger this resetting or deactivating behaviour (VanRullen et al. 2005).

3.7 Discussion and Open Questions

This section will touch on some of the recurring questions posed throughout the reviewed papers. This includes encoding and decoding of temporal spike train information, the effect of noise on the system, the possible use of stochastically firing neurons, threshold adaptation and the complexity of both the domains and the models used.

3.7.1 Encoding

It is necessary to encode an analogue signal and convert them into a form that is compatible with a SNNs spike train input. States of a virtual environment often involve the use of large numbers of scalar values representing things such as pixel colour and intensity or distance and direction to an object in the world. These must be converted to some temporal mapping of that scalar in the form of a spike train. Often this is done by assuming the value is some mean firing rate and using that to generate a Poisson distributed spike train (Wang et al. 2016)(Gamez et al. 2013) often called rate coding. However, having some randomness involved with the encoding of the input will result in noise being injected into the system. More research is needed into the most effective and worthwhile methods of encoding input in SNNs.

3.7.2 Decoding

A SNNs output layer must be decoded from the temporal mapping of its spikes into some action selection. This can be done in an opposite way to rate coding whereby the mean firing rate can be extracted from a certain time window with the most frequently spiking output neuron, which is mapped to some action, will be selected (Gamez et al. 2013). This method would introduce a kind of reaction time in the system depending on the time window used to find the average firing rate. There are occasions where these mean firing rates will be identical depending on the time window used for extracting the mean firing rate, or the existence of noise in the system.

3.7.3 Noise

Although the effect of noise in a SNN is an open question, several of the reviewed papers identify that the noise forces exploration in the environment. Exploration is an important aspect of optimising agent control in a virtual environment and is one of the cornerstones of reinforcement learning Sutton and Barto (2018). Additionally, this noise can help to decrease the likelihood of identical output decoded firing rates when making a decision on the action to take. Olmsted (2011) and Daucé (2009) deem noise to be a crucial aspect of their networks so much so that it is separately injected through the use of randomly spiking neurons integrated into the system.

3.7.4 Stochastically Firing Neurons

The three definitions for SNN models, namely the LIF, IZ and the H-H models, will deterministically fire the neurons every time their neuron potential exceeds their threshold. This means that when the network is exposed to the same inputs over multiple occurrences, it should fire in the same way every time. There is evidence that the human brain does not work entirely in this way. It appears that when analysing individual neurons when exposed to the same stimulus on multiple

occurrences, there is noise that appears in the resultant spike trains (Buesing et al. 2011) (Domingos 2018). This would imply that biological neural systems are not deterministic and instead stochastic. There are suggestions that these stochastic properties emerge from complex biological phenomena, difficult to model in computationally efficient ways (Tuma et al. 2016). Simplified methods of modelling this noise can be done by making the threshold not a trigger but some probabilistic region, whereby the closer the neuron potential is to the threshold the higher the probability of spiking. Daucé (2009) highlights the importance of stochastically firing neurons for exploration and identifies that noise can contribute to the stochastic nature of firing. So instead of utilising a probabilistic region for neuron threshold, injecting noise is enough to produce randomly firing neurons. It is worth investigating whether probabilistic neuron thresholds could replace the supposed need for noise injection for agent control in virtual environments.

3.7.5 Threshold Adaptation

Vitanza et al. (2015) is the only paper reviewed that both trains the synaptic weights as well as the threshold values. They demonstrated that utilising reward based threshold adaptation with STDP for weight training showed much better results than exclusively using STDP. Applying threshold adaptation along with other forms of synaptic weight training such as evolutionary or R-STDP algorithms would be a first for agent control in virtual environments and is an important area of future research.

3.7.6 Model Complexity

Many of these papers conduct their experiments in neuromorphic simulations as opposed to directly implementing them on neuromorphic hardware. Wunderlich et al. (2019) conducted the only experiment in which their algorithms were tested on neuromorphic hardware. They observed that their training algorithm performed almost 100 times faster on neuromorphic hardware as it did in neuromorphic sim-

ulation. This demonstration of large efficiency gains when compared to the neuromorphic simulation shows the greater need to test algorithms of these forms directly on neuromorphic hardware and compare their efficiencies to similar ANN training algorithms. Additionally, the majority of the papers utilised small SNNs with less than 100 neurons. Comparing that to state of the art ANNs for virtual agent control whereby they often utilise hundreds of thousands of neurons (Silver et al. 2018), it is crucial that networks of this size with several hidden layers of neurons be tested to see if the training methods outlined continue to function in deep SNNs. The wider use of neuromorphic computing will assist in testing networks of these comparable sizes.

3.7.7 Domain Complexity

Largely, the tasks attempted by the reviewed papers are simple virtual environments with minimal objectives. Several attempted to replicate basic animal behaviour such as foraging for food (Kamali Sarvestani et al. 2013) (Olmsted 2011) or navigation and collision avoidance in relatively uncomplicated environments (Müller et al. 2018). Video games that were utilised were simplified versions, fully observable and deterministic environments (Wang et al. 2016) (Wunderlich et al. 2019). There are no examples of environments of a stochastic nature and those that are partially observable contain simplified input such as with (Gamez et al. 2013). Demonstrations on more complex environments are important for the testing of the promising algorithms demonstrated in the papers reviewed. In order for the uptake of neuromorphic computing for complex virtual spatio-temporal problem sets to be widely investigated by the research community, the demonstration of these SNN optimisation strategies need to be applied to problems of greater complexity. Future work is needed to implement efficient algorithms for the training of an agent to play a variety of the Atari 2600 suite of games (Mnih et al. 2013) and to be able to match performance of current state of the art ANN methods. Exploring a SNNs ability to generalise gameplay across a number of these games is also an important future step.

3.8 Summary

This chapter outlined and reviewed the current state of the literature in the field of SNNs. It looked at both their general use in the wider machine learning domain, as well as their specific use in the RL domains. Various open questions were identified, most of which will lie outside the scope of this research however, were notable for possible future work. Questions over encoding and decoding methods as well as the effect that the length of exposure (otherwise referred to as the state exposure period) has on network performance are questions that fall within the scope of this research. This further strengthens the framework for what should be tested when experimenting with the SNNs. In order to answer those open questions, a comparitive analysis of the different identified methods are needed. It is often the case in the literature that the author will select a type of encoding or decoding method and a certain state exposure period without much justification. This furthers the significance of this research as comparatively analysing the performance of these different methods will allow for a more informed selection of the components of the network architecture before applying it to their specific problems. Additionally, depending on the types of encoding methods used, the open question on noise in the network will be relevant. Through testing, identifying which encoding methods produce the most noise as well as the effect that the noise has on performance will help with answering that open question. The following two chapters will attempt to address some of these open questions through experimentation as they relate to the research questions identified in Chapter 1.

Chapter 4

Methodology

This chapter contains a description and explanation of the methodologies, techniques and code-base used for the Experimentation chapters to follow. It includes a detailed description of the virtual environment problems used, the methods of encoding and decoding, a definition and description of the state exposure period and the code base used for experimentation. More specific explanations supporting individual experiments will be covered in the appropriate sections in Chapters 5 and 6.

4.1 Problem Descriptions

This section outlines the two virtual environments used during testing. The first subsection is a description of the Cart Pole problem and the second subsection describes the Lunar Lander environment supplied by the python OpenAI Gym (Brockman et al. 2016) framework.

4.1.1 Cart Pole

The goal of the cart pole problem, also known as the inverted pendulum problem, is to balance an upright pole on a cart that can either move left or right at each step in the episode. The episode is terminated if the angle of the pole is greater

than 12 degrees from the Y axis, the cart reaches further than 2.4 units from the origin point or the episode length is greater than 200 steps. A reward of 1 is received by the agent for each step of the episode including the terminating step. The problem is considered solved if an agent receives a total episode reward of greater than or equal to 195 over 100 consecutive episodes. However, the first experiment conducted had an episode length of 1000 and an average reward of 900 or more over 100 consecutive episodes as the goal. This was changed for future experiments to bring it in line with benchmark comparisons in the literature as well as decreasing the time taken to run each experiment. The Cart Pole environment produces a state space of four float values representing the cart position from center point, the cart velocity, the pole angle and the current velocity of the tip of the pole. Negative state space values represent the current state in terms of movement to the left and positive to the right. With an Izhikevich neuron model, negative inputs mean that the input neuron will rarely, if ever spike and will therefore not pass this information onto future layers. It is therefore required to eliminate negative values from the state space. This removal of negative input was handled differently in different experiments and will be addressed in those applicable sections. A screenshot of the virtual environment is provided in Figure 4.1 and the original state space is described in Table 4.1.

Input Num	**Description**	**Min**	**Max**
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	-0.209 rad (-12 deg)	0.209 rad (12 deg)
3	Pole Velocity at Tip	-Inf	Inf

Table 4.1: *The original state space that the environment produces*

The restrictions on the cart position and pole angle are due to the failure of the trial if they are to exceed those values, i.e., either the cart hits the edge of the screen or the pole has fallen over.

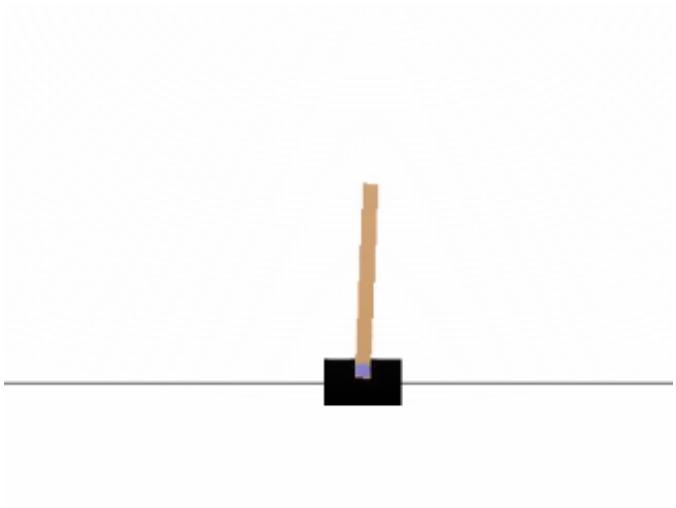

Figure 4.1: *A screen shot of the Cart Pole experiment.*

4.1.2 Lunar Lander

The Lunar Lander problem requires the landing of a lunar module in a 2D environment within a landing zone. The landing zone is always in the same position however the surrounding landscape is procedurally generated in each episode. It includes four actions being the left, right and main thrusters as well as an action for doing nothing. The episode terminates if the lander crashes or comes to rest. A reward of 200 or greater over 100 consecutive episodes is required for the trial to be a success. Around 100 to 140 reward comes from moving from the top of the screen to the bottom and also for having zero speed. An additional 10 reward points come from each leg making contact with the ground. If the lander body touches the ground, it is considered crashed and gets -100 reward and if both legs are in contact and the body is at rest, an additional 100 points are awarded. Similarly with the Cart Pole problem, input values that could be negative or positive need to be transformed into a different set of inputs to remove negative values when using this problem with the Izhikevich neuron model. The transformation methods used during experimentation are detailed in the next section. A screenshot of the lunar lander environment is provided in Figure 4.2 and the original state space of the lunar lander problem is outlined in Table 4.2.

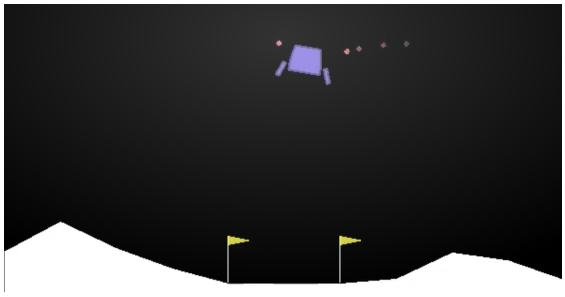

Figure 4.2: *A screen shot of the Lunar Lander environment.*

Input	Description	Min	Max
0	Module X Position	$-\infty$	∞
1	Module Y Position	$-\infty$	∞
2	X Velocity	$-\infty$	∞
3	Y Velocity	$-\infty$	∞
4	Module Rotational Angle	-π	π
5	Module Angular Velocity	-∞	∞
6	Left leg contact with ground	0	1
7	Right leg contact with ground	0	1

Table 4.2: *The transformed state space for Lunar Lander to remove negative inputs.*

4.2 Input Encoding

As briefly described in the previous section, state spaces that include negative values must be transformed such that all inputs into the SNN using Izhikevich model neurons are positive. Two methods of transformation are discussed and used in the experimentation including binary encoding and double encoding. This section will outline these methods and demonstrate the transformation of the state spaces on the two problems utilised in this research. The equation detailed in Equation 2.1

does allow for spikes to occur with negative input however it requires much more negative stimulation than it does positive. Additionally, this was posited to be an unintended consequence of the Izhikevich formula as it does not appear to occur in biological neurons. Therefore, some form of transformation of these inputs needed to take place. The initial thought was to take the absolute value of the inputs in order to transform all to positive values however the sign of all inputs is important information in determining which direction you would want to move either the cart or the lunar module in. So absolute values would remove that information from the network. In order to preserve the sign whilst keeping all inputs positive, two methods of input transformation have been applied and tested.

4.2.1 Binary Encoding

The idea behind binary encoding, a method that did not appear in the literature, was to double each input for both tasks with the first input of each pair becoming the magnitude of the original input value, and the second becoming a binary signal where 0 is used for a negative value or occurences to the left of the environment and a 1 representing an input value that is positive or to the right of the virtual environment. An example of transforming the cart pole state space is outlined in Table 4.3.

Input	**Description**	**Min**	**Max**
0	Cart position	0	2.4
1	Cart right of center	0	1
2	Cart velocity	0	∞
3	Cart moving right	0	1
4	Pole angle	0	0.209
5	Pole leaning right	0	1
6	Pole velocity at tip	0	∞
7	Pole velocity to the right	0	1

Table 4.3: *The transformed state space in order for the inputs to work with the SNNs*

This binary encoding method was used for the intial experimentation in the cart pole environments however from more detailed analysis of the literature, a method of double encoding was utilised for the majority of experiments.

4.2.2 Double Encoding

Double encoding (Wiklendt et al. 2009)(Markowska-Kaczmar and Koldowski 2015) similarly splits a single input value where there is a possibility of a negative value and replaces it with two input values. The first input represents the positive magnitude of the input value, like in binary encoding, and the second input represents the negative magnitude of the original input value. If, for example, a value of 1.3 was parsed to the network as one of the state space input values, the first input of the pair would be 1.3 and the second input of the pair would be 0. If the value was instead -1.3, the first input would be 0 and the second would be 1.3. A demonstration of the input transformation of the lunar lander environment is detailed in Table 4.4.

