Autonomous Artificial Intelligence for Robotics Applications

The Limits of Classical Conditioning

Author

Aidan Belton-Schure

51769738

Supervisor

Dr. Andrew Starkey

A dissertation submission in partial fulfilment of the requirements of the award Bachelor of Engineering at the University of Aberdeen

School of Engineering, University of Aberdeen

2021

Abstract

Artificial intelligence (AI) is used in a variety of systems. However, few of them rely solely on AI to perform the entire task. Deep learning, the current state-of-the-art in AI, can perform tasks. The Markov decision process provides a framework for deep learning to learn to perform a task. Classical conditioning is the mechanism animals use to associate stimuli and actions with a goal. Classical conditioning and the Markov decision process have shared mathematical and theoretical relationships. A study is designed that tests deep learning methods on a variety of video games. The results show that deep reinforcement learning successfully learns a variety of tasks with little modification. However, the study has limitations due to computational expense and the amount of data required. The study is used to create an experiment to test the relationship between deep reinforcement learning and classical conditioning. The author designed the experiment to control for hyperparameters, random effects, maze shape, and maze size. The experiment trains a robot to perform tasks in a maze. The aim of the experiment is to prove the relationship between classical conditioning and the Markov decision process. The results indicate that classical conditioning and Markov decision processes share similar limitations by failing to perform a task that requires planning and self-identification of the goal. Therefore, deep reinforcement learning cannot be autonomous due to the Markov decision process framework. The results demonstrate where the current state-of-the-art AI systems fail and indicate ways to make them more autonomous.

Contents

List of Figures

List of Tables

Table 1: List of Abbreviations

Abbreviation	**Phrase**
A2C	Advantage Actor Critic
ACE	Adaptive Critic Element
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASE	Associative Search Element
CNN	Convolutional Neural Network
CR	Conditioned Response
CS	Conditioned Stimulus
DPG	Deep Policy Gradient
DQN	Deep Q-Learning Network
MDP	Markov Decision Process
NN	Neural Network
SGD	Stochastic Gradient Descent
UCS	Unconditioned Stimulus
XOR	Exclusive Or

1 Introduction

1.1 Motivation

As computing power has increased and become cheaper, artificial intelligence (AI) systems have been developed to perform tasks once done by humans. AI models usually do not complete the entire job but make up a small portion of the overall system. For example, many robots use AI-driven image recognition as an input to a human-designed controller. Few AI methods process and control the entire task.

Major companies and institutions are pursuing performance improvements of AI systems. Most focus on using experiments with high-performance computers and large quantities of data, leading to complex and challenging experiments to replicate. While computationally expensive approachs demonstrate novel capabilities, simpler experiments can indicate more clearly the performance of AI methods.

Advances within the field could lead to improvements in healthcare, industry, and agriculture. The wide range of fields AI could affect amplifies any improvement made. Therefore, understanding the methods, mechanisms, and attributes applied in this field has significant value. Solving the problem of solving problems is the ultimate aim and goal of autonomous AI.

1.2 Aims and Objectives

While fully autonomous AI will not be designed for a long time, assessing the current methods will detail which avenues need to be explored. Deep reinforcement learning is the state-of-the-art AI system for robotics (Fenjiro and Benbrahim, 2018). This work aims to explore the major forms of deep learning and understand the pros and cons of each method. This thesis also seeks to demonstrate analogous behaviour between deep reinforcement learning and conditioned responses within animals. The final aim is to illustrate the limits of deep reinforcement learning.

The objectives of this work are:

- Clearly define autonomous artificial intelligence.
- Review neural networks and deep reinforcement learning.
- Show the relationship between the Markov decision process and classical conditioning.
- Demonstrate the current successes of state-of-the-art AI.
- Create and test a valid experiment for the hypothesis that *classical conditioning and deep reinforcement share limitations for task performance, adaptability, and robot navigation.*

This thesis covers the definitions of intelligence, AI, and autonomy. The background section describes the deep learning methodology. The biological and mathematical foundations of classical conditioning is explained. This project demonstrates that deep learning combined with Markov decision processes creates methods that learn to perform tasks. The mathematical relations between conditioned responses and classical conditioning is analysed. A study on the OpenAI gym is run, where deep reinforcement learning methods learn to play video games. A final experiment that aims at demonstrating the relationship between deep reinforcement methods and classical conditioning is setup. The results are analysed and discussed. Avenues for further research are proposed.

2 Background

2.1 Definitions

Artificial intelligence (AI) has been researched since the 1940s, and academics still disagree on the precise definition of autonomous AI. A robust scientific test requires knowledge of the exact attribute measured to create a valid experiment. To this end, the most pertinent terms are reviewed, analysed, and defined. The goal of defining these terms is not to argue there are no other adequate definitions but to clearly state the definitions used in this thesis. Intelligence, artificial intelligence, and autonomy are stated in this section.

2.1.1 Intelligence and Artificial Intelligence

The first popular definition and test for intelligence was proposed in 1950 by Allan Turing. In Turing's paper *Computing Machinery and Intelligence*, he argues that a machine could be defined as thinking if it passes an imitation game (Turing, 2009). In the game, the interrogator can ask two players any question to determine whether a player is a machine or a human. The goal of each player is to convince the interrogator that the other player is a machine. Turing argues that should a machine win the game, it can be considered a thinking machine. The imitation game, more often called the Turing test, has been criticised on numerous occasions (Saygin et al., 2000).

Searle (1980) disputed the Turing test with the famous thought experiment, the *Chinese Room*. Searle's thought experiment involves an AI which is capable of writing in Chinese. The AI is good enough to trick any person into believing it is a native Chinese writer. If a human, who does not know Chinese, follows the AI's instruction set, they would convince any person to believe it is a native Chinese writer. Therefore, Searle argues that as nobody could distinguish between a thinking native Chinese speaker and an unthinking English speaker following an AI's instructions, a person cannot classify thinking and unthinking agents. Searle concluded that as humans cannot distinguish intelligence based on conversation, the Turing test was not a reliable measurement of thought.

Marvin Minsky argues that intelligence is the combination of many unintelligent sub-agents (Minsky, 1988). The requirement of every sub-agent being unintelligent within Minsky's definition begs further questions. What sort of combinations create intelligent agents? Why would many intelligent agents not create an intelligent agent? How does one measure the resulting intellect? These questions create an unscientific theory as it is unmeasurable and unclear.

A review of 70 definitions of intelligence led Legg et al. (2007) to conclude that a suitable definition of intelligence is: *"Intelligence measures an agents ability to achieve goals in a wide range of environments"*. Legg et al. (2007) introduce a new attribute to intelligence: the agent must perform more than one task and the task needs to be completed in many environments. Hernández-Orallo (2017) identified three different approaches to the measurement and definition of intelligence to create greater consensus on the topic. Hernández-Orallo proposes the three standard measures of intelligence are human discrimination, benchmarks, and peer confrontation.

Chollet (2019) built on the work of Hernández-Orallo and proposed a concrete definition of intelligence. Chollet argues that skill is a product of intelligence with controlled prior knowledge, experience, and the difficulty of the task. Therefore, as skill is the product of intellect, the acquisition of skill is the measurable attribute of intelligence. Chollet fully defines the intelligence of a system as: *"its skill-acquisition efficiency over a scope of tasks, with respect to priors, experience, and its generalisation difficulty."* (Chollet, 2019) The concept of intelligence as skill acquisition efficiency means that learning is a vital component of intelligence. Generalisation is the ability of an agent to transfer a learnt skill to another environment or task. Chollet's definition specifies that a newborn infant is an intelligent agent because it can acquire skill.

This thesis uses Chollet's definition of intelligence as it is pragmatic, measurable, and objective. The agents in this thesis will be subjected to a battery of tests to measure the agent's ability to succeed at multiple goals across changing environments. The rate at which the agent learns is the underlying metric used to determine a successful agent.

As explained by Wang (2019), there is little difficulty in defining the term artificial. Therefore, the definition of AI used for the remainder of this work is a human-made agent who acquires skill to perform tasks.

2.1.2 Autonomy

Research for the definition and measurement of autonomy is relatively new compared to the research on intelligence. No major consensus exists for the meaning of autonomy, let alone its measurement.

Smitherst (1992) argues that many definitions of autonomy are tied to folk psychological notions and constructs, which cannot be objective. Smithers proposes that an autonomous robot does not need a symbolic representation or understanding to be au-

tonomous but can be understood as a dynamic system, resulting in a broad definition that an autonomous robot is simply a robot performing a task. As a result, even a human-controlled robot would be considered autonomous.

More constrained definitions exist which stipulate that an autonomous agent must learn entirely independently of the human designer (Pfeifer and Scheier, 2001). Bradshaw et al. (2013) makes two points counter to Pfeifer and Scheier (2001): first, autonomy is not a single attribute and consists of self-directness and self-sufficiency. Second, autonomy may not be possible in all cases. As there is an infinite number of possible environments, one set of rules could not create an autonomous agent in all of them.

Ezenkwu and Starkey (2019) break an autonomous robot down into high and low-level attributes, which must occur without the designer's intervention. The low-level characteristics are learning, context-awareness, actuation, perception, and decision-making. The high-level features are domain-independence, self-motivation, self-identification of goals, and self-recoverability. Low-level attributes are fundamental mechanisms which are explicity programmed. High-level attributes are behavioural and are not explicity programmed but occur emergently through learnt behaviour.

The definition of autonomy used from now on is based on Ezenkwu and Starkey's description. Context-awareness and self-recoverability are replaced with abstraction. As both are fully encompassed by learning an abstract representation of the environment and itself. The modification is made because it is more general and simpler.

Table 2: Attributes of Autonomy

Low-Level Attributes	**High-Level Attributes**
Learning	Domain-independence
Actuation	Self-motivation
Perception	Self-identification of goals
Decision-making	
Abstraction	

Table 2 shows the attributes of autonomy that will be used for the remainder of this thesis. Autonomy will not be considered a discrete attribute but will be regarded as continuous. Currently, no direct measurement of autonomy exists, instead, subjective arguments are used to debate to what degree the agent is autonomous.

2.1.3 Final Definitions

Intelligence, AI, and autonomy are clearly defined. All three words have a broad range of definitions that demonstrate the immature nature of the field. With time the terms will likely become more specific and generally agreed upon according to Wang (2019).

Table 3: Definitions of Intelligence, Artificial Intelligence, and Autonomy

Term	**Definition**
Intelligence	"A measure of skill-acquisition efficiency over a scope of tasks with respect to priors, experience and its generalisation difficulty."(Chollet, 2019)
Artificial Intelligence	A man made agent which aquires skill to perform tasks.
Autonomy	An agent who is self-motivated and can self-identify goals. The agent can perceive, interact, make decisions, and learn its environment, regardless of the domain and without human action.

Table 3 gives the final definitions used for the remainder of this work, no other definitions are used. Note that a definition of abstraction is out of the scope of this report. However, further analysis into a formal mathematical definition would be helpful in the design of novel learning mechanisms. It is also noted that the definitions given are anthropocentric (regarding humankind as central) as all the definitions frame the problem in terms of performing tasks, perceiving, and interacting with an environment as a single agent. It could be argued that many species such as bees, birds, and other swarm animals behave intelligently as a collective but not as a single agent (Kennedy, 2006; Leimeister, 2010). The limitation is acknowledged, and if more research shows that intellect is tied to social interactions, the definition should be updated.

From the definitions given, a central concept arises; the manner in which the robot gains the skills is of high interest as this allows the system to be adaptable and general. The direct skills gained from any system is not of great interest in this dissertation. In the simplest of terms, the central point of autonomous artificial intelligence is learning. This thesis explores the methods, mechanisms, and attributes in which learning takes place.

2.2 Deep Learning

The current state-of-the-art AI systems is all premised around the neural network (NN) (Fenjiro and Benbrahim, 2018). As the reader is assumed to have an engineering background, a brief explanation of the biological framework follows below before the introduction of artificial neural network's (ANN).

Figure 1 shows a neuron. It operates by transmitting an electrical signal from the axon (input) to the dendrite (output). Synapses are used to control and modulate the electrical current the neuron receives from other neurons. Synaptic plasticity is the change in the synapses conductivity. The change in conductivity is the primary mechanism for learning. Synaptic plasticity is dependent upon the firing of the synapse. The dendrites connect to axons of other neurons to propagate the signal further. Neurons drive animal's perceptions, actions, and reactions. Neurons tend to organise into layers of neurons, creating neural networks. Many researchers are inspired by biological neurons to develop systems that mimic neuronal attributes.

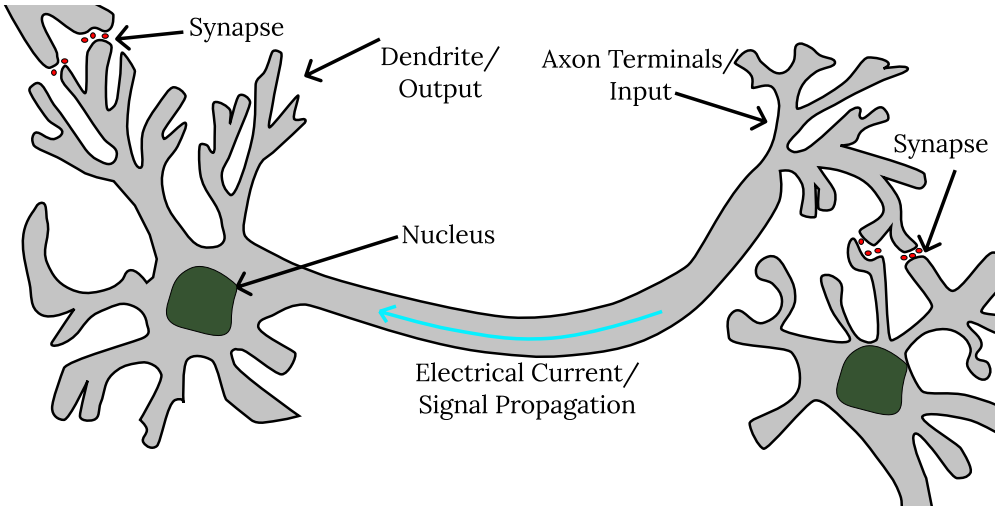

Figure 1: Biological neuron in cell body (gray), nucleus (dark green), and synapses (red dots between joints).

