University of Manchester School of Computer Science Project Report 2021

Training Spiking Neural Networks to play Atari games

Author: M. Abdelrazek
Supervisor: Dr. S. Furber

Abstract

Training Spiking Neural Networks to play Atari games

Author: M. Abdelrazek

This project investigates the abilities of spiking neural networks (SNNs), specifically in the context of playing Atari games. This is done by multiple experiments using different models. The experiments use elements from hebbian learning in addition to traditional reinforcement learning methods. It is found that Reward Modulated Spike Timing Dependent Plasticity (MSTDP), and Backpropagation on SNNs coupled with Q-learning to be the most optimistic models for a good design. Furthermore, future improvements and discussions of results are provided.

Supervisor: Dr. S. Furber

Acknowledgements

I would like to thank Prof. Steve Furber for his continuous support , insights , and supervision. Additionally, many thanks to Dr. Andrew Gait , and Mr. Andrew Rowley for his support with jupyterlab and with integrating the gym with spinnaker. Finally I would like to thank my family and friends for their support.

Table of contents

1 - Introduction

This project aims to implement and compare different ways to train different Spiking Neural Network Architectures to play classic Atari 2600 [11] games using openAI gym [2]. The main focus has been on Two games (Atari Pong and Breakout), and the non-Atari game Cartpole [1] from the classic reinforcement learning literature is used once. Overall, the main features of the project are the following:

- Transforming frames from the game into a better representation
- Using the produced frames as input to the spiking neural network
- Comparing How different Spiking Neural Network (SNN) models approach the problem and how they are trained
- Comparing Results from the different techniques between themselves and an ANN that was made to perform the same function

On the First Two points, using images as an input makes the problem much harder than, for example, supplying the coordinates of the ball and the opponent. From the point of view of seeing Neural Networks as Function Approximators [7], a function with (84x84) pixels as an input will be harder to approximate than a function with only the coordinates. However, this is more similar to the way humans learn to play games, as in automatically extracting coordinates from the images present. This also emphasises the importance of input image quality. That is why a significant chunk was spent trying to modify the frames to better represent what is going on in-game in as compact a form as possible.

Because SNNs are peculiar and different from traditional ANNs (Artificial Neural Networks), the use of different Training Algorithms and different network structures is utilised to try to understand how can biologically inspired networks learn without the use of labelled data, only by evaluating the effect of its actions on the environment.

1.1 Motivation

The motivation behind this project is multi-part. The problem is attractive due to its faint similarity with General Intelligence. With the similarity in approaching almost any Atari game with the same network (with the need to train). The use of spiking neural networks and their unexplored properties is one more aspect of the project that is exciting.

1.1.1 Why Spiking Neural Networks

The main interest of AI research is essentially replicating the functions of the human brain. It is usually the problems that human brains are exceedingly competent at, like controlling the human body, which is a very complicated nonlinear dynamic system [3] that looks orders of magnitude harder than controlling an ATARI game. Human Brains are also excellent at object

recognition and classification, needing very few examples to learn a pattern. ANNs have indeed achieved tremendous improvements in these tasks [8]. However, One aspect of the brain that usually goes unappreciated is its power efficiency needing about 12 watts of power to run [6]. While Training ANNs requires power-hungry GPUs and TPUs, AlphaGo [9], for example, requires the use of 85KW to train [2].

Neuromorphic hardware [4] (hardware that tries to mimic the human brain's architecture) and the Spiking Neural Networks designed to run on it achieve much better power efficiency than an ANN. This increased efficiency means inference, and maybe training can be performed on low-power devices, Increasing the applicability of AI, and opening new opportunities for business use.

Understanding how Spiking Neural Networks train may also give a push toward finding out how to achieve Human-level performance on tasks or give insight into how human brains learn to perform tasks.

Proximity to the biggest neuromorphic super-computer (SpiNNaker [4]) at the University of Manchester is also a great plus as it gives access to hardware and experts.

1.1.2 Why Reinforcement Learning

Trying to solve the task on hand, the thought of using Supervised Learning was dismissed, as while it is more accessible and usually more straightforward to implement. In this case, having saved frames with the right action to take can be a time-consuming process, given the significant number of possible states in the image. This problem was overcome using a way to try to keep the paddle under the ball; however, This was only done to compare supervised learning's performance as otherwise, it did not look like the right way to approach a control problem given that we have a score that can be used as a reward and a limited number of possible actions every frame.

In Feb 2015, Nature's article on DeepMinds paper ("Human-level control through deep reinforcement learning") [9] generated a vast amount of interest. Since the network only takes images as input, it is not restricted to one Atari Game and can be trained in multiple games (unlike the state-based ones that were special for every game). This embodies the versatility of reinforcement learning, where massive pre-labeled datasets are not needed, and only a metric of how good an agent is doing in its environment is required. Nevertheless, reinforcement learning is computationally expensive and does not do well in specific control cases.

1.1.3 Third AI Winter

Access, quality, and availability of datasets are increasing [5]; however, the computing power required to process them is not keeping up [10]. This, coupled with ANNs reaching the maturity stage, may mean a third AI winter. Finding a way to make training more efficient will help AI research remain economically reasonable and reduce its carbon footprint. Additionally, the new paradigm SNNs provide may help in making breakthrough discoveries.

1.2 Project Aims

The aim of this project was to experiment with different methods of performing reinforcement learning on Spiking Neural Networks (SNNs) in order to learn to play Atari 2600 games using game frames as the input. In pursuing this aim, specific objectives have been set:

- Learning:
    - to understand the fundamental difference between SNNs and ANNs
    - to understand training algorithms and methods relating to SNNs and how they compare to backpropagation
    - to understand more about neuromorphic computing's challenges and possible advantageous

- Research:
    - to read literature on general reinforcement learning problems, SNN implementations of reinforcement learning, and general SNN implementations of non-reinforcement learning problems
    - to identify commonly training algorithms for SNNs
    - decide on network designs based on the complexity and feasibility

- Experimentation:
    - to use as many different techniques as time allows with different designs and mechanisms to capture the range of the possible solutions.
    - fairly evaluate each network's performance and gather data about its usability and what is holding it back.
    - find convincing improvements that can improve performance in future iterations.

1.3 Project Structure

This report is a Four chapter report. Organised as follows:

- **Chapter 1** contains the Introduction, defining the problem, stating the way that it will be approached, and the objectives that are hoped to be achieved.
- **Chapter 2** contains the Planning and Literature review section. Where The project's plan is discussed and the relevant literature reviewed

- **Chapter 3** contains the background and theory required to understand the decisions made during the design and experimentation section with as few problems as possible.
- **Chapter 4** contains the experiments performed, network descriptions and the results obtained along with possible explanations for results with examples of overcoming difficulty.
- **Chapter 5** contains the summary, conclusion, and evaluation of the experiments done during the fourth chapter.

1.3 Project Planning

As the previous chapter was concerned with what is being planned for achievement. This part is concerned with how the plan was actually placed on the schedule, and how the response to setbacks and problems happened. Along with how the project was bokeh into smaller more manageable goals to make the most of the time present.

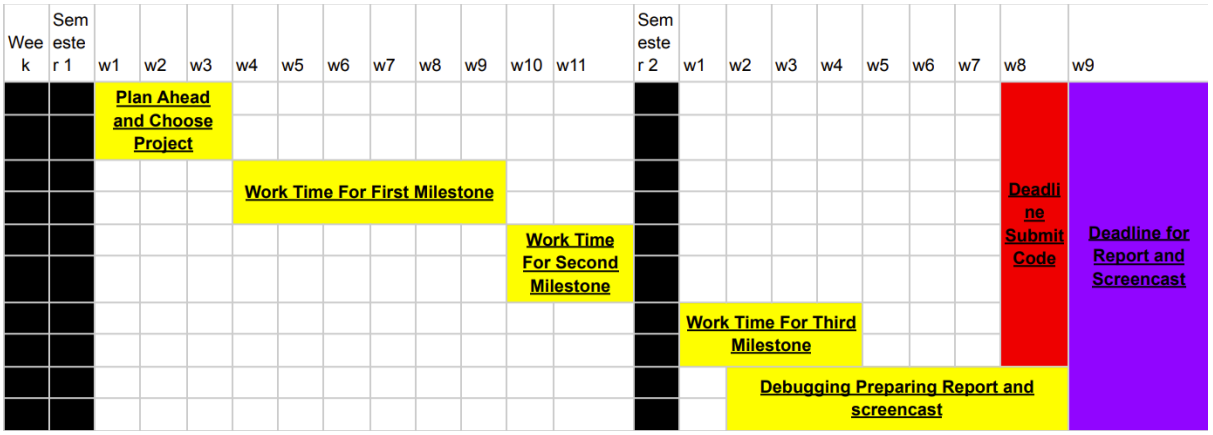


Figure 1 : A Gantt chart created on week 0 with the intention of planning time

The milestones created at the beginning are pretty different from those that were chosen later with learning more about SNNs and the difficulties associated with training them and achieving results. The milestones were changed to involve using different architectures rather than improving on one architecture, as planned to do in the beginning. This was mainly because the First experiment's results were so poor with little-to-no learning happening. This led to the change to the project being more about experimenting with different models that succeed at different things rather than building a single model that solves the task completely.

The first few weeks included a deep dive into relevant literature discussed in the following section. This was needed to find out how others approached the problem, and their results, giving me insight into how to approach the problem at hand differently.

I was advised by my supervisor early on that a lot of changes will probably need to be made to the design of the SNN as it is tough to get it the right the first time, so the decision was

made to make the set the first milestone to creating a workbench that is independent of the Neural Network being run at the time to speed up the implementation of different ideas.

This workbench involved a wrapper to the provided OpenAI gym class to facilitate the running of the games. These functions process the inputs and convert them to spike trains, functions that select actions from the outputted spikes from the network, a training loop, a testing loop, functions that save the complete game as a GIF to look at progress, and evaluation metrics like distribution of scores.

Setbacks happened, especially relating to the time it takes to train the SNN and figure out whether it's working, but with the help of SpiNNaker's software support team and an external library, the problem was made less detrimental to the quality of results obtained. Also, when random/poor performance was obtained it was very hard finding out why the network wasn't working or producing results given how hard it is to figure out issues from the spike graphs (graphs that show the activity of every neuron in terms of the spikes produced by it.). This is why the second and third milestones were further broken into experimental ideas that were formulated after the re-visiting of literature and looking for new reinforcement learning techniques.

1.4 Literature Review and Related Work

Although Spiking Neural Networks are the most broadly adopted brain-inspired computing model , the research area is relatively new and a lot of the concepts are not well understood (especially in the domains of reinforcement learning on SNNs). However, there is still a good amount of relevant literature that will be beneficial to the experiment.