Input	Description	Min	Max
0	Positive X position magnitude	0	∞
1	Negative X position magnitude	0	∞
2	Positive Y position magnitude	0	∞
3	Negative Y position magnitude	0	∞
4	Positive X velocity	0	∞
5	Negative X velocity	0	∞
6	Positive Y velocity	0	∞
7	Negative Y velocity	0	∞
8	Positive module rotational angle	0	π
9	Negative module rotation angle	0	π
10	Positive module rotational velocity	0	∞
11	Negative module rotational velocity	0	∞
12	Left leg contact with ground	0	1
13	Right leg contact with ground	0	1

Table 4.4: *The transformed state space for Lunar Lander to remove negative inputs.*

4.3 State Exposure Period

As SNNs are required to produce spike trains for action selection, it is necessary to expose networks to a particular state for a certain period of time. Often this is described in millisecond exposures however for the purposes of this research, the state exposure period will simply reflect the number of network execution cycles that the network undergoes over the course of a single step in an episode. A single network execution cycle occurs as follows.

1. The episode state is exposed to the input neurons.
2. The input neurons calculate their new potentials based on the injection from the episode state.
3. The outputs (either 0 if no spike occured or 1 if a spike did occur) from the current layer are calculated and their injection into the next layer connected neurons is multiplied by the connection weights.
4. The injection is then received by the next layer neurons causing the calculation of the next layer neuron membrane potentials.
5. Repeat steps 3 and 4 until reaching the output layer.
6. The output layer membrane potentials are calculated and any resulting spikes are recorded.

This network execution cycle is repeated n times, where n is the state exposure period. When the number of execution cycles that are requested are run, or a condition of one of the decoding methods is met, the spike trains of the output neurons will then be used to determine which action the agent should take.

4.4 Output Decoding

As described in Section 2.5, decoding methods for SNNs can be broken into two categories, namely temporal and rate coding. This section will discuss in more

detail the specific decoding methods used during experimentation. In all decoding methods used, each action that the agent can take in the environment is associated with a single Izhikevich neuron in the output layer of the network. During each of the episode steps and the time period of exposure, the output neurons' action potentials are recorded and analysed to determine the action to take. The first to fire (F2F) and rate decoding methods will be explained in more detail in the following as well as the outlining of both of their reset variant decoding methods. In the following experimentation chapters, the decoding method used in each of the experiments will be explicitly stated in both the graphs used in the results as well as during the experiment explanations.

4.4.1 First To Fire (F2F)

The first to fire (F2F) decoding method is a temporal coding method whereby the specific spike timing of an output neuron determines the action to be taken during that episode step (Li and Tsien 2017). During the exposure period of a single step in the environments episode, the first output neuron to undergo an action potential will cause the network to select the action that is associated with that output neuron. As soon as this action has been determined, the agent will provide the RL environment with the action to take. This agent will then receive a reward dictated by the rules of the problem and a new episode step will begin. The exposure period of the episode step to the network will determine the maximum length of time that the network will wait for an output spike. If no spike occurs in this timeframe, then a random action will be selected and the episode will progress to the next step. If an output spike does occur within the timeframe, then the episode step will be progressed before the entire exposure period has elapsed. If two or more output neurons spike on the same network time-step, then a random action will be selected out of the output neurons that spiked.

4.4.2 Rate

The rate decoding method assumes that the carriage of information in the SNN is dependent on the frequency of spikes occurring over the state exposure period. Similarly to F2F, rate decoding has a single output neuron associated with each of the possible actions that an agent can take within the environment. When the full state exposure period has elapsed, the output neuron that has spiked the most frequently over that period will be the action that is taken by the agent. If no output neuron was to fire over that period, then a random action would be selected and the episode would progress to the next step. If two or more output neurons were to spike with the same frequency over the state exposure period, then a random action will be selected from the set of output neurons that spiked the most frequently.

4.4.3 F2F Reset and Rate Reset

The reset variants of both F2F and rate decoding methods will clear the neurons within the SNN of the residual charge remaining from the previous steps in the episode whenever an action is determined. For example, using the normal F2F decoding method, after one output neuron fires, the action is selected and the episode progresses to the next step. The charge in the neurons from the previous exposures will remain. With the F2F reset decoding method, the residual charge will be cleared and the network is reset to its original conditions awaiting input from the next episodes steps state. The reset variants of F2F and rate decoding work in the same way as one another in that after each step, the network is reset to its default state.

4.5 Codebase

This section will outline the codebase used for the running of the experiments including the use of the Open AI gym (Brockman et al. 2016) interface and the developed SNN codebase accessible at `https://github.com/andrewrafeUTS/SNNTechnicalAppendix`.

4.5.1 Open AI Gym

The Open AI gym (Brockman et al. 2016) is a framework provided by Open AI for developing and comparing reinforcement learning algorithms. The framework provides access to a large number of virtual environments with a consistent interface across all problems. The two environments utilised and described in Section 4.1 are both accessible through the Python interface after importing the gym and numpy (Harris et al. 2020) packages. These environments can be accessed using the following python commands:

```
gym.make('CartPole-v1')
```

```
gym.make('LunarLander-v2')
```

4.5.2 Spiking Neural Network

Due to the unavailability of well documented SNN code and to improve ease of interfacing with the gym (Brockman et al. 2016) package, the code to control the SNNs was developed for this research. The code was based off of the Izhikevich (Izhikevich 2003) neuron model (described in Section 2.4.2.) and was inspired by the formation of networks by adding layers of neurons to a network object and then connecting those layers together with specified connection weights like in the python package pandas (pandas development team 2020). SNNs are constructed by sequentially adding layers of a specified neuron type. These neurons types can either be input encoder neurons, used for the input layer of the network, or izhikevich nodes, neurons that abide by the equations outlined in Section 2.4.2.

These networks when executing, can have one of the four decoding methods for determining action selection, namely f2f, f2f reset, rate or rate reset. More detailed explanation of this code with provided examples is accessible through GitHub at `https://github.com/andrewrafeUTS/SNNTechnicalAppendix`.

4.6 Summary

This chapter outlined standard methods and techniques used across all experiments in the following two chapters. It introduced the two virtual environments explored provided by the Open AI gym (Brockman et al. 2016) interface by describing the standard state spaces and reward structures. Secondly, it provided an explanation and examples of the two state space transformation methods used, namely the binary encoding and double encoding methods. Thirdly, the state exposure period methodology was defined, a method utilised in all experiments conducted in the following chapters. Fourthly, the four standard decoding methods tested were explained in detail with specific examples given. Finally, a description and link to the codebase was provided.

The following two chapters will provide the results of the experimentation of evolutionary and natural learning algorithms applied to SNNs of varying network structures, state exposure periods, encoding methods and decoding methods as they relate to answering the proposed research questions. The specific algorithms will be described in more detail in those sections as they vary greatly across different experiments.

Chapter 5

Evolutionary Experiments

This chapter contains a detailed description of the Evolutionary experiments conducted with SNNs. It covers the initial experimentation where a SNN is compared to an ANN using the same evolutionary algorithm. The second experiment explores the effect that changing the encoding method has on the performance of the SNN. The third experiment conducts similar trials to those covered in the first two experiments in more depth to be able to effectively answer RQ1 and RQ2. The final evolutionary experiment trials different types of encoding methods found in the literature to further provide evidence for RQ2.

5.1 Experiment 1 - Initial Experimentation

The first of the Cart Pole experiments was in response to RQ1 and RQ2. Firstly, it was to investigate whether there is a SNN with no hidden layers that exists that can solve the Cart Pole RL environment. Secondly, it included initial experimentation of a direct encoding method where the input values were passed into the first layer Izhikevich neurons with a simple scaling applied. Thirdly, the temporal first-to-fire (F2F) decoding method was used for action selection. Finally, although not explicitly stated in the research questions, a simple comparison of a single layer ANN was used to compare the performance of SNNs against a widely used network architecture.

5.1.1 Description of Networks

In order to abide by the methodology constraints, the ANN and SNN network structures must be of the same size with the same number of connections. The networks consisted of 8 neurons in the input layer and is fully connected to the final output layer of 2 neurons. These networks therefore have no hidden layers and a total of 16 weighted connections. The inputs are consistent with those identified in Table 4.3 however these inputs are transformed for input into the SNN with a simple multiplication of 10 times. This is to ensure that spiking occurs in the input layer and is not lost by the leaking of the neurons.

5.1.2 Description of Genetic Algorithm

The populations for each of the tests were 100. These 100 networks all start off with random weights between -1 and 1 or -10 and 10 for the greater weight experiments. After each generation, the top 20 of the population are selected for crossover where two parents are selected with each individual weight of the next generation agent created by randomly selecting between the two parent agents. For all of the weights of this new agent, there is a 10% chance that the weight will be increased by a small random amount (between 0 and 0.1 or 0 and 1 for the greater weights experiment), a 10% chance of a decrease in weight by the same amount and finally a 1% chance of a flip to the sign of the weight. Every new agent over every generation undergoes the same transformation. As this was conducted as an initial exploratory experiment, only 1 trial for each of the provided network types was conducted. Although this seems limiting to draw any useful conclusions, it was helpful in determining the direction of future experiments and this algorithm was used in future experiments to create more meaningful conclusions.

5.1.3 SNN Decoding Method

As described in Chapter 2, the F2F method will choose the action based on the first neuron to fire in the output layer after the start of the exposure of a new state.

In order to prevent long execution times, a maximum exposure period was set with that parameter becoming one of the measured characteristics of the network.

5.1.4 Results

The first decoding method to be tested extensively was the F2F method. It was found that the larger the time window to capture the spike, the greater the agent's ability to perform better over time. As can be seen from Figure 5.1, the SNN with an exposure period of 50 and 100 time-steps per state achieved better than the time periods of 5, 10 and 20. However, they were still unable to achieve the 900-fitness goal that the ANN was able to achieve in 43 generations.

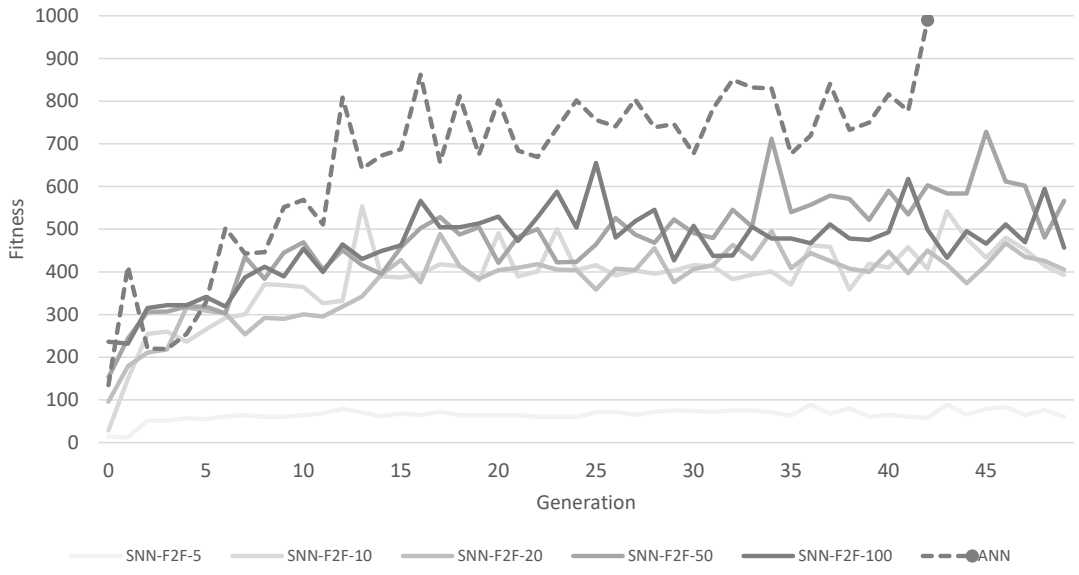

Figure 5.1: *The best fitness achieved each generation. The random mutations on the SNNs weights are identical to the mutations of the ANNs.*

It was identified that the weights of the SNN were too low to encourage frequent firing of the output layer and therefore in smaller time windows, even in the case of 50 exposures, the output layer was not regularly firing. So, for the second experiment, the same parameters for the SNNs were tested however this time, the magnitude of the random mutations and the initial weights were 10 times that

of the original experiment. Figure 5.2 shows how this increase to the weights drastically improved the performance of the 20, 50 and 100 time-step exposure SNNs with the 50 and 100 exposure networks being able to reach the goal within the 50-generation limit at generation 39 and 22 respectively.

Increasing the mutation chances and the original weights in the ANN by 10 times didn't seem to have an effect on the ANN network. This is thought to be because the magnitude of the signal isn't as important in the ANN as the inputs are passed onto further layers every iteration anyway. There is no threshold at which it passes on this information as is the case with the SNNs.

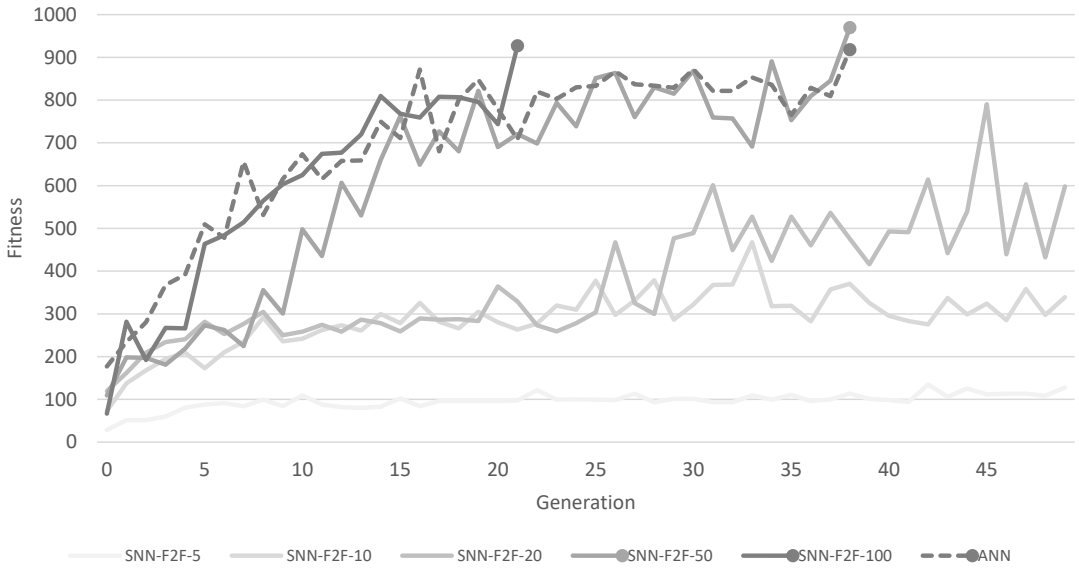

Figure 5.2: *The best fitness achieved each generation. The random mutations on the SNNs weights were 10 times more than those for the ANN signified by the GW label. The exposure times of 5, 10, 20, 50 and 100 were tested with a first-to-fire (F2F) decoding method for the SNNs.*

Although this initial experiment is relatively simplistic, some key areas of focus for future experimentation were identified. Firstly, the need to have large weights compared to those of ANNs was identified. Although larger weights are more effective for first-to-fire decoding methods, will it be that other decoding methods also require the larger weights. It makes sense to have larger weights in order to

encourage spiking behaviour in the network, however, when attempting to maximise efficiency, which this research will not cover, the goal would be to minimise spiking frequency therefore minimising computation. Both negative weights and magnitude of network weights have been identified as key areas for further research. Secondly, it appears that the shorter exposure periods do not produce adequate spiking activity in the input layer and therefore does not translate to much, if any, spiking activity in the output layer leading to the inability to make an informed decision. There is more research needed into getting lower-level exposure periods to work more effectively.

