THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Deep Reinforcement Learning for Partially-Observable Markov Decision Processes

Cameron Gordon 42370057

Supervisor: Dr Nan Ye

A thesis submitted for the degree of Masters of Science at

The University of Queensland in 2020

School of Mathematics and Physics

1 Abstract

This thesis has investigated the use of Deep Reinforcement Learning for classic Partially-Observable Markov Decision Process problems such as Tiger, Rock Sample, and Tag. It has examined a number of deep reinforcement learning algorithms and techniques found within recent literature. This has included testing Deep Q-Networks (DQN), Deep Recurrent Q-Networks (DRQN), Action-specific Deep Recurrent Q-Networks (ADRQN), the use of 'flooding' regularisation described in Ishida et al. (2020), the use of pre-initialisation of expert experiences, and the use of prioritised experience replay described in Schaul et al. (2015).

The contributions of the thesis include development of a Python POMDPX parser and simulator to enable the use of deep learning libraries such as Keras on POMDPX problems, and a POMDPX to OpenAI Gym environment converter. An extension to ADRQN is proposed in which step rewards are included with the input of observation-action history, termed RADRQN. Benchmark comparison is made to leading tree-based POMDP solvers such as DESPOT found in Ye et al. (2017). Of the tested models, the combination of pre-initialisation of expert experiences and prioritised experience replay produces the best results for the deep reinforcement learning models. Trained policies involving multiple step behaviours are observed for each of the tested POMDP problems, however results are well below the performance of benchmark algorithms such as DESPOT, POMCP, or SARSOP. Trained policies show path dependence on experiences encountered early in the training cycle.

All code and data used in this project is made available at https://github.com/iciac/POMDP.

Contents

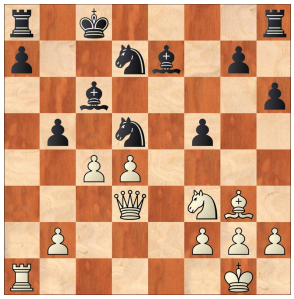
List of Figures

List of Tables

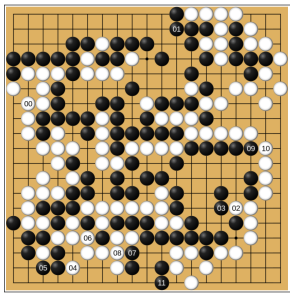
List of Algorithms

2 Introduction

Partially-observable environments comprise the majority of situations that a decision-making agent must contend with in the world [1, 2]. Robotic measurements may be faulty or unknown. Humans must navigate an epistemically uncertain world from birth. Businesses must react decisively on the basis of limited market intelligence. How to handle this limited observability is both a practical question, and one of great contemporary interest to artificial intelligence [3]. Historical benchmarks for artificial intelligence problems include Chess, Checkers, Go, Othello, and Backgammon [4]. More recently benchmarks have shifted to include video games, such as Atari, Doom, and Starcraft [5, 6, 7]. This shift has in part been driven by a desire to model more complicated domains, and to examine artificial agents contending with difficulties introduced by partial-observability.



(a) Kasparov v Deep Blue game 6 final position, 1997. Credit: The Future of Things



(b) Lee Sedol v AlphaGo game 2 final position, 2016. Credit: Wikipedia

(c) Breakout, a key game in the Atari2600 Arcade Learning Environment.

Figure 1: Historical benchmark problems

Recent advances in Deep Reinforcement Learning (DRL), which combines deep artificial neural networks with techniques found in reinforcement learning have led to tremendous results on key benchmark problems. However, current DRL methods generally deal with a fully observable environment or use a fully observable approximation. This motivates the current thesis, which seeks to examine how DRL algorithms perform on problems framed as Partially Observable Markov Decision Processes (POMDPs), including those specifically developed for POMDPs such as Deep Recurrent Q-Networks (DRQN) [8]. In particular, it is intended to examine DRL performance on classic POMDP problems such as Rock Sample in order to compare these results to performance obtained from non-neural POMDP solvers such as DESPOT and POMCP which use tree-based simulations [9, 10].

Our experiments show that a simple neural network is able to produce policies comprising complex multi-step behaviour on POMDP problems. On the Tiger problem performance is comparable to tree-based solvers in some instances. On larger POMDP problems such as Rock Sample and Tag, the best performance obtained by our models is still well below com-

parable benchmarks. Policy performance is inconsistently obtained over repeated training cycles, indicating the presence of local optima and path dependence. Through this thesis we have examined a number of recent DRL algorithms and techniques noted for improved performance on both fully and partially observed problems. Best performance is obtained for models incorporating additional information from step actions and rewards, prioritised experience replay, and for those initialised with demonstrated policies from expert algorithms.

This thesis is structured as follows: first, a mathematical description of POMDPs is provided. Secondly, an overview of deep reinforcement learning is provided, including a description of key algorithms and techniques examined in this thesis. Third, we provide a description of the POMDPX parser and simulator developed for this thesis, along with model extensions such as expert buffers, flooding regularisation, and reward concatenation. Results are included from 1,100 trained policies obtained across the Tiger, Tag, and Rock Sample problems. Next we compare the performance of our models to tree-based planning algorithms. Finally, we discuss potential explanations for our results and possible extensions to our experiments including a POMDPX to OpenAI Gym converter, different exploration strategies, and combination solvers using both planning and deep reinforcement learning techniques.

3 Literature Review

3.1 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process is a generalisation of a Markov Decision Process (MDP) in which the agent cannot directly observe the state of the world, and instead must make decisions on the basis of an observation (Kaelbling et al. (1998)) [2]. This uncertainty presents computational difficulties for many planning algorithms [3]. In this section we provide a mathematical description of POMDPs.

3.1.1 Mathematical Definition

A Partially Observable Markov Decision Process (POMDP) definition is provided in Kaelbling et al. (1998), which builds upon previous descriptions in the operations research literature (e.g. Åström (1965) who examined control problems with incomplete state information) [2, 1]. We follow Kaelbling et al. (1998) closely in our description of a POMDP, which is defined as the tuple $<S, A, R, T, O, \Omega>$:

- S : State set of the environment
- A : Action set of the agent
- Ω : Observation set of the environment
- $R : S \times A \to \mathbb{R}$ - Reward given for an action a in state s
- $T : S \times A \to \Pi(S)$ - State-transition function. $T(s, a, s')$ is the transition probability of moving from state s to state s' following an action a. $\Pi(S)$ represents the set of all probability distributions across world states.
- $O : S \times A \to \Pi(\Omega)$ - Observation function. $O(s', a, o)$ is the probability that o will be observed when an action a leads to state s'. $\Pi(\Omega)$ represents the set of all probability distributions across observations.

A discount factor $\gamma \in [0, 1]$ and planning horizon H is often provided in POMDP descriptions. A MDP is defined similarly to a POMDP as the tuple $<S, A, R, T>$. While a MDP is fully-observable, a POMDP only receives indirect information on the state from the observation. Hence, the agent must develop a belief $b(s)$ on the state of the environment. We can define the belief space $\Pi(S)$ as the set of all possible probability distributions over the state space. Note that while the state space S may be discrete and finite, the belief space is infinite except when S contains only one element. Figure 2 shows a POMDP as a decision network. A policy π_t assigns an action to a belief at time t. A direct mapping may be prohibitively expensive to compute, which is a principal difficulty encountered in POMDP problems.