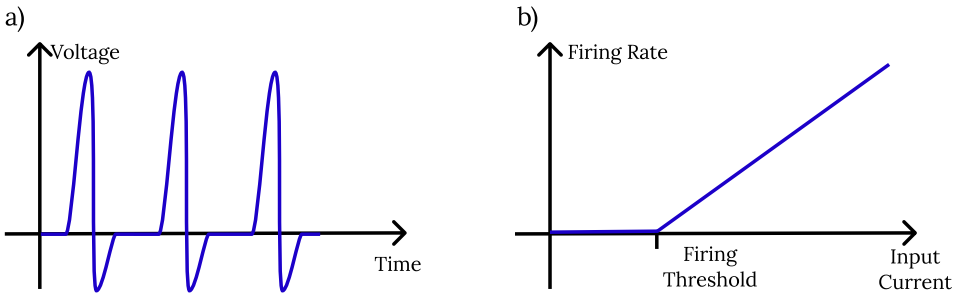

Figure 2: Categories of neuron firing models. a) Spiking neuron with x-axis representing the time and the y-axis representing firing voltage. b) Firing rate neuron with the x-axis representing the frequency of firing and the y-axis representing the neurons input current.

Neurons have all or nothing firing behaviour. Electrical stimuli excite the neuron; if the stimulus is sufficiently strong, the neuron creates a fixed voltage spike. The frequency of these spikes is proportional to the current the neuron receives. Figure 2 illustrates the difference between the spiking and frequency behaviour of neurons. Figure 2a) shows the all or nothing spiking behaviour of a neuron. Figure 2b) shows, that as the neuron receives more input current, its firing frequency increases. Note that the current must be above a firing threshold in order to have a linear response. Many neuron models exist for both spiking and rate-based neurons. In this thesis, for simplicity, all neurons are rate based as seen in Figure 2b).

Different types of ANN's and their application to function approximation are covered below. Function approximation is the way in which machines are taught to perform tasks. When presented with examples, a system can learn to estimate a function by approximating the input-output mapping.

McCulloch and Pitts (1943) showed that a mathematical model of neural circuits could model logical operations. In 1950 Alan Turing argued that a machine could mimic

learning by simulating a child's brain (Turing, 2009). Turing later expanded the use of biologically inspired methods to modelling the patterns on animals fur. Turing successfully predicted the type and structure of the patterns on animals' fur by solving a chemical diffusion equation. (Turing, 1990). While not directly related to AI, Turing demonstrated the modelling of biological systems mathematically, a central concept in AI. McCulloch, Pitts, and Turing lay the conceptual foundations for biologically inspired mathematics and AI.

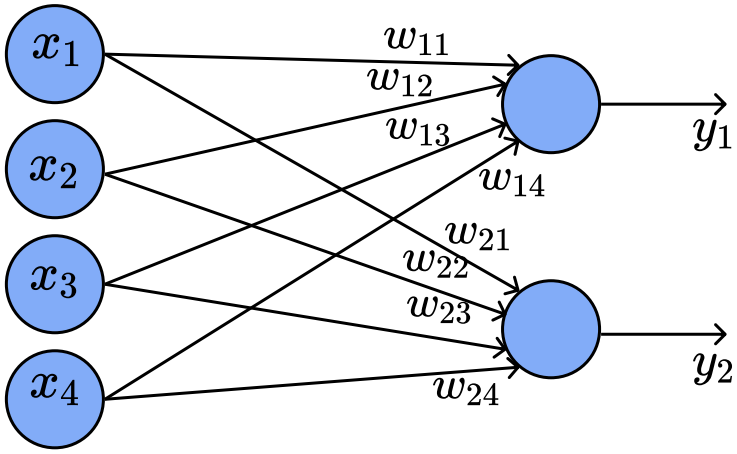

Figure 3: Perceptron model where x's are input stimuli from other neurons, w's are the weight given from one neuron to the stimuli, and y's are the neurons outputs after a non-linear activation function.

A neural model, called the perceptron was created by Rosenblatt (1958) which modelled the neurons synaptic connections as weights. Figure 3 shows a single layer two perceptron ANN. Perceptrons are trained to discriminate between classes of mutually exclusive items by giving examples of what inputs mapped to which output. This training method could only work for one layer of neurons. The perceptron modelled a neuron's output as:

$$y_j = \sigma(\sum_{i=1}^{m} w_{ji}x_i + b_j), \tag{1}$$

where y_j is the output of neuron j, σ is a non-linear output function, w_{ji} is the weight given to input i by neuron j, x_i is the signal from input i, m is the total number of inputs to the neuron, and b_j is a thresholding offset for neuron j.

Minsky and Papert (1969) demonstrated that a single layer of perceptrons could not approximate the exclusive or (XOR) function. The inability to approximate the simple XOR function posed a major setback for the development of ANNs. Minksy noted multiple layers could estimate the XOR function, however no suitable training mechanism existed. Rumelhart et al. (1986) proposed training multiple layers with stochastic

gradient descent (SGD) and a chain rule operation called backpropagation. The exact calculation of backpropagation is omitted for brevity. SGD minimises a loss function by calculating the change in loss with respect to a parameter within the ANN. A loss function is a measurement of error between the networks output and the desired output. The change in network parameters is calculated as,

$$\Delta\theta_{ji} = -\alpha\frac{\partial L}{\partial\theta}, \tag{2}$$

where α is the learning rate, L is the networks loss, and θ is a parameter, such as a weight or bias, within the ANN.

Once deep neural networks could be trained, more complex functions could be approximated. It was proven that a ANN is a universal function approximator; such that given infinite depth or width, ANN's can approximate any function (Pinkus, 1999).

Various architectures, novel layers, and optimisers have been developed to improve the speed and quality of learning. Two popular architectures extend ANN's to temporal/sequential and spatial problems. For temporal and sequential problems, long short term memory cells use a recurrent gating mechanism to allow information to flow from one timestep to another (Hochreiter and Schmidhuber, 1997). For image or other spacial data, convolutional neural networks train filters to perform tasks (LeCun et al., 1989).

Deep neural networks have improved greatly thorough many additional learning mechanisms. Various SGD-like optimisers have been created to speed up learning; most use running averages or adaptive learning rates (Sutskever et al., 2013; Loshchilov and Hutter, 2018). Learning rate schedules have also shown acceleration of learning and improved results (Loshchilov and Hutter, 2016; Smith, 2017). Randomly preventing neurons from firing has been shown to regularise the network's learning (Hinton et al., 2012). Normalising the outputs of the neurons between layers was shown to increase the rate of learning (Ioffe and Szegedy, 2015; Ba et al., 2016). Neural architectures such as residual and dense networks, have shown different connectivity schemes between layers improve the learning of deep neural networks (He et al., 2016; Huang et al., 2017).

The number of combinations and permutations of optimisers, learning rates, regularisation methods, normalisation, and network architectures has caused an explosion in the possible ways of training a deep neural network. The success of deep learning is not due to a simple one size fits method but relies upon the designer to construct a good neural architecture, combined with the various techniques just described (LeCun et al., 2015). This begs the question, if a significant effort is needed to custom design and tune the network to the task, are deep neural networks autonomous?

In the context of robotics, an agent should be acting and perceiving the environment in order to perform tasks. The material presented above focuses on the optimisation of

a loss function. However, the goal is to construct a framework that allows the agent to achieve some goal through its actions; this is covered in 2.4.

The techniques described above are also not biologically plausible. Deep learning with backpropagation requires each neuron to know all the other neurons' weights (Rumelhart et al., 1986). There is no biological evidence for this to be the case. The purpose of this work is to juxtapose the difference between biological learning and the deep reinforcement framework. In the next section the biological learning methods are described.

2.3 Hebbian Learning

While the perceptron and foundations of deep learning were being studied, other researchers pursued more biologically plausible models. Donald Hebb, in 1949, proposed neurons behave plastically and change their efficacies when stimulated (Hebb, 2005). The more a synapse is stimulated, the stronger it becomes. This is best described in the mantra: *that which fires together wires*. While Hebb never mathematised the rule, it can be expressed as,

$$\Delta w_{ji}(t) = \alpha y_j(x) x_i(t), \tag{3}$$

where, $\Delta w_{ji}(t)$ is the change in weight between neuron j and input i at time t, and α is the learning rate. Hebb's rule is associative; if two stimuli happen simultaneously, the neuron learns to correlate the inputs. The more correlated the inputs, the larger the neuron's response. For example, the smell and look of particular objects can become associated. Hebbian methods are unsupervised, they do not map inputs to output. Their weights change without a goal, unlike SGD.

While Hebbian theory does present a high-level explanation of unsupervised learning of associative representations, it fails to provide a mechanism for decision making and action. Animals, through reward, can develop conditioned behaviour. Russian psychologist Ivan Pavlov showed that a dog could be trained to salivate when a bell is rung (Pavlov and Gantt, 1928). Pavlov's dog is the most famous example of classical conditioning, but it is a general learning mechanism (Dickinson and Mackintosh, 1978). The association between an input and a reward can be learnt using a weight update,

$$\Delta w_{ji}(t) = R(t)(\alpha y_j(x) x_i(t)), \tag{4}$$

where $R(t)$ is the reward given to the agent for transitioning between the current and previous state. Equation (4) does not include a term for learning temporal associations. Stimuli must occur simultaneously for the neuron to learn an association between the

inputs. In Pavlov's experiment, the bell was rung before the dog received the food; therefore, classical conditioning learns temporal associations. Synaptic traces have been proposed to explain temporal behaviour. Synaptic traces use a short-term running average to allow neurons to learn a temporal association. The mechanism for synaptic traces is currently an active area of research within neuroscience and is considered a plausible theory (He et al., 2015). The synaptic trace of the neuron's input and output in vector notation is,

$$\bar{x}(t+1) = \lambda\bar{x}(t) + x(t) \tag{5}$$

and

$$\bar{y}(t+1) = \beta\bar{y}(t) + (1-\beta)y(t) \tag{6}$$

where $\lambda \in [0,1]$ and $\beta \in [0,1)$ control the rate the synaptic traces change at each timestep. The weight update rule for synaptic traces is

$$\Delta w_{ji}(t) = \alpha(y_j(t) - \bar{y}_j(t))\bar{x}_i(t). \tag{7}$$

When combined with reward the weight update becomes:

$$e_{ji}(t+1) = \delta e_{ji}(t) + (1-\delta)y_j(t)x_i(t), \tag{8}$$

$$\Delta w_{ji}(t) = \alpha R(t)e_{ji}(t). \tag{9}$$

Equation (9) learns the inputs which lead to a reward defining an associative search element (ASE).

The brain does not directly receive the reward from food. The mind controls its motivation using neurotransmitters. The primary reward neurotransmitter is dopamine. Mice, when deprived of dopamine, starve to death even when presented with enough food; as the mice have no drive to eat (Szczypka et al., 1999). Barto and Sutton (1982) showed that when reward is joined with synaptic traces the adaptive neurons weight update is,

$$\Delta w_{ji}(t) = \alpha(R(t) + \gamma r(t+1) - r(t))\bar{x}_i(t), \tag{10}$$

which learns the association between the temporal difference of the reward and the expected future reward and the running average of the input. Equation (10) teaches a neuron to learn the value of a given state and forms the adaptive critic element (ACE). Classical conditioning is only one way in which the brain learns to perform tasks. While

it is an effective way to train animals, they are not autonomously performing the task. Conditioned responses reinforce previous actions and therefore lack planning, intellect, and identification of the goal. So any system which is based upon a conditioned response cannot be autonomous.

2.4 Reinforcement Learning

As mentioned at the end of section 2.2, deep learning does not provide a framework for problems relating to robots. Reinforcement learning is the process of learning the actions which lead to the most reward.

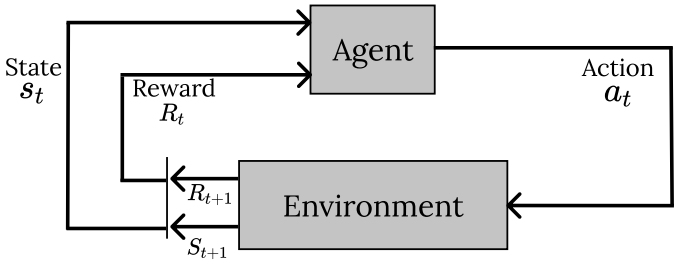

Figure 4: Block diagram of reinforcement learning framework. The agent receives the state s_t from the environment and makes an action a_t. The environment gives a reward to the agent R_t for the action.

Figure 4 shows the block diagram for reinforcement learning. At each timestep the agent choses an action from a sensory state. The environment then rewards the agent action. The goal of the agent is to maximise the total reward gained. The current state-of-the-art relies on the Markov decision process (MDP) (Sutton and Barto, 2018, p. 223).

The MDP framework is represented by a tuple of $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of possible actions, $P \in [0, 1]$ is the probability of transitioning from one state to another given an action, R is the expected reward for transitioning between two states, and $\gamma \in [1, 0)$ is the discount factor which controls the discounted future reward calculation (François-Lavet et al., 2018).

The Markov decision process assumes that previous states and actions do not effect the probability or reward of transitioning between two states,
$\mathbb{P}[S_t|S_t, A_t] = \mathbb{P}[S_t|S_1, ..., S_t, A_1, ..., A_t]$ and $\mathbb{R}[S_t|S_t, A_t] = \mathbb{R}[S_t|S_1, ..., S_t, A_1, ..., A_t]$. The total discounted reward is expressed as,

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1}, \tag{11}$$

where G_t is the total discounted reward and γ is the discount factor which gives greater value to more immediate rewards rather than longer term future rewards. The value of a state and the quality of an action can be calculated as:

$$v(s) = \mathbb{E}[G_t|S_t = s], \tag{12}$$
$$Q(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a], \tag{13}$$

where $v(s)$ is the value of the state s and $Q(s,a)$ is the quality of action a given the state s. A policy the agent is taking is specified,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s], \tag{14}$$

where $\pi(a|s)$ is the probability of the agent taking action a given state s. Both the value and quality of the actions can be redefined in terms of the agents policy,

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{15}$$
$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{16}$$

The optimal policy, π_*, of a MDP can be defined as $\pi_* >= \pi, \forall\pi$. The optimal Q-value function can then be defined as $Q_*(s,a) = \max_{\pi\in\Pi}(Q_\pi(s,a))$ where Π is the set of all policies. The optimal policy can also be defined in terms of the optimal Q-value function $\pi_*(s,a) = \arg\max_{a\in\mathcal{A}}(Q_*(s,a))$. So knowing the optimal Q-value function allows the agent to follow the optimal policy. Substituting equation (11) into (15) and (16) then bootstrapping the future value gives,

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma v_\pi(s_{t+1})|S_t = s], \tag{17}$$
$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t + \gamma v_\pi(s_{t+1})|S_t = s, A_t = a]. \tag{18}$$

As ANN's rely on minimising some loss function, Equations (17) and (18) can be rearranged into the temporal difference equations,

$$\delta_t = \mathbb{E}_\pi[R_t + \gamma v_\pi(s_{t+1})] - v_\pi(s), \tag{19}$$
$$\delta_t = \mathbb{E}_\pi[R_t + \gamma v_\pi(s_{t+1})] - Q_\pi(s,a), \tag{20}$$

where δ_t is the temporal difference error between the expected future reward and the current value. By minimising δ_t an ANN can learn an estimated function for the value of a state or quality of an action. The value of the next state can be estimated from the quality by selecting the highest quality action in the next state $v_\pi(s_{t+1}) = \max_a(Q_\pi(s_{t+1},a))$.