An Izhikevich SNN using an evolutionary algorithm for training was able to match the performance of an ANN using a similar evolutionary process on the Cart Pole RL benchmark. This shows promise for SNNs where the performance seems able to match and exceed ANNs when using F2F decoding methods with exposure times of 50 or 100 iterations. It also sets a strong foundation for the ability of SNNs going forward with evidence being collected for both RQ1 and RQ2 when more vigorous experimentation is conducted on training and decoding methods.

5.2 Experiment 2 - Transformed Input Space

To align with the standard benchmarking rules for the cart pole problems, future experiments including this one will be capped to episode lengths of 200 steps with a successful network being able to achieve an average fitness of 195 or more over 100 consecutive episodes.

5.2.1 Input Transformation

The agent is traditionally fed with four input variables however these have been transformed into eight inputs to remove negative values as is the requirement with the used encoding method and with Izhikivech neurons. The final inputs are outlined in Table 5.1. This differs from the previous experiment in that instead of using binary encoding to identify the sign, a method called double encoding

Input	Description	Min	Max	Scale Factor
0	Cart Position Right of Center	0	2.4	0.208
1	Cart Position Left of Center	0	2.4	0.208
2	Cart Speed Right	0	∞	0.250
3	Cart Speed Left	0	∞	0.250
4	Pole Angle to the Right	0	0.209	2.40
5	Pole Angle to the Left	0	0.209	2.40
6	Pole Tip Speed Right	0	∞	0.250
7	Pole Tip Speed Left	0	∞	0.250

Table 5.1: *The Transformed State Space to Remove Negative Input Values.*

(Wiklendt et al. 2009)(Markowska-Kaczmar and Koldowski 2015) was used where a single input value that has the possibility of including negative values is split into two neurons where one represents the positive magnitude, and the other the negative magnitude. This was conducted as a pre-processing step when a new state was supplied to the agent. Additionally, an observation from the previous experiment was that some neurons in the input layer were spiking much more frequently than others due to them having averagely larger magnitudes. Therefore, for each input, a scale factor (which was selected empirically) was selected to normalise the range of input data and prevent individual neurons from dominating the spiking activity in the network. These scale factors and adjusted state space are identified in Table 5.1.

5.2.2 Method

The population of networks (N=50) were exposed to the problem for five consecutive episodes with a maximum episode length of 200 steps with no change to

connection weights. The average total reward of these five episodes is used as the achieved fitness of the agent. The top 20% of agents (N=10) are identified using these fitness scores and they are prevented from experiencing weight changes in the next generation. The other 80% of agents (N=40) obtain their new weights through a random crossover of two of the agents from the top 20%. These weights are then randomly mutated according to a set of predefined rules. These rules are 10% chance of increasing the weight value by a random amount between 0 and 1, 10% chance of decreasing the weight value by a random amount between 0 and 1, and a 1% chance of flipping the sign of the weight. The top agent is then subjected to 100 consecutive episodes with no weight changes to evaluate its average fitness. If that average fitness is above 195 out of episodes with a max length of 200, then the algorithm terminates and has been successful. Otherwise, the generations will continue as described until 150 generations have been conducted and the algorithm terminates as a failure.

5.2.3 Results

The results shown in Figure 5.3 display the average fitness for the best agent in each generation over **10** separate evolutionary processes for each of the time exposure periods tested.

Those networks that used time exposures of 100, 50 and 25 were able to solve the problem successfully. The more exposures of a state that the networks were given, the fewer generations were needed to solve the problem. For the networks given 10 state exposures, they showed a similar increase in fitness initially but were not able to reach the solved fitness level. The network with 1 state exposure per step was unable to improve and resulted in comparable fitness to agents that select random actions. This was due to very little spiking activity occurring in the output layer resulting in the taking of random actions on a majority of steps in an episode.

Although the encoding method was suitable in creating well performing agents in this approach, the somewhat arbitrary conversion of the input values to spiking

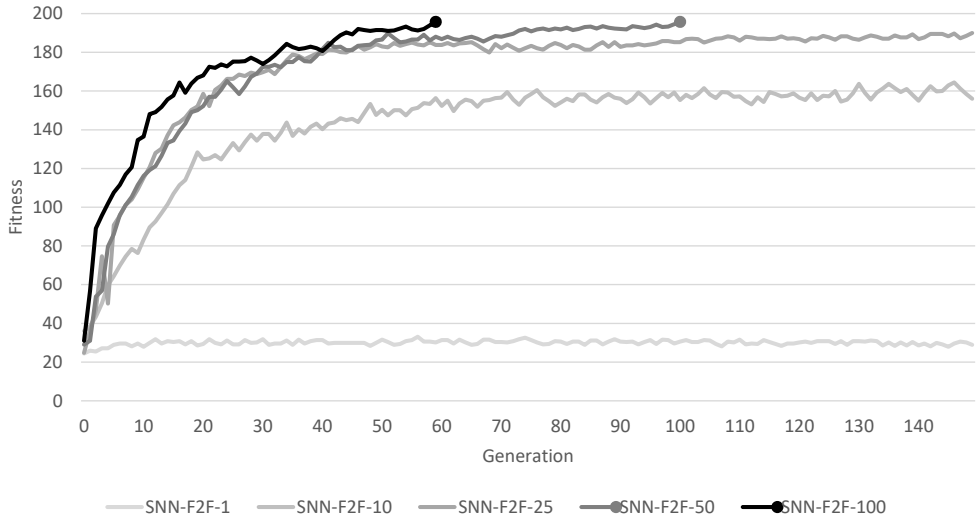

Figure 5.3: *Average fitness over 10 separate executions of the algorithm for the best agent in each generation for each of the time exposures tested.*

probabilities may not be suitable for other problems and therefore, more general encoding approaches for problems with differently scaled state space values should be developed and tested.

5.3 Experiment 3 - Comparitive Analysis

As outlined in the previous experiment, the arbitrary encoding of input variables into the input layer of the network poses a problem for generalising the evolutionary processes. In this section, a novel input encoder is proposed, and a detailed analysis of different decoding methods and exposure periods is conducted. It is worth noting that, for the Cart Pole inputs, the Scale Factors outlined in Table 5.1 is not used but the removing of negative inputs in the pre-processing step is still done. In the conduct of this experiment, the Lunar Lander problem was also attempted.

5.3.1 Input Encoder

One of the major differences with SNNs compared to ANNs is how information is parsed into and extracted from the network, otherwise known as encoding and decoding respectively. For encoding, the state information from the reinforcement learning problem must be converted into a spike train to be fed into the network. The original Cart Pole and Lunar Lander state spaces are made up of positive and negative input values often denoting direction where negative is positional or velocity in the left direction and positive in the right direction. Therefore, for each of the inputs where a negative value is possible, these have been split into two input values where one represents the left direction and the other the right also known as double encoding. If the original input value was negative, then the left direction input would be the positive magnitude of the original value and the right direction input would be zero. If the input was positive, then the left direction input would be zero and the right direction input would be the same value as the original input value.

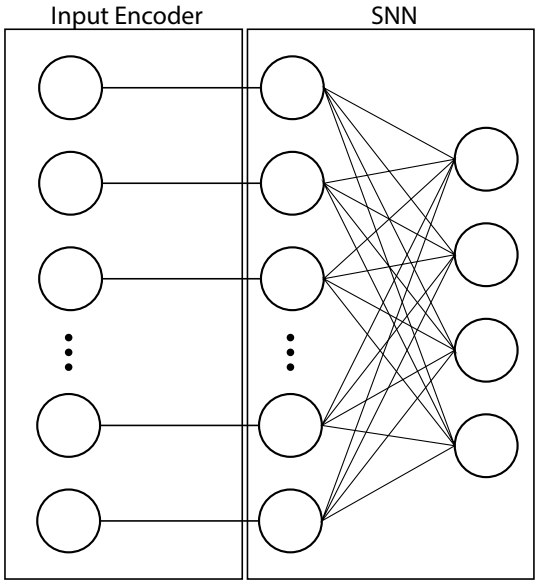

Figure 5.4: *An example of the input encoders connection to the SNN. In this example, the SNN has no hidden layer as the Input Encoder layer are not made up of IZ neurons and simply act as a pre-processing step for state space information.*

A similar issue with the inputs of both problems when compared to pixel input information is the non-normalized nature of them. Where pixel information is often represented with inputs between 0.0 and 1.0, these could be scaled equally among all inputs to encourage spiking behaviour in the input layer. However, both of the problems have differently scaled inputs, some of which often become greater than 1 and some that never cross a threshold of 0.1. Therefore, scaling them in a uniform way leads to unequal importance being placed on particular input values. Therefore, an additional input encoder layer was added to the network which acts as a simple multiplier of the inputs by some learned weight and injected into the first SNN layer of the network demonstrated in Figure 5.4. This mapping of input encoder to SNN first layer nodes is one to one, as in each initial layer SNN node has one input encoder node which is directly attached to it with some weight value. This is the scale that the input is changed by before being injected into the SNN. This is not being counted as a layer in the network as it is simply a pre-processing step of the input before being fed into the network however the weights of the input encoder are learned through the same evolutionary process used by the SNNs.

5.3.2 Method

For each of the experiments conducted, a population of 50 initially random agents were produced with weight values ranging from -20.0 to 80.0 for Izhikevich neurons and input encoder layer weights ranging from 0.0 to 150.0. Each trial had a maximum of 50 generations or would be concluded when the problem solution requirements were met. Each agent conducted 5 separate episodes and the average of all episode rewards was the fitness of that agent for that particular generation. If any of the networks achieved the goal over the 5-episode average, that network was run through 100 consecutive episodes to determine if the overall goal was achieved as set out by the problems. If it was, then the experiment was concluded otherwise the evolutionary process would continue. After all agents conducted their individual trials, the top 10 agents were prevented from undergoing any evolutionary changes. For the remaining 40 agents, their weights were replaced by

the random crossover of two of the top 10 agents. Each weight was replaced by either the first parent ($P = 0.5$) or second parent ($P = 0.5$) weight value. They then underwent a random mutation phase where each weight in the network was transformed according to Equations 5.1, 5.2 and 5.3.

$$X \sim U(0.0, 1.0) \tag{5.1}$$

$$w' = \begin{cases} w + 10\alpha & \text{if } 0.00 \leq X < 0.01 \\ w - 10\alpha & \text{if } 0.01 \leq X < 0.02 \\ w + \alpha & \text{if } 0.02 \leq X < 0.12 \\ w - \alpha & \text{if } 0.12 \leq X < 0.22 \end{cases} \tag{5.2}$$

$$\alpha' = \alpha\lambda \tag{5.3}$$

X refers to some mutation chance value selected from a uniform distribution, w' is the new transformed weight and w is the weight after crossover, α is the learning rate, which is 10.0 at the beginning of training, α' is the new learning rate after decay λ which is constant at 0.99. X is calculated separately for each weight. These values were selected empirically as a result of conducting a range of hyperparameter experiments. These experiments also demonstrate a benefit in using a decaying learning rate. For each network, each weight is updated according to Equation 5.2 once per generation. The learning rate update from Equation 5.3 occurs once per generation.

For both Cart Pole and Lunar Lander, a random benchmark trial was run as the control. This random benchmark involved running agents in batches identical to those used in the evolutionary trials whereby over 50 generations, 50 batches of 5 episodes were run with an agent that selects a random action at each episode step. Over the single generation, the best average fitness of a 5-episode run was recorded as the fitness of that generation. Also, for both problems, exposure periods of 20, 40, 60, 80 and 100 were used. Networks with and without hidden

		Highest Achieved Fitness / Solved Generation			
Decoding Method	**Exposure Period**	**No Hidden**	**16 Hidden**	**32 Hidden**	**Random**
F2F	20	196.80 / 14	196.82 / 37	196.55 / 45	
	40	197.10 / 13	195.27 / 26	196.41 / 61	
	60	**197.00 / 6**	**196.00 / 21**	197.31 / 36	
	80	196.28 / 7	195.36 / 23	**197.07 / 30**	
	100	**197.87 / 6**	195.47 / 24	197.25 / 31	
F2F RESET	20	195.75 / 16	199.49 / 21	195.47 / 18	
	40	198.07 / 7	**197.52 / 11**	196.52 / 20	
	60	197.1 / 8	198.13 / 14	195.42 / 17	
	80	195.56 / 7	196.22 / 17	**197.27 / 12**	
	100	**198.54 / 6**	**199.52 / 11**	195.48 / 17	47.0667 / -
RATE	20	195.31 / 31	195.54 / 56	195.58 / 31	
	40	195.45 / 14	195.26 / 23	198.27 / 22	
	60	195.27 / 7	196.38 /23	196.85 / 26	
	80	196.16 / 9	195.08 / 19	196.43 / 19	
	100	**197.55 / 6**	**195.00 / 12**	**195.22 / 18**	
RATE RESET	20	196.92 / 18	197.20 / 20	196.84 / 22	
	40	197.25 / 7	197.60 / 24	196.81 / 21	
	60	198.00 / 9	197.91 / 17	195.31 / 17	
	80	196.72 / 5	**198.13 / 12**	195.40 / 14	
	100	**195.09 / 4**	197.87 / 16	**197.27 / 12**	

Table 5.2: *Table of results for Cart Pole experiment showing the highest average fitness achieved and the generation it achieved the goal by each decoding method, exposure period and network structure with a goal fitness of greater than 195.0.*

layers were tested. For those multi-layered networks, a single hidden layer of 16 and 32 neurons was used. All networks used the Input Encoder method and F2F, F2F Reset, Rate and Rate Reset methods were used for decoding. Three separate trials were conducted for each of the decoding methods and each of the exposure periods and the average fitness of each generation were used for analysis.