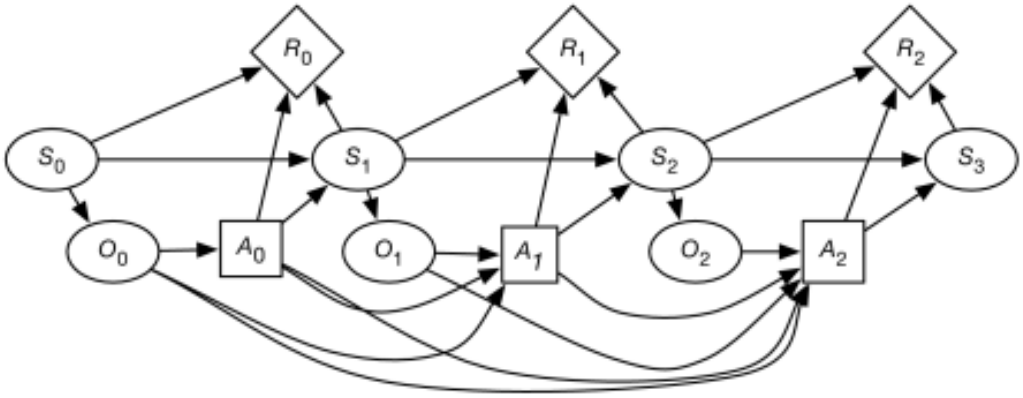


Figure 2: POMDP Decision Network. The true state S_t of the network is hidden to the agent, which instead observes O_t and takes action A_t, leading to reward R_t and state S_{t+1} Source: Poole and Mackworth (2017) [11]

The objective function for the agent is to maximise the total expected discounted reward across the duration of the problem. Specifically, we want to find a policy π maximising: $E\left[\sum_{t=0}^{H} \gamma^t r_t\right]$ [2].

Kaelbling et al. (1998) notes that $b(s)$ is a sufficient statistic for the past history of the problem. Using this property the problem can be reformulated as a *belief MDP* which retains the Markov property of being memoryless. As with MDPs this reformulation may be used to calculate an optimal policy π^* by applying simple dynamic programming techniques such as value iteration. Value iteration for a fully-observable MDP involves determining an optimal value $V_t^*(s)$ for each state, which is the maximum expected total discounted reward that can be achieved from s when planning for a horizon of t. The optimal policy is defined by:

$$\pi_t^*(s) = \text{argmax}_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}^*(s') \right] \tag{1}$$

We include the value iteration algorithm (Algorithm 1) from Kaelbling et al. (1998) as an example of a known optimal policy for MDPs, equivalent to exhaustive search. This is polynomial with respect to both A and S [3]. The dimensionality of the action, state, and belief state make calculating an optimal policy impractical in practice. This issue is referred to as the *Curse of Dimensionality* [3], and motivates the use of a wide class of POMDP solvers, including the Deep Reinforcement Learning techniques examined in this thesis.

3.1.2 Exploration-Exploitation Trade-Off

Many convergence results for solving POMDPs using reinforcement learning algorithms require that each state-action pair is sampled an infinite number of times in the limit [3]. As a result, practical solution techniques for POMDPs face an exploration-exploitation trade-off,

Algorithm 1: Value Iteration (Source: Kaelbling et al. 1998) [2])

Set $V_0(s) = 0$ for all s and set small ε;
for *all* t **do**
  for *all* $s \in S$ **do**
    for *all* $a \in A$ **do**
      $Q_t^a(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}(s')$
    end
    $V_t(s) = \max_a Q_t^a(s)$
  end
  until $|V_t(s) - V_{t-1}(s)| < \varepsilon$ for all $s \in S$
end

in which the solver must trade between exploiting a currently perceived optimal solution and exploring for improvements. The overarching goal of these methods is to balance this convergence with computational limitations. This issue is well described in the literature (see Sutton and Barto (2018)), so we will limit this section to outlining two strategies for handling the issue as applied in deep reinforcement learning [3].

The first is the ε-greedy method, which addresses the issue by introducing random action selection. Define $Q_t(a)$ as an estimated value for taking action a at time t. The *greedy* action is given by: $A_t = \underset{a}{\text{argmax}}\ Q_t(a)$. ε-greedy selects a random action with probability small ε, and the greedy action with probability $(1-\varepsilon)$. So long as $\varepsilon > 0$, each action will be sampled sufficiently in the limit. Sutton and Barto (2018) cite Watkins (1989) as a potential first usage, however the technique is likely older [3]. The simplicity and effectiveness of ε-greedy has led to its extensive use in reinforcement learning algorithms (see Algorithm 2: Q-Learning, Algorithm 3: Deep Q-Networks) [12, 3]. Annealing methods which start with a high ε term that is progressively decreased over training are also common [4]. This will lead to greater random exploration in earlier phases of training.

The second method is to directly control the exploration parameter through control mechanisms. This may involve multiple levels of exploration. An example of this is found in Badia et al. (2020), which uses a meta-controller to vary exploration terms depending on the specifics of a problem. At the problem level exploration is determined by an exploration-discount rate pair. For each problem iteration this pair is selected by a meta-controller, which explores different parameter settings by itself following an ε-greedy selection algorithm. This technique is an area of active enquiry, and its use led to above human-level performance across a key benchmark of 57 Atari Learning Environment problems in Badia et al. (2020). This suite involved environments with varied characteristics, including sparse rewards and

long-term credit assignment issues, requiring exploration to be varied for the problem level [13].

3.1.3 Types of POMDPs

The generality of POMDPs for defining problems is well known [14]. This has led to focus on specific subtypes of POMDPs, combining congruent fields of research. Here we outline a few of the important variants. Worth noting is that the distinction between these variants is not strict and each may be considered a case of the more general POMDP definition.

	Single Agent	Multi-Agent
General Case	POMDP	Partially-Observable Stochastic Game
Fully-Observed	MDP	Extensive Form Game
Mixed Observability	MOMDP / SMDP	Factored Observation Game

Table 1: Common descriptions of POMDPs

A special factored form of the POMDP introduced in Ong et al. (2009) separates fully and partially observable state-space components in a *Mixed Observability Markov Decision Process* (MOMDP) [15]. Separating these components into disjoint subspaces enables a more compact representation of the belief-space. A system with fully-observed parameters is rarely encountered in reality, so the authors additionally introduce the concept of *Pseudo Full Observability* which treats certain partially-observable parameters as if fully-observable. This decomposition to varying components of certainty has similarities to the idea of *Semi-Markov Decision Processes* (SMDP) found in Lane et al. (2002) [16].

It is possible to redefine a multi-agent game (often defined as an *Extensive Form Game*) as a single agent POMDP under certain conditions (the intuition is that moves from the opposing N players are drawn from an unobserved probability distribution) [17]. The OpenSpiel testing environment developed by DeepMind is embedded within this framework [18]. Parallel lines of research in game theory additionally enabled the convergence of Stochastic Games and POMDPs to Partially-Observable Stochastic Games (POSG), while further extending this approach to a factored representation has resulted in Factored Observation Games (FOG) [19, 20].

Finally, it is worth noting that fully-observable problems with an initial hidden state variable are classed as POMDPs. As Whiteson et al. (2011) note, drawing the initial state of a specific MDP environment M from a generalised environment $G =< \Theta, \mu >$ (where Θ is a set of environments, μ defines a distribution across the environment parameters, and $\theta \in \Theta$ represents the parameters of M) means that the generalised environment G will be a POMDP even if the specific MDP M is fully-observable. In this situation, the specific draw

θ represents a hidden state factor defining the initial state [21]. This has implications for testing suites such as OpenAI Gym, which randomises starting conditions for fully-observed problem instances. All problems in the suite may be considered POMDPs [22].

3.2 Algorithms for POMDPs

3.2.1 Overview

Recent progress in algorithms for both partially and fully-observable problems has been rapid, driven by a combination of a increases to computing power and breakthroughs across multiple fields. Interest in Deep Reinforcement Learning in particular has been aided by high-profile successes in benchmark problems including AlphaGo [23], the self-playing AlphaZero [24], and AlphaStar [5].