Three types of research papers are looked up, papers on general reinforcement learning where reinforcement learning concepts are being explored generally to benefit from the recent developments of the field, papers on general applications of SNNs in order to learn more about the training algorithms used in those applications, and finally papers about reinforcement learning on SNNs to find about the State of the art , and evaluation metrics used.

- **Spike-Timing-Dependent-Plasticity (STDP)-based applications,**
    - In their paper Diehl, P.U. and Cook, M. (2015) used a SNN with STDP and lateral inhibition to classify MNIST digits, achieving a performance close to that of ANNs; this acts as a reference that STDP actually works in learning patterns.

- **Reward-based Learning by Three-factor STDP,**
    - In their paper Myung Seok Shim and Peng Li [47] proposed the use of Reward-Modulated Spike-Timing-Dependent-Plasticity (MSTDP) to train a robot in collision detection. The robot uses simulated sensor data from sensors

placed on the body of the robot. They found out that MSTDP outperforms a baseline SNN, and proposed the use of Multiplicative-MSTDP where the synaptic weight is updated based on the product of four components: the current weight, learning rate, reward, and eligibility trace and that was found to outperform the traditional Additive-MSTDP model.

- **ANN to SNN Conversion for RL,**
    - In their paper Patel et al.[48] they used matching the graded activations of analog neurons and converted them to firing rates of spiking neurons. When the trained Deep-Q-Network (DQN) is converted to an SNN. They found out that this process creates a trade-off between accuracy and efficiency. Which results in longer inference latency being needed to improve accuracy. As far as the relevant research is concerned [35], for RL tasks, the converted SNNs cannot achieve better results than ANNs.

- **Spiking-Deep-Q-Learning,**
    - Several sources implement backpropagation on spiking neural networks and it has been the go to way to train (multi-layer SNNs as they are more effective than ANN to SNN conversions) using different ways, Using non-spiking neurons and performing backpropagation on membrane voltages is the method used in [35;25]. Backpropagation through time is another method that was used in [53] by Guo et al.

- **Co-learning of Hybrid Framework,**
    - In their paper Tang et al. [50] propose a hybrid framework, composed of a spiking actor network (SAN) and a deep critic network. Through the co-learning of the two networks, these methods [51; 52] avoid the problem of value estimation using SNNs. Hence, these methods are only on actor-critic models, however the energy consumption in the training process is much higher than other standard SNN methods [49].

2 - Background

This section is meant to cover all the ground needed to follow along smoothly with the implementation sections. Attempted at packaging the material so that relevant information is under the same heading. This may not be as easy as a lot of this was learned at different times of the year. Comparing and contrasting ANNs and SNNs will be an excellent start as it is the knowledge that most of my choices are drawn upon in the subsequent sections.

2.1 Artificial Neural Networks (ANNs)

This section is meant to give the background about ANNs. Although an ANN was used in Experiment 3, the main reason for including this section is that it provides a way to suggest methods to use/train SNNs as ANNs are very successful and achieving similar success is important for SNNs to be useful.

The beginnings were in a perceptron algorithm invented by Franc Rosenblatt 1957 [28] ; this was the first attempt to model a biological neural network. Eventually, multi-layer neural network nodes became possible thanks to the dramatic increase in computing power, the presence of more efficient GPUs (Graphics Processing Units) [25] , and the increase in the availability of large datasets used for training ML models [32] . In the 2000s-2010s, these developments gave rise to Deep Learning (DL) [22] , an ANN design based on the presence of multiple layers that can approximate exceedingly complicated functions.

Artificial Neural Networks (ANNs) are multi-layer fully-connected neural nets. They are made of an input layer, hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. The network weights and biases are adjusted in order to minimise the error/loss function; this allows the Network to model functions and probability distributions.

The most basic ANN architecture is the Single-Layer-Perceptron (SLP) then MLPs are basically multiple SLPs joined together. MLPs form the basis of the uses of ANNs and they are explained below.

2.1.1 MLP (Multi-Layer-Perceptron)

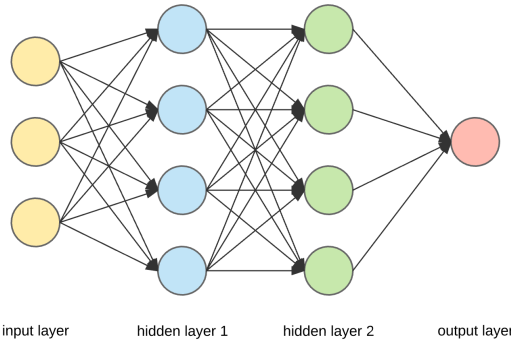


Figure 2: MLP

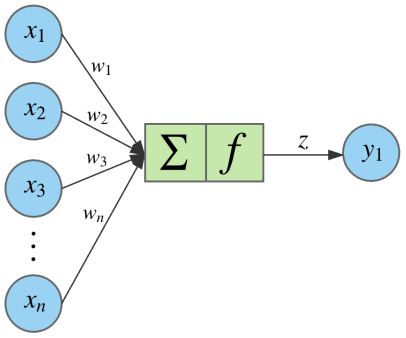


Figure 3: Single neuron as a function

A single node takes the weighted sum of the inputs and passes it through a non-linear activation function. This is the node's output, which becomes another node's input in the next layer. The process is repeated as the values move from one layer to the next.

The process can be done using matrix multiplications, and the outputs are used to fine-tune the Network. These matrix multiplications need a large number of parallel computations that consume power. Techniques to address this have been tried (using TPUs instead of GPUs [31]).

2.1.2 CNN (Convolutional Neural Nets)

MLPs are good but they are weak at tasks like image recognition, and as the task has a heavy dependence on image processing for state identification, CNNs make good candidates for use.

The base work on modern convolutional neural networks (CNNs) was in the 1990s, inspired by neurocognition. Yann LeCun et al., in their paper [23] , demonstrated that a CNN model which aggregates more specific features into progressively more complicated features could be successfully used for handwritten character recognition.

These networks have convolutional layers where a mask/kernel is passed over the data (usually images) and produces a smaller output matrix. The weights on the mask are to be trained using backpropagation. These filters are excellent in tasks like processing the frames of an atari game as they filter out noise from images and keep only the most essential parts. CNN's usually include pooling layers. These are layers that are meant to make the detection position invariant. Pooling layers aren’t used when using a CNN in one of my experiments because pooling affects the position in an image and the position that we are trying to extract.

2.1.3 Backpropagation

An Area where SNNs struggle is the presence of a robust training algorithm that is relatively quick to run. This problem was solved in ANNs using backpropagation. The trend in new research papers seems to be finding backprop equivalents that can work on SNNs.

The backpropagation algorithm is probably the most fundamental building block in an ANN. It was first introduced in the 1960s and almost 30 years later (1989), popularised by Rumelhart, Hinton, and Williams [30] .

Backprop underpins most of the advancements in deep learning. It is used to find an approximation of the global optimum of the weights in a Neural Network. The way it functions is after a forward pass where inputs run through a network. The Network's prediction is passed to the loss function, and the expected values are known beforehand. This loss is propagated using a backward pass (which utilises the chain rule to estimate gradients), trying to reduce loss using gradient descent. These features make backpropagation a supervised learning algorithm by default, as it needs loss values to be calculated [23] .

Backprop can run through the problems of vanishing gradients and exploding gradients (modernly solved using gradient clipping and alternative neuron designs), which may be an area where local learning (as in STDP (Spike-Timing-Dependent-Plasticity)) does not face problems.

This makes backpropagation a biologically implausible learning algorithm (as in it's unlikely that Neurons in living organisms learn using this way), Simply because it requires two-way connections with forward and backward passes needed to propagate error which is not present in biological neural networks. Also, it has the feature of stopping time and going back to change parameters, which is made possible in a deterministic ANN. However, in a biological network, this is impossible. Finally, Neurons in biological networks can only affect the neurons directly connected through synapses, leading to learning being limited to local neighbours only, making a global algorithm affecting synapses through multiple layers unrealistic [29] .

2.1.4 Function Approximators

This section ties in with the parts about backpropagation and MLPs as it explains why backpropagation yields good results especially in the cases where Deep networks are used. This is because backpropagation helps find the weights that fit the activation functions over the distribution of the data. Wrapping this section up with a small example that shows it in action.

The ability of Neural Networks to perform classification tasks and achieve human-level performance in prediction and labelling tasks is due to the Network's ability to model the

distribution of the data it is trained on. This model can be probability distributions like, for example, My First Network, which was trained to play Pong (against a hard-coded adversary). The game had six possible actions. Interpreted the Neural Networks' final output layer as a probability that this action is the best possible action given the state of the game (represented as the image input). The Network had to model the probability(action / Game_State) for every action to achieve the task correctly.

Backpropagation is very good at training neural networks because it finds the weights that make the activation functions fit the distributions best.

I will try to demonstrate this here:

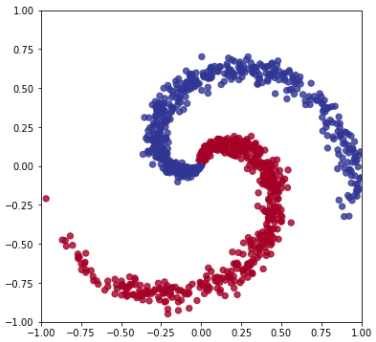


Figure 4: A randomly generated 2 class spiral

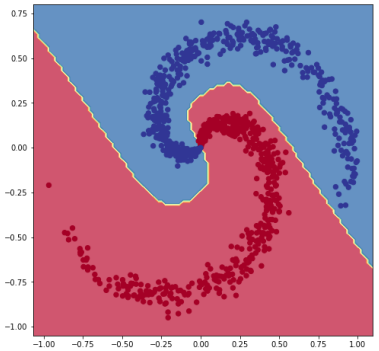


Figure 5: The learned Distribution from the ANN

This understanding of the powerful feature in function approximation was later used in the Deep-Q-Learning Model I implemented.

2.2 Spiking Neural Networks (SNNs)

This section is meant to demystify the SNNs that are used extensively in the following experiments.

Spiking neural networks (SNNs) try to mimic natural neural networks more closely. Neurons in an SNN do not take inputs that are real numbers like traditional ANNs; rather, they can only take in binary spikes that affect the membrane voltage and that neurons spike only if the voltage passes the threshold. To enable spiking to represent a useful set of values, SNNs incorporate the concept of time into their operating model. As a result, SNNs do not transmit information at each propagation cycle but rather transmit information only when a membrane potential – an intrinsic quality of the neuron related to its membrane electrical charge – reaches a specific value, called the threshold.