5.3.3 Results

Over all four decoding methods, all exposure period intervals and over both the single and multi-layered network combinations, the SNN was able to produce goal

		Highest Achieved Fitness / Solved Generation			
Decoding Method	**Exposure Period**	**No Hidden**	**16 Hidden**	**32 Hidden**	**Random**
F2F	20	190.93 / -	33.59 / -	-6.33 / -	
	40	184.24 / -	27.09 / -	-32.34 / -	
	60	**192.98 / -**	6.04 / -	**44.16 / -**	
	80	89.18 / -	**35.51 / -**	25.56 / -	
	100	136.17 / -	20.68 / -	27.15 / -	
F2F RESET	20	148.79 / -	26.90 / -	8.49 / -	
	40	**199.31 / -**	**157.42 / -**	**45.50 / -**	
	60	194.85 / -	63.52 / -	-2.01 / -	
	80	176.84 / -	129.51 / -	20.32 / -	
	100	188.74 / -	156.28 / -	29.39 / -	
RATE	20	172.08 / -	13.42 / -	16.52 / -	-71.3948 / -
	40	**192.04 / -**	-9.42 / -	1.87 / -	
	60	184.68 / -	10.44 / -	9.21 / -	
	80	59.78 / -	**70.07 / -**	**67.91 / -**	
	100	128.73 / -	44.90 / -	2.15 / -	
RATE RESET	20	185.46 / -	77.04 / -	51.16 / -	
	40	57.47 / -	**131.91 / -**	**68.64 / -**	
	60	**202.26 / 92**	54.72 / -	64.59 / -	
	80	167.50 / -	52.84 / -	53.97 / -	
	100	133.73 / -	41.02 / -	61.37 / -	

Table 5.3: *Table of results for Lunar Lander experiment showing the highest average fitness achieved and the generation it achieved the goal (if goal was achieved) by each decoding method, exposure period and network structure with a goal fitness of greater than 200.0.*

achieving agents in the Cart Pole problem as shown in Table 5.2. For single layer networks, there is no significant difference in the generations needed to reach the goal across all four decoding methods. The only slight difference appears to be in a better capacity for F2F and F2F Reset methods to solve the problem in fewer generations in lower exposure periods. When looking at an exposure period of 20, F2F and F2F reset were able to solve the problem in 14 and 16 generations respectively whereas Rate and Rate Reset decoding methods took 31 and 18 generations respectively. This is largely due to Rate based decoding methods at lower exposure periods tending not to produce multiple spikes per output neuron and therefore

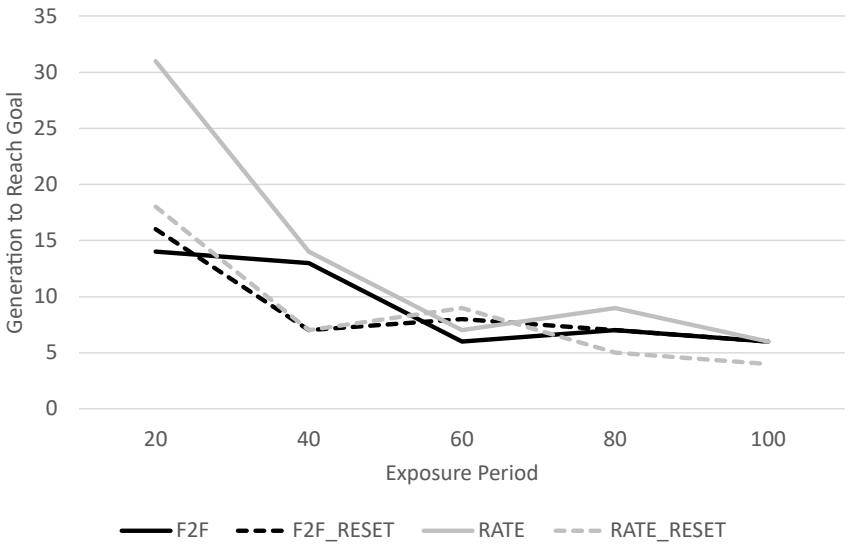

Figure 5.5: *The generation to reach the goal for single layer networks across all exposure periods and decoding methods for the Cart Pole problem.*

a calculation of the rate of firing of the neurons often results in the taking of a random action due to the rates across both output neurons being the same. As greater exposure periods are used, the output neurons are more likely to produce more variance in the rate of spiking of the output neurons and that disadvantage goes away.

When looking at the number of random moves taken by a goal achieving agent, as demonstrated in Figure 5.8, it is clear that the longer the exposure period, the less random actions that are taken. Random actions occurred if the decoding method over the exposure period was not able to determine an action to take. The occurrence of random actions is far fewer when using temporal decoding methods like F2F and F2F Reset in higher exposure periods. For lower exposure periods, occurrences of random action taking is fairly consistent between all decoding methods with rate-based decoding selecting random actions only slightly more often than temporal decoding.

As is demonstrated in Table 5.4, a clear negative correlation has emerged between the exposure period and the number of generations needed to reach the goal in

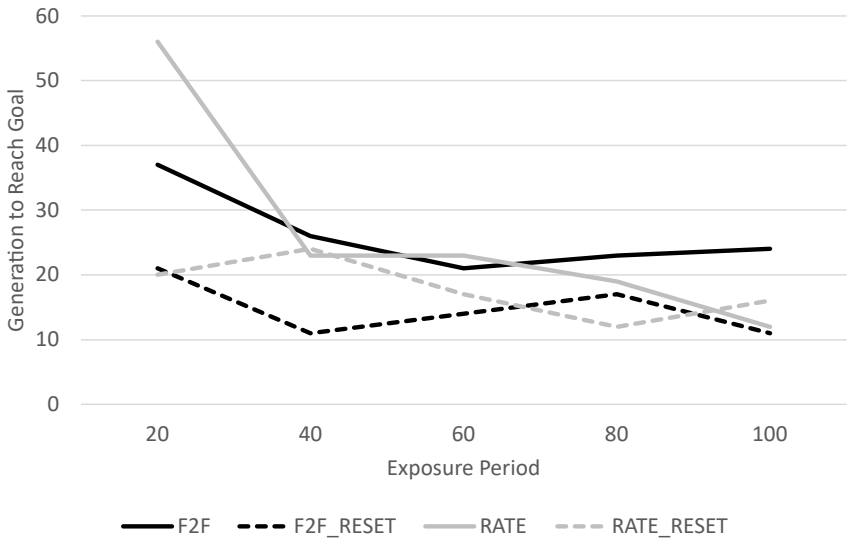

Figure 5.6: *The generation to reach the goal formultilayer networks with 16 hidden neurons across all exposure periods and decoding methods for the Cart Pole problem.*

the Cart Pole problem. This negative correlation is consistent across all decoding methods and network structures used in this experiment. This trend appears to level off at the higher exposure periods indicating that there may be an optimal exposure period for this problem and introduces a trade-off between the execution time of the network and the number of generations that will be needed to reach the goal. The larger the exposure period, the more execution time is needed as more network steps are required per episode step. This levelling off appears to occur at around the 60-exposure period mark as evidenced in Figure 5.5, 5.6 and 5.7.

For the Lunar Lander problem, no multi-layer network was able to achieve the goal with fitness rarely reaching above 100. The F2F Reset and Rate Reset decoding methods were able to achieve the highest average fitness across all exposure periods when compared to the other decoding methods as can be seen in Table 5.3. The single layer networks came close to achieving the goal state in almost all of the decoding methods and exposure periods utilised and was able to averagely achieve the goal in one occurrence with the use of the Rate Reset decoding method at 60 exposure periods. The fact that this occurred at generation 92 out

of the tested 100 generations indicates that given more time, other network structures may have been able to solve the problem. Introducing more neurons in the multi-layer networks meant that the highest achieved fitness over those 100 generations were consistently lower in networks with more neurons. This is expected as more neurons means more synapse weights to learn and more generations that are needed to improve fitness.

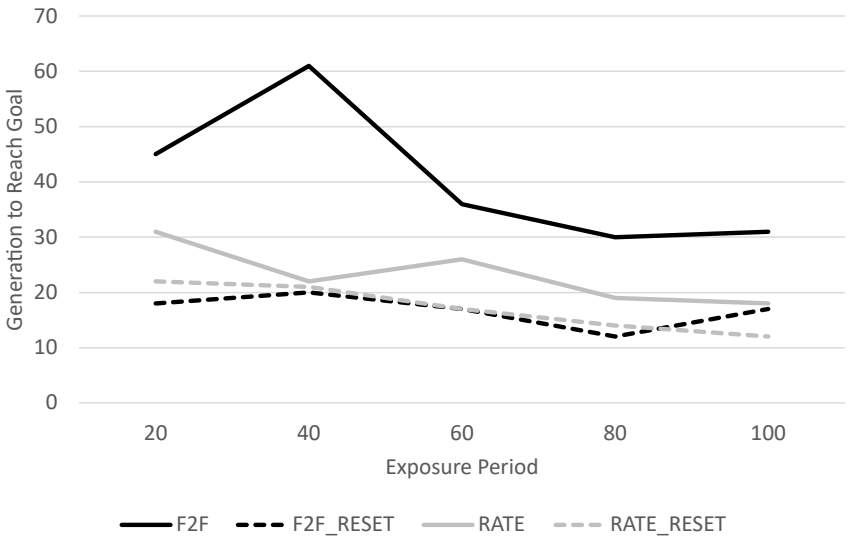

Figure 5.7: *The generation to reach the goal for multilayer networks with 32 hidden neurons across all exposure periods and decoding methods for the Cart Pole problem.*

	NO HIDDEN	**16 HIDDEN**	**32 HIDDEN**
F2F	-0.88	-0.73	-0.73
F2F RESET	-0.77	-0.52	–0.54
RATE	-0.84	-0.85	-0.86
RATE RESET	-0.85	-0.70	-0.99

Table 5.4: *Correlation Coefficients for Cart Pole problem between the generation that each trial reached completion and the exposure period used for that trial.*

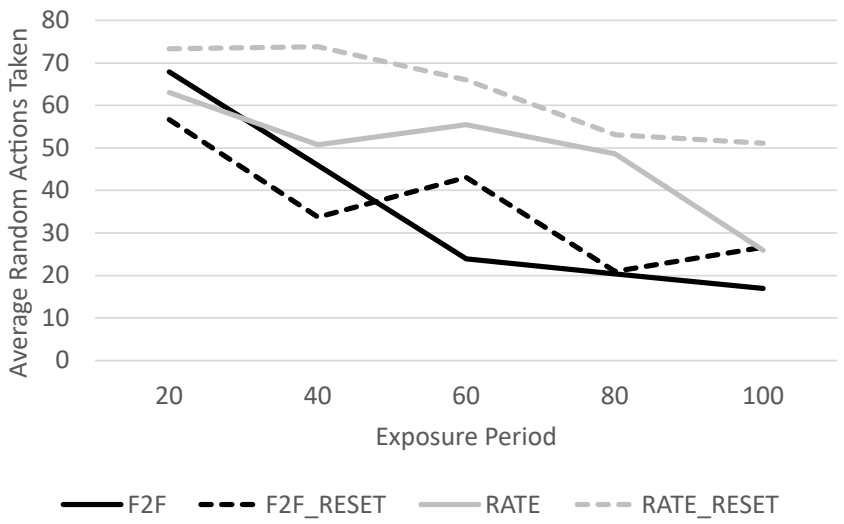

Figure 5.8: *The number of random actions taken on average for each of the exposure periods tested and decoding methods used for single layer networks in the Cart Pole problem.*

5.3.4 Discussion

Possible trends emerged in the negative correlation between the exposure period and the generations taken to reach a certain fitness when looking at the results of the Cart Pole experiment. This suggests that minimising the chance of random actions being chosen due to none of the output neurons firing in the output layer or, in the case of rate decoding, the same number of spikes over more than one output neuron over the exposure period, hinders the rate of learning in the network. Due to the failure of most Lunar Lander trials to reach the goal, it makes it difficult to confirm this trend and therefore requires more investigation. Similarly, the number of trials and exposure periods tested prevents strong conclusions being made about this correlation. It is worth expanding the number of trials and exposure periods tested to formulate better evidence of this. Additionally, investigating whether this trend continues in non-evolutionary training methods such as with Hebbian learning processes Hebb (1949) including Reward based Spike Timing Dependant Plasticity Bing et al. (2018) and with alternate reinforcement learning problems. This would determine whether this trade-off between network

efficiency, by lowering the number of exposure periods, and network accuracy by increasing the number of exposure periods is a universal trait of SNNs.

The input encoder proposed in this paper was successful at learning state space data pre-processing before being fed into the IZ SNN for the Cart Pole problem and for some individual Lunar Lander trials when using all tested decoding methods. This is significant for evolutionary learning as it can use the same algorithm for both learned pre-processing and network learning. As long as negative inputs are removed from the state space, this input encoder method can be used without any other pre-processing of state space information. Although this method may work well for evolutionary training of SNNs, future investigations into training those pre-processing weights in the input encoder using a non-evolutionary framework is needed for comparison.

It is clear that the reset variants of both F2F and Rate decoding methods achieves better fitness over less generations when compared to their non-reset variants, which is evident by both the Cart Pole and Lunar Lander networks. This is most likely a result of not needing information of previous states to select the optimal action in the current state. Not resetting the network to its default between steps introduces noise into the next state interpretation. This may not be the case for all problems where having some residual knowledge of previous states may be useful. It does however suggest that in these two problems, that previous state knowledge is not important. The failure of the Lunar Lander networks to reach the goal fitness is most likely a result of a lack of generations in training or incorrect network shape specifically evident by the poor fitness of the multi-layer networks. Additionally, the noisiness of the results of the Lunar Lander experiments indicates that more individual trials for each set of parameters needs to be included.

5.4 Experiment 4 - Encoding Method Comparisons

Although the Input Encoder method worked effectively in the previous experiment, there will be challenges using this method with a non-evolutionary approach due to the non-spiking behaviour of that pre-processing layer. Therefore, other methods of encoding should be compared and evaluated against each other to find the most appropriate method that could be used in non-evolutionary learning due to their inclusion of spiking activity.

5.4.1 Other Methods of Encoding

Unless specified otherwise, all of the encoding methods tested in this experiment will use Double Encoding (Wiklendt et al. 2009)(Markowska-Kaczmar and Koldowski 2015) to remove the negative input values. This method simply just splits the positive and negative components of a state input into two inputs representing those positive and negative components. The exception to the use of this will be using the Binary Flag Encoding introduced in Experiment 1 and Table 4.3 however this will be explicitly stated. Fundamentally, the decoding methods can be split into two main categories: direct encoding methods and probability based encoding methods.