Important advances have resulted from the combination of reinforcement learning (RL), and deep artificial neural networks (ANNs). RL has been investigated from early days in operations research [1], and has roots in psychological research (e.g. Pavlovian responses and Skinner Boxes) and neuroscience [3]. Algorithms combining RL with ANNs, such as TD-Learning led to early breakthroughs in artificial intelligence such as TD-Gammon (the first program to play at a human level in Backgammon) [25, 3, 26]. In following sections we'll first provide an overview of ANNs and deep learning, followed by key ideas within reinforcement learning, and finally the combined field of Deep Reinforcement Learning (DRL).

3.2.2 Deep Neural Networks

The use of artificial neural networks is based on the structure of neuronal cells. The basic structure of a neuron involves a cell body (the soma), dendrites, synapses, and an axon. Neurotransmitters released between synapses change the internal electrical charge within the soma. Once this shifts from the resting membrane potential of -70mV to a critical value of around -55mV to -50mV, gated ion channels within the soma open enabling an all-or-nothing action potential to be released through the axon [27, 28]. Dendrites enable the connection of a neuron to the axons of a large number of other neurons (in humans, the average neuron has approx. 7,000 synaptic connections) [28]. Artificial neurons attempt to replicate this structure.

While various models of neuronal firing exist in the literature, (including the differential equation based Hodgkin-Huxley model) the model most commonly applied in ANNs is closely related to the *integrate-and-fire* model found in McCulloch and Pitts (1943) and *perceptron* introduced in Rosenblatt (1958) [29, 27, 30]. The *integration* is given by $y = \mathbf{x}^T \mathbf{w}$ where y is the output, $\mathbf{x}$ is the input vector, and $\mathbf{w}$ is a vector of weights. The *firing* is given by a threshold activation function $f(y)$ [31]. A common activation is the rectifier linear unit (ReLU) $f(y) = max(0, y)$, popular in deep neural networks due to its performance and com-

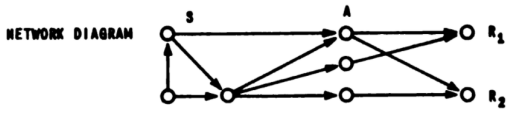


Figure 4: Three-layer perceptron in Rosenblatt (1962)

putational efficiency - however other activations such as the sigmoid function $f(y) = \frac{1}{1+e^{-y}}$ are also common [32, 33].

The structure of hierarchical, deep neural networks is largely based on the structure found in the visual cortex in which information is filtered and processed by sequential layers of neurons [34, 28]. Individually, single-layer perceptrons are able to learn linearly separable functions, but are unable to learn more complicated representations such as non-linearities or invariance to transformations [35]. While perceptrons are often considered synonymous with the single-layer version, some authors contend that Rosenblatt's conception of a perceptron was flexible enough to permit the existence of multiple layers [36]. Indeed, Rosenblatt (1962) shows a network that is described within the book as a three-layer perceptron [37]. As it is, multilayer perceptrons (MLPs) with at least one hidden layer are known to approximate any continuous real function [38].

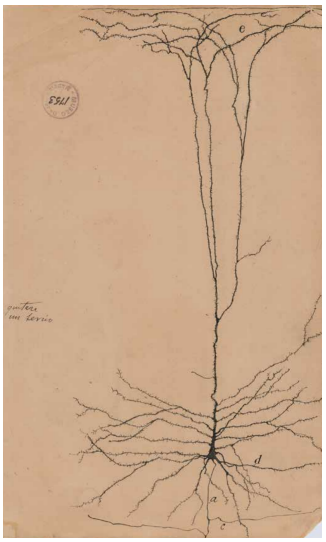


Figure 3: Drawing of a neuron by Santiago Ramón y Cajal, clearly showing the cell body (soma), dentrites, and axon.

Before defining deep learning, we'll first note the role of a technique that opened the way for research to extend beyond MLPs to more complicated architectures: the backpropagation algorithm described by Rumelhart, Hinton, and Williams (1986). The backpropagation algorithm consists of two parts: a forward pass in which a network calculates a predicted output and error relative to the true output; and a backward pass in which weights in the network are changed to minimise this error function with respect to the weights through gradient descent. The authors define the error function as $L = \frac{1}{2}(Y_{\text{Predicted}} - Y_{\text{Target}})^2$. Gradient descent updates network weights by successive use of the chain rules: $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_i}\frac{\mathrm{dy}_i}{\mathrm{dx}_i}$ and $\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial x_i}\frac{\partial x_i}{\partial w_{ij}}$ where i, j refer to indices of specific neurons; y_i refers to the output of neuron i; x_i is the input to neuron i; and w_{ij} is the weight between neurons i and j. This method enabled the practical training of weights in MLPs and led to a resurgence of research in the field [39, 35].

LeCun, Bengio, and Hinton (2015) popularised the term *deep learning* to refer to learning by MLP structures characterised by a large number of layers (e.g. 5-20) [35]. These networks contain many millions of weight parameters, while the recent GPT-3 has 175 billion [40]. The authors point to 2006 as the reemergence of deep network research, with fast advances in image recognition and speech recognition. Applying convolutional layers and pooling led to additional improvements (informally, a convolution passes inputs through a filter in the form of the discrete convolutional function, while pooling outputs the max or average activation within a region). The use of deep convolutional networks has led to successes across multiple domains, including Deep Reinforcement Learning to which we will return to shortly [26, 35].

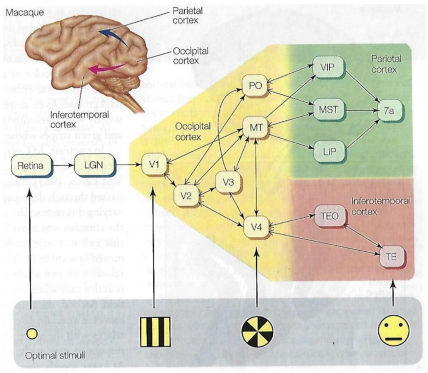


Figure 5: Hierarchical structure of the visual cortex, showing the increased complexity of representation at each layer of neurons. Source: Pinel (2017)

3.2.3 Reinforcement Learning

In this section we give a brief overview of key techniques and considerations in reinforcement learning. Reinforcement learning is a learning technique in which an agent explores an environment and modifies its behaviour in response to obtained rewards. Sutton and Barto (2018) define it as a third branch of machine learning alongside supervised and unsupervised learning. The early development of reinforcement learning was in the field of animal psychology, particularly influenced by the association learning experiments of Ivan Pavlov. In the 1950's, reinforcement learning principles became influential in optimal control theory, in which it was integrated with dynamic programming techniques for the solution of MDP and POMDP problems [3, 1]. We briefly outline below key techniques which have emerged from Reinforcement Learning which have been particularly influential, especially in recent work on Deep Reinforcement Learning. We have already provided an overview of Value Iteration (see: Algorithm 1). More detailed techniques in Reinforcement Learning exist, including those which incorporate planning algorithms (e.g. SARSOP, DESPOT), however we aim here to give a brief taste of the fundamental concepts before moving onto DRL [9]. We follow Sutton and Barto (2018) closely for our notation.

Concept 1 (Temporal Difference (TD) Learning)

TD-Learning examines the difference between a current value and the predicted value for a given state. TD-Learning has neural analogues to the role of dopamine in behavioural formation [3]. Tesauro (1995) combined TD-Learning and a simple neural network to achieve

human-level performance in Backgammon [25]. The TD-Error δ_t is defined as:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2}$$

The back-up function for TD-Learning is given by:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3}$$

Which can be rewritten as:

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t \tag{4}$$

Where $V(S_t)$ is the value of a state at time t, α is the learning rate, and δ_t is the TD-Error. Through the back-up function, rewards for future states are propagated to the stored value of earlier states [3].