The most prominent spiking neuron model and the one used in this project is the leaky integrate-and-fire model. In the integrate-and-fire model, the momentary activation level (modelled as a differential equation) is normally considered the neuron's state, with incoming spikes pushing this value higher or lower until the state eventually either decays or - if the

firing threshold is reached - the neuron fires. After firing, the state variable is reset to a lower value.

On the surface, this may seem like an unnecessary complication. As we have moved from continuous outputs to binary, these spike trains are even harder to interpret than the outputs/inputs of traditional ANNs. However, spike trains offer us an improvement in processing Spatio-temporal data. The spatial aspect arises from local learning, where neurons only interact with those in their local neighbourhood, so SNNs inherently process chunks of the input separately (Meaning that the noise tolerance of SNNs is usually higher). The temporal aspect refers to spike trains happening over time, so what is lost in binary encoding is gained in the temporal information of the spikes. This allows us to process temporal data naturally without the extra complexity that RNNs add. It has been proven that spiking neurons are fundamentally more powerful computational units than traditional artificial neurons [5].

Given these modifications, SNNs produce much sparser networks that need less activity as neurons are either activated or not. While this is still inefficient to simulate von Neumann architecture. Neuromorphic hardware tries to solve this using a different architecture that uses neurons being idle to save power.

SNNs use a neuron model that efficiently models the behaviour of the neuron while multiple models exist; the one used in this project is defined in the next section.

2.2.1 Leaky-integrate and Fire Neurons

In the leaky integrate-and-fire model. The membrane voltage V is described by:

$$\frac{dV}{dt} = fn(Vr - V + IR)$$

Where V_r is the rest voltage, V is membrane voltage, R is membrane resistance, and I is the input current.

When the neuron's membrane potential V crosses its membrane threshold V_{thres}, the neuron spikes and the membrane potential is back to V_r. after the reset, the neuron is in a refractory period (used to limit the rate of firing) and cannot spike.

2.2.2 Challenges SNNs introduce

1) **Encoding and decoding data into/from spike trains**; data needs to be encoded into a binary spike train to be inputted into the Network. The spikes must be meaningful, as simply using the binary representation of continuous values will not make sense. I have used two methods in my project for encoding.

a) Pulse encoding presynaptic neurons pulse/spike at different times to represent different values, usually in the form of the higher the value, the earlier the spike is in a specific n ms interval. Pulse encoding is more powerful than rate encoding [5] in terms of the wide range of information encoded by the same number of neurons and its ability to be more power-efficient on neuromorphic hardware as neurons are idle most of the time in the interval and only spike once.

b) Rate encoding, the rate at which the input neurons fire, represents the value of the input. The higher the rate, the higher the value [5]. Rate encoding is very useful in cases as it Is more resistant to noise, usually leading to higher accuracy and enabling transferring weights from ANNs to SNNs as the rates better represent values[29] .

For decoding, Only used population decoding as the output is broken into populations of equal size, and the number of spikes in each population is passed through a sigmoid layer to produce probabilities.

2) **Effectively training the neural Network**; Since spike trains are not differentiable, gradient descent is unusable on SNNs. This means we cannot use regular error backpropagation. As of now, there is a lack of a training algorithm (that is widely adopted) that can achieve good results, especially on the deepest networks. The most popular algorithm (That is used extensively in my project) is Spike-Timing-Dependent-Plasticity (STDP). STDP is an unsupervised algorithm. The lack of an efficient Supervised learning Algorithm is, in my opinion, what is between SNNs and more mainstream adoption.

3) **Lack of standard SNN datasets and evaluation Metrics**; in most papers that I have referenced, datasets ported directly from traditional ANN research and the high dependence on accuracy as the sole evaluation metric (while it is already known that ANNs can exceed human performance in some cases) makes it very hard to choose which Architecture/Training algorithm is better to use from the papers.

2.2.3 Spike-Timing-Dependent-Plasticity (STDP)

STDP [31] is an unsupervised learning algorithm that follows Hebbian learning rules summarised by the mnemonic: those who fire together, wire together; and those who fire out of sync, lose their link. This local-learning [31] algorithm is only concerned with the presynaptic and postsynaptic neurons around a synapse. If the postsynaptic neuron fires before the presynaptic neuron, then the connection between them is weakened. How much it weakened depends on Δt (the difference in time between t_{pre} (time at which presynaptic neuron spikes) and t_{post} (time at which postsynaptic neuron spikes) . In the opposite case,

where the presynaptic neuron spikes before the postsynaptic neuron, the synapse weight increases using Δt again.

This should make features of the input images look similar to have a similar spike pattern in the SNN, similar to what a clustering algorithm would do. For example, this learning is hard to control and figure out which neurons refer to a specific class in a classification task. The locality of the learning also introduces issues where it is not clear if the Network is converging and whether global optimums can be reached.

STDP is biologically plausible as it has been proved to happen in real biological neurons [24]. This may mean that networks trained using STDP can provide insight into biological neurons.

2.2.4 Dopamine-Modulated STDP (MSTDP)

In most reinforcement learning problems, rewards usually come after the actions that produce them. For example, in the case of an ATARI game like breakout, a reward from hitting the brick wall comes after deciding to move the bat towards the ball earlier. It is of utmost importance to the task that only the paths that lead to the reward are the ones that are reinforced. Following on through Izhikevich's EM [30] paper on MSTDP, where he discusses the use of MSTDP to solve the problems of patterns no longer being there when the reward arrives and that all neurons and synapses are active during the waiting period for the reward; how this is achieved is explained below.

Each synapse gets an eligibility trace C [30] . This trace is modified based on subsequent spikes using the normal STDP rule. Until any dopaminergic spikes arrive or the eligibility trace C decays, then the concentration of dopamine and the eligibility trace are used to calculate the change in the weights of the synapses.

This setup is why MSTDP is also called three-factor STDP, as it adds Dopamine concentration (D) to the t_{post} and the t_{pre} parameters. The eligibility trace blocks activity from affecting the calculation and allows neurons to be remembered until a reward is achieved. MSTDP elegantly captures the reinforcement learning problem.

2.2.5 Supervised Learning on SNNs

In one of the experiments it was needed to perform supervised learning on SNNs, I approached it using an ANN as the last layer. The ANN is meant to learn what the populations correspond to in the classes using backpropagation. The SNN works more as a feature extractor than an end-to-end solution in this case. SNNs trained using STDP are good feature extractors as local learning means different image parts get different weights like CNN masks.

2.2.6 Backprop on SNNs

The last experiment performed uses a library called spikingjelly [25]. The spiking jelly library performs backpropagation on SNNs. They do it by setting the firing threshold of the neurons to infinity. This results in no neurons firing and allows the membrane voltages to be used as outputs from the final layer. The voltages are differentiable; this means that backpropagation and SGD can be used.

By doing this, we lose some of the useful features of SNNs. Biological plausibility is lost as time has to be stopped in order to send a backward pass and perform backpropagation, In addition to power efficiency as we end up doing matrix multiplications again.

2.3 Neuromorphic computing

SNNs deliver their power efficiency promises when run on neuromorphic hardware.

It is the use of hardware that mimics the action of biological neurons. This is quite different from the traditional von Neuman architecture, as neuromorphic hardware needs to be massively parallel and event-driven. Many different architectures can be used to achieve this, from spintronics [27] and other hardware-based implementations to software-based implementations.

2.3.1 SpiNNaker

Based at the University of Manchester, SpiNNaker simulates biological neurons using about 518,400 chips, each with 18 low-power ARM cores (Power efficiency chosen to offset increased demand from implementing the neurons using software [26]), local memory, and the SpiNNaker router that handles the communication with other chips.

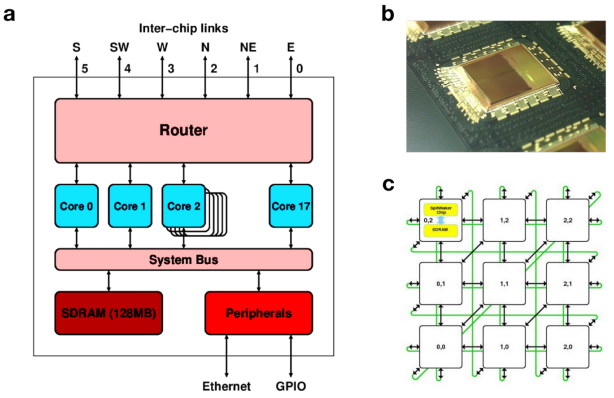


Figure 6: The architecture of a SpiNNaker chip (a), a single chip on a 48 node board (b), and a scheme of SpiNNaker's connectivity (c).

This communication happens asynchronously (a chip sends a signal to another chip but does not require an answer) and in parallel (all chips communicate simultaneously). This gives

SpiNNaker a great degree of parallelism that cannot be achieved by conventional hardware architectures and allows it to simulate the functions of neurons.

2.5 Open AI gym

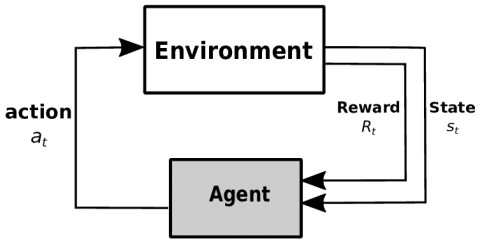
Gym [26] is a toolkit for developing and comparing reinforcement learning algorithms. It provides access to games that can be interacted with through a step function that takes action from the possible action space of each game. It also provides image output in the form of frames where the frames show the consequences of an action.

2.6 Reinforcement Learning

After understanding SpiNNaker and Spiking Neural Networks, Reinforcement Learning is the second biggest part of the project. As explained before, reinforcement learning was the chosen learning method for this project because it fits the use case and the absence of a properly evaluated supervised training algorithm on SNNs.

Reinforcement Learning and games have had quite a history together, from Samuel's checkers player [24] as one of the first learning programs to TD-Gammon, where reinforcement learning's first big success was reached. With the recent development of DeepMinds AlphaZero, Mastering different games like Chess and Go [32].

2.6.1 General RL



The general idea implemented in the first few experiments performed was using the SNN (the agent) to find the probability of an action given the state of the current game (represented as the frame saved). The reward was then used to reinforce the correct/incorrect decisions using different ways to do this depending on whether STDP/MSTDP was being used.

2.6.2 Deep-Q-Learning

2.6.2.1 Q-Learning

A value-based reinforcement learning method depends on assigning each possible action a Q-value (a measure of the action's quality) given the state. The classical way involves a lookup table, and the training process is as follows. Initialising the Q-table with the Q values.