Direct Encoding (DE)

DE was the method used in Experiment 1 and Experiment 2 in this Chapter. Essentially it uses the direct state space values produced by the problem and feeds them into the first layer of the network which comprises of regular IZ neurons. As identified in Experiment 1, using the base values supplied by the problem was inadequate. It required a simple scale to be applied to those inputs in order to encourage spiking activity in the input layer. Therefore, the 10x scale factor for inputs will be used before feeding them into the first layer.

Direct Encoding with Binary Flag Encoding (DBE)

The DBE method will be the same as described in DE however will utilise the Binary Flag Encoding instead of the Double Encoding that is used by all the other methods. This will also use the 10x scale factors to encourage spiking behaviour in the input layer.

Direct Scaled Encoding (DSE)

The DSE method will utilise the scale factors described in Table 5.1. The inputs will be scaled by those factors, then multiplied by 10 like in the DE method and then will be fed as injections into the first layer of regular IZ neurons.

Probability Encoding (PE)

The PE method involves utilising the input values not as direct injection into the network, but as a probability of causing a spike to occur at any individual timestep in the first layer. Take for example, the 2nd input value of 0.43. This would cause the 2nd input neuron to spike with a 43% chance. If the value is greater than 1.0, then the input neuron is guaranteed to spike on that timestep.

Normalised Probability Encoding (NPE)

The NPE method is similar to the PE method except the state input values are normalised to sum to 1 and then used as probability of causing spikes to occur on input neurons.

Scaled Probability Encoding (SPE)

SPE is similar to PE except the values are scaled by their scale factors described in Table 5.1. Those scaled values then act as probabilities of spiking for the input layer neurons.

Scaled Normalised Probability Encoding (SNPE)

SNPE is a combination of the NPE and SPE methods where the inputs are scaled by their scale factors, normalised and then used as probabilities of spiking for the input layer neurons.

5.4.2 Method

Due to the introduction of more parameters needed for testing, the exposure periods tested in Experiment 3 and the four decoding methods also compared in Experiment 3 will be selected from the best performing methods whilst attempting to minimise processing time required for the conduct of the experiments. Therefore, an exposure period of 60 with the F2F Reset and Rate Reset decoding methods will be used. The exposure period was selected due to the reduced number of generations required to reach the goal and the diminishing returns of the longer exposure periods. Additionally, the two decoding methods chosen were the Reset types of the F2F and Rate decoding methods due to the more rapid initial improvement demonstrated. Additionally, only networks with no hidden layers will be tested. The conduct of this experiment will only look at the Cart Pole problem due to the ability for all types of network parameters in the previous experiment to solve this problem.

All networks will undergo the same evolutionary algorithms set out in the previous experiment such that comparisons can be made easily between these methods and the Input Encoder method. The combination of encoding method and decoding method with an exposure period of 60 network iterations per timestep, will each run 5 separate trials with the results representing the average fitness per generation over those trials.

5.4.3 Results

All methods of Direct Encoding using both F2F Reset and Rate Reset were able to achieve the goal of 195 or more over 100 consecutive episodes as can be seen

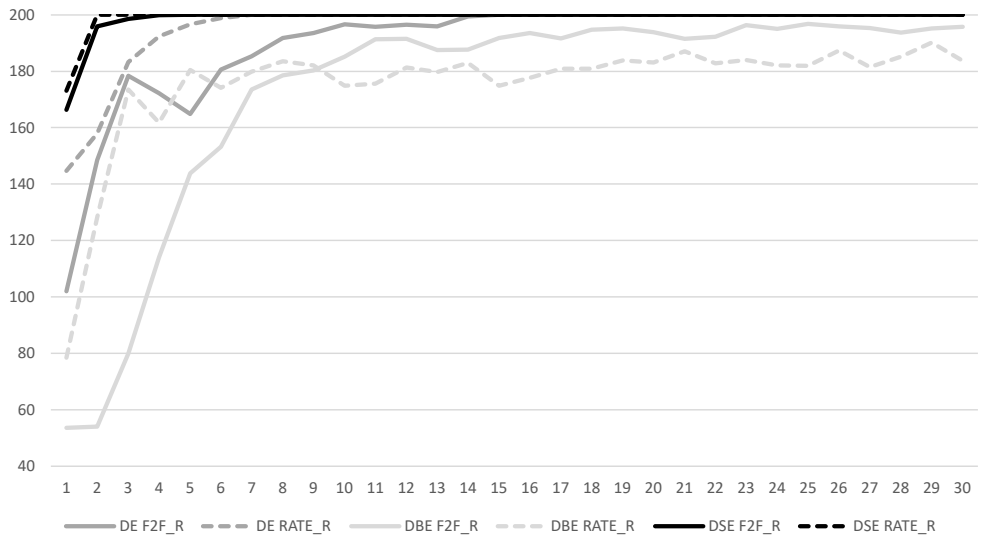

Figure 5.9: *A comparison of direct encoding methods using a network with no hidden layers in the Cart Pole problem.*

in Figure 5.9. This includes the DE, DBE and DSE encoding methods. DE and DSE both utilised Double Encoding methods to remove negative input however, DBE which did not use the traditional double encoding method but a binary flag to represent whether the original input was negative or not, was successful as well. DSE outperformed all other methods of encoding with rapid improvement in the first few generations and reaching the goal after only a few generations. DBE performed the worst out of the direct encoding methods but still better than all other forms of probability encoding. The other forms of encoding methods, namely the probability encoding types were not successful in reaching the Cart Pole goal as demonstrated in Figure 5.10. These methods however did come close to reaching the goal and showed steady improvement over the initial generations. The SNPE method performed the best out of all of the probability encoding methods with the SPE performing the worst. PE and NPE methods achieved very similar results to one another.

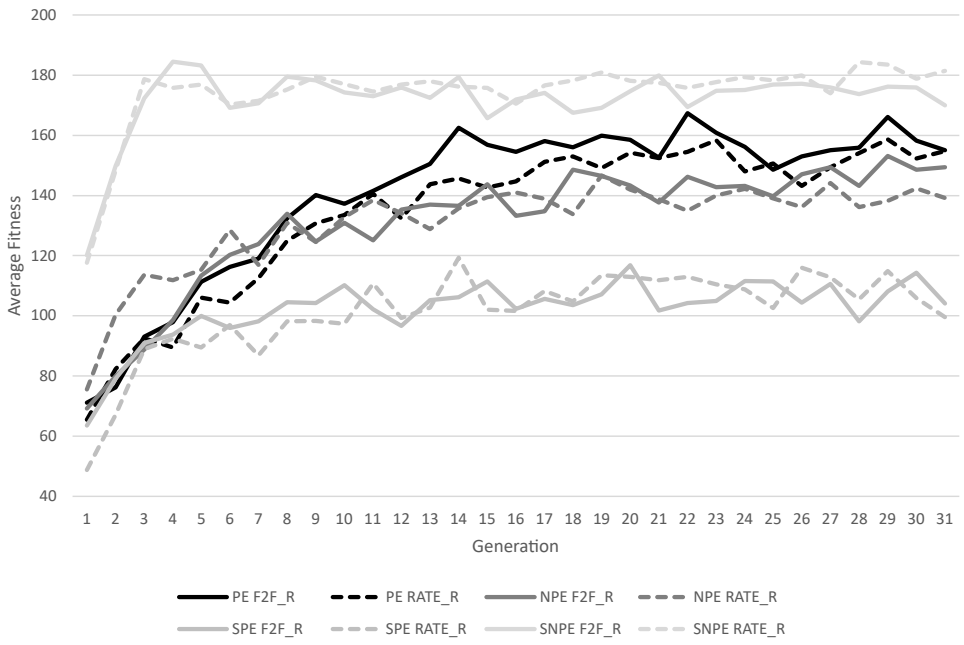

Figure 5.10: *A comparison of probability encoding methods using a network with no hidden layers in the Cart Pole problem.*

5.4.4 Discussion

Direct encoding methods achieved far better results than the probability encoding methods. DSE performing the best suggest that having standardised inputs between 0 and 1 produce more consistent spiking behaviour in the input layer and does not cause only a few inputs to have a dominating effect on the spiking behaviour of the entire network. Double Encoding methods appear to be better at handling negative input conversion than the binary flag encoding methods evidenced with the better performance of the DE and DSE methods when compared to the DBE method. This difference in performance is however relatively small for this problem.

Comparing direct encoding with probability encoding suggests that the noise created by having probability-based spiking in the input layer leads to worse action taking agents. This may be because it is harder to learn weights when spiking activity in the input layer is not consistent across generations. SNPE methods,

which produce less spiking activity in the input layer due to the normalisation of the input vector, causes better learning performance in the initial generations. This when compared to SPE, which causes much more spiking activity in the input layer, suggests that having too much spiking activity reduces the ability of the networks to make informed action selections.

5.5 Summary

This research has identified some interesting trends that need to be investigated further. Firstly, the reset variants of the F2F and Rate based decoding methods seem to have a steeper initial learning than other methods especially evident in multi-layer networks. Secondly, a higher exposure period tends to produce goal reaching agents in less generations than lower exposure periods, specifically evident in the Cart Pole experiment. This therefore introduces a trade-off between network accuracy and network efficiency when choosing an exposure period for a SNN. Thirdly, the input encoder method proposed allows for evolutionary training of networks and evolutionary learning of state pre-processing using the same algorithm, simplifying the need for complex pre-processing algorithms and making it a more general method for input encoding. Fourthly, although there is no clear benefit from using either temporal decoding or rate-based decoding, temporal decoding methods like F2F seem to produce better performing agents at lower exposure periods when compared to Rate based decoding. Finally, DE and DSE methods seem to be the most suitable encoding methods where evolutionary training of weights is not possible. Both achieved similar results to those as the Input Encoder method but could be used in non-evolutionary training.

Chapter 6

Hebbian Based Experiments

This chapter includes a detailed description of non-evolutionary experiments conducted with SNNs. Due to the difficulties of producing goal achieving agents in the Lunar Lander problem in the previous chapter, this chapter will focus solely on the use of the Cart Pole problem described in Section 4.1.1. These experiments were conducted to further address RQ2 and start the investigation into RQ3.

6.1 Experiment 5 - R-STDP

This experiment was based off of the use of R-STDP in experiments conducted by both Mozafari et al. (2018) and Bing et al. (2018). Both demonstrated the use of R-STDP on relatively simplistic reinforcement learning problems and therefore demonstrating its use on Cart Pole, arguably a slightly more advanced problem, will be useful. Before conducting the experiment, it was hypothesised that the continuous reward structure of the Cart Pole problem will pose a problem for the direct application of this algorithm as no negative rewards are given and therefore applications of modulating either Long Term Potentiation (LTP) or Long-Term Depression (LTD) by the reward signal will not happen. Regardless, testing the affect R-STDP from the literature will help determine some initial problems with the non-evolutionary approaches to training.

6.1.1 Method

Networks using F2F Reset and Rate Reset decoding methods with the best performing DSE encoding method from Experiment 4 were used for the conduct of this experiment. All networks were exposed to the state for 60 exposure periods due to determining this exposure period as a good trade-off between possible performance and limited execution time.

$$X \sim U(0.0, \alpha) \tag{6.1}$$

$$\Delta w_{pre,post} = \begin{cases} X & \text{if } 0 <= t_{post} - t_{pre} < 10 \\ -X & \text{if } -10 < t_{post} - t_{pre} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

For those spiking patterns where the pre-synaptic neuron spike was followed by a post-synaptic spike more recently than it was preceded by the post synaptic spike, the weights were increased by a random value obtained from a uniform distribution described in 6.1 which will be called long term potentiation (LTP). Additionally, if the pre-synaptic neuron spike was preceded by a post-synaptic neuron spike more recently than it was followed by a post-synaptic spike, then the weights were decreased by the random value from 6.1 which is referred to as Long Term Depression (LTD). α refers to the learning rate of the network which for this experiment was 0.1. These weight changes were conducted every episode for 5000 episodes and the average of blocks of 100 episodes were recorded and used for analysis.

6.1.2 Results

Figure 6.1 clearly shows that R-STDP was incapable of being applied directly to the Cart Pole problem with networks producing results in line with expected fitness of agents selecting random actions. All tests showed the explosion of weights in the network causing spiking the output neurons to spike every network iteration meaning that random actions were being taken almost all of the time. This suggests that some mechanism to control weight expansion using STDP methods is needed.

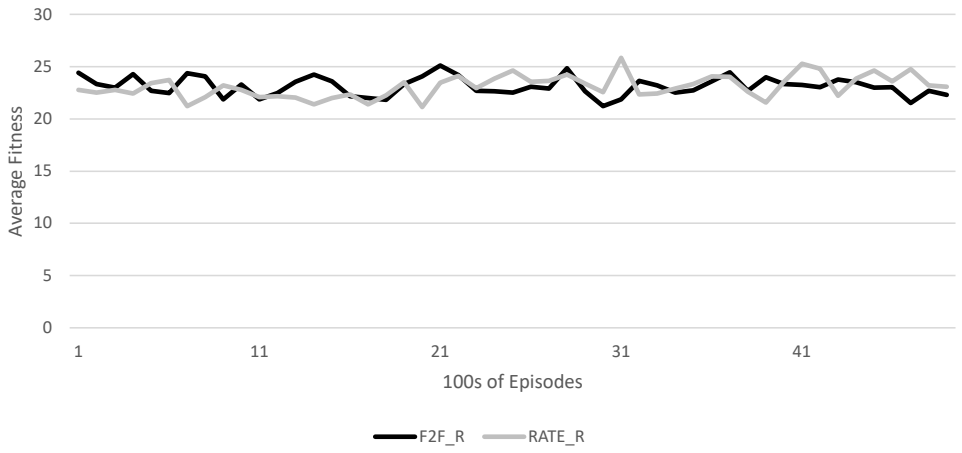

Figure 6.1: *Average fitness per 100-episode blocks for networks in the Cart Pole problem using R-STDP for learning with exposure periods of 60.*

The results of different forms of weight constraining methods can be seen in the next experiment.

6.2 Experiment 6 - R-STDP Weight Constraining

This experiment was conducted to address the weight explosions identified in the previous experiment. There is a possibility that the results of the previous experiment are linked to the weight explosion problem and therefore it is important to test methods of constraining the weights and seeing whether that improves the performance of the literature based R-STDP method. This experiment will address three different weight constraining methods, namely Pre-Synaptic Normalisation, Post-Synaptic Normalisation and Weight Decay. The first two deal with the locking the outgoing or incoming weights of the neurons to some value. Therefore, the weights change with accordance to the R-STDP algorithm and these new weights then undergo a normalisation process to force the new weights to sum to that maximum value. For the final decay method, the weight changes happen according to the R-STDP algorithm and then the weights will decay by some decay value forcing the weights to remain closer to zero. So for negative weights , the decay forces an

increase to the weight values and for positive weights, the decay forces a decrease to the weight values. These three methods will be tested and then compared back to the non-weight constraining experiment conducted previously.