Concept 2 (Q-Learning and Double Q-Learning)

Q-Learning is a method for learning a table of state-action pairs $Q(s,a)$ (the 'Q-Table'), which was first described in Watkins (1989) [41, 3]. The method involves iterative updates of the Q-Table based on observed rewards. As can be seen in Algorithm 2, the method essentially applies TD-Learning on the state-action pairs. Watkins and Dayan (1992) provides a proof that the Q-Learning algorithm converges with probability 1 to the optimal policy, assuming that each entry in the algorithm is sampled an infinite number of times [42, 3]. This resampling requirement limits the practical application of Q-Learning to small MDP problems, and motivates approximation methods (e.g. DNNs) to handle larger problems. A common policy applied to ensure continued exploration for Q-Learning is ε-greedy, which selects a random action with probability ε and a greedy action (the current best action as per the Q-Table) with probability $(1-\varepsilon)$. The key update for Q-Learning is given by:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{5}$$

Action value entries within the Q-Table can be optimistically biased, meaning that the action becomes overvalued by the model. This is because $\max_a Q(S',a)$ is used as a proxy for the maximum expected value of the next state (Hasselt (2010)) [43]. This may degrade performance on stochastic environments. To account for this a common extension of Q-Learning is to apply Double Q-Learning which uses two Q-Tables and changes the key update of the form [43, 3]:

$$Q_1(S_t,A_t) \leftarrow Q_1(S_t,A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \max_a Q_1(S_{t+1},a)) - Q_1(S_t,A_t)] \tag{6}$$

$$Q_2(S_t,A_t) \leftarrow Q_2(S_t,A_t) + \alpha[R_{t+1} + \gamma Q_1(S_{t+1}, \max_a Q_2(S_{t+1},a)) - Q_2(S_t,A_t)] \tag{7}$$

Where the algorithm updates Q_1 with probability 0.5 at each step and Q_2 otherwise. The reasoning here is to separate the *estimation* of the action value and the *selection* of an action

at each time step. This avoids a maximization bias that occurs where these two processes are combined in single Q-Learning. The resulting Double Q-Learning does not fully solve the issue of bias within the algorithm, however it does reduce the severity. As with single Q-Learning, Double Q-Learning converges to an optimal policy in the limit [43].

Algorithm 2: Q-Learning (Source: Sutton and Barto (2018) [3])

Set $\alpha \in (0,1]$ and small $\varepsilon > 0$
Initialise $Q(s,a)$ for all $s \in S$, $a \in A(s)$
for *each episode* **do**
    Initialise S;
    for *each step in episode* **do**
        Choose A from S using policy on Q;
        Take action A, observe R, S';
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$;
        $S \leftarrow S'$;
    end
    until S is terminal ;
end

3.2.4 Deep Reinforcement Learning Models

Deep Reinforcement learning combines techniques from both reinforcement learning and deep neural networks. Below we give a brief overview on Deep Q-Networks (DQN), Deep Recurrent Q-Networks (DRQN), and Action-specific Deep Recurrent Q-Networks (ADRQN). We also highlight the use of Prioritised Experience Replay (PER), which has been shown to be a key component of performance in the literature (e.g. see Hessel et al. (2018)) [44, 45].

Concept 3 (DQN)

Deep Q-Networks (DQN) combine Q-Learning with a Deep Neural Network (DNN). The DNN estimates the value of state-action pairs $Q(s,a)$ for network weights θ. Gradient descent is applied to update weights to minimise the error between the predicted $Q(s,a)$ value and observed values. Algorithm 3 shows the pseudocode of the algorithm combined with experience replay (Lin (1992)), which maintains a buffer of previously seen experiences [12].[1] While TD-Gammon represents an early link between ANNs and learning algorithms, the modern source paper for DQN is considered to be Mnih et al. (2013), which was further extended in Mnih et al. (2015) to achieve human-level performance on Atari Learning Environment benchmarks [46, 12]. Similar to Q-Learning, the technique has been extended to a Double Deep Q-Network (DDQN) to reduce estimation bias (Hasselt et al. (2015)) [47].

[1] Note that x in the original algorithm refers to pixel input from an Atari screen, while ϕ refers to a preprocessing step.

Algorithm 3: Deep Q-Learning with Experience Replay (Source: Mnih et al. (2013) [12])

Initialise replay memory D to capacity N
Initialise $Q(s,a)$ with random weights
for *t, T* **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  for *each step in episode* **do**
    With probability ε select a random action a_t ;
    Otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$;
    Take action a_t, observe reward r_t, and image x_1;
    Set $s_{t+1} = s_t$, a_t, x_{t+1}, and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D ;
    Sample random minibatch transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D ;
    Set

$$y_j = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$;
  end
end

Concept 4 (DRQN)

Deep Recurrent Q-Networks (DRQN) includes a recurrent layer (e.g. LSTM) before the final output layer in a DQN. Recurrent models have been shown to perform well for tasks requiring memory or sequential representation across multiple domains [48, 8]. The model is passed an input of a history of observations $(o_1, o_2, \ldots, o_t)$. The DRQN algorithm was introduced in Hausknecht and Stone (2015), which showed improved performance relative to the non-recurrent DQN on Atari problems which had frames removed from the observed environment by 'flickering'. Applying a secondary Q-Network (DDRQN) has been shown to improve performance [8].

Concept 5 (ADRQN)

Action-specific Deep Recurrent Q-Networks (ADRQN) are an extension of DRQN which include actions along with observations as the model input. Concretely, the model is passed an unprocessed input of a history of action-observation pairs $((a_0, o_1), (a_1, o_2), \ldots, (a_{t-1}, o_t))$. In theory, this leads to a closer coupling of actions, observations, and outcomes. The model is introduced in Zhu et al. (2018), which showed improved performance relative to DRQN and DDRQN on Atari problems when frames were removed from the observed environment

through 'flickering' [49]. In later sections, we detail tests to a natural extension to this method in which the step reward is included in the action-observation history.

Concept 6 (Prioritised Experience Replay)

Prioritised Experience Replay (PER) is a technique introduced in Schaul et al. (2016), which maintains a buffer of experiences that are resampled during the training cycle of a DQN algorithm [44]. Experiences are stored as a tuple of the state, action, reward, next state, and the absolute TD-Error $< s_t, a_t, r_t, s_{t+1}, |\delta_t| >$. The method extends Experience Replay (which uses random sampling from stored experiences), by prioritising experiences perceived to be important for training. The *priority* of an experience i is defined as $p_i = |\delta_i| + \varepsilon$ where ε is a small minimum priority value. The probability of sampling an experience i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{8}$$

Where α is a hyperparameter that determines the degree of prioritisation, usually set at $\alpha = 0.5$ [50, 44]. Note that $\alpha = 0$ leads to a uniform distribution. The use of Prioritised Experience Replay is highly influential. In the *Rainbow* experiments performed in Hessel et al. (2018), PER is noted as the most important algorithmic modification for DQN with performance improvements over other techniques such as Double Q-Learning, Dueling Networks, and Distributional Networks [45]. Experience replay without prioritisation was incorporated in early DQN implementations such as Mnih et al. (2013) and (2015), which can be seen in Algorithm 3 [12, 46]. All DRL models examined in this thesis use either PER or the original Experience Replay introduced in Lin (1992) [51].

4 Methodology

4.1 Overview

In this section we provide an overview of our methodology for testing deep reinforcement learning algorithms on POMDPs. Experiments have been conducted on Tiger, Rock Sample, and Tag - three benchmark problems within the literature. These problems were specified in POMDPX, a file format for specifying details of POMDP problems. A Python parser and simulator for POMDPX files was developed for this thesis, which has extended an existing implementation to handle key terms and wildcards. This has enabled a wider variety of problems within benchmark POMDPX repositories to be simulated directly in Python.