Choose the action with the highest Q value, perform the action and measure the reward that results from the environment. The Q value is updated based on the reward using the Bellman equation.

$$Q_*(s,a) = \sum_{s'} P_{ss'}^{a}(r(s,a) + \gamma \max_{a'} Q_*(s',a'))$$

Bellman equation

2.6.2.2 Adding the Deep aspect

In Deep-Q-learning, instead of a lookup table, a neural network is used that is supposed to estimate the Q-value of each action given the state. This setup makes much more sense when using images as states, as neural networks are known to learn image representations well.

So the process is slightly modified but still follows the same ideas as those in traditional Q-learning. An exploration stage is needed first. The agent randomly chooses between selecting a random action and using the initialised neural Network to find the action with the highest Q-value. This data is stored in a cache with the state (image), action chosen, and the reward produced. Next, an experience replay is set up where random samples are extracted from the experience cache, and the updated Q-values are calculated using the Bellman equations. The loss between the updated Q-value and the one predicted from the Network is used to update it.

The process is repeated, and as the Network gets better, it makes it less likely to make a random action.

2.7 PyNN

A Python package for simulator-independent specification of neuronal network models [27] . PyNN is interpreted as a backend that can run on SpiNNaker. PyNN allows defining the neurons, their types, and their connections.

3 - Design and Implementation

Approaching the problem from different angles gives me a better understanding of SNNs and Reinforcement learning using them. Each Approach was made as an experiment. Each experiment has its agent design and lessons learned from previous experiments to make the Model a better model.

3.1 Experiment 1: STDP is not the deal

3.1.1 Network Design:

This was my first SNN design ever, so It was designed to look like regular feedforward neural networks with an input layer, two hidden layers, and an output layer with ideas from the work of [43]. The output layer was designed such that every 100 neurons corresponds to an action. As this is a reinforcement learning problem, adding the reward as an extra input to the network (with catastrophic consequences later on). Only the action that leads to the reward is reinforced by the input, as there is a single reward connection for each 100 output neurons representing an action.

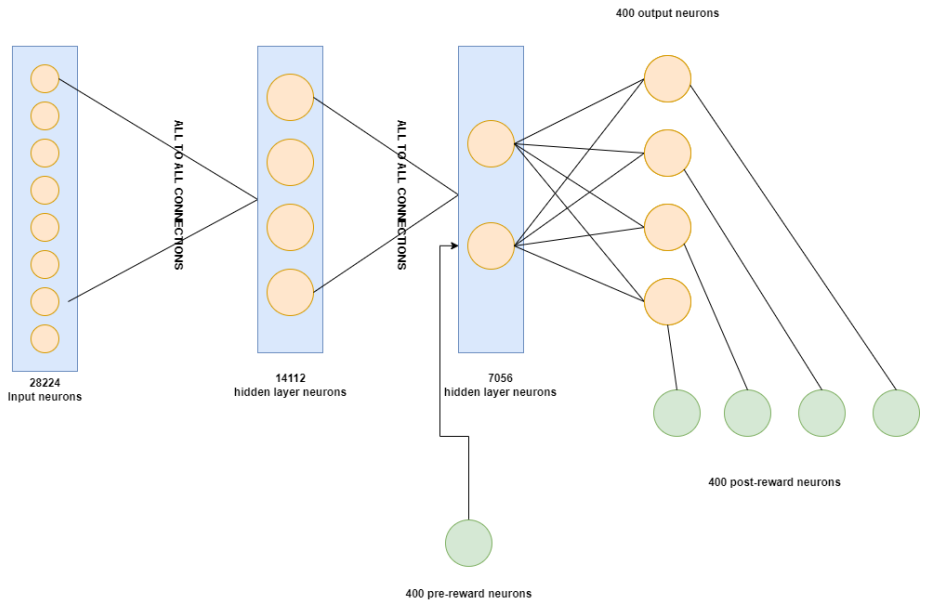


Figure 7: Description of the SNN design

Each reward neuron is connected to the output neurons of its corresponding action, and a pre-reward neuron is connected to the ultimate hidden layer. Then the laws of STDP can be exploited to reinforce measures that produce rewards. After every action, the gym environment returns a reward value. If the reward value is zero (for example, in Breakout, when no bricks are hit, we do nothing), if the reward is >0 (as in we hit a brick), then the pre-reward neurons that represent only the last action send spikes to the ultimate hidden layer then after a delay, the post-reward neurons are meant to send a spike this promotes STDP to

strengthen the synapse between the last hidden layer and the output neurons belonging to the action. If the reward < 0, then the post-reward neurons related to the action send spikes before the pre-reward ones do that, causing STDP to weaken the connection, so this action is likely to be taken.

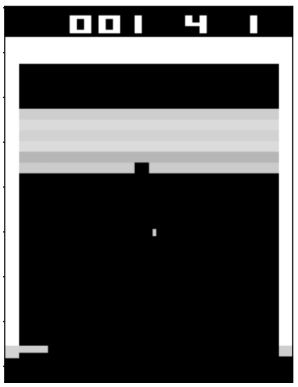


Figure 8: grayscale breakout frame

The inputs are frames modified to become grayscale and resized to 84x84 dimensions. This was meant to reduce the number of neurons used to represent the images to make the learning process easier. The pixel values are used as the rate in a Poisson rate encoded spike train (adds a random distribution to the values).

For outputs, a voting system is used as the network is run for 40ms (with a time step of 1ms), and the action whose neuron population produces the most spikes is the one chosen to be the top action. Increasing the running time increases the spikes obtained in the output population (which probably makes the network more noise-tolerant). However, it doesn't significantly change the chosen values in the end.

To prevent the same games being played over and over again such that the neural network only learns a specific sequence of actions that do well every game. The first 100 frames are given random actions so that the starting point is different every game.

3.1.2 Results:

Scores are calculated using the accumulated reward through the game. The gym environment doesn’t return negative rewards when lives are lost, so a negative change in the agent's value is used as a negative reward when training the network; however, only the positive rewards are added up . The breakout game has different scores for the different bricks, so partial rewards are possible.

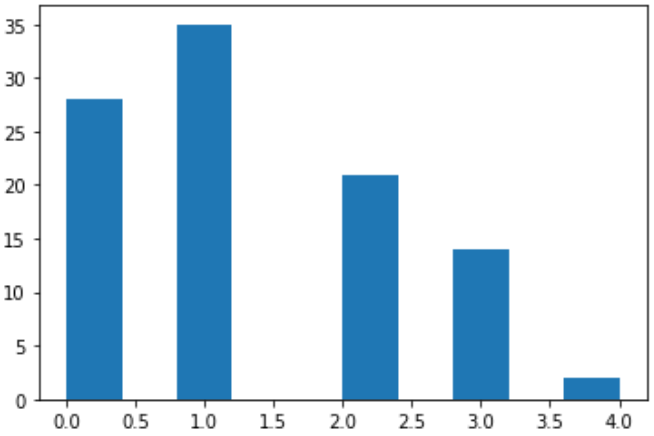


Figure 9: random agent mean 1.21, std 1.1

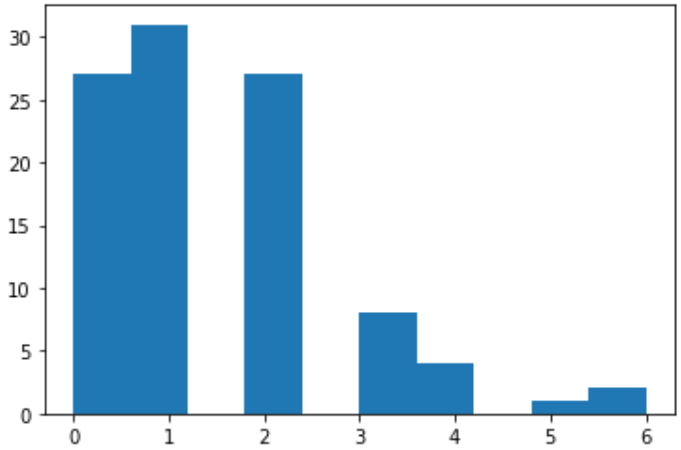


Figure 10: STDP agent mean 1.42 std 1.31

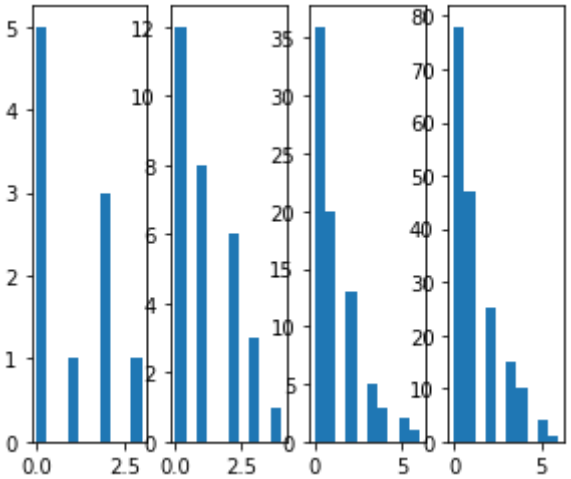


Figure 11: Training STDP agent over 10,20,50,100 games

The results suggest that little to no learning is happening in this network; the distribution of scores is similar to that of a random number generator, with most of the games of the hundred played ending scoreless. The mean and variance of the score are also consistent with the

random agent. For additional proof that no learning is happening, the mean score doesn't change when trained for 10,20,50 and 100 games.

3.1.3 Possible Explanation:

This lack of learning is probably because of the reinforcement learning, the fact that rewards are the result of actions made way ahead of the reward, for example, if the agent decides to move right so that the paddle hits the ball and the ball bounces upwards if the agent moves, while the ball is travelling those unrelated actions (or the action of staying still), will be reinforced while the action that caused the reward is ignored. This network/STDP lacks the short-term memory that is needed to eliminate bias in reward assignment to actions as mentioned in [33]. Moreover, the fact that a combination of actions is usually the reason for the reward and that each action must be reinforced proportionally to how much it contributes to the reward, which this setup doesn't do and fully rewards the last action.
Also, the grayscale snapshot is not a good representation of the current state of the game, as the direction of the moving ball and its speed cannot be guessed from it. Nevertheless, there may not be enough layers for adequate feature extraction, but the number of neurons is large enough already. Lastly, There may be an issue with the way outputs are select outputs and a function that uses a threshold, ratios, or some weightings to the values of the spikes at each action (as some actions are more likely than others).