6.2.1 Method

The following experiment was conducted inline with the same method from the previous experiment using Equation 6.2 to deal with episodic weight changes. The same method was used so that easy comparisons between the experiments are possible. After an episode of training using R-STDP, the weights are then altered according to the following weight constraining methods.

Pre-Synaptic Normalisation

Will force the outgoing weights from a neuron to sum to a specified amount. The pre-synaptic normalisation amount used in this experiment was 100.0.

Post-Synaptic Normalisation

Will force the incoming weights to a neuron to sum to a specified amount. The post-synaptic normalisation amount used in this experiment was 200.0.

Weight Decay

After each episode, will decay weights by a specified proportion with positive weights decreasing by that proportion and negative weights increasing by that proportion in order to stabilise the weights around 0.

$$w' = w - w\lambda \tag{6.3}$$

The weight decay described in Equation 6.3 occurs at the conclusion of an episode and after the R-STDP weight changes are conducted. The w' refers to the new weight, w refers to the current weight and λ refers to the weight decay value which should always be between 0 and 1. The weight decay value used for this experiment is 0.05.

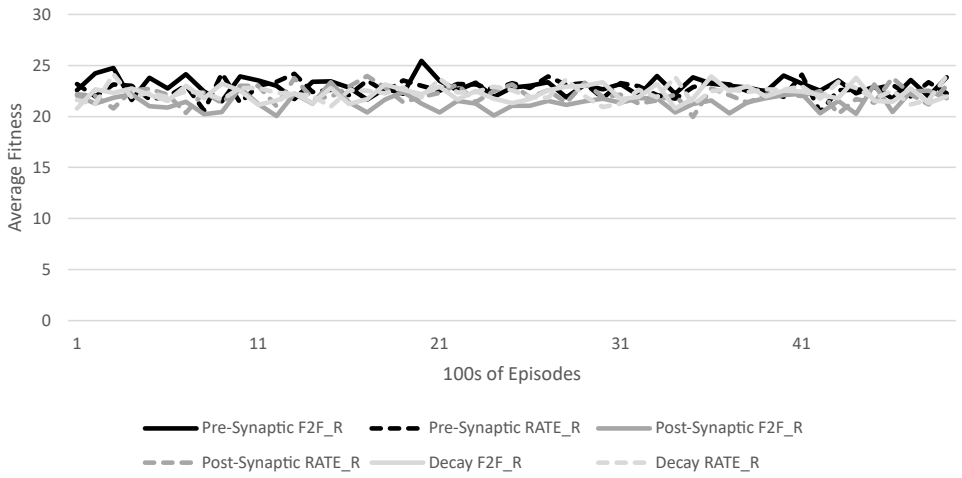

Figure 6.2: *Average fitness per 100-episode blocks for networks in the Cart Pole problem using R-STDP with Weight Constraining methods.*

6.2.2 Results

Utilising these weight constraining methods produced almost identical results to those networks that used no constraining methods and operated similarly to agents that selected random actions each step from previous evolutionary experiments. This suggests that the weight constraining methods are not to blame for the poor performance of the R-STDP method on the Cart Pole problem. Fundamentally, the issue may arise from the reward structure of the Cart Pole problem where it receives continuous rewards regardless of the outcome of the episode or step. Future work to experiment with different reward structures for the Cart Pole problem is needed. It is worth noting that network weight values using the weight decay method better reflected similar weights to those learned through evolutionary methods and demonstrated a much more consistent spread of positive and negative weight values. Both the pre and post-synaptic normalisation methods often resulted in one weight dominating and reaching the maximum normalisation value with others being very small or 0. This is similar to the weight explosion problem from Experiment 5, where some or one weight ended up becoming very large therefore dominating the network spiking activity producing too much spiking activity.

Although the literature based R-STDP algorithm was incapable of producing well performing agents, the comparison of produced weights suggests that a weight decay method for constraining the weight explosion problem is more desirable than using the pre or post-synaptic normalisation methods.

6.3 Experiment 7 - EF-STDP

This section proposes a training method inspired by Spike Timing Dependant Plasticity (STDP) in the literature and the evolutionary techniques used in the previous chapter. This method is called Episodic Fitness based Spike Timing Dependant Plasticity (EF-STDP). This section utilises the transformed state spaces for Cart Pole described in Table 5.1 and utilises the DSE encoding method.

6.3.1 Method

The population of networks (N=50) were independently exposed to the Cart Pole problem for 100 learning cycles with a maximum episode length of 200 steps. The spiking patterns, total reward, and action history was recorded for each of 100 consecutive episodes on the same network with no changes to the weights during those consecutive episodes. Using the total reward information, the top 20% of episodes were identified, and the spiking patterns of those episodes were used to determine the changing of the weights. For those spiking patterns where the pre-synaptic neuron spike was followed by a post-synaptic spike more recently than it was preceded by the post synaptic spike, the weights were increased by a random value obtained from a uniform distribution described in Equation 6.1 which will be called long term potentiation (LTP).

For those pre-synaptic spikes that were preceded more recently by a post-synaptic spike than one that followed, underwent a weight decrease of a random uniform value, again described by Equation 6.1, which will be called long term depression (LTD). This is formalised in Equation 6.2. An inverse rule was used for the bottom 20% of episodes where LTD was used when the post-synaptic spike followed the

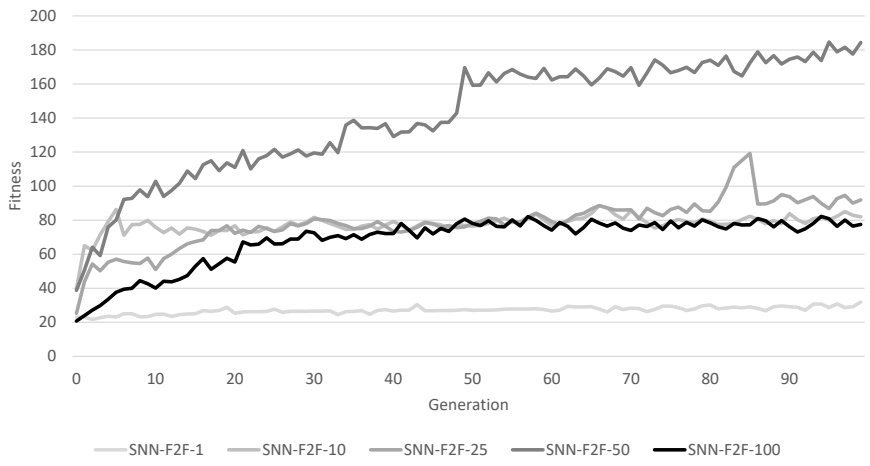

Figure 6.3: *Average fitness of 5 separate trials over each learning cycle for each of the exposure periods tested.*

pre-synaptic spike and LTP was used when the post-synaptic spike preceded the pre-synaptic one. In order to prevent the explosion of connection weights either to really large positive values or large negative values, the network, after undergoing a learning cycle, used the weight decay technique specified in Experiment 6 and described in Equation 6.3.

6.3.2 Results

For trials with 1 exposure, the same trend was observed when compared to the EA method as the minimal spiking activity in the input layer led to random action selection by the output layer in most instances. For 10, 25 and 100 exposure periods, an upwards trend in agent fitness was observed however they appeared to get stuck in local optimums. The only exposure period that had some trials succeed was 50 and this may be due to a favourable α value.

The DSE method was effective at producing networks with adequate spiking behaviour to produce agents with enough information to make a decision. However, it required the explicit transformation of the initial values to values between

0 and 1. This means that this method may be more difficult to generally employ on large numbers of input variables with differently defined maximum values. This same problem was encountered in Experiment 2 when using evolutionary processes but could not be overcome as simply as with the Input Encoder used in Experiment 3 as training those weights without evolution is difficult. The Input Encoder weights don't spike, they simply multiply the incoming injection to the input layer and therefore the processes of STDP cannot be applied. However, the method used in this experiment may be useful for pixel-based information with consistent maximum values across all inputs. This was also tested with a normalized probability encoding where the eight values were normalized to sum to 1 and these new values used as the probability of spiking in one instance of exposure. Even when multiplied by some scaling factor, the normalized probability method produced uneven weighting on those inputs that were often much larger than others. The normalized probability method was observed to not produce enough spiking activity in the input layer therefore producing less spiking in the output layer leading to worse performance by agents using this method.

A rate based decoding method was used where the output neuron with the highest average spiking rate over the exposure period was selected. For those instances where neither output neuron spiked, or they spiked the same amount in the exposure period, then a random action was selected. This explains the observed behaviour for the 1 exposure period trials as it corresponds to what is expected for random action selection. This method appears to be effective in selecting actions in the other exposure period situations. It appears that with more information (i.e., longer exposure periods), the networks were better at selecting the appropriate action to take and this is evident through the less training cycles needed to reach the learning goal using the EA method for those with longer exposure periods. It is not clear whether this is related to more information being input or allowing for more spiking behaviour in the output layers to make a more informed decision.

The stagnation of fitness in the EF-STDP method without reaching the goal average fitness of 195 indicates the networks are becoming stuck in a local optimum.

This stagnation is most likely a result of a lack of exploration of the state space. Additionally, the initial weights of the network are important as some of the trials never exceeded the ability of random action selection whereas others that did improve tended to have a rapid improvement in the early learning cycles with stagnation later in training. In some instances, after stagnating for large periods of time, rapid increases in fitness were observed. This signifies a breaking out of those local optimums possibly due to the randomness in the weight changes. This requires further work to improve for the lack of exploration in the current algorithm.

Additionally, more work is needed in identifying better performing α values and how different α values perform for different η values. This may include recrafting equation 6.1 as it is observed that some α values work better for lower exposure periods and others work better for higher exposure periods but a single range of α values that work well over all η values has not been found.

6.4 Experiment 8 - EF-STDP Search for Better Learning Rates

In line with the discussion from the previous experiment and the accompanying results, it was clear that the calculation for the learning rate outlined in Equation 6.1 was not great at producing appropriate learning rates for various exposure periods. From observation, with differing α values, different exposure periods performed better than the rest. Therefore, it is important to discover how the α values are related to η (exposure period). The equation in the previous experiment to decide the learning rate of the was linear as can be seen in Equation 6.4.

$$lr = \frac{\alpha}{\eta} \tag{6.4}$$

As this did not result in good fitness for a range of different η values, the best relationship to calculate the learning rate depending on the exposure period may

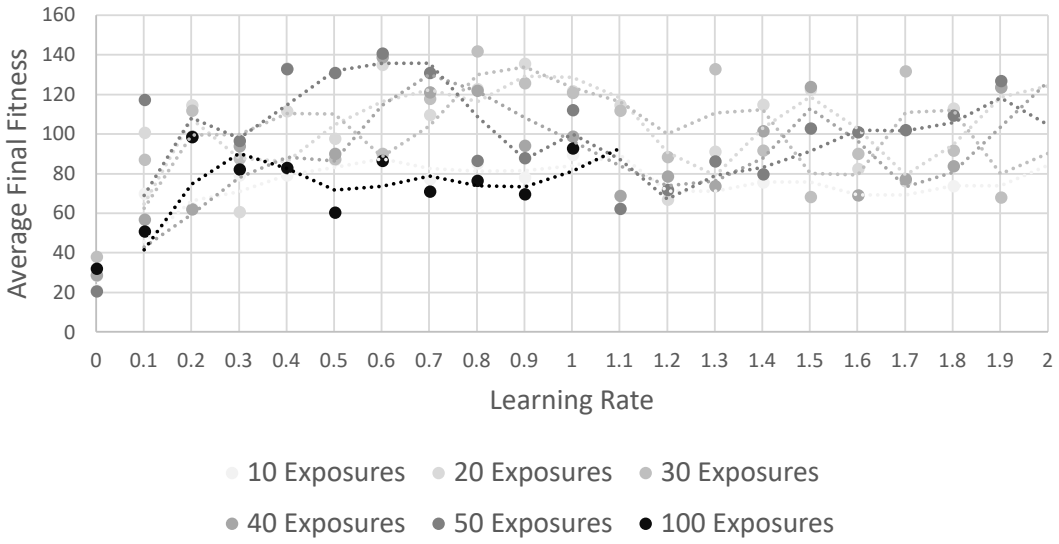

Figure 6.4: *The average final fitness for the exposure period and learning rate combinations after 100 learning cycles*

not be linear. Therefore, an exploratory experiment needed to be conducted to determine the best learning rate equation for the EF-STDP method.

6.4.1 Method

The value of X as identified in Equation 6.1 was not calculated based on that equation but instead multiple X values were used for all exposure periods tested in the previous experiment. This is to identify a possible link between learning rate and exposure period and in order to find a more appropriate equation for a general α value among all η values. For η values of 10, 20, 30, 40 and 50, learning rates ranging from 0.001 to 2.001 were used with increments of 0.100. Six trials were conducted for each combination of learning rate and η value where 100 learning cycles were conducted. The average final fitness of these combinations was plotted as can be seen in Figure 6.4.

The moving averages of a period of 2 were plotted and the resultant learning rate of the peak of the moving average was then plotted in Figure 6.5

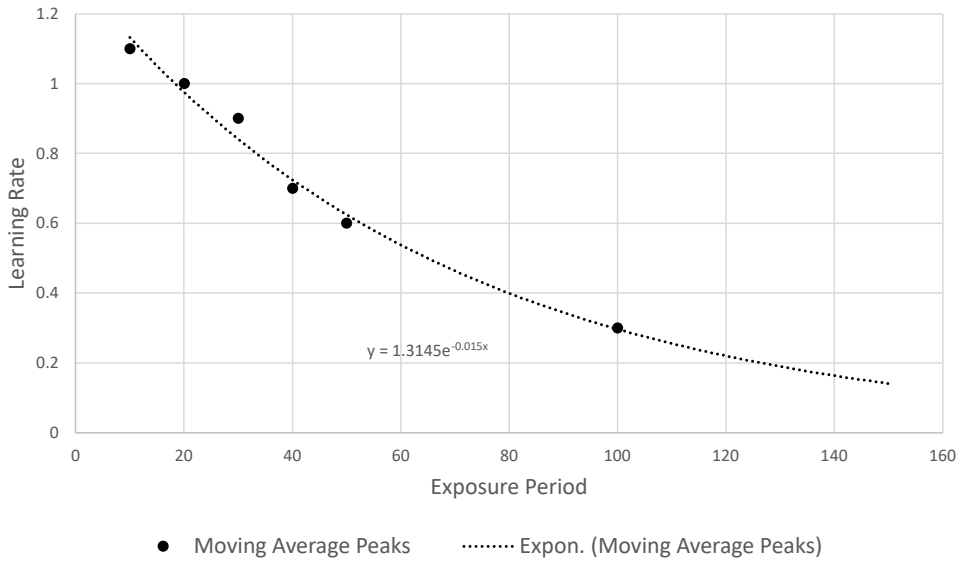

Figure 6.5: *The learning rate fitness peaks of the moving average.*

6.4.2 Results

From Figure 6.5, an exponential line of best fit was the most appropriate predictor of the learning rate that should be used for a specific exposure period that would result in the production of agents with the best fitness. Therefore, for the EF-STDP method, the selection of a learning rate will be done using equation 6.5.

$$lr = 1.3145e^{-0.015\eta} \tag{6.5}$$

More extensive trials are needed to identify if this is the best way of calculating learning rates in terms of the exposure periods of the networks.