This section is structured as follows: first, we cover details of the POMDPX file format and the Python parser and simulator developed for this project. Secondly, we provide a description of the Tiger, Tag, and Rock Sample problems. Third we outline the algorithms and techniques that have been examined in this thesis, and the testing approach including hyperparameter tuning. The techniques examined in this thesis includes the use of 'flooding' regularisation recently introduced in Ishida et al. (2020) [52], the use of pre-initialised memory buffers filled from an 'expert' policy, and prioritised experience replay. The algorithms tested have included many variants of DQN, DRQN, ADQN, ADRQN, including a natural extension to ADRQN that includes the step reward within the observation that we term *RADRQN*.

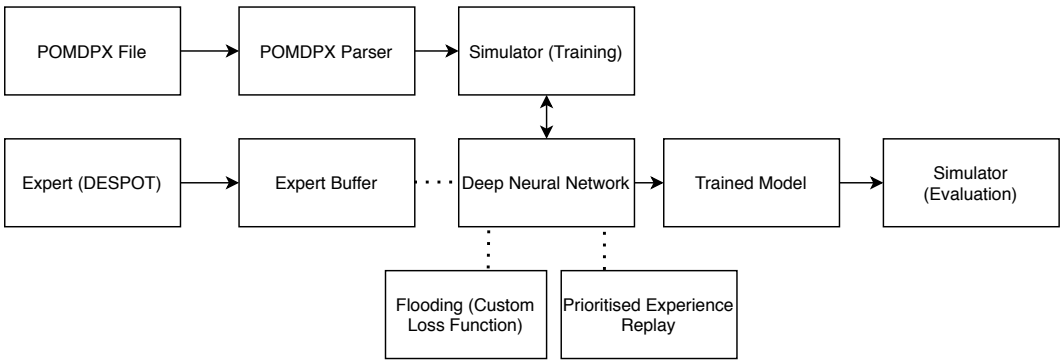


Figure 6: Flowchart showing the interaction between the different modules.

4.2 POMDPX Parser

POMDPX is a file format for specifying POMDP problems [53]. The format is based on XML and allows direct encoding of key parameters and dynamics of a problem, including the Variables, State Transition, Observation Function, and Reward (example code below). Documentation of the format is provided by the *Approximate POMDP Planning Toolkit* (APPL) developed at the National University of Singapore [53]. The original toolkit was de-

veloped in C++, and a number of implementations of key benchmark problems are specified in the POMDPX format.

POMDP Name	**Source**
Autonomous Underwater Vehicle	S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee (2009) [15]
Homecare	H. Kurniawati, D. Hsu, and W.S. Lee. (2008) [54]
Underwater Navigation	H. Kurniawati, D. Hsu, and W.S. Lee. (2008) [54]
Rock Sample	T. Smith and R. Simmons. (2004) [55]
Tag	J. Pineau, G. Gordon, and S. Thrun (2003) [56]
Tiger	L.P. Kaelbling, M.L. Littman, and A.R. Cassandra (1998) [2]
Battleship	Silver and Veness (2010) [10]
POCMAN	Silver and Veness (2010) [10]

Table 2: List of classic POMDP problems. Source: APPL [53].

To enable testing across these benchmark problems using Python libraries, we sourced a parser from Github (https://github.com/larics/python-pomdp) to extract the relevant data structures [57]. This parser was able to handle basic POMDPX files, but did not include wildcard (i.e. the '*' character, which provides for all possible values in a given position in the syntax, and the '-' character which repeats assignment for multiple objects) or keyword (i.e. 'uniform' and 'identity', providing shorthand for matrix values and distributions) terms as specified in the APPL documentation [58]. See Figure 7 for an example of this syntax. As part of this thesis, the original parser was extended to enable these terms to be specified. This is useful as many of the benchmark problems provided on the APPL website (see Table 2 for example benchmarks) require these terms. The codebase for this project is provided as a public Github repository at https://github.com/iciac/POMDP.

To run the parsed POMDPX files a simulator was built. At each time step it handles the state-transition $T(s'|s,a)$, observation $O(o|s',a)$, and reward $R(s,a)$ functions. It additionally provides for the concatenation of the fully-observable components of the state and observation enabling mixed observability problems such as MOMDPs to be specified. Histories of observations are able to be passed to DRL algorithms (e.g. DRQN) to determine step actions.

4.3 Problem Descriptions and Benchmarks

Testing has been performed on Tiger, RockSample, and Tag POMDPX files. The Tiger problem is a classic control problem described in Kaelbling et al. (1998) [2]. The problem

```
<StateTransitionFunction>
    <CondProb>
        <Var>rock_1 </Var>
        <Parent>action_rover rover_0 rock_0</Parent>
        <Parameter>
            <Entry>
                <Instance>ame * - - </Instance>
                <ProbTable>identity </ProbTable>
            </Entry>
        </Parameter>
    </CondProb>
</StateTransitionFunction>
```

Figure 7: Example POMDPX code showing the use of wildcard and keyword terms to specify a State Transition Function. Source: APPL POMDPX Documentation.

involves an agent deciding between opening two doors - behind one is a pot of gold, behind the other a tiger. At each time step, the agent chooses between three actions: opening the *left* or *right* door, or *listening* for the tiger's growl which provides an imperfect observation to the beast's position. Opening the door with the tiger leads to a reward of -100, opening the door with the gold leads to a reward of 10, and listening incurs a reward of -1. Listening has a probability of 0.85 of correctly indicating the tiger's position.

The Rock Sample problem is described in Smith and Simmons (2004) [55]. The problem involves a robot on a $m \times n$ grid. A number of ores exist on the grid, with either 'good' or 'bad' quality. The robot may move in the four cardinal directions and is equipped with a long-range sensor for imperfectly testing each of the rocks. For a map of k ores the robot selects from the actions: $\{N, S, E, W, Sample, Check_1, \ldots, Check_k\}$. The robot receives a reward of 10 for sampling a 'good' ore, a reward of -10 for sampling a 'bad' ore, a reward of 10 for exiting the map to the East, and a reward of -100 for exiting the map from another direction.

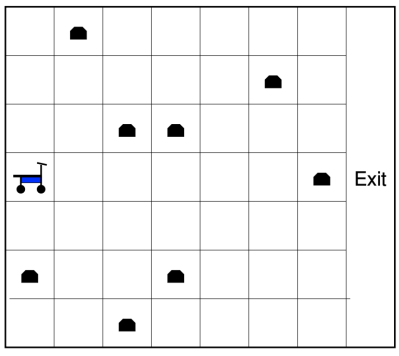


Figure 8: RockSample (7,8). Source: Smith and Simmons (2004)

The Tag problem is described in Pineau et al. (2003) [56]. The problem involves a robot and target on a grid of 29 positions. The position of the robot is fully observed, however the position of the target is hidden unless both are on the same grid square. The target is biased

towards moving away from the robot at each time step with probability = 0.8. The robot selects from the actions $\{North, South, East, West, Catch\}$ and receives a reward of -1 for moving, 10 for successfully tagging the target, and -10 for using *Catch* when the target is in a different square.

Benchmark results for these problems found within the literature are provided in Appendix B: Comparative Results. To record direct benchmark comparison we have used the DESPOT POMDP solver described in Ye et al. (2017) [9]. For the Tiger problem, we average the policy performance of DESPOT at step 15 over 150 policy evaluations [9]. Rock Sample (3,1) is tested for 10 steps over 150 iterations. Note that Rock Sample (3,1) is a toy problem with a known optimal solution and an expected average score of 15. As will be discussed in following sections, despite the simplicity of this problem tested deep reinforcement learning algorithms often had difficulty in converging to the correct solution. Tag and Rock Sample (7,8) are each also tested for 30 steps, 150 iterations. We additionally list results from manual human control on Tiger and Rock Sample (3,1) as a further benchmark.