3.1.4 Challenges Faced:

Each game is about 1000 frames and takes about 500 seconds to run and return results for one game. So playing more than 100 games was a difficult thing to do as the Jupyter terminal seems to freeze if kept running for too long. Challenges were faced in saving the PyNN model weights in order to avoid retraining every time, but due to the sheer number of synapses, the file produced is too big that it crashes the interface, and a smaller file produced was not being read properly.

3.2 Experiment 2: A slight change

3.2.1 Network Design:

This time two changes were introduced to experiment with One's design. First, lateral inhibition (inspired by the Dihel and Cook paper [36]) is used to try to improve the performance of STDP; this requires the introduction of inhibitory and excitatory synapses.

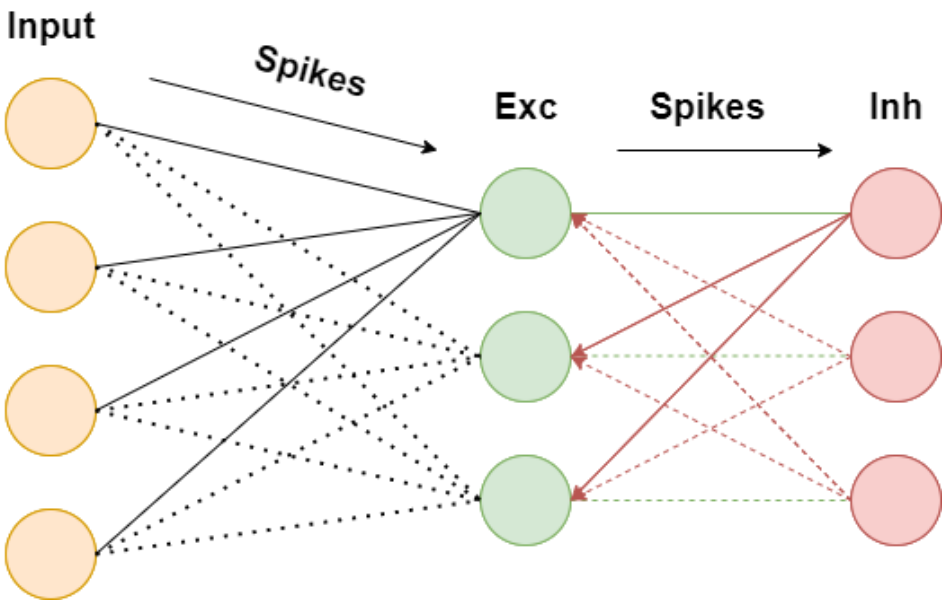


Figure 12: Network design using lateral inhibition

When an inhibitory neuron is excited, it is meant to inhibit all the other Excitatory Neurons so that we get clearer winners.

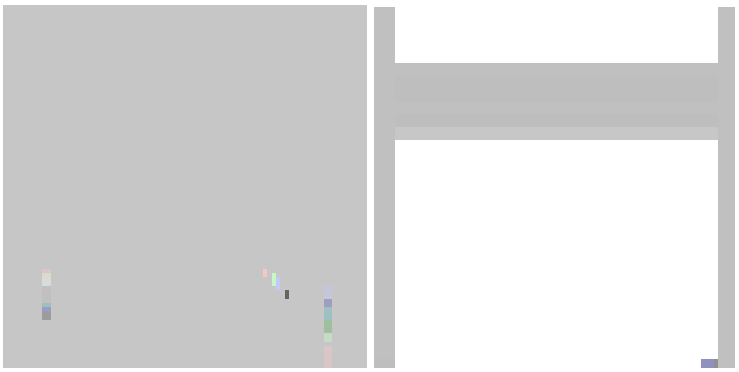
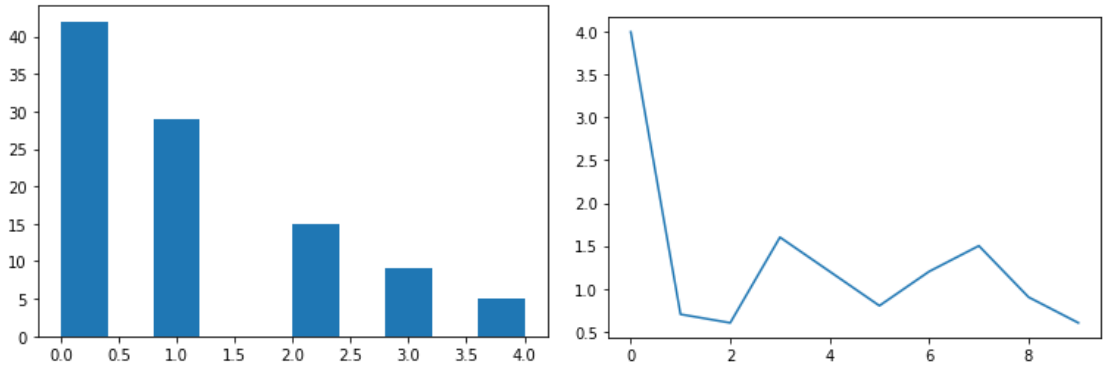


Figure 13: The processed images for pong and breakout

Also, I have improved preprocessing of the frames by using the ϕ function from Minh et al. [9]. This function saves the last four frames of the image and stacks them on top of each other, passing each frame (greyscale images) as a channel(Red, Green, Blue, Alpha) to form an image that carries more information about the state enabling the direction of movement and speed of the ball to be found. This increases the size of input by x4, but I believe the improvements to the state are worth it.

3.2.2 Results:



(a) (b)

Figure 14: (a) The distribution of scores over a 100 test games, (b) The average score in the last ten games during training

Very little to no learning was achieved again (when compared with experiment 1 and a random baseline). The problems that relate to reinforcement are probably what's holding back the performance. This probably needs a solution that goes beyond STDP.

3.2.3 Explanation:

The problems mentioned in experiment One are still holding back the performance, and the new design doesn't do well with the problems due to forgetting which actions contributed to the reward.

3.2.4 Challenges created:

Same problems as experiment One with Training time limiting the possible number of samples that can be run.

3.3 Experiment 3: MSTDP a little better

3.3.1 Network Design:

This design is inspired by the works of Evans [51] and Hazan [52]. All the synapses are MSTDP synapses, and they are connected to the dopaminergic neurons in an all-to-all connection. The dopaminergic neurons are connected to the environment where they spike if a reward is greater than zero. The lateral inhibition idea is kept as it was documented to improve STDP in the work of Dhiel and Cook [50].

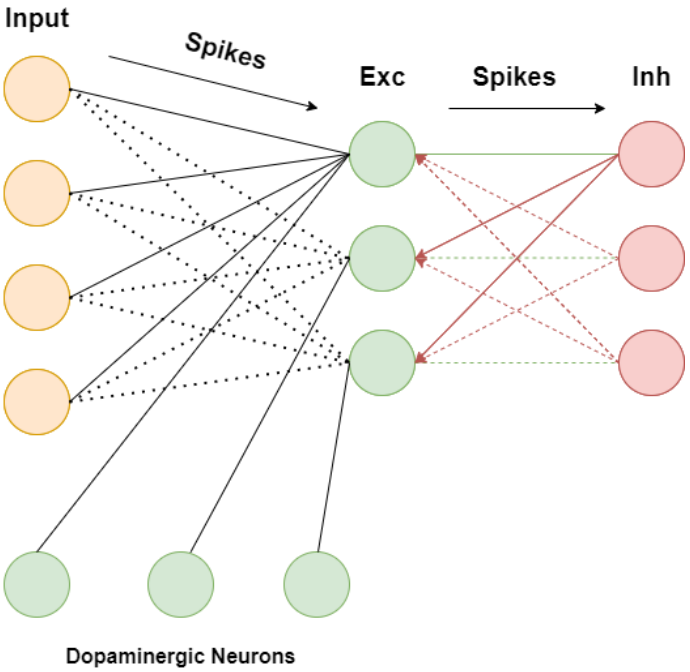
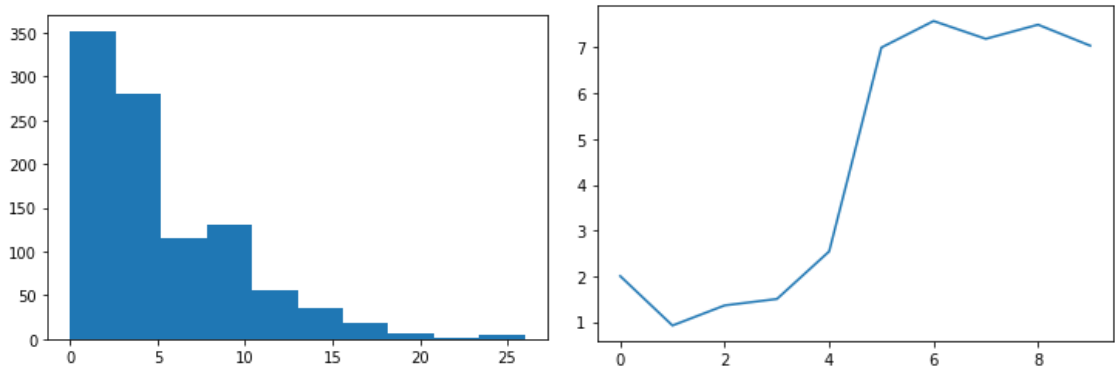


Figure 15: Neural Network uses MSTDP

The inhibitory neurons are used as output neurons, and the spikes are counted to choose the action (with the hope that lateral inhibition makes winners clearer).

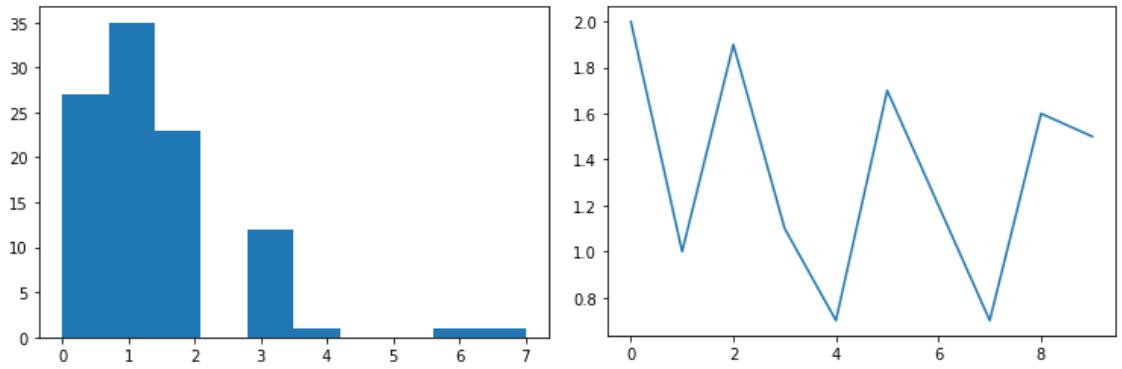
Inputs use the stacked format from Experiment 2 and each pixel is used as the rate value in a Poisson encoded spike train. For the running Time, MSTDP looks like it requires a longer running time probably due to the added complexity of eligibility traces and the waiting time each synapse takes before updating the synapses so 60ms was used as it was seen that a clear winner could be identified most of the time at this running time.