6.5 Summary

R-STDP algorithms found in the literature cannot be easily applied to more complex RL problems as identified in Experiment 5 and 6. Most likely, the poor performance of the algorithm was a result of the continuous reward signal received by the agent in the Cart Pole problem as hypothesised in Experiment 5. Experimenting with alternate reward structures is necessary if R-STDP is to work in

its current form. Additionally, due to the inability of these networks to perform well in the Cart Pole problem, an analysis of different weight constraining methods is difficult. Although it was found that weight decay constraining methods produce the most similar weight distributions to those in the evolutionary experiments, rating its performance alongside the other weight constraining methods is not possible. The novel EF-STDP algorithm showed that spike timing dependant learning does allow agents to perform better than random action taking however, no goal achieving agent was produced. This is assumed to be due to a lack of exploration of the state space as is necessary with traditional ANN RL algorithms to prevent the networks from becoming stuck in local optimums. In conclusion, spike timing Hebbian training methods requires much more work to make SNNs a viable alternative to the state-of-the-art ANN RL methods.

Chapter 7

Conclusions and Future Work

The aim of this thesis was to explore the use of spiking neural networks (SNNs) within the reinforcement learning (RL) domain. In order to achieve this, it was broken into three research areas. Firstly, to test whether the structure of a SNN was capable of solving problems within this domain in both single and multi-layer formats, and this was done by using well established evolutionary methods. Secondly, if the SNNs were able to solve these types of problems, were there non-evolutionary methods that were capable of training the weights of these networks to provide a solution to the tested problems. Thirdly, to test different network structures, weight constraining, encoding, and decoding methods to determine which provide better outcomes for the learning process and which are capable of solving the problems tested.

Initial experimentation looked at the comparison between using evolutionary processes to train a standard artificial neural network (ANN) and compare that to a SNN with the same network structure. This identified an initial issue of the SNN in that the magnitude of the input values into the first layer are important in order to encourage spiking activity in the network compared to the ANN where this is not an issue. Increasing the magnitude of the state space values of the problem produced SNNs that were capable of reaching the goal state. Interestingly, this goal state for higher exposure periods of 100 were reached in far fewer generations than that of the ANN.

Due to the identification that injection of just a few state space values dominated the spiking behaviour of the network led to the transformed input space experiment. This experiment attempted to scale the input values to only operate between 0 and 1. This led to a more even spread of spiking activity in the input layer and therefore, no set of input values dominating the network. As a result of the scaling of the input values, networks with lower exposure periods, namely the 25 exposure periods were also able to achieve the problem goal, which is preferable as lower exposure periods lead to more efficient network learning and execution as less network iterations are required to make a decision.

Because of the non-normalized nature of the input values in both the Cart Pole and Lunar Lander experiments, the empirically tuned state space values required a large number of pre-processing experiments to decide on the required scale factors of the input values. This therefore led to the development of the novel Input Encoder method which attempted to use the same evolutionary process for training the weights of the network, to also train the scale factors of the input values. This method was successful over all decoding methods and exposure periods tested when used with the Cart Pole problem. The Lunar Lander testing created networks that were able to solve the problem, however, averagely did not produce goal reaching agents.

One of the major issues with using an Input Encoder for training networks in non-evolutionary methods is the training of those scale factor weights cannot take advantage of traditional spike-based learning as the pre-processing layer does not exhibit the same spiking behaviour as the main network. Therefore, an experiment was conducted to test a range of different encoding methods using the same evolutionary processes to see which of those encoding methods were capable at producing goal achieving agents. Methods of Direct Encoding (DE) and their variants were able to produce these goal achieving agents in all circumstances in the Cart Pole problem. Probability Encoding (PE) methods and their variants were not able to produce these goal achieving agents however did show initial learning and a stagnation at high fitness levels.

Reward Modulated Spike Timing Dependant Plasticity (R-STDP) methods outlined in the literature were incapable of producing agents that showed any sign of initial learning in this research. Although R-STDP was routinely mentioned in literature as a non-evolutionary based training method for SNNs in RL problems, this was only demonstrated on simple problems and was not, in its current form, able to produce goal achieving agents in Cart Pole through the experimentation conducted in this research. Therefore, it was decided to attempt to use Spike Timing Dependant Plasticity (STDP) in conjunction with evolutionary fitness-based methods for training. This led to the development of the novel Episodic Fitness based Spike Timing Dependant Plasticity (EF-STDP) algorithm. This method did show rapid initial learning using Direct Encoding methods and rate-based decoding. However, the agents appeared to become stuck in local optimums suggesting that the networks lacked exploration of the state space and therefore were not able to achieve the goal. Non-evolutionary training methods including STDP methods for RL requires more work to make SNNs a viable alternative to ANNs for RL problems and although attempted in this research, produced the weakest answer to all of the research questions proposed.

In the remainder of this chapter, the research questions will be outlined and answered with reference to the experiments conducted throughout this project, as well as a description of the limitations of this research and some possible areas for future work.

7.1 Answers to Research Questions

This section will answer the three research questions posed in Chapter 1 and will do so with reference to the experimentation conducted and outlined in Chapter 4 and 5.

7.1.1 RQ1: Are SNN structures suitable at solving RL problems?

RQ1 - A: Using evolutionary approaches for training, are SNNs without hidden layers effective at solving RL problems?

Most of the answers to this question can be found in Experiment 3 in Section 4.4. Utilising both the Cart Pole and Lunar Lander RL problems, individual single layer networks utilising a range of different encoding and decoding methods were able to achieve solutions to the problems. Cart Pole, although the simplest of the two problems, allowed for the quick learning using an evolutionary method to produce a goal achieving agent. In all circumstances, the single layer networks produced a solution to this problem. Therefore, for simple RL problems, single layer SNNs are effective to use as a network architecture to produce a solution. When looking at a more complex RL problem like Lunar Lander the answer is not as clear. Although individual networks were able to achieve solutions to this problem, some were not within the specified generational limit. However, it is still clear that there existed a solution to this problem using a single layered SNN. Therefore, we can conclude that single layer SNNs are effective at solving a range of RL problems. However, this research does not conclude that these single layer networks are suitable at solving all RL problems. It can be assumed, just like with regular ANNs, some problems require the use of a hidden layer to solve.

RQ1 - B: Using evolutionary approaches for training, are SNNs with a hidden layer effective at solving RL problems?

Similar to the answer above, the evidence for this answer comes from Experiment 3 in Section 4.4. Utilising the simple RL problem of Cart Pole, all networks that utilised a hidden layer with either 16 or 32 hidden neurons were able to achieve the goal fitness. Comparing that to the more complex Lunar Lander problem, none of these same network structures were able to achieve a solution. However, the networks did show improvement over generations using the evolutionary algorithm and achieved average finesses well above that of the random benchmark. Therefore, this research can conclude that multi-layer SNNs are effective at solving simple RL problems and can be used to produce well performing agents in more complex RL problems. However, similar to the previous answer, it cannot be concluded that SNNs are an effective architecture at solving all RL problems.

7.1.2 RQ2: What network constraining and network coding methods improve training in RL environments?

RQ2 - A: What affect does the exposure period of the state space to the network have on network performance?

Throughout all evolutionary experiments from Chapter 4, a clear negative correlation emerges between the length of the exposure period and the generation for the network to achieve the goal fitness. Additionally, for those networks unable to reach a solution, the greater the exposure period of the network, the more rapid the initial fitness increases over generations. Although this phenomenon appeared throughout these experiments, it appears that this trend would not continue in exposure periods greater than a certain threshold. For example, in Experiment 3 in Section 4.4 with exposure periods of 80 and 100, this trend does not appear. It can therefore be suggested that there is a certain exposure period where the benefit of steeper learning becomes non-existent or not important. Not only is this important for judging how to make the most effective SNNs, but it also introduces a trade

off when executing these networks on traditional computing hardware. This trade off becomes between how steep the learning is compared to the time of execution for each step in the episode. It is also clear when looking at Experiment 1 and Experiment 2 (Section 4.2 and 4.3 respectively), that there are a lower bound of exposure periods that produce ineffective networks for the specific RL problem. For example, using a Direct Encoding method in Experiment 1, exposure periods of 1, 5, 10 and 20 were incapable of producing solutions to the Cart Pole problem. This is proposed to be because there is not enough spiking activity in the network over that time frame to be able to choose a well-informed action. Therefore, not only is there an upper bound to the benefits of higher exposure periods, but there also exists a lower bound under which networks cannot make correct or informed decisions on what action to take in the environment.

RQ2 - B: What methods are effective at decoding spike train signals into action selection?

The two main categories of decoding method explored in this research are Temporal and Rate based decoding. The former specifies that the precise spike timings are important for action selection and the latter suggests that the frequency of spikes occurring in a neuron is important for action selection. Two temporal decoding methods were utilised, namely the First to Fire (F2F) and First to Fire Reset (F2F R) and two rate based decoding methods were used, namely Rate and Rate Reset (Rate R) methods. No clear difference in the performance between the two main categories emerged throughout the research. However, the main distinction between the two occurred at lower exposure periods. F2F and F2F R appeared to perform better at exposure periods of 20 and 40 when compared to Rate and Rate R. This was assumed to be because lower exposure periods do not give the network enough output layer spiking activity to produce multiple spikes over the exposure time period and is therefore more likely that a random action will be taken in rate-based methods. For exposure periods of 60, 80 and 100, no clear difference between Temporal and Rate based methods emerged. The more interesting result

came from the comparison between the normal and reset counterparts of the two decoding methods. From the results of Experiment 3, it is clear that the reset counterparts of both F2F and Rate decoding methods caused more rapid learning reaching goal achieving agents in far fewer generations. Most likely this is due to less noise from previous episode steps remaining in the networks and they can therefore make a more informed decision on that episode step in isolation from other steps. However, this may not be the case for other RL problems where knowledge of previous states is useful for deciding an action.

RQ2 - C: What methods are effective at encoding state space information into spike train signals?

From the results of both Experiment 3 and Experiment 4, it appears that direct encoding methods perform much better than probability based encoding methods. The novel Input Encoder proposed in Experiment 3 performed well at both the Cart Pole and Lunar Lander problems and is essentially the same as the Direct Scaled Encoding (DSE) method explored in Experiment 4. The difference being that the scale factors of the Input Encoder were learned and the scale factors of the DSE algorithm were hand crafted. Both injected the scaled input values directly into a first layer of IZ neurons. Probability based encoding methods produced more noise in the network as the same state would produce different spiking activity in the input layer on different trials. It is therefore more difficult for a network to learn the weights due to that noise. Therefore, methods that use Direct Encoding into a SNN with scaled inputs that also utilises Double Encoding to remove negative input values seems to be the most appropriate encoding method for RL problems out of the methods tested.

RQ2 - D: What methods effectively constrain the strengthening and weakening of synapse weights?

Due to the unrestricted nature of weight changes in non-evolutionary training leading to the weight explosion problem, methods of constraining the weights are

necessary. Due to the difficulty in getting non-evolutionary training working on the problems explored in this research, it is hard to answer this question conclusively. Methods of pre-synaptic normalisation, post-synaptic normalisation and weight decay were tested. Pre- and post-synaptic normalisation methods often led to the pooling of weights at their threshold values which caused networks to have very similar weights across all synapses. On the other hand, weight decay methods provided a method for keeping a range of different synapse strengths spread throughout the network creating weights that look more like those produced through evolutionary training. This brought with it is own challenges of introducing a new hyperparameter that needed to be tuned which could, if too big, make all the weights too small and if too small, create the same weight explosion problem as identified with no constraining method. Although these methods could not be tested extensively due to the inability to produce working algorithms for the Cart Pole or Lunar Lander problems, observationally, this research suggests that weight decay methods are more effective than the pre- and post-synaptic normalisation methods.

7.1.3 RQ3: Due to the inability to use gradient based algorithms for the training of SNNs, are non-evolutionary learning techniques effective at solving RL problems?

This question has the least evidence sourced through this research compared to the other questions and therefore a conclusive answer is not able to be given. More research in this area is required to form a better answer to these sub questions and will be explored in the Limitations and Future Work section.

RQ3 - A: Can learning methods based on spike timing effectively solve RL problems?

Although not able to be shown in this research, the literature suggests that for simple RL problems, STDP variants are capable of solving RL problems. However,

those methods used in the literature appear not to be able to be easily applied to more complex RL problems. This requires far more research into non-evolutionary training techniques to answer conclusively.

RQ3 - B: Can learning methods that incorporate aspects of evolutionary and spike timing training effectively solve RL problems?

The novel EF-STDP algorithm proposed in this research showed more promising results when compared to the traditional R-STDP. This method showed initial improvement in the early episodes of training however appeared to get stuck in local optimums. This suggests that this method lacks the crucial exploratory capabilities of other traditional ANN RL algorithms. Applying random mutations networks with stagnating fitness may be able to add this exploratory ability however was unable to be tested in this research. This question will also be further explored in the Future Work section of this Chapter.

7.1.4 Research Questions Conclusions

The evolutionary experiments conducted in Chapter 4 allowed for conclusive answers to RQ1 and RQ2 (excluding RQ2-D). However, the inability to demonstrate well performing agents trained through non-evolutionary methods led to difficulties answering RQ3. This research therefore suggests that much more work must be done in future to be able to answer RQ3 conclusively.

7.2 Limitations

This section will cover the limitations of this research in respect to answering the research questions in the previous section. The limitations can be broken into two main categories: technical limitations and algorithmic limitations.