Equations (19) and (20) are rearanged into update rules,

$$v(s_t) \leftarrow v(s_t) + \alpha[R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s)], \tag{21}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma \max_a(Q_\pi(s_{t+1}, a)) - Q_\pi(s, a)], \tag{22}$$

where $v(s_t)$ and $Q(s_t, a_t)$ are approximated by an ANN with the temporal difference error from Equations (19) and (20). The temporal difference equation can be treated as a loss function. The quality of state-action pairs can be used to find the optimal policy. Another option is to search for the optimal policy directly. By increasing the probability of taking an action that leads to a higher future value, the agent can learn the optimal policy. These sets of methods are called policy gradient. Policy gradient uses gradient ascent, as opposed to gradient descent, by maximising the log probability of actions that lead to the highest discounted future reward. This can be written as:

$$\nabla_\theta L(\theta) = \mathbb{E}_\pi[G_t \nabla_\theta \ln(\pi_{theta}(a|s))]. \tag{23}$$

where ∇_θ is the ANN's gradient and $L(\theta)$ is the cost function.

2.5 Comparing Classical Conditioning and Deep Reinforcement Learning

Above the mathematics behind biologically plausible classical conditioning have been described and the MDP has been explained. It has been explained how deep learning can use the MDP framework to learn an optimal policy. In the table below the weight update and the loss functions are displayed together to show the similarities between these methods.

Table 4: Comparison of Classical Conditioning and Markov decision process equations

Method	**Learnt Parameter**	**Equation**
ACE	Input Value	$\Delta w_{ji}(t) = \alpha(R(t) + \gamma r(t+1) - r(t))\bar{x}_i(t)$
ASE	Best Action	$\Delta w_{ji}(t) = \alpha R(t) e_i(t)$
Q-Learning	Action Quality	$L = \lvert R_t + \gamma \max_a(Q_\pi(s_{t+1}, a)) - Q_\pi(s, a)\rvert$
Value Estimation	State Value	$L = \lvert R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s)\rvert$
Policy Gradient	Best Action	$L = -G_t \ln(\pi_\theta(a\|s))$

Table 4 shows the different methods to allow comparison. The table shows that biologically plausible learning is related to the MDP framework. The equation describing ACE's weight update is similar to the loss function of Q-Learning and value estimation. The primary difference is ACE uses a running average of the input. The relationship

between ASE and policy gradient methods is less obvious. Frémaux and Gerstner (2016) demonstrated that classically conditioned Hebbian learning with synaptic traces approximates policy gradient methods. This indicates that the learnt MDP is an approximation of a conditioned response. As deep learning uses MDP's as its framework, if MDP's are related to classical conditioning, deep learning cannot be an effective method for autonomous artificial intelligence. This thesis seeks to experimentally test whether deep learning behaves like a classically conditioned agent.

3 Methods

Following, the various methods to be tested are described. As the goal of this work is to show the relationship between classical conditioning and deep reinforcement learning, detailed descriptions of both types of learning are shown. All methods are programmed in Python 3.7 using PyTorch 1.8.1, and Numpy. The methods are implemented such that they can be computed on the central processing unit and the graphics processing unit depending on the circumstances. Psudo-code for each method is included in Appendix A. The code for every experiment and method is available in Appendix E

3.1 Terms and Dilemma

3.1.1 Hyperparameters

A significant term that must be introduced is hyperparameters. These are constant values which control the learning behaviour of the method. The term hyperparameter is used to distinguish between the designer-controlled constants and the internal parameters of the network. Examples of hyperparameters are learning rates, discount factor, and eligibility trace constants. Examples of parameters are the weights and biases inside the network.

3.1.2 Exploration vs Exploitation

A central problem to reinforcement learning is the trade-off between exploring the environment to find the actions with the most rewards and executing what is already known to get rewards. The dichotomy leads to a necessary addition to each method. There must be a non-zero random chance to perform any action at any timestep. Each method has its technique for implementing non-deterministic actions. This thesis aims to explore the general framework of reinforcement learning and understand its limitations.

3.2 Reward Modulated Hebbian Learning

Classical conditioning is a behavioural phenomenon that can be observed within animals. Within neuroscience and AI, the conditioned behaviour is modelled using reward modu-

lated Hebbian learning (Barto et al., 1983; Gerstner et al., 2018). The reward modulated Hebbian methods used within this work will be described within this section.

Such biologically plausible algorithms are not simulations of biological neurons. They emulate segments of biological behaviour and abstract the learning mechanics. The methods below do not mirror neuronal behaviour within the brain. Instead, they utilise abstracted learning mechanisms to emulate conditioning and associative learning.

3.2.1 Associative Search Element

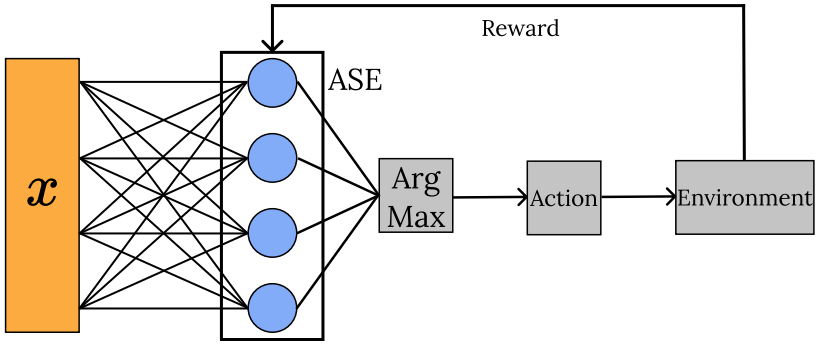

Figure 5: Block diagram of associative search element. The associative search element takes an input x, each neuron in the ASE represents an action, the arg max function selects the neuron which has the highest output. This dictates the action taken. The ASE receives feedback from the environment with a reward to update its weights.

Figure 5 depicts the block diagram of a network using an associative search element (ASE). The diagram shows that the input is fed to the Hebbian network. The highest value neuron specifies which action is taken. The ASE receives feedback from the environment through reward. Only the neuron whose action is chosen has its eligibility trace updated. The feedforward section of the network is defined as,

$$y(t) = W_{ASE}(t)x(t) + \xi, \tag{24}$$

$$a(t) = \arg\max y(t) \tag{25}$$

where, $a(t)$ is the action chosen and, ξ is guassian noise added to the neurons ouput to induce exploration. The weight update in vector notation is,

$$\Delta W_{ASE}(t) = \alpha R(t) e(t). \tag{26}$$

The eligibility traces are computed slightly differently to Equation (8), seen in Section 2.3. The modification is because the Hebbian update rule is unstable, a small stimulus will increase the weight, and the weight will increase the effect of the stimulus. The

positive feedback loop creates a rule which goes to $\pm\infty$. Oja (1982) proposed solving this problem using a normalisation term which grows faster than the neurons output. Oja proved the equation converges to the highest variance eigenvector, by performing principal component analysis. This gives another interpretation of the ASE, as an adaptive element which finds the principal component of the inputs weighted by the reward. The modified eligibility trace becomes

$$e(t+1) = \delta e(t) + (1-\delta)(y(t)x(t) - y^2(t) \cdot W_{ASE}), \tag{27}$$

where $-y^2(t) \cdot W_{ASE}$ is Oja's normalisation term.

3.2.2 Adaptive Critic Element

As mentioned in Section 2.3, brains create an internal reward scheme. This biological system inspires the Actor-Critic system. In an Actor-Critic architecture the Actor makes decisions while the Critic evaluates the action and informs the Actor on whether it was a good or bad choice. An Actor-Critic architecture allows the agent to receive an internal reward based upon the agent's estimation of the value of a state. The goal is to smooth the reward function and to improve learning. In environments with sparse rewards, the Critic can learn a smooth reward function with the temporal difference equation. The ASE choses the actions, the adaptive critic element (ACE) then gives positive or negative reward based on the expected value of these actions.

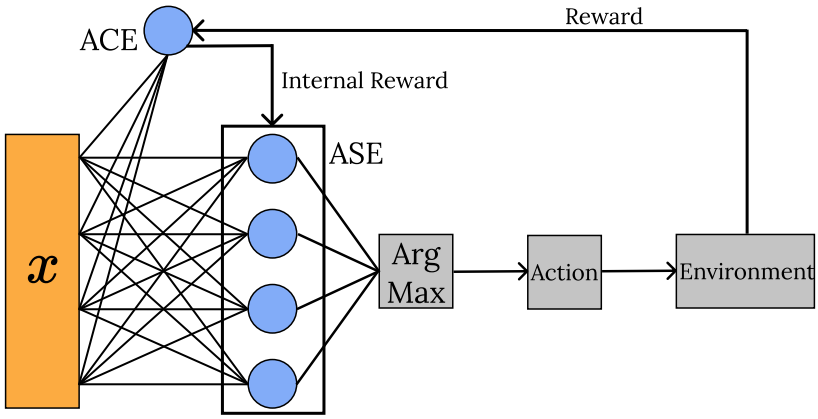

Figure 6: Block diagram of adaptive critic element giving the associative search element an internal reward. The ACE receives a sensory input x and, using the discounted predicted reward, updates its weights. The temporal difference is used as an internal reward for the ASE which works the same as in Figure 5.

Figure 6 shows the block diagram of the ASE-ACE architecture. The ACE receives as input the same sensory information as the ASE but attempts to learn the association between the temporal difference equation and the input. The ASE functions as usual but

uses the temporal difference of the Critic as the reward. The value prediction is computed with,

$$r(t) = W_{ACE}(t)x(t), \tag{28}$$

where $r(t)$ is the ACE's value estimation of input $x(t)$. No Gaussian noise is needed for the ACE; the ACE does not make actions but instead learns the state-value function. The weight update rule is also modified to incorporate Oja's normalisation term for stability. The new equation is:

$$\Delta W_{ACE}(t) = \alpha([R(t) + \gamma r(t+1) - r(t)]\bar{x}(t) - y^2(t) \cdot W_{ACE}). \tag{29}$$

A synaptic trace on the input is used to learn temporal associations. The trace is calculated as the running average of the input,

$$\bar{x}(t+1) = \lambda\bar{x}(t) + (1-\lambda)x(t). \tag{30}$$

The internal reward is calculated using the temporal difference equation,

$$\hat{r}(t) = R(t) + \gamma r(t+1) - r(t) \tag{31}$$

where $\hat{r}(t)$ is the ASE-ACE's internal reward.
Equation (26) is altered to use the internal reward and becomes,

$$\Delta W_{ACE}(t) = \alpha\hat{r}(t)e(t). \tag{32}$$

3.3 Deep Reinforcement Learning

Deep reinforcement learning is the current state-of-the-art in AI on rewarded tasks (Fenjiro and Benbrahim, 2018). They have been shown to beat the best players of Chess, Go and StarCraft 2 (Silver et al., 2017; Vinyals et al., 2019). Each category of deep reinforcement learning has a plethora of variations and permutations. The specific implementations described in this thesis are chosen as they demonstrate a reasonable base method.

3.3.1 Q-Learning with Experience Replay

Q-learning initially started in the form of dynamic programming (Watkins, 1989). Each state action pair was given a unique value. In small discrete domains, this is reasonable. However, when large, continuous inputs are given, the size of unique states explode. Consider a 32x32 image with 255 possible pixel values that produce 2×10^{2464} possible unique states, a greater value than the number of electrons and protons within the universe which is 10^{80} (Eddington, 1923). To solve this problem, machine learning techniques, like artificial neural network's (ANN), can be used to abstract features within the image to infer the value function. The original Q-learning approach updated the value estimation at every timestep with the most recent experience (Sutton and Barto, 2018). The method was improved by storing the past experiences and each timestep drawing a random batch of experiences to learn from (Mnih et al., 2013). The experience replay approach is chosen as it is versatile in the environments it can be used in.

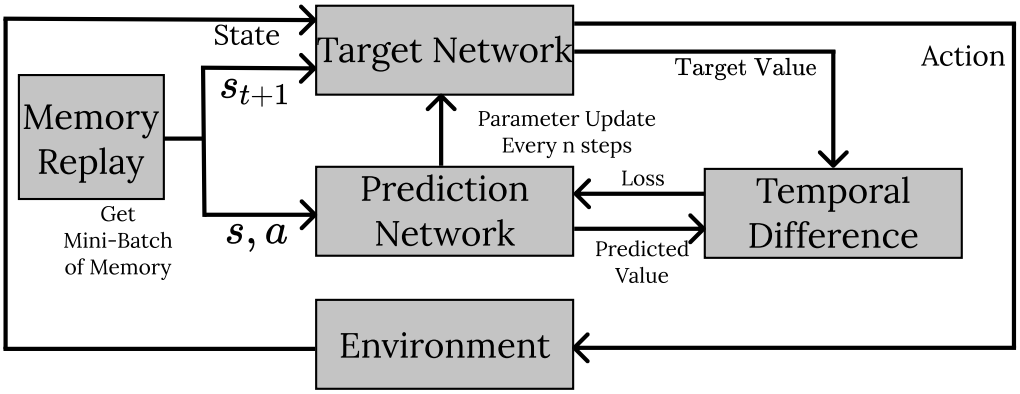

Figure 7: Block diagram of Deep Q-Learning with Memory Replay. The target network receives an input from the environment and makes an action. The memory replay stores past state, action, reward, state pairs. Each timestep a batch of experiences are drawn from memory replay. The state and action are used by the prediction network to calculate $Q_\pi(s, a)$. The future state is used by the target network to calculate $Q'_\pi(s_{t+1}, a)$. The result is used to compute the loss of the prediction network.