3.3.2 Results:



(a) (b)

Figure 16: (a) Score distribution using Bindsnet on GPU (b) Average score over last 100 games during training



(a) (b)

Figure 17: (a) Score distribution using SpiNNaker (b) Average score over last 20 games during training

From the results, we can see that MSTDP is indeed learning something related to the game as the distribution of scores is different from the random agent, and the agent is learning as the game progresses though it seems to plateau very early.

This improved learning means that the agent has used the rewards to improve its decision making the improvement may look minor and insignificant it is still a nudging towards the right direction.

3.3.3 Explanation:

MSTDP's three-factor STDP tries to address the problem discussed in Experiment 1, As in if a reward is achieved after the action is taken for a while; The eligibility trace C keeps the path to the action ready to become reinforced (until it completely decays) when the reward comes. This makes the learning of better quality than that achieved only by the use of STDP. However, the quick plateau required further investigation where there is quite a number of possible reasons why that is so:

1. **Catastrophic forgetting [34] ,**(documented to happen frequently in problems of this scope)**,** unlike humans, neural networks learn things sequentially. This means that learning a new thing such as chasing the ball by moving right before it falls outside the frame, can affect the weights that control the task of moving in the opposite direction. This problem may have also been amplified by lateral inhibition, where decisions that have little use such as the actions taken while the ball is traveling to the brick-wall lead to inhibition of other decisions that are of use (Need to look more into this effect may also be limited by MSTDP's).

2. **Poor feature extractions,** when processing images normally, CNNs are used. CNN's convolutional filters used by CNN's use multiple filtering layers with different filters to perform the extraction. This is not available here as there is only one layer between the input layer and the excitation layer. The processing of images is very important in state identification which is vital for choosing the action.

3. **No way to account for negative rewards,** This network doesn't include a way that dopamine concentration can be reduced if the agent loses a life. This means that missing the ball is not punished while only hitting the bricks is rewarded this may mean that the agent is plateauing because it is losing all lives very quickly. While it is essentially the same task, as keeping the ball in play involves not losing lives, that's why this suggestion is given less weight.

4. **The ratio of inhibitory to excitatory neurons,** Dhiel and Cook [36] uses a 4:1 excitatory to inhibitory neuron ratio (copying observations from human biology). A one to one ratio was used in this experiment and different ratios may have helped.

3.3.4 Challenges created:

The training time problem has been addressed this time as the network was showing learning promise, discovering new ways to run the networks that make the inference time faster.

- **Running on spinnaker,** using the guidance provided by the SpiNNaker software support team. The way the network runs is changed to improve the speed; Previously, the network used to run every frame, selecting the chosen action. Sending the action to the environment, collecting reward, then running the network again on the new frame. This way is inspired by the way deterministic ANNs are run on similar tasks that were met during my degree. This includes so much time wasted as the SpiNNaker simulator re-initializes the network for every run, wasting time in doing so. This coupled with the fact that each game is one average of a thousand frames made it very hard to run multiple games.

  The new method involves running the network forever and then sending spikes to it using features provided by PyNN to interact with a running SNN. This exploitation of the temporal aspect of the SNNs with a problem involves going through frames with time and the reduction of time wasted reinitializing new networks. This allowed me to run twice as many games in the training phase than in the experiment one.

- **Running on the GPU,** using the library Bindsnet [38] (that was edited so that it allows the pipeline to work with my frames). This library simulates MSTDP using Pytorch [42] and allows the use of CUDA to speed up the process to make the game run faster than the real-time game. This allowed me to run 1000 training games that weren't that useful as the plateau was actually real.

3.4 Experiment 4: ANN2SNN

3.4.1 Network Design:

In an attempt to compare and contrast different ways to implement/train SNNs. After reading the work of Bodo Rueckauer, and Shih-Chii Liu [29] on their tool SNN-toolbox [46] it made so much sense to try using it. As the advancements already made in Training ANNs may prove beneficial when the conversion happens and lead to better performance. The SNN-toolbox repository provides an example [45] where a CNN designed to recognize MNIST is then converted to an SNN that can run on SpiNNaker and maintain high performance. So It was planned to design a CNN that tries to play the games pong/breakout.

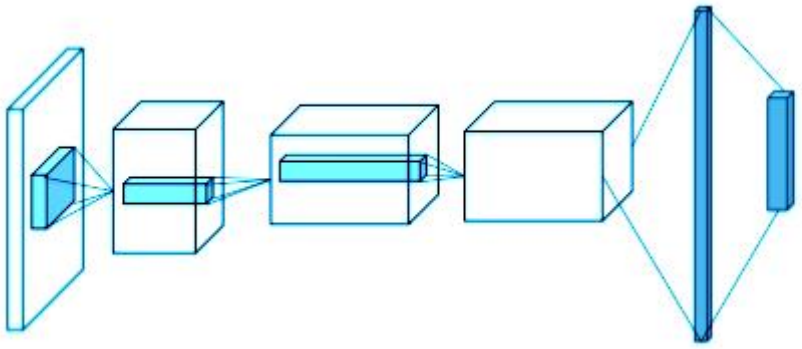


Figure 18: CNN design

The CNN structure was the same one used in Minh et al [9].The CNN was trained using a labelled data set that was created using a technique that uses a mode in the gym's pong function that returns x,y-coordinates of the ball and the bat the action that moves the bat towards the ball was the one always picked. In the pong game, there are two types of up and down movements the fast one and the normal speed ones, this required setting an arbitrary threshold distance between x_{bat} and x_{ball} for the fast actions to be chosen.

The training data consisted of 100,000 labelled frames that were sampled randomly and used as training data.

3.4.2 Result:

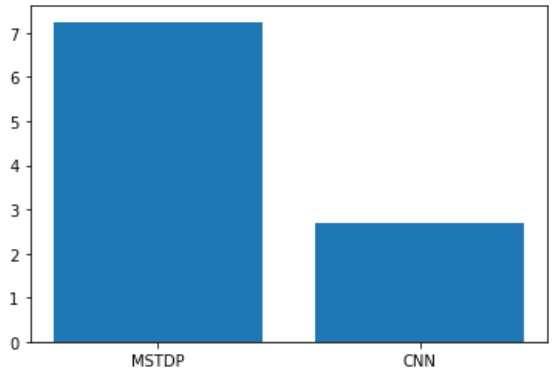


Figure 19: CNN average score over 1000 games vs the MSTDP one

The results here were underwhelming. When tested, an accuracy of 40% was obtained, which means that it knows the correct direction to go only 40% percent of the time (showing either how hard the state identification task is or that my 100,000 frames of training data are not enough). This is better than random but doesn't look like a good performance. When CNN is running the game. The performance is less than that of MSTDP, as shown below:

3.4.3 Explanation:

The poor performance is probably the result of the inadequacy of the moving towards the ball strategy as following the ball while it is going up leads to going too many side-ways and when the ball is deflected to move to the other side the paddle probably can't catch it. In addition, the absence of rewards as an input may mean that the strategy cannot be optimized to the choices made in-game.

The problem with the low accuracy may be due to not having enough training data.

3.4.4 Challenges created:

Due to the poor game performance and the movement away from reinforcement learning (which is believed to be the best way to approach this problem), it was decided that not using SNN-toolbox to convert it to an SNN instead it was decided to use the time to implement the next two experiments.

3.5 Experiment 5: Q-learning

3.5.1 Network Design:

This was inspired by the works of Chen et al. (2022) [35] where a similar idea was used with the exception of not using backpropagation and the original DQN paper by Mnih et al. [9]. In addition to the idea presented by Dmitry Krotov in his lecture on Biologically Inspired Neural Networks [39], the network was conceptualised.

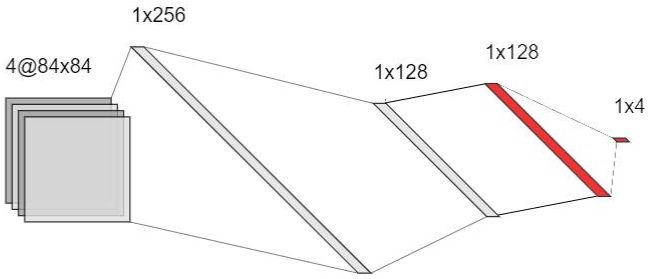


Figure 20: The Spiking DQN structure

The DQN training process explained in section 2, involves two stages the first stage where the network makes decisions and explores the solution/state space and stores the findings in a cache, and in the next step called experience replay a random sample of the experience is chosen and the difference between the predicted Q value and the updated Q-value calculated using the Bellman equation is used to train the network (a supervised learning task), but instead of using an ANN and then converting it to an SNN like what was done in Chen et al. (2022) [35]. The idea used is provided by Dmitry Krotov [39]. Where STDP is used in all the layers except for the last one. The training process goes as follows:

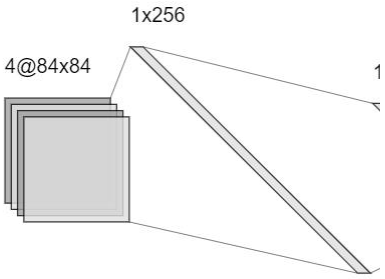


Figure 21: The one-layer network

1. Frames (from the experience replay) are randomly sampled into a one-layer network using pixel values as rates in a Poisson-encoded spike train. STDP performs its job modifying the initialised weights, weights are frozen as static synapses after all the images are passed to the network

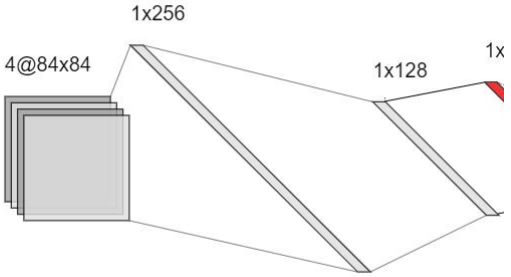


Figure 22: The Second layer is added

2. The second layer is added to the frozen first layer and a random sample of the frames is inputted again, allowing STDP to do its job again. Then freeze the weights by creating a new network with static synapses.