This gives the following set of comparison benchmarks:

Benchmark	Tiger	Tag	RS (3,1)	RS (7,8)
DESPOT (discounted)	8.71 (2.28)	-7.05 (0.53)	13.2 (0.35)	20.06 (0.35)
DESPOT (undiscounted)	13.45 (3.14)	-9.27 (0.90)	15.4 (0.41)	40.6 (1.07)
Human (undiscounted)	4.8 (1.3)	-	13.2 (0.25)	-

Table 3: Benchmarks for tested problems using the DESPOT solver in Ye et al. (2017). Human benchmarks also recorded for the smaller POMDPs examined (Tiger and Rock Sample (3,1). Average scores recorded (standard errors in brackets).

4.4 Model Extensions

The POMDPX environment has allowed us to examine DRL algorithms found within the literature (DQN, DRQN, ADQN, ADRQN) along with the following extensions: 'flooding' regularisation described in Ishida et al. (2020), pre-intitialisation / 'expert-buffers', and an extension to ADRQN that includes rewards within the observation that we term RADRQN. A brief overview of these techniques is provided below.

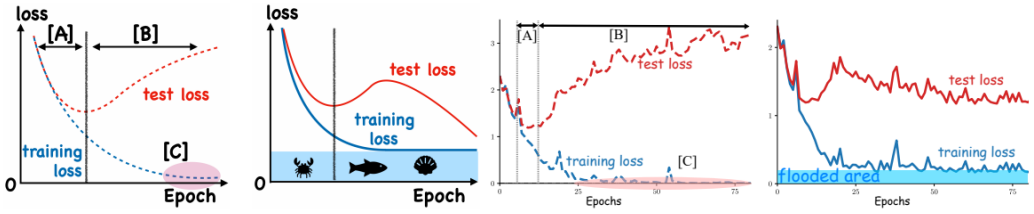
4.5 RADQN / RADRQN

A natural extension of the ADRQN introduced in Zhu et al. (2018) is to concatenate the step reward to the action-observation history. In this way, the model is passed an input of a history of reward-action-observation pairs $((a_0, o_1, r_1), (a_1, o_2, r_2), \ldots, (a_{t-1}, o_t, r_t))$. The justification to this approach is to tighter couple the representation of previous actions,

observations, and outcomes. This is motivated by noting that the reward for taking action a in observed state o may change depending on the previous history. As an example, in the Rock Sample problem, a 'good' rock will turn to 'bad' after being correctly sampled. While it may be eventually learned by sampling and reward transmission through a backup function that taking the same action twice in the same square may lead to a different outcome, an agent may more easily observe this by noting that a reward of 10 has already been obtained in the previous step. To our knowledge, this extension of ADRQN is a novel approach.

4.6 Flooding

The use of flooding reguarlisation is described in Ishida et al. (2020) as a method of handling overfitting within trained models [52]. The algorithm is simply described: for a learning objective function $J(\theta)$, a regularization term b is added as: $\widetilde{J}(\theta) = |J(\theta) - b| + b$, where θ are the model parameters and $\widetilde{J}(\theta)$ is the regularised objective function. This adds a floor to the objective function. Once the floor is reached through gradient descent the model will continue train leading to expected increased generalization. Note that $b = 0$ corresponds to the original objective function. The below image gives the intuition of this approach. The authors experimentally show performance improvement on image classification tasks including MNIST and CIFAR-100.



(a) w/o Flooding **(b)** w/ Flooding **(c)** CIFAR-10 w/o Flooding **(d)** CIFAR-10 w/ Flooding

Figure 9: Intuiting behind flooding regularisation. Training continues following the training loss reaching the flooding value. This enables a random walk over parameters, with the possibility of further test performance improvement. Source: Ishida et al. (2020) [52].

To our knowledge flooding has only been applied using supervised learning on image classification. Overfitting may also be observed within reinforcement learning, with an agent potentially converging to a fixed policy on a deterministic environment and sensitive to small perturbations [59]. In stochastic or partially-observable environments there is a need for the resulting policy to maintain generalization. To test this in the context of deep reinforcement learning we applied flooding values of 0, 0.1, 0.25, 0.5, 2, and 5 to the tested problems.

4.7 Pre-initialisation / Expert Buffer

The use of human experiences as initialisation is a common technique in deep reinforcement learning, as demonstrated in Silver et al. (2015) [23]. Systems that have access to demonstration data have been shown to improve performance on DRL algorithms, including reduced training time as shown in the Deep Q-Learning from Demonstrations (DQfD) model in Hester et al. (2017) [60]. In order to investigate the use of demonstration experiences, we incorporated two pre-initialisations of the memory buffer for prioritised experience replay. One involves an 'expert buffer', involving a pretrained policy that partially fills the model memory buffer (buffer size: 2,000 steps) and biases the learning process to positive expectation policies. The 'expert buffers' for Rock Sample (3,1) and Tiger were constructed by manually playing the problems and recording the human policies for 10 iterations, copied to 100 iterations (1,500 steps) and 330 iterations (1,420 steps) respectively. For Rock Sample (7,8) and Tag, an expert policy was formed by running DESPOT, a tree-based POMDP solver described in Ye et al. (2017) for 150 iterations [9]. As can be demonstrated in the below results, the approach leads to a policy including the optimal sequence of actions being learned on Rock Sample (3,1) which was not able to be demonstrated on other tested experiments despite a wide range of hyperparameter tuning. Memory buffer initialisations based on random sampling was also tested to trial an expert-agnostic pre-initialisation, however did not demonstrate meaningful experimental results.

4.8 Model Architectures and Hyperparameters

Four literature models (DQN, DRQN, ADQN, ADRQN) were tested on the Tiger, Rock Sample and Tag problems. We additionally examine a reward concatenation extension to ADRQN termed as RADQN / RADRQN. Each model takes a history input of the previous N steps of observations. The ADQN and ADRQN models combine each observation with the previous action, while the RADQN and RADRQN models also include the previous step reward. The DQN, ADQN, and RADQN models have the structure (Dense(50), Dense(50), Dense(50)) with ReLu activations. The DRQN, ADRQN, and RADRQN models have the structure (Dense(50), Dense(50), Dense(50), LSTM(50)) with ReLu activations. The loss function used is the mean square error, and the Adam optimizer described in Kingma and Ba (2014) is used with a learning rate of 0.001 [61].

Hyperparameter tuning was conducted through the use of a grid search on batch size (32, 64, 128), learning rate (0.01, 0.001, 0.0001), network layer width (5, 15, 50), training periods, and input history lengths. Prototype model architectures involving convolutions, deep networks (e.g. more than 5 layers), wide networks (network layer width of 300), skip connections, and training delays were trialed, however these had little observable change and were not used for repeated experiments. Training is conducted for $T = 150$ to $8,000$ episodes for each

model. Each episode is a full 'run' of the problem instance. Epsilon-greedy annealing is conducted for each model such that an initial $\varepsilon = 1$ is annealed to $\varepsilon = 0.01$ after 50% of the training time (ε-decay$= \exp\left\{\frac{\ln 0.01}{0.5T}\right\}$).

Testing is conducted by setting $\varepsilon = 0$ and evaluating the fixed network policy on randomly initialised runs of the problem for the evaluation period of 300 episodes. For each trained policy the average episode score over the evaluation period is reported, along with the minimum and maximum episode score. As repeated experiments with the same parameter and hyperparameter settings were observed to produce different final policies, experiments were repeated to observe distributions for our analysis. In total a dataset of 1,100 final policies was developed for the following results.