Figure 7 shows the block diagram for double deep Q-learning with memory replay. Two networks are used to estimate different sections of the temporal difference equation. The reason is that this stabilises the learning process as the target value is not constantly changing for the same input (Van Hasselt et al., 2016). Every few timesteps, the prediction networks weights and biases are transferred to the target network. The resulting equation for double deep Q-learning is,

$$L = |R_t + \gamma \max_a (Q'_\pi(s_{t+1}, a)) - Q_\pi(s, a)|, \tag{33}$$

where $\max_a (Q'_\pi(s_{t+1}, a))$ is the target networks value estimation of the next state and

$Q_\pi(s, a)$ is the prediction networks value estimation of the current state.

Exploration of the environment is introduced using the epsilon greedy policy. This means that each timestep there is a probability, ϵ, that the agent will take a random action. The ϵ value is slowly decayed linearly until it reaches a minimum value ϵ_{min}. When a random action is not being taken, the highest Q-valued action, $a_t = \max_a(Q'_\pi(s_{t+1}, a))$, is chosen.

3.3.2 Policy Gradient

Policy gradient attempts to learn the optimal policy directly by performing gradient descent based on an episodic memory using the REINFORCE algorithm (Williams, 1992). Sutton et al. (1999) extended the method to use function approximators, such as deep neural networks.

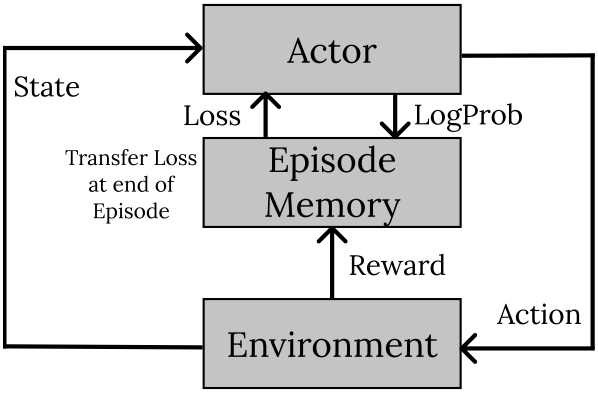

Figure 8: Block diagram of episodic Policy Gradient. The policy gradient method uses a network which selects which action to take for each state. The logarithmic probability of the action being taken and the reward associated with it is stored in sequential memory. After an episode the loss is calculated and the weights updated using gradient ascent.

Figure 8 shows the block diagram of the method. The Actor receives the state and makes an action. The logarithmic probability of making the action is stored in the episode. The environment tells the agent the reward, which is also stored in the memory of the episode. At the end of the episode, the loss is calculated and updated using gradient ascent. As PyTorch has built-in systems for gradient descent, a negative sign is added to the loss function to invert the step direction of the gradient. This produces the loss equation,

$$L = -G_t \ln(\pi_\theta(a|s)), \tag{34}$$

where $\ln(\pi_\theta(a|s)$ is the log probability that the network chooses the action and G_t is the discounted cumulative reward for choosing the action over the episode. The method relies upon a probability of choosing an action, so the network must output a probability

distribution. A SoftMax function is used to transform the network's output into a distribution (Goodfellow et al., 2016, p.180). Drawing actions from a probability distribution also solves the exploration and exploitation problem as no action has an absolute zero chance of being chosen.

3.3.3 Advantage Actor Critic

Like the ACE-ASE method, value and policy methods can be combined using the temporal difference as the policies methods reward. Advantage actor critic (A2C) is a synchronous implementation of OpenAI's A3C algorithm (Mnih et al., 2016).

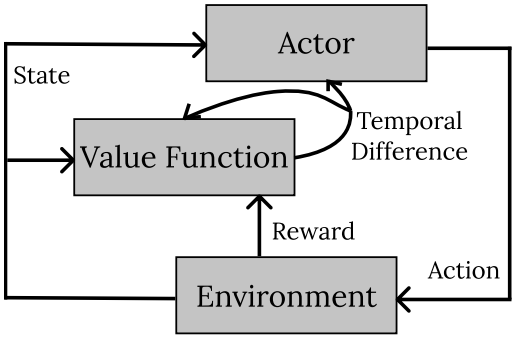

Figure 9: Block diagram of Actor-Critic architecture. The Actor receives the input state and selects an action. The reward for the action is given to the value function which computes the temporal difference. The Actor uses the temporal difference as an internal reward.

Figure 9 shows the general Actor-Critic framework. The Actor is identical to that in Figure 8. The value estimation does not use memory replay, consists of one network and is updated every timestep. The temporal difference and loss of the Critic is calculated as:

$$\delta_t = R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s) \tag{35}$$

$$L_{Critic} = |R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s)| \tag{36}$$

The loss of the Actor is almost identical to Equation (34),

$$L_{Actor} = -\delta_t \ln(\pi_{theta}(a|s)) \tag{37}$$

where δ_t replaces G_t.

4 Open AI Gym

An autonomous robot or agent interacts with its environments by sensing the world, performing actions and receiving feedback. The agent does not necessarily need to be embodied within a robot to test the core attributes. Video games have all the desired characteristics. They have scores that the player should maximise, images to perceive, and buttons to press. The OpenAI gym was created to provide a benchmark for researchers to compare their reinforcement learning systems (Brockman et al., 2016). The OpenAI gym is a collection of reinforcement learning environments, with video games, control problems, and robotics problems. The goal of the following study in the OpenAI gym's framework is to evaluate the performance of deep learning methods and understand limitations in OpenAI's benchmark.

4.1 Experimental Setup

Two games and one control problem are tested. The three environments are Lunar Lander, Cart Pole, and Pong. These methods are chosen as they present a range of actions, reward schemes, and state values.

The OpenAI gym follows the framework laid out in Figure 4. At each timestep an action is given to the environment, observations, and rewards are received. Each environment has differing action spaces $\mathcal{A}$, state spaces $\mathcal{S}$, and Reward functions $\mathbb{R}(S_t|S_t, A_t)$. The hyperparameters, and network architecture used for each method and environment can be found in Appendix B. Ten runs of the Lunar Lander, and Cart Pole and three runs of the Pong environment are performed. Fewer iterations of the Pong environment are performed due to runs taking between one and three days. Each environment is episodic. In each environment there is a win, loss, and timeout condition; each will reset the episode. This thesis is not concerned with the neural architecture which is used in each method, as the goal is to understand the attributes of each of them. The hyperparameters and network architecture used with each method is available in Appendix B. Each method uses a modified implementation of SGD called AdamW which accelerates learning (Loshchilov and Hutter, 2018). The neural networks use a rectified linear activation function as it is currently the most standard activation function (Nair and Hinton, 2010). All methods are computed on the graphics processing unit for reduced computing time. The deep learning methods used are deep Q-learning network (DQN), deep policy gradient (DPG), and advantage actor critic (A2C).

4.1.1 Lunar Lander

Figure 10a) shows the Lunar Lander environment. The goal of the game is to navigate the space ship between the yellow flags. The agent can push the lander left, right or up using

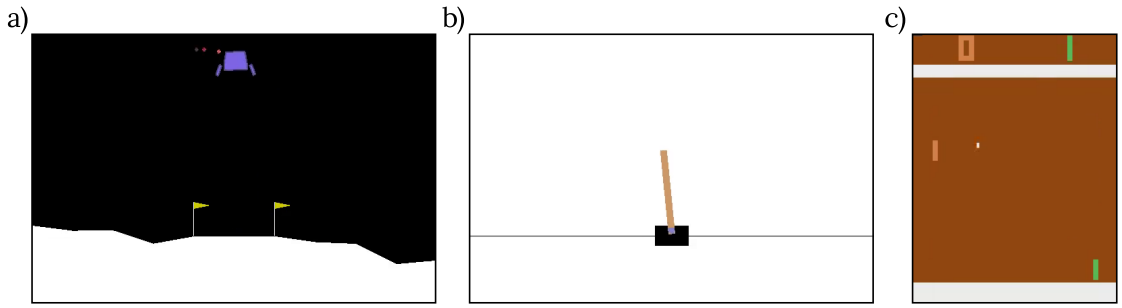

Figure 10: Images of the OpenAI environments used. a) Lunar Lander the blue box is the lander and the yellow flags is the landing zone. b) Cart Pole control problem, the brown pole must be kept upright by the black cart. c) Pong video game the white ball is bounced between the brown and green paddels.

thrusters. For each frame where the agent uses the downwards thruster, it is penalised -0.3 points. The agent is rewarded 140 points for landing with near-zero velocity on the landing pad. If the agent crashes or comes to a standstill, the agent will receive -100 or +100 points, respectively. Each leg that touches the ground when the agent lands is worth 10 points. The points are used as the agents reward function. The only modification is for the DPG method, the reward is scaled to be between -1 and 1, so that it improves stability. The agent receives as input the lander's position, velocity, angle, angular velocity, and which legs have made contact with the ground.

4.1.2 Cart Pole

Figure 10b) shows the Cart Pole environment. Unlike Lunar Lander and Pong, this is a classic control problem. The agent must keep the pole vertical and not let it tip below -15 degrees or move too far away from the centre. At each timestep, the agent can apply a force to the cart moving it left or right. For each frame the pole is upright, the agent receives a reward of 1. They are informed of the state of the cart's position and velocity, and the poles angle and angular acceleration.

A modification of the reward scheme is necessary for the deep DQN as each frame produces a reward of 1. Therefore, all actions of all states are of the same value. The modified reward function applies a reward of -1 when the agent crashes.

4.1.3 Pong

Figure 10c) shows the game Pong. Compared to the Lunar Lander and Cart Pole problem, Pong is the most challenging. The agent receives the pixel inputs from the game and must learn what each object is and how to play the game. The goal of Pong is to hit the white ball around the opponent's paddle and into the goal behind it. Each time the player scores, it receives a reward of 1. The player controls the paddle on the right-hand side of the screen by moving it up, down, or keeping it still. To speed up computation time, each

action is executed for 4 frames. As a single frame is not sufficient to know the direction the ball is moving, the agent uses the past four frames as an input. The image given to the network is cropped, grayscaled, and resized to a 84x84 image.

Each method will use a convolutional neural network (CNN). CNN's use 2-dimensional filters to learn features in an image. The resulting features are then used for classification. The weights of the filter are trained using stochastic gradient descent (SGD), in the same way as a regular artificial neural network (ANN). The ANN used is identical to the one desribed by Mnih et al. (2013). To simplify some aspects of the technical implementation, wrappers from OpenAI's baselines code repository are used to force start the Pong game, rather than wait for the player to press a start button and update 4 frames per action (Dhariwal et al., 2017).

4.2 Results

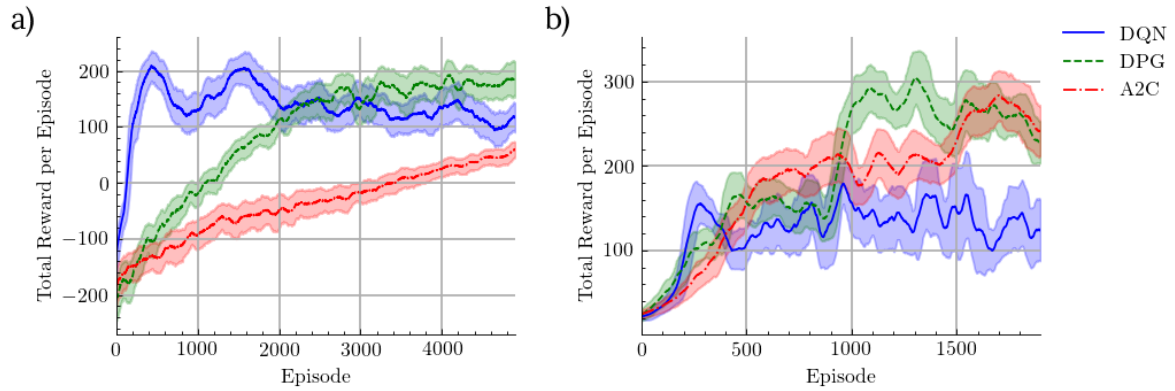

Figure 11: Lunar Lander and Cart Pole results showing the running average of 100 episodes across 10 samples. Upper and lower lines indicates one standard deviation limit between runs. a) Lunar Lander total reward for each episode. b) Cart Pole total reward per episode. Legend applies to both graphs.

Figure 11a) shows the results from the Lunar Lander experiment for the DQN, DPG, A2C algorithms. The DQN result shows the highest peak results and learns the fastest, achieving a total of 200 points within 500 episodes. The figure indicates as more training is given the agent becomes worse. The reason for this is that it is overfitting, which is when rather than learning the general rule the network memorises input to output values. In effect, it has found a low error representation of the Markov decision process (MDP) but it is too complicated and is not representative of the larger problem. The DPG learns reasonably quickly, converging to an asymptote, almost monotonically. A2C learns slower than DPG and does not converge within 5000 episodes. The reason for the slower convergence is that the Critic network must first learn the value of each action then train the Actor. In effect the Critic produces a bottleneck for the training of the Actor.

Figure 11b) shows the results from the Cart Pole control problem. The DQN quickly

converges to a local minima and stays there. It is likely that with enough time, or a change in hyperparameters the local minima will be escaped. DPG and A2C perform similarly, with A2C training slightly slower, indicating A2C may in general learn slower. Both DPG and A2C achieve similar optimal policies, and both peaks are within the other's variance, so any difference in peak value could be attributed to random chance. Therefore, the learnt policy of both methods are similar.