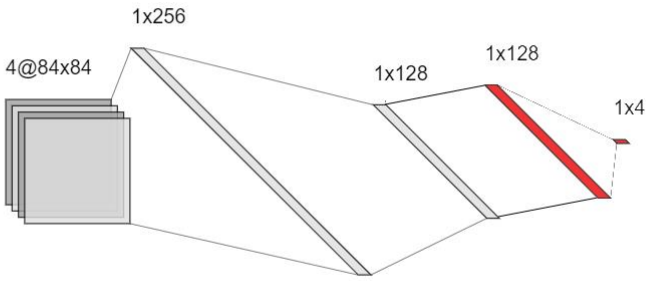


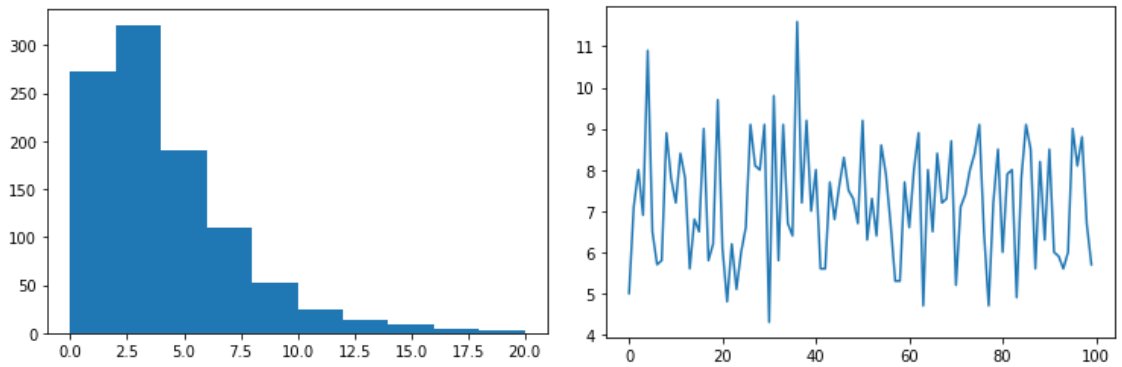
Figure 23: The final layer

3. The final layer is a PyTorch Feedforward layer; this is the layer where the supervised learning is supposed to happen; the updated Q-values calculated are used to perform backpropagation on this layer.
4. When the exploration-replay cycle is repeated the STDP networks no longer need training and are kept as static as they are and only the final layer is optimised.

In this setup, the SNN part acts more like a feature extractor where the local learning features of STDP are exploited to perform feature extraction akin to that of a CNN.

The outputs are picked using the one with the highest Q-value as outputted by the last layer.

3.5.2 Results:



(a) (b)

Figure 24: (a) Score distribution over 1000 test games (b) Average over last 100 test games

Again the results are underwhelming as the network has been trained to 100,000 frames from approx. thousand games during exploration. The results seem to be random as there is no increase followed by a plateau like that of MSTDP but more like non-/very-slowly converging learning going on. The average score over a 1000 test games is second best to that of MSTDP.

3.5.3 Explanation:

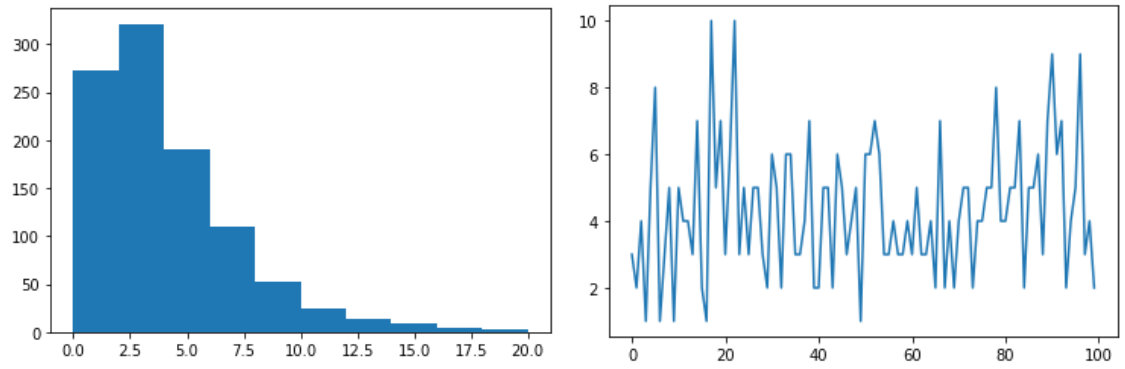
There can be a multitude of reasons why this setup doesn't work properly, First the hindering back by the freezing process as the improvement in feature detection cannot happen by weight adjustment. This may be the reason that the updated Q-values aren't being properly learned. Also, Q-learning is known for its need for a lot of training samples, especially in games where losses/rewards happen over multiple frames like Breakout. Finally, the network may not be deep enough to properly model the probability distribution of the Q-values given the extracted features from the frame in a reasonable way. The Minh et al. [9] network uses 4 convolutional layers, and 2 feed-forward layers.

3.6 Experiment 6: Backpropagation Through Membrane Potential

3.6.1 Network Design:

Using the same network design as Experiment 5, and the library spikingjelly [14] allows me to perform backpropagation (by setting the V_{thresh} to infinite and performing SGD Stochastic Gradient Descent on membrane voltages during backpropagation).

3.6.2 Result:



(a) (b)

Figure 25: (a) Score distribution over 1000 test games (b) Average over last 100 test games

The results are the best achieved. The non-convergence can probably be seen as a Q-learning problem as the scores are also fluctuating.

3.6.3 Explanation:

The Fluctuation problem is the result of the use of the same network to estimate the updated Q-value and the Q value. As a consequence, there is a big correlation between the updated Q and the weights we are changing. Therefore, it means that at every step of training, our Q values shift but also the target Q-value shifts. So, as we get closer to the target-Q-value the changing parameters push it further and this leads to a huge fluctuation in scores.

The lack of convolutional layers and the not deep enough arguments from the previous experiments are probably the reasons here. Also, Duelling DQNs were used in Minh et al. [9] to separate the estimation of the value of the state and the reward values expected from taking each action knowing that the Q-value is the sum of both. It is said to increase stability and speed up learning in the paper cited.

4 - Evaluation , Conclusion , and Summary

This section will include a summary of the experiments and the contributions achieved, followed by a quick comparison of the results to the best performing models proposed in the literature. Finally conclusions and future work ideas are introduced to the reader.

4.1 Summary:

The report covers the experiments that tried to address the task of training a Spiking Neural Network in order to achieve human level performance on atari games that require complex control. The networks get the inputs as images containing frames from the game. And should output the suggested actions to be taken.

The best performing model was the one that used Backpropagation on a DQSNN (using the membrane potentials and Q-learning) [14]. Followed by the model that uses MSTDP as a close second (which also benefits from less fluctuations than the Q-learner). Those results produced in this report are below human averages and performances of linear learners.

STDP has not been adequate enough to address the task of reinforcement learning effectively. While It achieved some learning when used with a Single Layer Perceptron (SLP) in experiment 5, the results were underwhelming.

The contributions of this project are summarised below:

- Proposing models that can be improved upon to produce better results
- Experimentation with different training algorithms and reinforcement learning methods and identifying the most promising ones as (MSTDP, and Backpropagation trained Deep-Q-SNN)

4.2 Comparing Results to State of the Art and other relevant work:

4.2.1 ANN-based relevant example:

The best performing ANNs are a lot, With the best being the ones that use Asynchronous Advantage Actor Critic actor critic models (A3C) (which learns by performing policy modification based on the rewards) such as MuZero [54] and AlphaZero [32]. These networks require huge amounts of power to train and consume 200 Million frames in training data, showing that the direction ANNs are moving towards may become unsustainable in the future.

4.2.2 SNN-based relevant example:

The best performing SNNs underperform the best ANNs, but still outperform my results. The best performing SNN [35]; Uses a method similar to that of experiment 5, but uses a different encoding scheme and improvements on the Backpropagation on Non-spiking neurons. However, according to [35] SNNs are better at resisting Adversarial attack.

4.3 Conclusions and Future Work:

This section presents conclusions and gives ideas for future work.

4.3.1 Conclusion:

Reinforcement learning is a difficult task with a lot of problems native to it, and SNNs also introduced their own problems making the intersection of the tasks a lot harder. In this report different approaches to the problem were tried and addressed their issues. There is no evidence that along with improvements and Future work, SNNs can easily replicate performance produced by top ANNs. This means that if added to edge and low-power devices a RL SNN will learn complicated control tasks. This can be applied to manufacturing , autonomous robots , and many other robotic applications.

4.3.2 Future Work:

1. **Minibatch Processing,** A major hurdle in this project was the slow training time of the SNN, to improve this we can borrow from techniques used in modern ANNs, one of those is batch processing where a batch of the images are accepted as an input to the network and are all processed at once. In their paper Saunders, D. et al. [55] proposed a method where the network is replicated by the batch size B , then the change in weights that results in each batch is averaged and the average value is the one used to update the original network. This setup makes more sense on GPU based implementations of SNNs as the parallel processing power of GPUs can be utilised but on edge devices it would be unreasonable to expect them to be able to do batch processing.

2. **Improve input encoding quality,** poisson encoded pixel values do not seem as an ideal input to an SNN because of how arbitrary the decision to represent them this way was. Papers that used sensor data to produce spikes have shown good performance, and maybe changing the input type to sensor data may improve performance. Also stacking images like what was done in experiments 2-4 may not be the ideal method given the temporal qualities of SNNs instead keeping STDP online during running tests will probably give the network a better ability as it may learn internal representations of previous states

3. **Multiplicative-MSTDP,** In their paper Shim et al. [47] used multiplicative-MSTDP and said it leads to solutions of higher quality and higher accuracy. The idea behind multiplicative-MSTDP allows the control of learning rates and allows the availability of a scalar reward rather than the binary reward used in my experiments.

4. **Make SNNs deeper,** Deeper ANNs can learn more approximate functions in higher detail than shallower ones and the trend seems true in SNNs too as the best performing SNN models were significantly deeper than the models I use. This introduces a training problem as training a deeper network is more time consuming and would restrict the benefits of the local learning training algorithms.

5. **Better Reward Functions,** a reward function that gives a better indication of the reward quality will help the network identify better reward generating opportunities quicker, leading to a better performance overall.