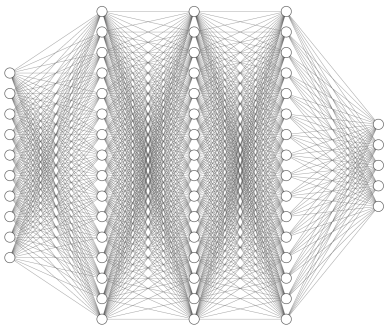


Figure 10: Network diagram for the non-recurrent models. The input layer is a one-hot encoding of the input N-step observation history. There are three fully-connected hidden layers. The output layer contains neurons corresponding to the number of actions. The recurrent models contain an additional LSTM layer between the final hidden layer and the output layer.

5 Results

In this section we examine the performance of our algorithms and parameter specifications on the tested problems. We first examine the maximum final policy scores on the problems across all algorithms, and compare this to benchmarked policies. This is followed by examining the distribution of evaluated policies achieved by the tested algorithms, and compare performance of algorithms incorporating an Expert Buffer or Prioritised Experience Replay. Finally we note training time considerations of the tested algorithms.

5.1 Maximum Scores

	Tiger	**Tag**	**Rock Sample (3,1)**	**Rock Sample (7,8)**
Score	10.72 (0.28)	-25.31 (0.85)	15.03 (0.29)	10.5 (0.28)
Model	ADQN	RADQN	ADQN	ADQN
Training Period	8000	150	300	400
Expert Buffer	-	✓	✓	✓
PER	-	✓	-	✓
Network Width	50	50	50	15
History Length	15	10	10	30
Max Steps	15	30	30	30
Flooding Value	0	0	0	0
Human	4.8 (1.3)	-	13.2 (0.25)	-
DESPOT*	8.71 (2.28)	-7.05 (0.53)	13.2 (0.35)	20.06 (0.35)
DESPOT**	13.45 (3.14)	-9.27 (0.9)	15.4 (0.41)	40.6 (1.07)

Table 4: Maximum scores (evaluated average) on the tested problems. Network width refers to the number of neurons in the dense network layers. * discounted, ** undiscounted

The maximum scores (averaged over the evaluated policy) achieved for the tested problems across all parameter values are given in Table 4. The maximum policy score is chosen as a comparison as it indicates whether comparable performance to other algorithms is able to be achieved by our models. Policy distributions are provided in Figure 11 to indicate generalised performance over multiple trained policies. Each of the maximum scores are achieved by ADQN or RADQN models. Expert Buffers feature in the top performance on Tag, Rock Sample (3,1), and Rock Sample (7,8). Prioritised Experience Replay features in the Tag and Rock Sample (7,8) problems. Performance on Tiger (10.72) exceeds the recorded human result (4.8) and discounted DESPOT score (8.71), but is less than the undiscounted DESPOT score of 13.45. In POMDP literature it is common to compare discounted scores, however this is principally for forward-looking planning algorithms determining a discounted policy expected value. Our evaluation method looks at average recorded scores over an evaluation period. As a result the undiscounted DESPOT score is the truer comparison. Performance on

Rock Sample (3,1) of 15.03 is higher than the recorded human experience and discounted DESPOT score of 13.2, and comparable to the undiscounted DESPOT score of 15.4. Performance on Tag (-25.31) and Rock Sample (7,8) is substantially lower than both the discounted (-7.05 and 20.06 respectively) and undiscounted (-9.27 and 40.6) DESPOT comparisons.

5.2 Algorithm Scores

Algorithm	Tiger	Tag	Rock Sample (3,1)	Rock Sample (7,8)
RADQN	1.79	**-25.31**	10	10.13
ADQN	**10.72**	-25.85	**15.03**	**10.5**
DQN	-2.68	-26.46	10.67	10.13
DRQN	-15	-28.46	10.13	10
ADRQN	-15	-28.61	10	10
RADRQN	-15	-27.81	10	10

Table 5: Maximum scores (evaluated average) for each algorithm on the tested problems.

In Table 5 we give the maximum score obtained by each of the algorithms on the tested problems. Algorithms incorporating action or reward information are observed to have improved performance. Taking the maximum achieved score gives an indication of the best observed performance, but does not give a general performance indication of the algorithm as multiple runs may produce different final policies. In Figure 11 we provide the distribution of policies obtained under the different algorithms. We can immediately note the recurrent networks converging on 'fixed' policies, which will be examined more closely in the Discussion section. We may note here indications that Expert Buffer and Prioritised Experience Replay affect the evaluated performance with wider distributions and higher policy scores.

In addition, we provide the distribution of maximum and minimum iteration scores over evaluated policies in Figure 12. This provides an indication of the complexity of the observed behaviour, which may be missed by examining the maximum average score or distribution of scores alone. As an indication, we can note that the maximum iteration score achieved by policies on Rock Sample (7,8) is 40 (indicating sampling 3 rocks correctly and leaving the environment). The average score for the same trained network however is below 10, due to negative iteration results on other testing iterations. It is worth observing that algorithms incorporating Expert Buffers and Prioritised Experience Replay achieve higher maximum and lower minimum episode scores as the agent interacts with partially-observed components of the problems. This is considered in more detail in the 'Fixed Policy' section of the discussion.