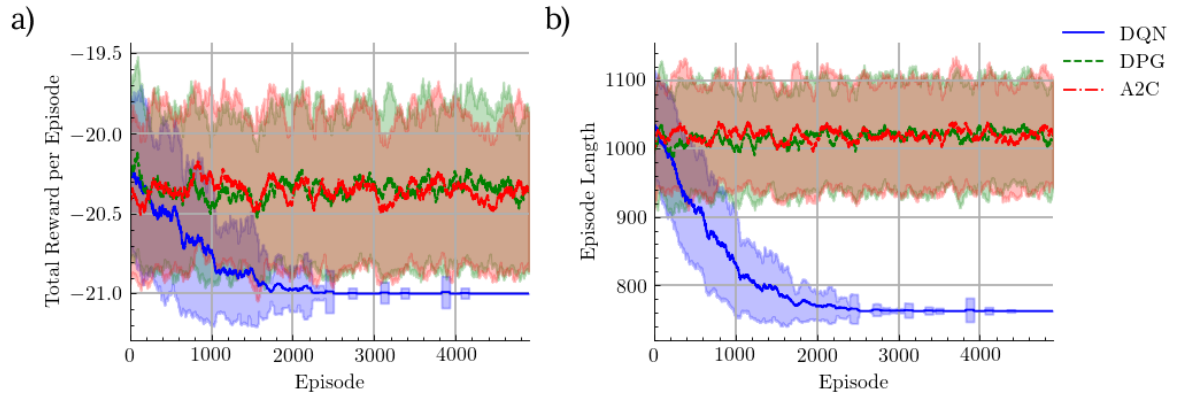

Figure 12: Pong results showing the running average of 100 episodes across 3 samples. Upper and lower lines indicates one standard deviation limit between runs. a) Pong total reward for each episode. b) Pong episode length for each episode. Legend applies to both graphs.

Figure 12a) shows the total reward the agent receives per episode for Pong. The DQN fails to learn an effective policy and converges to the worst possible score of -21. Figure 12b) shows the DQN method's episode length going to 762 frames, which occurs when the opponent scores with one hit every time. Therefore, the agent is not learning to play the game and has become stuck and is not returning the ball. A likely cause is a sparse reward function. A possible solution could be to increase the batch size and replay memory. This would allow the agent to learn a more complete idea of each images relationship to another. As the implementation is based on Mnih et al. (2013), a comparison of memory size, batch size and training time can be made. Mnih used a memory size of one million frames, a batch size of 32, and a training length of 10 million frames; they achieved a final average reward of 20 compared to -21. The Q-learning implementation used in this work had a replay memory of 0.1 million frames, a batch size of 32 and a training length of approximately 4 million frames. As the batch size and training length were similar, the likely cause of the difference is the memory size. The smaller replay memory is due to limited memory being available to the author.

In Figure 12a) the DPG and A2C methods show similar performance. Both do not appear to be learning any skills at the task. Figure 12b) shows a possible increase in episode length, a sign it is stopping the opponent from scoring. However, the slight trend lays within one standard deviation of the runs, indicating that it could be attributed to random chance. Much longer running times are likely needed for both methods to see any

convergence to a reasonable policy.

4.3 Discussion

The experiment demonstrated that deep reinforcement learning is able to learn to perform, complex tasks across a range of environments. It showed that A2C is more similar to a policy gradient method than a value method. This is interesting as it uses both systems, so the Actor dominates the methods behaviour. As a general rule the experiment suggested, A2C converges slower than DPG. It was shown DQN is more sensitive and can overfit leading to worse performance with training time.

All three methods struggled to process image data to play Pong. As other researchers have successfully trained deep reinforcement agents to play Pong it can be inferred that greater computational resources are needed to replicate their results. This indicates that the methods used do not scale well with lower computing power. The results also explain why Deep Minds AlphaStar, an AI which plays StarCraft 2, required 200 years worth of game playing experience (the game has only been available for 11 years) (Vinyals et al., 2019). This demonstrates that deep reinforcement learning methods require large quantities of experience to become adequate at a task.

The inefficiency at complex tasks can be explained given the relationship between deep reinforcement methods and classical conditioning. The eventual reward is associated for every frame and image. No abstraction of the overall goal, environment or agents actions is being learnt. Therefore, a dense sampling of the game must be given to the agent to learn the association between pixels and reward. The delay between hitting the ball and scoring is causing the agent to associate the wrong actions with the reward or penalty. The mostly zero reward function also means the player does not receive a lot of feedback on what a good or bad action is. This indicates the expected behaviour for complex environments beyond pong.

The general behaviour shows that with the correct reward scheme, enough training time, and valid hyperparameters, each method can converge to a reasonable solution.

No processing of the inputs was needed to achieve good performance for either the Lunar Lander or Cart Pole problem. Different inputs are used with little designer intervention which is impressive and demonstrates some autonomy. This indicates that the MDP framework does allow for learning skills in a wide range of conditions. There is an essential distinction between the method being general and the agents being general. The experiment did not look at taking an agent's adaptation to a new task, so the agent's generality cannot be inferred. However, the experiment is valid for the method's adaptability.

The Pong environment indicated that hyperparameters must be controlled. This produces a dilema. If a designer optimises the system by running a sizeable sweep of hyper-

parameters on an environment, the agent will be highly specialised to the problem. At a high level this hyperparameter tuning is similar to wrapping an optimiser, hyperparameter fitting, around an optimiser, stochastic gradient descent. At what point is the solution found through the meta-optimised hyperparameters rather than learned through the neural network? Hyperparameter tuning for the above tasks is not computationally feasible. Each run can take up to one hour and multiple runs would be needed per set of hyperparameters. No assumptions about a linear relationship between the hyperparameters could be made, as shown by Smith (2018). Therefore, testing in such environments prevents comprehensive, systematic studies from being performed without high-performance computer clusters.

The computational limit is not simply an experimental problem, but also a practical problem for robots. An autonomous AI robot would require all learning to be performed as it explored its environment. A high-performance computer will be too large to be embedded into any such robot. This limitation will continue even though computational power is continuously growing making embedded chips faster.

5 Maze Navigating Robot

From the results in Section 4, a stronger, more valid, and computationally inexpensive experiment is created. The experiment's hypothesis is that *classical conditioning and deep reinforcement learning share limitations for task performance, adaptability, and robot navigation.*

5.1 Experimental Setup

Human infants develop object permanence at around 8 to 12 months of age (Huitt and Hummel, 2003). The infant, using previous sensory inputs, creates an internal model of the world. When an object is omitted from view, infants understand that it continues to exist and have an intuition about its position. In contrast, Markov decision processes (MDP) rely upon the assumption that previous actions and states do not affect future reward. The MDP's assumption is not reflective of the real world. A robot cannot have a complete understanding of the world at every point in time. The vast majority of available robotic sensors are cameras, proximity sensors, and temperature sensors, all of which measure local variables.

For an effective and practical set of experiments, a partially observable environment is constructed. The robot must navigate through the maze and find the location of a goal. At each timestep it only has information about the local maze area, whether there is a wall in the eight grids which surround it. Figure 13 shows an example of the robot sensing a small portion of the maze. The sensory information is complete enough to allow for a robot

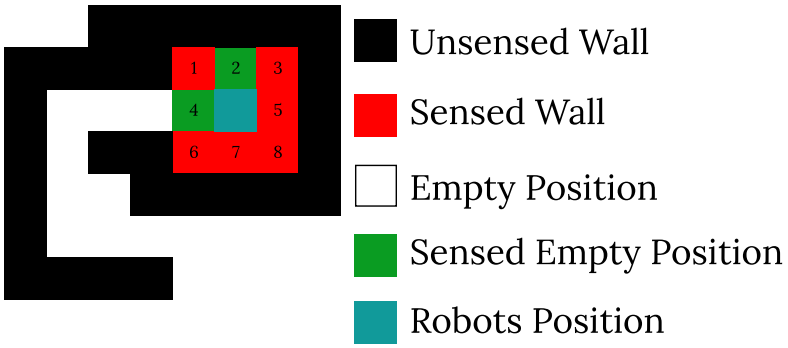

Figure 13: An example of sensed parameters in a portion of the maze. The robot senses two empty spaces indicated with a green block and six filled spaces indicated with the red block. The white spaces and green spaces show where the robot can move, and the teal shows the current space the robot is in.

to always move into a new position. The labyrinth is a discrete gridworld environment. The robot may move up, down, left, or right. As the robot only has limited information about the environment, it must use multiple timesteps to discern the differences between multiple places in the maze.

The problem of partially observable environments has been addressed before (Oliehoek, 2010; Littman et al., 1995). Partially observable environments are solvable with MDPs and form the partially observable Markov decision processes (Monahan, 1982).The most common method of solving such a problem is keeping the previous k-senses and using them as inputs into the robot. While more advanced methods, using recurrent neural networks, exist the purpose of this experiment is to test the relationship between classical conditioning and MDPs, therefore a simple solution is adequate for the task (Hausknecht and Stone, 2015). The robot's sensing system presents a good test for the agent's ability to handle different inputs which represent the same maze position.

Figure 14 shows the mazes that are used for the experiment. The green and gold squares show the starting and rewarded locations. The numbers indicate at which stages in the experiment which location is used. Each maze is designed by hand and has a different structure and pattern. Maze generating algorithms have similar structures and would create five mazes with similar shapes.

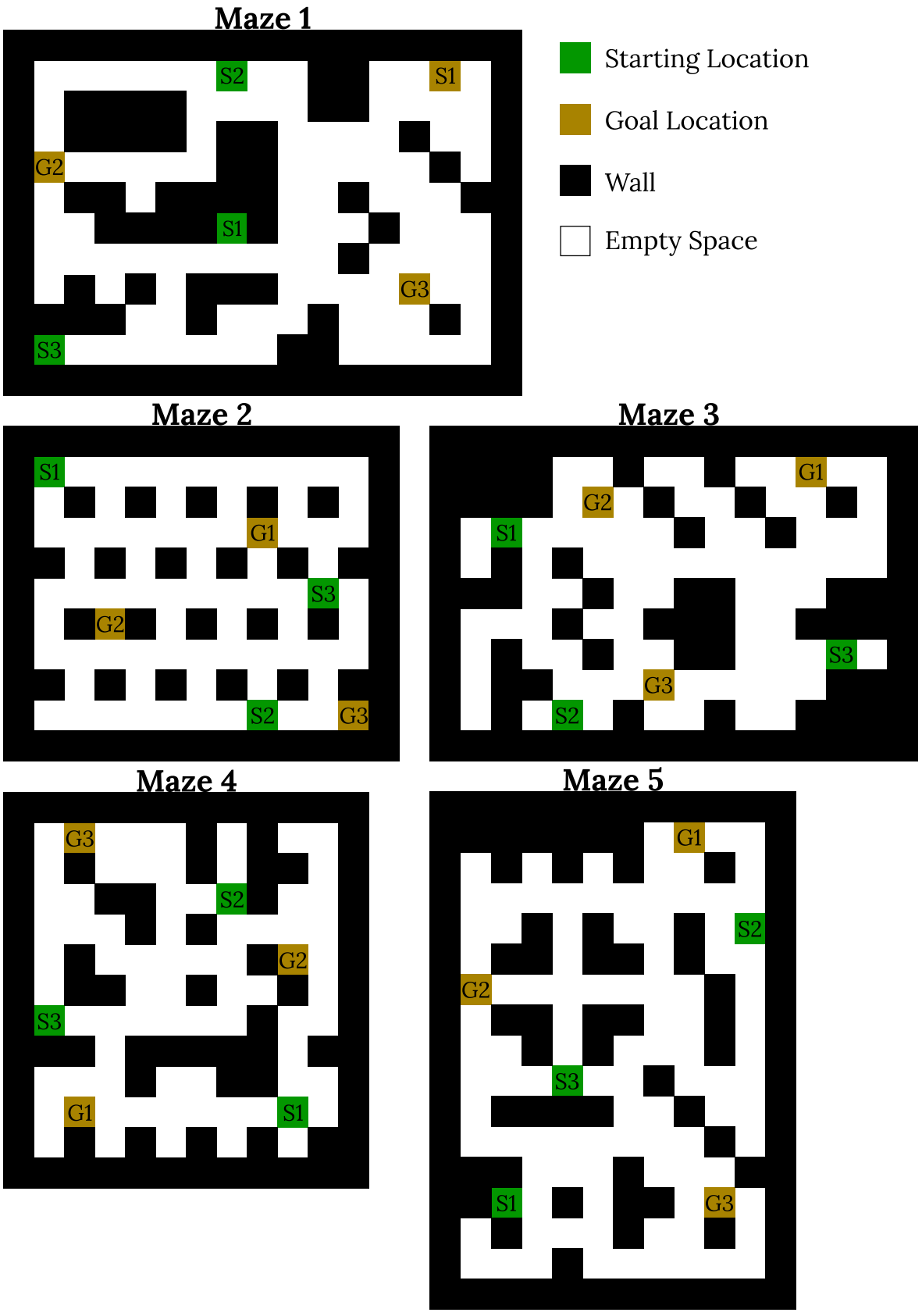

Figure 14: Mazes used for testing comparison. Green boxes show starting locations while gold boxes show goal locations. S1, S2, S3, indicate the starting locations while G1, G2, G3 indicate the goal locations. The number indicates which starting positions and goal positions are randomly chosen during different tests.

A battery of tasks are being used to test the robot's adaptability and generality. Each experiment tests the robots ability to navigate between two points in a maze. The agent is rewarded when it reaches the final position. The agent is placed back at the starting location once it has reached the goal. The possible starting positions and goal positions change from task to task. Figure 15 shows the sequence of tasks the robot must perform in the maze. Task 1 is the most simple where the agent must move from a fixed starting point in the maze to a fixed endpoint in the maze. Each time it reaches the end, it is sent back to the starting point. Task 2 has a moving goal, the goal location is randomly chosen between three possible maze positions. The agent is reset to the same start location. Task 3 has a moving start and a fixed end. The goal location is the same as in task 1. Each time the agent gets to the goal, the agent is sent to one of three possible starting locations.

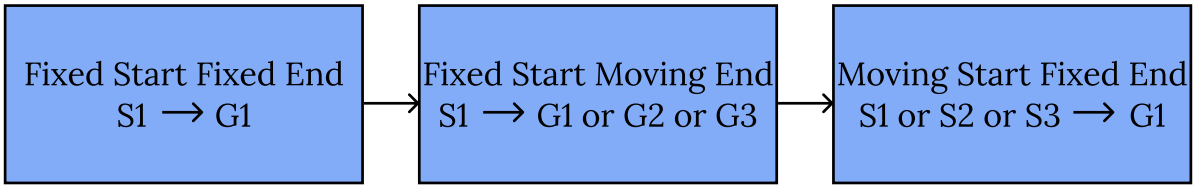

Figure 15: Sequence of tests being performed. Each box shows the starting and ending locations for each segment of the test.