Reference list

1. AG Barto, RS Sutton and CW Anderson (n.d.). *OpenAI Gym: the CartPole-v0 environment*. [online] gym.openai.com. Available at: https://gym.openai.com/envs/CartPole-v0/.
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016). *OpenAI Gym*. [online] arXiv.org. Available at: https://arxiv.org/abs/1606.01540 [Accessed 24 Nov. 2019].
3. Cavanaugh, J.T., Guskiewicz, K.M. and Stergiou, N. (2005). A Nonlinear Dynamic Approach for Evaluating Postural Control. *Sports Medicine*, 35(11), pp.935–950.
4. Furber, S. and Bogdan, P. (2020). *SpiNNaker: A Spiking Neural Network Architecture*. [online] now publishers, Inc. Available at: https://www.nowpublishers.com/article/DownloadEBook/9781680836523?format=pdf [Accessed 27 Apr. 2022].
5. GOV.UK. (n.d.). *Increasing access to data held across the economy*. [online] Available at: https://www.gov.uk/government/publications/increasing-access-to-data-held-across-the-economy [Accessed 20 Apr. 2022].
6. Jabr, F. (2012). *Does Thinking Really Hard Burn More Calories?* [online] Scientific American. Available at: https://www.scientificamerican.com/article/thinking-hard-calories/.
7. Liang, S. and Srikant, R. (2017). Why Deep Neural Networks for Function Approximation? *arXiv:1610.04161 [cs]*. [online] Available at: https://arxiv.org/abs/1610.04161 [Accessed 20 Apr. 2022].
8. Microsoft.com. 2022. [online] Available at: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/APSIPA_May2015.pdf> [Accessed 20 April 2022].
9. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1312.5602.
10. Rotman, D. (2020). *We're not prepared for the end of Moore's Law*. [online] MIT Technology Review. Available at:

https://www.technologyreview.com/2020/02/24/905789/were-not-prepared-for-the-end-of-moores-law/.

11. Wikipedia Contributors (2019). *Atari 2600*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Atari_2600.
12. Allred, L.G. and Kelly, G.E. (1990). *Supervised learning techniques for backpropagation networks*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/5726614 [Accessed 22 Apr. 2022].
13. Bi, G. and Poo, M. (1998). Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *The Journal of Neuroscience*, [online] 18(24), pp.10464–10472. Available at: https://www.jneurosci.org/content/jneuro/18/24/10464.full.pdf [Accessed 20 Nov. 2020].
14. fangwei123456 (2022). *SpikingJelly*. [online] GitHub. Available at: https://github.com/fangwei123456/spikingjelly [Accessed 22 Apr. 2022].
15. Littman, M.L. (1994). Markov games as a framework for multi-agent reinforcement learning. [online] ScienceDirect. Available at: https://www.sciencedirect.com/science/article/pii/B9781558603356500271 [Accessed 26 Apr. 2022].
16. Ghosh-Dastidar, S. and Adeli, H. (2009). Third Generation Neural Networks: Spiking Neural Networks. *Advances in Intelligent and Soft Computing*, pp.167–178.
17. Grollier, J., Querlioz, D., Camsari, K.Y., Everschor-Sitte, K., Fukami, S. and Stiles, M.D. (2020). Neuromorphic spintronics. *Nature Electronics*, 3(7), pp.360–370.
18. Hunsberger, E. (2018). Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition. *uwspace.uwaterloo.ca*. [online] Available at: https://uwspace.uwaterloo.ca/handle/10012/12819 [Accessed 22 Apr. 2022].
19. Izhikevich, E.M. (2007). Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex*, [online] 17(10), pp.2443–2452. Available at: https://academic.oup.com/cercor/article/17/10/2443/314939 [Accessed 23 Feb. 2020].
20. Jin, X., Rast, A., Galluppi, F., Davies, S. and Furber, S. (2010). *Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/5596372 [Accessed 22 Apr. 2022].

21. Jordan, M.I. and Mitchell, T.M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), pp.255–260.
22. Lecun, Y., Bengio, Y. and 4g332, R. (n.d.). *Convolutional Networks for Images, Speech, and Time-Series*. [online] Available at: http://www.iro.umontreal.ca/~lisa/pointeurs/handbook-convo.pdf.
23. LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep Learning. *Nature*, 521(7553), pp.436–444.
24. neuralensemble.org. (n.d.). *PyNN - NeuralEnsemble*. [online] Available at: https://neuralensemble.org/PyNN/ [Accessed 22 Apr. 2022].
25. Nowakowski, R.J. (1998). *Games of No Chance*. [online] *Google Books*. Cambridge University Press. Available at: https://books.google.com/books?hl=en&lr=&id=cYB-ra2T8i4C&oi=fnd&pg=PA119&dq=+Samuel%27s+checkers+player&ots=H2qIWFf_nm&sig=PvDaq1pn413ykmIxc9lmhnlzAfU [Accessed 22 Apr. 2022].
26. Oh, K.-S. and Jung, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, 37(6), pp.1311–1314.
27. OpenAI (2019). *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. [online] Openai.com. Available at: https://gym.openai.com/.
28. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), pp.386–408.
29. Rueckauer, B. and Liu, S.-C. (2018). *Conversion of analog to spiking neural networks using sparse temporal coding*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/8351295/ [Accessed 22 Apr. 2022].
30. Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, [online] 323(6088), pp.533–536. Available at: https://www.nature.com/articles/323533a0?error=cookies_not_supported&code=2926f83e-9c3a-46a7-9b28-b3d19d46768a.
31. Shahid, A. and Mushtaq, M. (2020). *A Survey Comparing Specialized Hardware And Evolution In TPUs For Neural Networks*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/9318136 [Accessed 22 Apr. 2022].
32. Zhang, H. and Yu, T. (2020). AlphaZero. *Deep Reinforcement Learning*, pp.391–415.
33. Bogacz, R., McClure, S.M., Li, J., Cohen, J.D. and Montague, P.R. (2007). Short-term memory traces for action bias in human reinforcement learning. *Brain Research*, 1153, pp.111–121.

34. Cahill, A. (2011). *Catastrophic Forgetting in Reinforcement-Learning Environments*. [online] ourarchive.otago.ac.nz. Available at: https://ourarchive.otago.ac.nz/handle/10523/1765.
35. Chen, D., Peng, P., Huang, T. and Tian, Y. (2022). Deep Reinforcement Learning with Spiking Q-learning. *arXiv:2201.09754 [cs]*. [online] Available at: https://arxiv.org/abs/2201.09754 [Accessed 25 Apr. 2022].
36. Diehl, P.U. and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9.
37. Evans, R. (2015). Reinforcement Learning in a Neurally Controlled Robot Using Dopamine Modulated STDP. *ArXiv*. [online] Available at: https://www.semanticscholar.org/paper/Reinforcement-Learning-in-a-Neurally-Controlled-Evans/20695f2897c6f04c870e6466b5d5952013b89eea [Accessed 24 Apr. 2022].
38. Hazan, H., Saunders, D.J., Khan, H., Patel, D., Sanghavi, D.T., Siegelmann, H.T. and Kozma, R. (2018). BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. *Frontiers in Neuroinformatics*, 12.
39. Krotov, D. (2019). *MIT 6.S191 (2019): Biologically Inspired Neural Networks (IBM)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=4lY-oAY0aQU [Accessed 25 Apr. 2022].
40. Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. and Liu, S.-C. (2017). Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. *Frontiers in Neuroscience*, 11.
41. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, [online] 518(7540), pp.529–533. Available at: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf.
42. PyTorch (2019). *PyTorch*. [online] Pytorch.org. Available at: https://pytorch.org/.
43. Roberts, P.D., Santiago, R.A. and Lafferriere, G. (2008). An implementation of reinforcement learning based on spike timing dependent plasticity. *Biological Cybernetics*, 99(6), pp.517–523.

44. Chu, T., Wang, J., Codeca, L. and Li, Z. (2019). Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control. *IEEE Transactions on Intelligent Transportation Systems*, pp.1–10.
45. Rueckauer, B. and Liu, S.-C. (2022). *NeuromorphicProcessorProject/snn_toolbox*. [online] GitHub. Available at: https://github.com/NeuromorphicProcessorProject/snn_toolbox/blob/master/examples/mnist_keras_spiNNaker.py [Accessed 25 Apr. 2022].
46. Rueckauer, B. and Liu, S.-C. (n.d.). *Spiking neural network conversion toolbox — SNN toolbox 0.5.0 documentation*. [online] snntoolbox.readthedocs.io. Available at: https://snntoolbox.readthedocs.io/en/latest/ [Accessed 25 Apr. 2022].
47. Shim, M.S. and Li, P. (2017). *Biologically inspired reinforcement learning for mobile robot collision avoidance*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/7966242 [Accessed 26 Apr. 2022].
48. Patel, D., Hazan, H., Saunders, D.J., Siegelmann, H.T. and Kozma, R. (2019). Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to the Atari Breakout game. *Neural Networks*, 120, pp.108–115.
49. Kim, J., Kwon, D., Woo, S.Y., Kang, W.-M., Lee, S., Oh, S., Kim, C.-H., Bae, J.-H., Park, B.-G. and Lee, J.-H. (2021). On-chip trainable hardware-based deep Q-networks approximating a backpropagation algorithm. *Neural Computing and Applications*, 33(15), pp.9391–9402.
50. Tang, G., Kumar, N. and Michmizos, K.P. (2020). Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware. *arXiv:2003.01157 [cs]*. [online] Available at: https://arxiv.org/abs/2003.01157 [Accessed 26 Apr. 2022].
51. Tang, G., Kumar, N., Yoo, R. and Michmizos, K.P. (2020). Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control. *arXiv:2010.09635 [cs]*. [online] Available at: https://arxiv.org/abs/2010.09635 [Accessed 26 Apr. 2022].
52. Zhang, D., Zhang, T., Jia, S., Cheng, X. and Xu, B. (2021). Population-coding and Dynamic-neurons improved Spiking Actor Network for Reinforcement Learning. *arXiv:2106.07854 [cs]*. [online] Available at: https://arxiv.org/abs/2106.07854 [Accessed 26 Apr. 2022].

53. Guo, W., Fouda, M.E., Eltawil, A.M. and Salama, K.N. (2021). Efficient Training of Spiking Neural Networks with Temporally-Truncated Local Backpropagation through Time. *arXiv:2201.07210 [cs, eess]*. [online] Available at: https://arxiv.org/abs/2201.07210 [Accessed 26 Apr. 2022].
54. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. and Silver, D. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839), pp.604–609.
55. Saunders, D.J., Sigrist, C., Chaney, K., Kozma, R. and Siegelmann, H.T. (2019). Minibatch Processing in Spiking Neural Networks. *arXiv:1909.02549 [cs]*. [online] Available at: https://arxiv.org/abs/1909.02549 [Accessed 27 Apr. 2022].