7.2.1 Technical Limitations

As identified by Davies et al. (2018), traditional computer hardware struggles to efficiently model SNN architectures as it does not allow for the abundant use of parallel processes. These networks are drastically more efficient on neuromorphic based hardware. However, through the conduct of this research, using neuromorphic computing was not possible due to the difficulty in accessing the technology as it is still being actively developed. Therefore, the magnitude of the experiments conducted had to be limited in order to deal with the inefficient processing of the networks on traditional hardware. This meant fewer trials per experiment had to be conducted therefore making it more difficult to identify trends as results data from average trial runs was often noisy due to the variations between trials. This also restricted the ability to run evolutionary algorithms for large numbers of generations due to the immense processing time required. This was especially evident when trialling with the Lunar Lander problem as attempting to analyse trends over 100 generations of training was impractical due to time it would take to process. Additionally, access to computing hardware was limited during the research period which also limited the number of trials able to be run, especially when paired with the slow execution time of SNNs on traditional hardware. Therefore, the technical limitations of traditional computing hardware meant the scale of experiments run had to be limited. This research attempted to achieve results that were adequate to identify possible trends whilst minimising the execution time of the networks.

7.2.2 Algorithmic Limitations

One of the major issues of an SNN using IZ neurons (Izhikevich 2003) is the number of hyperparameters surrounding the behaviour of individual neurons in the network. Namely the a, b, c, and d parameters outlined in Equation 2.1 and 2.2, as well as the peak voltage variable. It was therefore difficult to have an in-depth look at the effect that changes to those variables has on the performance of the network due to the technical limitations described in the previous subsection. Furthermore,

the inability to get forms of R-STDP working, hindered the analysis that could be done on that algorithm. This was most likely a result of the reward structure of the Cart Pole problem, but due to an inadequacy of time, testing differing reward structures such as discounting rewards or punishing rewards for certain actions could not be conducted. This made it difficult to draw any reasonable conclusions from the R-STDP experiments. When working on the novel EF-STDP method, it introduced more hyperparameters which made it increasingly difficult to extract useful analysis from it. The hyperparameter experiment conducted attempted to address this issue but the extensive testing of the results of this hyperparameter experiment was unable to be done due to time constraints. Majorly, the algorithmic limitations identified in this research is the range of different hyperparameters coupled with the slow execution times of the networks. This led to the inability to extensively run hyperparameter experimentation which may have resulted in worse performance for the networks.

7.3 Future Work

This section will cover possible future work to both continue to gather evidence for the research questions, and also further work in the field of SNNs for RL problems. The weakest answer produced in this research was around non-evolutionary methods of training SNNs. It is clear from the literature that uses of SNNs in either supervised or unsupervised learning environments are more prominent, however, they still are unable to reach the same performance and efficiency levels as their ANN counterparts. This is predominately due to the lack of research into this method, as well as an issue with simulating these networks efficiently on traditional computing hardware. When adapting algorithms for SNNs developed in the supervised and unsupervised learning domains for the RL domain, the literature clearly shows that these methods are well outperformed by the traditional back-propagation techniques of ANNs. With the development of new computing techniques, such as neuromorphic computing, the efficiency viability of SNNs will

be met and is predicted to be more efficient than current ANN techniques on traditional hardware (Wunderlich et al. 2019). Testing the methods developed in this research on neuromorphic hardware for reproducibility of results will paint a clearer picture as to whether there are benefits over existing hardware for network efficiency and will allow more detailed trials to be run. The requirement of making SNNs viable in RL domains comes from further development of these non-evolutionary techniques. This could be done by improving on current R-STDP methods, development of novel learning methods or the continuation of research into the proposed EF-STDP method.

Outlined in Experiment 3, some difficulties arose around the evolution of SNNs in the Lunar Lander problem. This could be addressed by running the networks for more generations than currently tested. A future experiment that does this will be able to further confirm the trends found when using the Cart Pole problem, such as the negative correlation between exposure period and generation to reach goal, as well as the trend of less random actions being taken when longer exposure periods are used. Additionally, investigating a range of different evolutionary algorithms including using methods of Neuro-Evolution of Augmenting Topologies (Stanley and Miikkulainen 2002) to alter the network structure during learning would be useful future work.

Izhikevich (2003) describes that changes to the parameters of the IZ neurons, as described in Equations 2.1 and 2.2, results in the replication of a range of differently behaving spiking neurons found in the brain. Testing combinations of these neuron parameters for different layers in the network would be useful to identify trends and to be able to analyse the effect that those parameters have on the performance of the networks. It could be that certain different types of spiking neurons may work better for shorter or longer exposure periods or may work better with a different type of spiking neuron in the input layer compared to the output layer. This would be a useful contribution to the development and analysis of the IZ SNN structure and is intended to be done in future. Additionally, investigating adapting parameters during learning has the possibility of making parameter

selection more generalised to be able to take on parameters that maximise agent fitness.

The work outlined in this section would be incredibly useful in furthering research into the use of SNNs for RL problems. As SNNs are representations of biological neurons, a deeper understanding of neuroscience may also lead to the breakthroughs needed to develop learning techniques for these biologically representative networks. Additionally, continuing to build on spike timing dependant learning methods to reach the performance levels of traditional RL training for ANNs is needed. In order for SNNs to offer itself as a viable alternative to traditional ANNs for RL problems, robust and effective training methods must be developed.

Bibliography

Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M. & Knoll, A., 2018, 'End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle', *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8.

Borowiec, S., 2016, 'Alphago seals 4-1 victory over go grandmaster lee sedol', *The Guardian*, vol. 15.

Braitenberg, V., 1986, *Vehicles: Experiments in synthetic psychology*, MIT press.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W., 2016, 'Openai gym', *CoRR*, vol. abs/1606.01540, ¡ `http://arxiv.org/abs/1606.01540`¿ .

Buesing, L., Bill, J., Nessler, B. & Maass, W., 2011, 'Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons', *PLoS computational biology*, vol. 7, no. 11, p. e1002211.

Burkitt, A. N., 2006, 'A review of the integrate-and-fire neuron model: I. homogeneous synaptic input', *Biological cybernetics*, vol. 95, no. 1, pp. 1–19.

Campbell, M., Hoane Jr, A. J. & Hsu, F.-h., 2002, 'Deep blue', *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83.

Cao, Y., Chen, Y. & Khosla, D., 2015, 'Spiking deep convolutional neural networks for energy-efficient object recognition', *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66.

Daucé, E., 2009, 'A model of neuronal specialization using hebbian policy-gradient with "slow" noise', Alippi, C., Polycarpou, M., Panayiotou, C. & Ellinas, G. (eds.) *Artificial Neural Networks – ICANN 2009*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 218–228.

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y., Wild, A., Yang, Y. & Wang, H., 2018, 'Loihi: A neuromorphic manycore processor with on-chip learning', *IEEE Micro*, vol. 38, no. 1, pp. 82–99.

Domingos, P., 2018, *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*, Basic Books, Inc., New York, NY, USA.

Eskandari, E., Ahmadi, A., Gomar, S., Ahmadi, M. & Saif, M., 2016, 'Evolving spiking neural networks of artificial creatures using genetic algorithm', *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 411–418.

Gamez, D., Fountas, Z. & Fidjeland, A. K., 2013, 'A neurally controlled computer game avatar with humanlike behavior', *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 1, pp. 1–14.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E., 2020, 'Array programming with NumPy', *Nature*, vol. 585, p. 357–362.

Hausknecht, M., Khandelwal, P., Miikkulainen, R. & Stone, P., 2012, 'Hyperneat-ggp: A hyperneat-based atari general game player', *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, ACM, New York, NY, USA, pp. 217–224, ¡ `http://doi.acm.org/10.1145/2330163.2330195`¿ .

Hebb, D. O., 1949, *The Organization of Behavior: A Neuropsychological Theory*, Lawrence Erlbaum Associates.

Hodgkin, A. L. & Huxley, A. F., 1952, 'A quantitative description of membrane current and its application to conduction and excitation in nerve', *The Journal of physiology*, vol. 117, no. 4, pp. 500–544.

Izhikevich, E. M., 2003, 'Simple model of spiking neurons', *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572.

Kaiser, J., Tieck, J. C. V., Hubschneider, C., Wolf, P., Weber, M., Hoff, M., Friedrich, A., Wojtasik, K., Roennau, A., Kohlhaas, R. et al., 2016, 'Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks', *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, IEEE, pp. 127–134.

Kamali Sarvestani, I., Kozlov, A., Harischandra, N., Grillner, S. & Ekeberg, Ö., 2013, 'A computational model of visually guided locomotion in lamprey', *Biological Cybernetics*, vol. 107, no. 5, pp. 497–512, ¡ `https://doi.org/10.1007/s00422-012-0524-4`¿ .

Kiselev, M., 2016, 'Rate coding vs. temporal coding – is optimum between?', .

Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998, 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324.

Lee, J. H., Delbruck, T. & Pfeiffer, M., 2016, 'Training deep spiking neural networks using backpropagation', *Frontiers in Neuroscience*, vol. 10, p. 508, ¡ `https://www.frontiersin.org/article/10.3389/fnins.2016.00508`¿ .

Li, M. & Tsien, J. Z., 2017, 'Neural code—neural self-information theory on how cell-assembly code rises from spike time and neuronal variability', *Frontiers in cellular neuroscience*, vol. 11, p. 236.

Markowska-Kaczmar, U. & Koldowski, M., 2015, 'Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game', *Soft Computing*, vol. 19, no. 12, pp. 3465–3478, ¡ https://doi.org/10.1007/s00500-014-1515-2¿ .

Mitchell, I., Huyck, C. & Evans, C., 2016, 'Planeural: spiking neural networks that plan', *Procedia Computer Science*, vol. 88, pp. 198–204.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M., 2013, 'Playing atari with deep reinforcement learning', *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al., 2015, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, p. 529.

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A. & Ganjtabesh, M., 2018, 'First-spike-based visual categorization using reward-modulated stdp', *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6178–6190.

Müller, J., Nawrot, M., Menzel, R. & Landgraf, T., 2018, 'A neural network model for familiarity and context learning during honeybee foraging flights', *Biological Cybernetics*, vol. 112, no. 1, pp. 113–126, ¡ https://doi.org/10.1007/s00422-017-0732-z¿ .

Neil, D., Pfeiffer, M. & Liu, S.-C., 2016, 'Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks', *Proceedings of the 31st annual ACM Symposium on Applied Computing*, ACM, pp. 293–298.

Olmsted, D. D., 2011, 'Foraging behavior in a 3-d virtual sea snail having a spiking neural network brain', *The 2011 International Joint Conference on Neural Networks*, pp. 90–94.

pandas development team, T., 2020, 'pandas-dev/pandas: Pandas', ¡ https://doi.org/10.5281/zenodo.3509134¿ .

Qiu, H., Garratt, M., Howard, D. & Anavatti, S., 2018, 'Evolving spiking neural networks for nonlinear control problems', *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, pp. 1367–1373.

Shim, M. S. & Li, P., 2017, 'Biologically inspired reinforcement learning for mobile robot collision avoidance', *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3098–3105.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. et al., 2018, 'A general reinforcement learning algorithm that masters chess, shogi, and go through self-play', *Science*, vol. 362, no. 6419, pp. 1140–1144.

Slade, S. & Zhang, L., 2018, 'Topological evolution of spiking neural networks', *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1–9.

Stanley, K. O. & Miikkulainen, R., 2002, 'Evolving neural networks through augmenting topologies', *Evolutionary computation*, vol. 10, no. 2, pp. 99–127.

Stevens, C. F. & Zador, A. M., 1998, 'Input synchrony and the irregular firing of cortical neurons', *Nature neuroscience*, vol. 1, no. 3, p. 210.

Strubell, E., Ganesh, A. & McCallum, A., 2019, 'Energy and policy considerations for deep learning in nlp', *arXiv preprint arXiv:1906.02243*.

Sutton, R. S. & Barto, A. G., 2018, *Reinforcement learning: An introduction*, MIT press.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A., 2019, 'Deep learning in spiking neural networks', *Neural Networks*, vol.

111, pp. 47 – 63, ¡ http://www.sciencedirect.com/science/article/pii/S0893608018303332¿ .

Tournament, U., 2004, 'Epic games', .

Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A. & Eleftheriou, E., 2016, 'Stochastic phase-change neurons', *Nature nanotechnology*, vol. 11, no. 8, p. 693.

van Hasselt, H., Guez, A. & Silver, D., 2016, 'Deep reinforcement learning with double q-learning', ¡ https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389/11847¿ .

VanRullen, R., Guyonneau, R. & Thorpe, S. J., 2005, 'Spike times make sense', *Trends in neurosciences*, vol. 28, no. 1, pp. 1–4.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Wu, Y., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D. & Silver, D., 2019, 'AlphaStar: Mastering the Real-Time Strategy Game StarCraft II', https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Vitanza, A., Patané, L. & Arena, P., 2015, 'Spiking neural controllers in multi-agent competitive systems for adaptive targeted motor learning', *Journal of the Franklin Institute*, vol. 352, no. 8, pp. 3122–3143.

Vreeken, J., 2003, 'Spiking neural networks, an introduction', .

Wang, G., Zeng, Y. & Xu, B., 2016, 'A spiking neural network based autonomous reinforcement learning model and its application in decision making', Liu, C.-L., Hussain, A., Luo, B., Tan, K. C., Zeng, Y. & Zhang, Z. (eds.) *Advances in*

Brain Inspired Cognitive Systems, Springer International Publishing, Cham, pp. 125–137.

Wiklendt, L., Chalup, S. & Middleton, R., 2009, 'A small spiking neural network with lqr control applied to the acrobot', *Neural Computing and Applications*, vol. 18, no. 4, pp. 369–375.

Wu, Y., Deng, L., Li, G., Zhu, J. & Shi, L., 2018, 'Spatio-temporal backpropagation for training high-performance spiking neural networks', *Frontiers in neuroscience*, vol. 12.

Wunderlich, T., Kungl, A. F., Müller, E., Hartel, A., Stradmann, Y., Aamir, S. A., Grübl, A., Heimbrecht, A., Schreiber, K., Stöckel, D., Pehle, C., Billaudelle, S., Kiene, G., Mauch, C., Schemmel, J., Meier, K. & Petrovici, M. A., 2019, 'Demonstrating advantages of neuromorphic computation: A pilot study', *Frontiers in Neuroscience*, vol. 13, p. 260, ¡ `https://www.frontiersin.org/article/10.3389/fnins.2019.00260`¿ .

Yanguas-Gil, A., 2018, 'Going small: Using the insect brain as a model system for edge processing applications', *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, ACM, New York, NY, USA, pp. 373–378, ¡ `http://doi.acm.org/10.1145/3194554.3194610`¿ .

Yee, E. & Teo, J., 2011, 'Evolutionary spiking neural networks as racing car controllers', *2011 11th International Conference on Hybrid Intelligent Systems (HIS)*, pp. 411–416.

Zambrano, D. & Bohte, S. M., 2016, 'Fast and efficient asynchronous neural computation with adapting spiking neural networks', .

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J. et al., 2013, 'On rectified linear units for speech processing', *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, pp. 3517–3521.