The test focuses on a range of tasks that will test each method's behaviour. The expected result is that the conditioned responses will be able to perform task 1 and task 3. As the act of going to the reward is reinforced, task 2 should not be solvable with classical conditioning. Planning and comprehension of the rules which control the task, need to be developed to solve task 2. Therefore, if deep learning processes succeed at task 1 and task 3 but fail at task 2, it shares similar limitations to classical conditioning and proves the hypothesis. Controlling other variables in this experiment improves the validity of the test.

Important variables to control for are hyperparameters, maze size, maze shape, and random effects. Before each method is tested, it has its hyperparameters optimised on a maze, not used in the test, by sweeping the hyperparameters values. The best hyperparameters are used on the test mazes. Five different mazes are used to control for maze size and shape. Each labyrinth has a different layout, size, goal locations, and start locations. Stochastic effects like weight initialisation, epsilon greedy policies, and noise are handled by sampling the robot's performance in the maze multiple times. The number of samples per maze is ten. The standard deviation can indicate both the sensitivity and the significance of results. Two controls are used. The first is a random policy in which the agent takes a random action at each timestep. The second is a random policy with a simple heuristic. The agent takes a random action that does not cause a collision into a wall. The second heuristic is used to validate that sequential state-action learning is needed to perform the tasks.

Each task's success metric is the total number of times it has reached the goal within a fixed time. The metric is chosen as it reflects the definition of intelligence used in this thesis, namely skill-acquisition with respect to priors and experience (Chollet, 2019). As the metric measures the agent while learning, the agent is incentivised to quickly converge to a reasonable policy. This metric's limitation is that the trade-off between policy optimality and convergence rate is a function of the experiment's time. The longer the time, the more valuable the final policy is. The shorter the time, the more critical the rate of convergence is. 2-layer deep Q-learning network (DQN), deep policy gradient (DPG), and advantage actor critic (A2C) methods are used to perform the first task. When both methods have successfully converged to a reasonable solution, it is used as the total time, per task, for the experiment. The methods will use unoptimised hyperparameters, so that it not bias the final result.

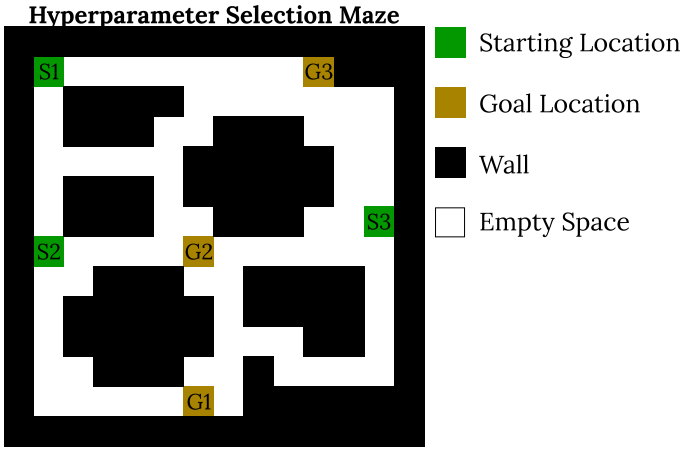

Figure 16: Maze used for optimisation of hyperparameters

Figure 16 shows the maze for hyperparameter optimisation. It is representative of the mazes shown in Figure 14. This shows a realistic methodology as the designer can estimate the type of environment the agent will be in, but not have the exact same environment. This controls for solving the environment through hyperparameter tuning, not through learnt behaviour.

Figure 17 shows the unoptimised 2-Layer Q-Learning, Policy Gradient, and Actor-Critic methods converge. All methods use the AdamW optimiser and rectified linear activation function as in the OpenAI gym (Loshchilov and Hutter, 2018; Nair and Hinton, 2010). Hyperparameters and network architectures are in Appendix C. Figure 17 shows that each method converges after around 100,000 timesteps in the maze. As a result this experiment will use 100,000 timesteps for each test. Therefore, each robot has a total lifetime of 300,000 timesteps. Each method is computed on the computer's central processing unit as the network sizes are too small to benefit from the graphics processing unit.

Methods based on classical conditioning and deep reinforcement learning are applied

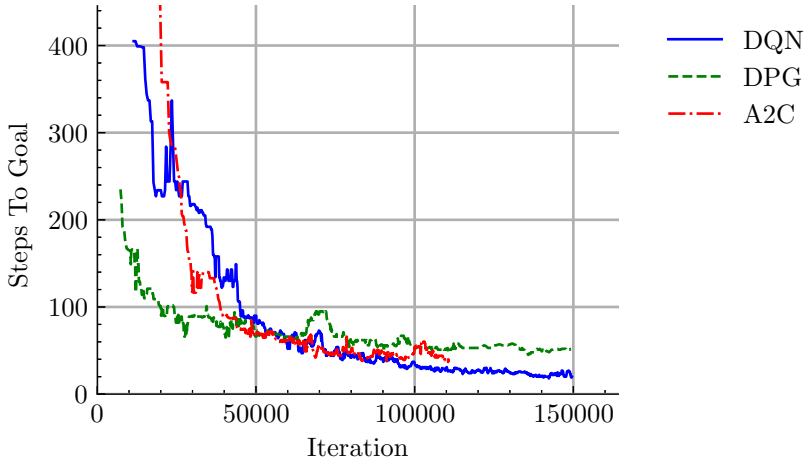

Figure 17: Amount of steps it takes the robot to find the goal per timestep. Convergence test for deep Q-learning with memory replay, policy gradient, and advantage actor critic methods for fixed start and fixed end test on the hyperparameter selection maze shown in Figure 16.

to this experiment so as to test the hypothesis that the limitations of deep learning are the same as classical conditioning. The deep reinforcement methods are DQN, DPG, and A2C. The classical conditioning methods are associative search element (ASE) and the Actor-Critic version with the adaptive critic element (ASE-ACE).

The hyperparameters are optimised, and the results are collected and averaged. The final hyperparameter values can be found in Appendix D. The total number of times the agent reached the goal is recorded and averaged between the 50 total runs per method. The average compute time for each method is recorded and averaged. The averaged standard deviation of each maze-method combination is calculated. The direct standard deviation is not used as it would indicate the variation across mazes rather than the method's sensitivity to initialisation weight and other random effects.

5.2 Results

Table 5: Results from Maze Experiment. The bolded sections indicate the best performing method for that attribute.

	Fixed Start Fixed End	**Fixed Start Moving End**	**Moving Goal Fixed Start**	**Total Across All Tasks**	
Method	Average Reward				Computation Time
Q-Learning 2-Layers	**3072 ± 386**	230 ± 208	**5021 ± 2647**	**8322 ± 2734**	13.4 mins
Policy Gradient 2-Layers	2158 ± 1801	43 ± 42	1513 ± 556	3714 ± 1990	3.3 mins
A2C 2-Layers	1991 ± 1922	21 ± 61	1247 ± 1070	3259 ± 2769	3.4 mins
Q-Learning 1-Layer	309 ± 286	76 ± 50	612 ± 213	997 ± 387	4.5 mins
Policy Gradient 1-Layer	2315 ± 853	51 ± 53	907 ± 381	3273 ± 1145	2.1 mins
A2C 1-Layer	1007 ± 740	40 ± 38	905 ± 318	1951 ± 795	2.1 mins
ASE	339 ± 334	63 ± 65	332 ± 280	734 ± 361	**0.9 mins**
ASE-ACE	546 ± 626	10 ± 28	13 ± 35	569 ± 662	2.2 mins
Random Actions	191 ± 26	137 ± 29	180 ± 25	508 ± 52	1.4 secs
Random Actions with Heuristic	318 ± 34	**232 ± 33**	309 ± 33	859 ± 62	1.2 secs

Table 5 shows the average reward of each method for each task. In the far right-hand columns, the total reward and total average computation time is recorded. Each method has ± of two standard deviations next to it. This indicates the tolerance and sensitivity of each method for each task.

Figure 18a) shows the average reward for all methods across each task. There is a clear difference between the AI models ability to perform each task. Task 1 and 3, with fixed goals, perform similarly. The robots are unable to learn task 2, where the goal location moves between three possible points. In effect, the methods are not sufficient to solve tasks that require foraging behaviour. The only method that could outperform the agents taking random actions is the DQN method with 2-layers, which achieved 93 more points than the control.

Figure 18b) shows the average reward for all methods across each maze. It demonstrates the need to control maze shape and size, as the methods were sensitive to the maze layout, where the average reward of Maze 4 is 3.9 times higher than Maze 3. This

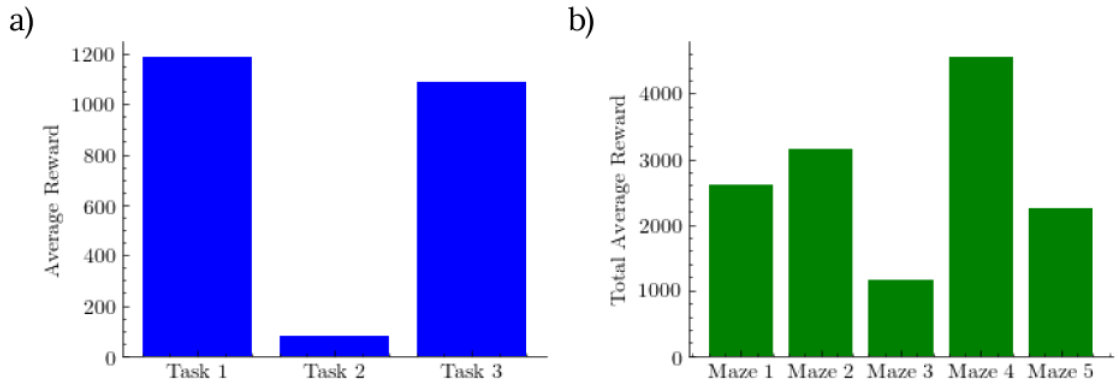

Figure 18: a) The average reward across all methods for each task. b) The average reward across all methods for each maze.

indicates a limitation in all the methods; they are highly sensitive to their environment.

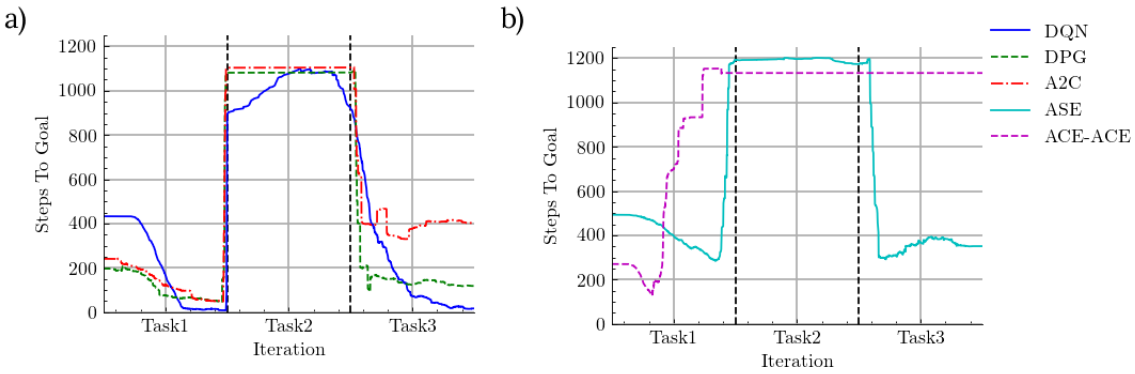

Figure 19: Amount of steps it takes the robot to find the goal per timestep for all methods. Results for 10 runs on Maze 1. The graph is filtered using a running median with a window size of 100 goals. a) Deep learning methods performance. b) Classical conditioning methods performance. The legend is applicable to both graphs.

Figure 19 shows the number of steps the agent took to get to the goal. Figure 19a) shows the performance of the deep learning methods and Figure 19b) shows the performance of the classical conditioning methods. All deep learning methods show similar performance. They quickly converge to a reasonable policy for task 1. Each method fails to learn an appropriate response for task 2. When the goal moves to another location, the agent stays around the position where the goal used to be, and as a result, does not explore the possibility that reward exists elsewhere in the maze. Once the reward returns to the original position for task 3, the agent begins to perform the task again. The agent does not appear to have problems with a moving starting position. A2C does show a drop in performance at task 3 compared to task 1. This is due to the agent sometimes becoming delusional after the task is changed. It repeatedly runs in loops or into walls. It is difficult to break as the agent is convinced that that is the optimal policy so the likelihood of sampling another action becomes small.

Figure 19b) shows the plots of the ASE and ASE-ACE methods. The ASE shows a

similar response to the deep learning methods. The ASE learns a conditioned response to going to the fixed goal. Once the goal begins moving between three positions, the method fails to learn a proper behaviour. In task 3, the agent is able to handle being restarted in different positions. However, the learning is less smooth and does not converge to a response efficiently. The ASE-ACE method behaves differently. It initially shows better performance than the ASE, but then becomes stuck, even before task 2. The reason for this is explored later. The figure shows that the classically conditioned response of the ASE is similar to the deep reinforcement learning methods. The ASE and deep learning schemes can handle a fixed goal regardless of whether the start is moved or not. However, no method can learn a reasonable policy for a goal that moves between three points.

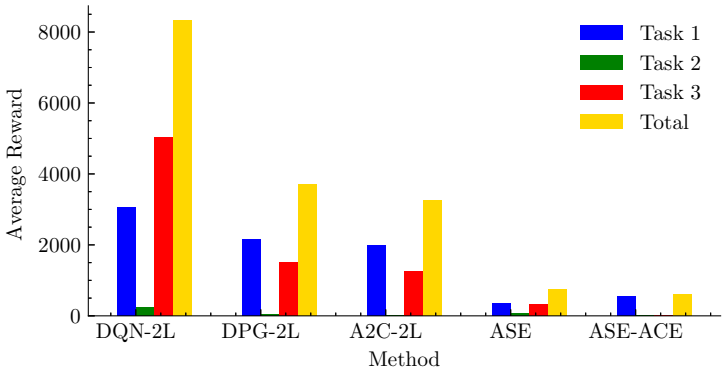

Figure 20: Task performance breakdown for the best performing networks of each method. Each bar indicates the performance for each task. The yellow bar shows the total performance across all tasks.

Figure 20 shows the average reward of some of the methods in graphical form. DPG and A2C have similar results, with worse performance in task 3 than task 1. A2C performs worse than DPG across every task, demonstrating that the addition of a value estimator does not lead to better results. ASE and ASE-ACE have a similar relationship. The ASE-ACE method is worse than just the pure ASE indicating that Actor-Critic's tend to perform worse than just policy gradient methods. Biologically plausible methods, ASE and ACE do not perform as well as deep learning methods.

Figure 21 indicates the sensitivities of the methods to each task. If the agent performed consistently, then the standard deviation is low. If the agent performed erratically, the standard deviation is high. The highest standard deviation is for the DQN method on the final task, shown in red. The results demonstrated that as the DQN method was being forced to adapt from task 2 to 3, it could become delusional and become stuck. The term delusional indicates that it becomes convinced that the action it is taking is valuable when it is not. Once again, the DPG and A2C methods have a shared attribute as they are more sensitive to task 1 than task 2. The ASE also follows this trend but has a lower standard deviation as its overall ability to perform the task is worse.

Table 6 shows the drop in performance between 2-layer and 1-layer networks. The

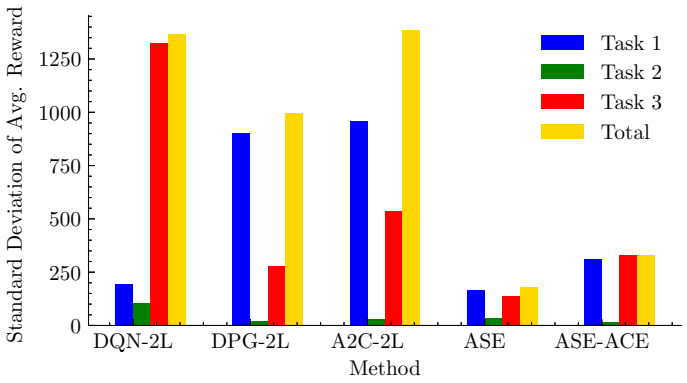

Figure 21: Standard deviation for the performing networks of each method. Each colored bar indicates the standard deviation for each task. Each bar indicates the variance for each task. The yellow bar shows the total variance across all tasks.

Table 6: Comparison between 2 and 1 Layer implementations. The values shown are the average reward across all tasks.

	2 Layers	**1 Layers**	**Difference**
DQN	8322	997	7325
DPG	3714	3273	441
A2C	3259	1951	1308
ASE	-	734	-
ASE-ACE	-	569	-

DQN method has the most significant reduction in performance, reducing from 8322 to 997 average total reward reducing to near biologically plausible methods. DPG does not suffer significantly from a reduction in network depth. However, A2C does indicate that value-based approaches require greater network depth. A2C is not as sensitive to the number of layers as DQN due to the Actor component: a modified DPG.

5.3 Discussion

The drop in performance between 2-layer and 1-layer value networks presents many questions. On the test maze in Figure 16, value estimators were trained using both Hebbian and stochastic gradient descent (SGD). After training, random actions in the maze were performed, the artificial neural network's (ANN) value estimation was recorded and averaged for each position in the maze. Heatmaps were generated from the value estimation, creating a value map.

Figure 22 shows the one layer Hebbian Critic, the ACE, has an expected error; positions that are similar to the goal have a higher value. The error is caused by the Hebbian methods being associative, like Pavlov's dog. This is the reason the ASE-ACE method fails, seen in Figure 19b). Locations which seem similar to the goal become confused with the actual goal location. It indicates a limitation with the method as it becomes

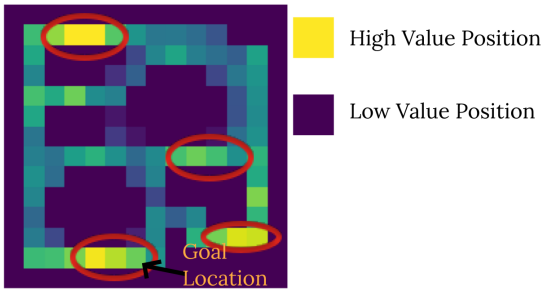

Figure 22: Error due to similarities between goal location and other parts of the maze in ACE value estimation.

easily confused between locations. This demonstrates why error feedback, like in SGD, is valuable as it will reduce these effects.

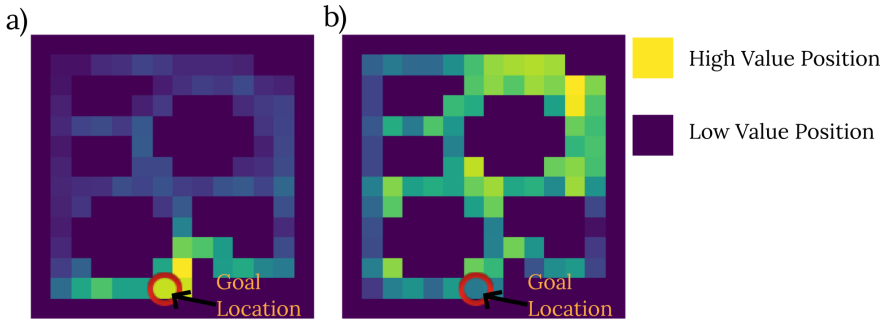

Figure 23: Average value estimated at each point in the hyperparameter maze by the Q-learning method with memory replay. The method was run only on the first test of a fixed start and fixed goal. a) Estimated value for 2 fully connected layers with batch normalisation in between. b) Estimated value for 1 fully connected layer.

Figure 23 shows a comparison between the average value estimation of a 2-Layer and a 1-Layer DQN. The 2-layer DQN correctly identifies the goal location, while the 1-layer DQN spreads value pseudo-randomly across the maze with a hotspot in the top right. Figure 23a) has a clean step of value from the starting location to the ending location. Figure 23b) has hotspots randomly located throughout the maze. The additional depth allows for a greater range of functions to be approximated, similar to the XOR problem highlighted by Minsky and Papert (2017). Depth is required to approximate certain functions. The designer must predefine the amount of network depth for the SGD algorithm and account for the computing load. Therefore, a Q-learning system requires greater human intervention to function appropriately. This makes the method lack autonomy. Figure 23 also suggests that depth could resolve the problems within the Hebbian network. No obvious extension to Hebbian learning exists for creating deeper Hebbian networks.

It is challenging to infer why policy gradient methods need less depth than value methods. A reasonable explanation is that the value network bootstraps into itself through the temporal difference equation. The value associated with each position is constantly

moving as a result. So depth would allow for abstraction of a position within the first layer then have a moving value estimate in the second. While this is plausible, it is challenging to test as there are no suitable methods to understand the workings within a deep neural network. This is a limitation of blackbox function approximators such as deep neural networks. The area of explainable deep learning is currently an new area of research (Choo and Liu, 2018).

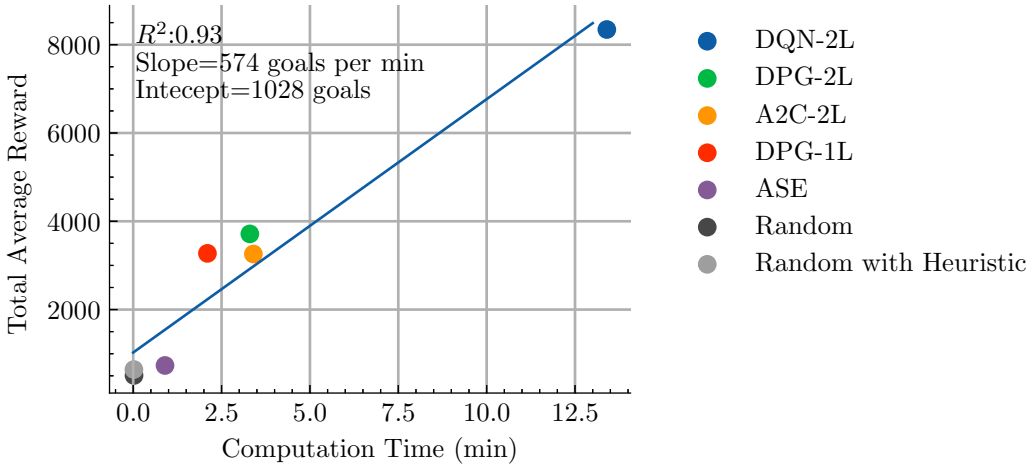

Figure 24: Relationship between the total number of times the robot gets to the goal and the time it takes for the program to run.

Figure 24 shows a plot of the methods total average reward with the amount of computation time. 1-layer value methods are omitted as the ANN's are not suitable for the method. The line shows a good fit to the data with an R2 of 0.93. The graphs suggest that the methods are purchasing skill with computational power, indicating that the methods do not engage in a different sort of learning. The experimental setup controlled for the skill-acquisition efficiency of the total amount of timesteps the robot had. Instead, the best methods resulted in the ones that were best able to use the computational resources. As the methods are buying skill with computational power, it demonstrates the lack of intelligence within each system. If the framework improves as more resources are given to it, the solution is brute-forced, not solved using clever and novel learning mechanisms.

The goal of this experiment was to test the hypothesis that *classical conditioning and deep reinforcement learning share limitations for task performance, adaptability, and robot navigation.* The results showed that all methods failed on the second task indicating that both classical conditioning and deep reinforcement learning are unable to handle moving targets. As the robots have certain behaviours reinforced, they will repeat the behaviour routinely unless given a negative reward. The inability to adapt is intrinsic to classical conditioning and MDPs, no simple or easy method will rectify this.

Actor-Critic architectures of A2C and ASE-ACE perform worse than the pure Actor methods, DPG and ASE. Actor-Critic architectures across both classical conditioning and MDPs have been shown to share similar limitations. The root reason for the problem

is that the Critic must be trained before the Actor, creating a bottleneck. The skill-acquisition efficiency is, therefore, lower as two networks need to be trained.

A limit in this experiment is that the behaviour of 2-layer deep Hebbian networks cannot be tested. This is because there is no conclusive mathematical framework for deep biologically plausible neural networks. Despite this, the inability of 1-layer value networks to correctly estimate the discounted value is shared by both methods. When methods for deep Hebbian networks are developed, this should be revisited. The results also differ significantly in the ability to learn the task. SGD optimisation outperforms Hebbian learning, indicating that the current biologically plausible learning theory might need to consider how neurons could implement error feedback.

The results indicate that the methods share similar limitations on tasks with moving goals, Actor-Critic architectures, and value estimations. The results agree with the similarity in mathematics shown in Section 2.5. The reframing of deep reinforcement learning as approximating classical conditioning allows for a better understanding of how the method will behave in various circumstances. Therefore, deep learning with MDP's cannot be autonomous AI as they learn conditioned responses. It also explains how it can be developed to solve more complex, adaptive problems as the method can be understood in a non-mathematical sense.

6 Future work

The results from this thesis lead to many possible avenues of future work. The experimental setup of a battery of computationally inexpensive tasks is fantastic for this application. The general experimental framework encourages robust adaptive robots. Two research avenues are worth pursuing: the first is learning without a classically conditioned or Markov decision process framework. The second is improving classical conditioning methods by unlearning behaviour.

To solve the moving goal task, planning and self-identification of the goal are paramount. Research into methods which can learn the general task, then plan what actions it needs to take. Whatever method used is likely to suffer from long computing times as planning involves branching possibilities, leading to a high number of possible combinations and events. Classical conditioning of behavioural sequences could mitigate the expense. If constrained to short, simple sequences, deep learning and conditioned responses are efficient.

Conditioned responses to subsets of the agent's behaviour is a reasonable approach. Learning to open a door, swing a racket, or move a wheel are all conditioned responses. They happen quickly, easily and without thought or planning. They are mainly learnt in an unsupervised way, based upon the novelty or utility of the behaviour. They can also be unlearnt. Research into how Actor methods like associative search elements and deep

policy gradient can unlearn a behaviour if the expected reward or behaviour does not occur would allow for greater adaptability. So when the goal is moved, the agent unlearns the behaviour to go straight to the goal, as the behaviour is now unlearnt exploration of the environment can then increase. The problems demonstrated in this thesis will be solved with research into planning, self-identification of the goal, and unlearning behaviour.

7 Conclusion

This work aimed to review and find the limitations of autonomous artificial intelligence (AI) systems for robots. The problem of autonomous AI for robotics was broken down into definitions. The acquisition of skill was identified as the core of an intelligent agent. This meant that learning was paramount to any effective AI solution. Deep reinforcement learning is the current state-of-the-art for problems related to autonomous learning. The number of possible permutations of networks, optimisers, and hyperparameters leads to a method which is heavily reliant on designer intervention. While deep reinforcement learning has desirable attributes, like being a universal function approximator, the Markov decision process (MDP) framework shows similarities to classical conditioning. If MDP's are related to conditioned responses, then they are not autonomous. A study on the attributes of deep reinforcement learning was first conducted.

Three OpenAI gym environments: Lunar Lander, Cart Pole, and Pong, were used in the study. The OpenAI gym showed that deep learning could be applied to a range of tasks, indicating the methods are successful for different action, state, and reward spaces. The Pong environment showed that a dense sampling of the player's experience is needed to learn more complex tasks. This demonstrated why reinforcement learning applied to StarCraft 2 required 200 years worth of playtime (Vinyals et al., 2019). Pong took three days to train, indicating a limitation of deep learning for AI that must be embedded into robotics. The study led to the conclusion that to test the relationship between classical conditioning and deep reinforcement learning, a well-controlled and computationally inexpensive task was needed.

A partially observable maze experiment was set up for a robotic agent. The experiment's goal was to see if the Markov decision process shared the same limitations as classical conditioning in task performance, adaptability, and robot navigation. Three tasks were given to the robot: first with a fixed start and fixed goal, the second with a fixed start and a moving goal, the third with a moving start and a moving goal. If the Markov decision process failed the second task and passed the third, it would demonstrate that it shares the limitations of conditioned responses.

The result demonstrated that the relationship between classical conditioning and Markov decision processes are apparent in the agent's behaviour. Conditioned responses do not prevent an agent from being autonomous if it forms a part of a more intelligent

system. However, by itself, it is unable to create autonomous artificial intelligence. The methods fail on tasks that require abstraction, planning, and self-identification of the goal.

An analysis of the computation times shows that the skill across all methods is proportional to the time to compute, indicating that the methods are purchasing skill with computing power. Given the interpretation of the Markov decision process as a form of classical conditioning, a biological understanding can be inferred. The best methods find improvement by repeating past experiences more frequently, strengthening the conditioned response. The methods are not utilising smarted learning mechanisms to solve the problem.

While the results do indicate the relationship between classical conditioning and deep learning, the experiment had a limitation. Biologically plausible Hebbian learning currently has no good theory for developing deep neural networks, unlike stochastic gradient descent. Therefore, a comparison between deep Hebbian and deep learning networks could not be made. A future possibility could be to research multi-layer Hebbian networks' methods and compare them to deep learning.

Future work into methods that can solve tasks with more complex goals is needed. This will be solved using planning mechanisms. Conditioned responses, like MDP policies, are computationally inefficient when applied to small tasks. Therefore, conditioned responses will likely continue to have a valuable place in training robots. However, unlearning a response needs to be researched as current methods do not update the robot's behaviour in changing environments.

Understanding the relationship between Markov decision processes leads to future research into the areas of planning, self-identification of the goal, and unlearning conditioned responses. By solving these follow-up problems, major advances in many fields can take place. Such as greater task automation, usage of companion robots for the elderly, and more efficient agricultural equipment.

8 Bibliography

Youssef Fenjiro and Houda Benbrahim. Deep reinforcement learning overview of the state of the art. *Journal of Automation Mobile Robotics and Intelligent Systems*, 12, 2018.

Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

Ayse Pinar Saygin, Ilyas Cicekli, et al. Turing test: 50 years later. *Minds and machines*, 10(4):463–518, 2000.

John R Searle. Minds, brains, and programs. *The Turing Test: Verbal Behaviour as the Hallmark of Intelligence*, pages 201–224, 1980.

Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

Shane Legg, Marcus Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

José Hernández-Orallo. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. *Artificial Intelligence Review*, 48(3):397–447, 2017.

François Chollet. On the measure of intelligence. *CoRR*, abs/1911.01547, 2019. URL `http://arxiv.org/abs/1911.01547`.

Pei Wang. On defining artificial intelligence. *Journal of Artificial General Intelligence*, 10(2):1–37, 2019.

Tim Smitherst. Taking eliminative materialism seriously: A methodology for autonomous. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, page 31. MIT Press, 1992.

Rolf Pfeifer and Christian Scheier. *Design Principles of Autonomous Agents*, pages 299–326. IEEE, 2001.

Jeffrey M Bradshaw, Robert R Hoffman, et al. The seven deadly myths of" autonomous systems". *IEEE Intelligent Systems*, 28(3):54–61, 2013.

Chinedu Pascal Ezenkwu and Andrew Starkey. Machine autonomy: Definition, approaches, challenges and research gaps. In *Intelligent Computing-Proceedings of the Computing Conference*, pages 335–358. Springer, 2019.

James Kennedy. Swarm intelligence. In *Handbook of nature-inspired and innovative computing*, pages 187–219. Springer, 2006.

Jan Marco Leimeister. Collective intelligence. *Business & Information Systems Engineering*, 2(4):245–248, 2010.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1):153–197, 1990.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

David E Rumelhart, Geoffrey E Hinton, et al. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143195, 1999. doi: 10.1017/S0962492900002919.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Yann LeCun, Bernhard Boser, et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Ilya Sutskever, James Martens, et al. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam, 2018. URL `https://openreview.net/forum?id=rk6qdGgCZ`.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

Geoffrey E Hinton, Nitish Srivastava, et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Jimmy Lei Ba, Jamie Ryan Kiros, et al. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, et al. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Yann LeCun, Yoshua Bengio, et al. Deep learning. *nature*, 521(7553):436–444, 2015.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

Ivan Petrovitch Pavlov and William Gantt. *Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals*. Liverwright Publishing Corporation, 1928.

A Dickinson and NJ Mackintosh. Classical conditioning in animals. *Annual review of psychology*, 29(1):587–612, 1978.

Kaiwen He, Marco Huertas, et al. Distinct eligibility traces for ltp and ltd in cortical synapses. *Neuron*, 88(3):528–538, 2015.

Mark S Szczypka, Mark A Rainey, et al. Feeding behavior in dopamine-deficient mice. *Proceedings of the National Academy of Sciences*, 96(21):12138–12143, 1999.

Andrew G Barto and Richard S Sutton. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4(3):221–235, 1982.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Vincent François-Lavet, Peter Henderson, et al. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Wulfram Gerstner, Marco Lehmann, et al. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

David Silver, Julian Schrittwieser, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Oriol Vinyals, Igor Babuschkin, et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, King's College, Cambridge United Kingdom, 1989.

Arthur Stanley Eddington. *The mathematical theory of relativity*. The University Press, 1923.

Volodymyr Mnih, Koray Kavukcuoglu, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Hado Van Hasselt, Arthur Guez, et al. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Richard S Sutton, David A McAllester, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

Ian Goodfellow, Yoshua Bengio, et al. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Volodymyr Mnih, Adria Puigdomenech Badia, et al. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Greg Brockman, Vicki Cheung, et al. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Prafulla Dhariwal, Christopher Hesse, et al. Openai baselines. `https://github.com/openai/baselines`, 2017.

Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

William Huitt and John Hummel. Piaget's theory of cognitive development. *Educational psychology interactive*, 3(2):1–5, 2003.

Frans Oliehoek. *Value-based planning for teams of agents in stochastic partially observable environments*. Amsterdam University Press, 2010.

Michael L Littman, Anthony R Cassandra, et al. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

George E Monahan. State of the arta survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

Jaegul Choo and Shixia Liu. Visual analytics for explainable deep learning. *IEEE computer graphics and applications*, 38(4):84–92, 2018.

Appendices

A Algorithms

A.1 Associative Search Element Psudo-code

Algorithm 1: Associative Search Element

Initialise weights randomly

for *t=1,T* **do**

- Calculate $y(t)$ from environmental inputs $x(t)$
- Select $a(t) = \arg\max y(t)$
- Execute action $a(t)$ in environment and receive reward $R(t)$ and input $x(t+1)$
- Update eligibility trace $e(t+1) = \delta e(t) + (1-\delta)(y(t)x(t) - y^2(t) \cdot W_{ASE})$
- Update ASE weights using environmental reward

end

A.2 Adaptive Critic Element Psudo-code

Algorithm 2: Associative Search Element and Adaptive Critic Element

Initialise weights randomly

for *t=1,T* **do**

- Calculate $y(t)$ from environmental inputs $x(t)$
- Select $a(t) = \arg\max y(t)$
- Execute action $a(t)$ in environment and receive reward $R(t)$ and input $x(t+1)$
- Calculate internal reward $\hat{r}(t)$
- Update eligibility trace $e(t+1) = \delta e(t) + (1-\delta)(y(t)x(t) - y^2(t) \cdot W_{ASE})$
- Update ACE weights using temporal difference
- Update ASE weights using internal reward $\hat{r}(t)$

end

A.3 Double Deep Q-Learning Psudo-code

Algorithm 3: Double Deep Q-Learning with Experience Replay

Initialise replay memory buffer $\mathcal{M}$ to maximum size N

Initialise weights randomly for Target Network and Prediction Network

for *t=1,T* **do**

- With probability ϵ select a random action a_t
- if not select deterministic action $a_t = \max(Q_\pi(s_t, a))$
- Decay ϵ till ϵ_{min}
- Get random mini-batch from $\mathcal{M}$
- Get target value $Q'_{t+1} = \max_a(Q'_\pi(s_{t+1}, a))$ from Target Network
- Get prediction value $Q_t = Q_\pi(s, a)$ from Prediction Network
- Calculate loss $L = |R_t + \gamma Q'_{t+1} - Q_t|$ and update weights
- Send action a_t to environment
- Recieve reward R_t and new state s_{t+1}
- Push tuple (s_t, a_t, R_t, s_{t+1}) to memory replay $\mathcal{M}$
- Every n-steps transfer Prediction Network parameters to Target Network

end

A.4 Policy Gradient Psudo-code

Algorithm 4: Deep Policy Gradient

Initialise Actor and Critic weights randomly

for *episode = 1, E* **do**

- **for** *t=1,T* **do**
  - Get action a_t from policy π_θ
  - Store probability of action $\ln(\pi_\theta(a|s))$
  - Send action a_t to environment
  - Recieve reward R_t and new state S_t
  - Store the temporal difference R_t
- **end**
- Compute G_t for each time step t in episode
- Calculate the loss $L = -G_t \ln(\pi_\theta(a|s))$
- Update weights

end

A.5 Advantage Actor Critic Psudo-code

Algorithm 5: A2C

Initialise Actor and Critic weights randomly
for *episode = 1, E* **do**
  for *t=1,T* **do**
    Get action a_t from policy π_θ
    Store probability of action $\ln(\pi_\theta(a|s))$
    Send action a_t to environment
    Recieve reward R_t and new state S_t
    Compute the temporal difference $\delta_t = R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s)$
    Store the temporal difference δ_t
    Update value function estimation weights with $L = |R_t + \gamma v_\pi(s_{t+1}) - v_\pi(s)|$
  end
  Calculate the loss $L = -\delta_t \ln(\pi_\theta(a|s))$
  Update weights
end

B Hyperparameter Values for OpenAI Gym

B.1 Lunar Lander

Deep Q-Learning Network:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.001
Memory Size	200,000
Update Frequency	1000
Batch Size	128
γ	0.99
ϵ_{min}	0.1
ϵ decay rate	$\frac{1}{40{,}000}$
Hidden Layer Width	128
Number of Hidden Layers	2
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	None

Deep Policy Gradient:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0
γ	0.99
Hidden Layer Width	128
Number of Hidden Layers	1
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

Advantage Actor Critic:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.01
γ	0.99
Hidden Layer Width	128
Number of Hidden Layers	1
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

B.2 Cart Pole

Deep Q-Learning Network:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.00001
Memory Size	20,000
Update Frequency	500
Batch Size	512
γ	0.99
ϵ_{min}	0.05
ϵ decay rate	$\frac{1}{20{,}000}$
Hidden Layer Width	50
Number of Hidden Layers	1
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	None

Deep Policy Gradient:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0
γ	0.99
Hidden Layer Width	128
Number of Hidden Layers	1
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

Advantage Actor Critic:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.01
γ	0.99
Hidden Layer Width	128
Number of Hidden Layers	1
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

B.3 Pong

Deep Q-Learning Network:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0
Memory Size	100,000
Update Frequency	100
Batch Size	32
γ	0.99
ϵ_{min}	0.1
ϵ decay rate	$\frac{1}{7{,}500{,}000}$
Number Convolutional Filters Layer-1	16
Filter Size Layer-1	8
Stride Layer-1	8
Number Convolutional Filters Layer-2	32
Filter Size Layer-2	4
Stride Layer-2	4
Linear Layer-3	256
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	None

Deep Policy Gradient:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0
γ	0.99
Number Convolutional Filters Layer-1	16
Filter Size Layer-1	8
Stride Layer-1	8
Number Convolutional Filters Layer-2	32
Filter Size Layer-2	4
Stride Layer-2	4
Linear Layer-3	256
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

Advantage Actor Critic:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0
γ	0.99
Number Convolutional Filters Layer-1	16
Filter Size Layer-1	8
Stride Layer-1	8
Number Convolutional Filters Layer-2	32
Filter Size Layer-2	4
Stride Layer-2	4
Linear Layer-3	256
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax

C Convergence Hyperparameters

Deep Q-Learning Network:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.01
Memory Size	10,000
Update Frequency	1
Batch Size	64
γ	0.99
ϵ_{min}	0.05
ϵ decay rate	$\frac{1}{150,000}$
Hidden Layer Width	100
Number of Hidden Layers	1
Normalisation	BatchNorm
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	None
k-previous inputs	12

Deep Policy Gradient:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.01
γ	0.99
Hidden Layer Width	100
Number of Hidden Layers	1
Dropout	0.6
Normalisation	LayerNorm
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax
k-previous inputs	12

Advantage Actor Critic:

Hyperparameter	Value
Learning Rate	0.001
Weight Decay	0.01
γ	0.99
Hidden Layer Width	100
Number of Hidden Layers	1
Normalisation	LayerNorm
Optimiser	AdamW
Hidden Layer Activation Function	ReLu
Output Layer Activation Function	SoftMax
k-previous inputs	12

D Final Hyperparameters for Maze

Deep Q-Learning Network:

Hyperparameter	2-Layer Value	1-Layer Value
Learning Rate	0.0001	0.01
Weight Decay	0.01	0.01
Memory Size	10,000	10,000
Update Frequency	1	1
Batch Size	64	64
γ	0.99	0.99
ϵ_{min}	0.05	0.05
ϵ decay rate	$\frac{1}{100{,}000}$	$\frac{1}{100{,}000}$
Hidden Layer Width	100	N/A
Number of Hidden Layers	1	0
Normalisation	BatchNorm	N/A
Optimiser	AdamW	AdamW
Hidden Layer Activation Function	ReLu	N/A
Output Layer Activation Function	None	None
k-previous inputs	8	20

Deep Policy Gradient:

Hyperparameter	2-Layer Value	1-Layer Value
Learning Rate	0.001	0.01
Weight Decay	0.01	0.01
γ	0.99	0.99
Hidden Layer Width	100	N/A
Number of Hidden Layers	1	0
Dropout	0	0
Normalisation	LayerNorm	N/A
Optimiser	AdamW	AdamW
Hidden Layer Activation Function	ReLu	N/A
Output Layer Activation Function	SoftMax	SoftMax
k-previous inputs	16	64
Episode Length	1500	1000

Advantage Actor Critic:

Hyperparameter	2-Layer Value	1-Layer Value
Learning Rate	0.001	0.01
Weight Decay	0.01	0.01
γ	0.99	0.99
Hidden Layer Width	100	N/A
Number of Hidden Layers	1	0
Normalisation	LayerNorm	N/A
Optimiser	AdamW	AdamW
Hidden Layer Activation Function	ReLu	N/A
Output Layer Activation Function	SoftMax	SoftMax
k-previous inputs	6	8
Episode Length	500	1000

Associative Search Element:

Hyperparameter	Value
Learning Rate	0.001
Eligibility Trace	0.9
Neural Noise Standard Deviation	0.5
k-previous inputs	32

Associative Search Element and Adaptive Critic Element:

Hyperparameter	Value
Learning Rate	0.001
γ	0.99
Eligibility Trace	0.9
Input Trace	0.7
Neural Noise Standard Deviation	0.5
k-previous inputs	32

E Code

All code is available online at the authors GitHub repository.

`https://github.com/AidanBelton/Deep_Reinforcement_Learning_Robotics`