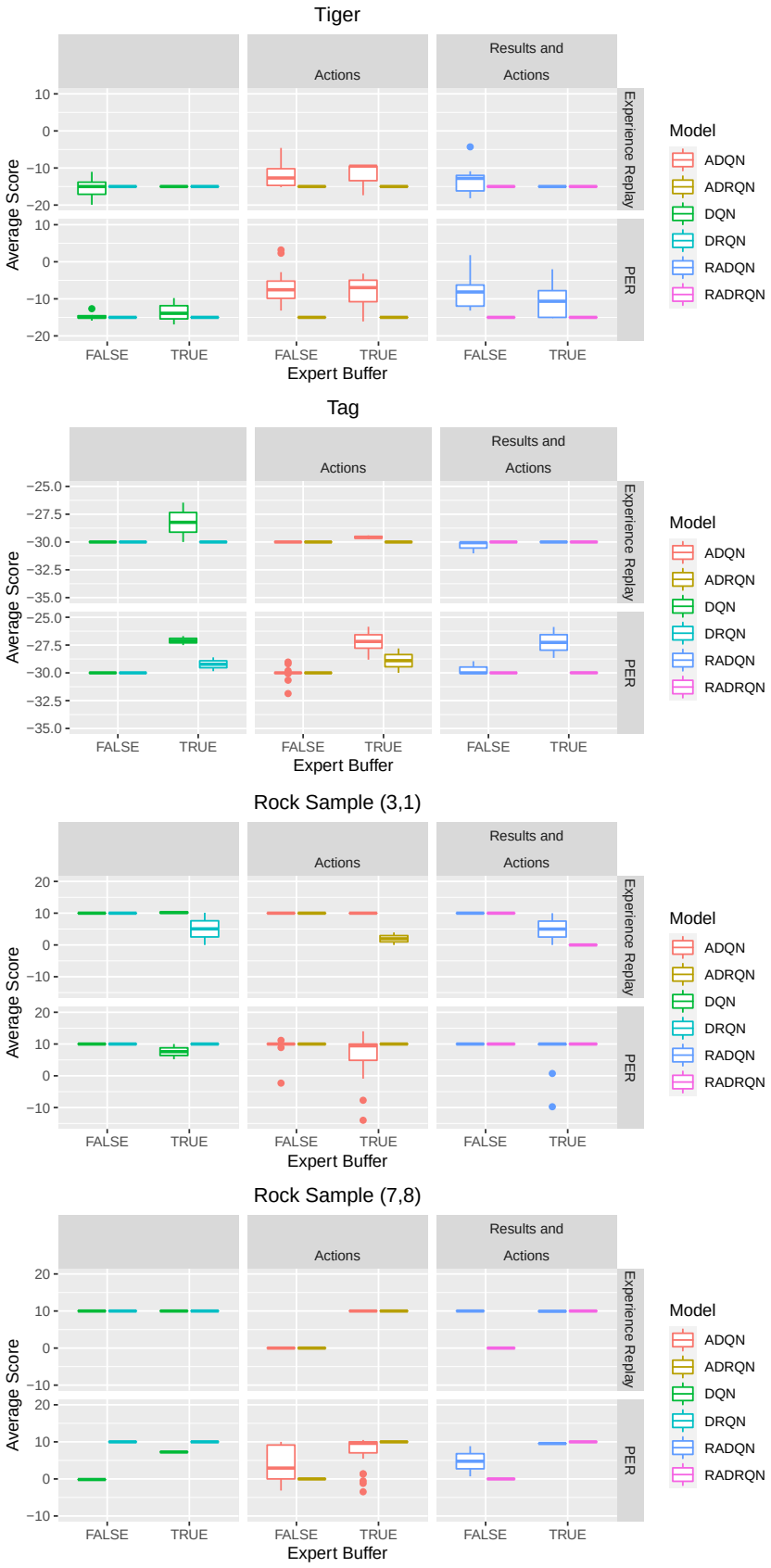


Figure 11: Distribution of evaluated policies. Policies for recurrent policies converge to fixed action sequences. Prioritised Experience Replay is associated with higher scores.

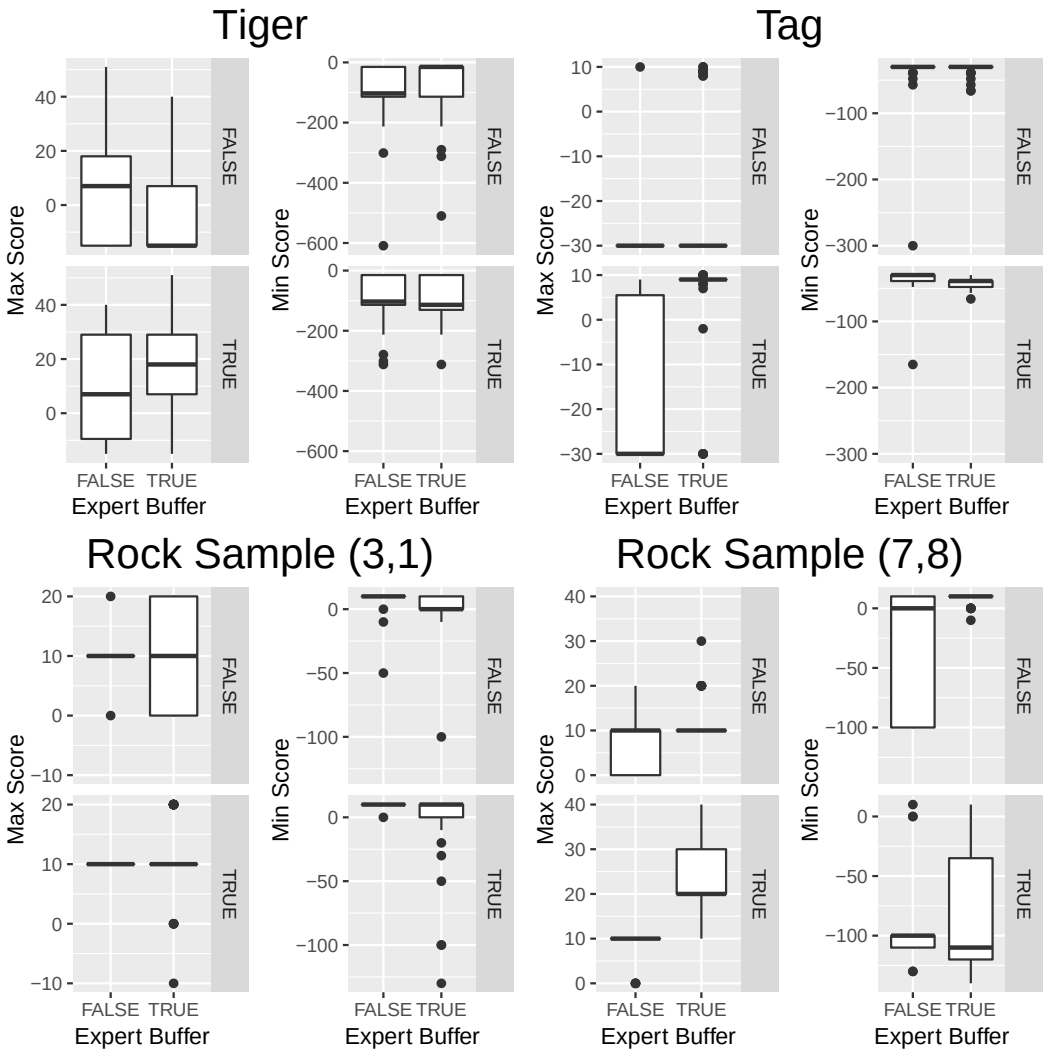


Figure 12: Left (max) and right (min) score obtained from evaluated policies on tested environments. Expert Buffer and Prioritised Experience Replay use leads to higher maximum and minimum values obtained on Rock Sample (7,8) and Tag.

6 Discussion

Results indicate that learning complex behaviours has occurred for the tested problems, however policy scores are substantially below comparison benchmarks within the literature. In certain instances, trained policies for the Tiger and Rock Sample (3,1) are able to converge to optimal policies. However, as the distributional of evaluated policies in Figure 11 shows, this is inconsistently obtained. In this section we discuss in more detail the policies that have been learned, and the observed differences between the tested algorithms and techniques. In addition, we highlight the training times for each of the tested algorithms. We also discuss the convergence of algorithms to 'fixed policies' that avoid worst-case results but do not receive rewards involving partially-observed information. We note the potential for improvements to be obtained by modifying the exploration strategy, and combining the algorithms with simulation-based planning algorithms. Finally, we provide details of an additional module developed in this thesis (a POMDPX to OpenAI Gym converter) which may be used for future research in classic POMDP problems.

6.1 Comparison to Tree-Based Planning Algorithms

Our results are substantially below literature results for tree-based planning algorithms such as SARSOP, POMCP, or DESPOT on the larger problems of Tag and Rock Sample (7,8). These algorithms incorporate simulation based on direct access to model dynamics. In contrast, the DRL algorithms examined by this thesis result in a policy as represented by network weights learned over the course of training. The final evaluated policy fixes these weights, and chooses each step action on this basis. As it incorporates no online simulation of future states and instead relies on an implicit representation of past experiences.

The lack of step-by-step online simulation in our examined DRL algorithms is a potential weakness compared to DESPOT and POMCP. Developing each state-action value prediction from the Q-Network during network training is computationally intensive relative to a Monte Carlo simulation of the problem. As such it is possible that planning based algorithms can incorporate substantially more information to value representations than the DRL algorithms given equivalent training time. As the following section 'Training Times' shows, the training of individual policies is a time-intensive process with no guarantee of improved performance. Rare state history sequences may be little unexplored during the learning process, which a forward looking simulation would help address if encountered. Incorporating Monte Carlo simulations into our algorithms at the training or evaluation stage may therefore lead to more information incorporated into the network to increase performance.

One advantage of the examined DRL algorithms over the online planning algorithms is the compactness of representation for the final policy. Online planning algorithms need to simulate at each step to determine the next action. While this is faster than offline training of

a network, once trained a DRL policy requires little further computation across the evaluation of the problem. The relative strengths of a deep network and simulation is recognised in the literature. Highly successful DRL algorithms, such as AlphaStar, AlphaZero, and AlphaGo use a combination of neural networks and simulation-based algorithms, such as Monte-Carlo Tree Search [5, 23, 24, 3]. Examining these combined algorithms is beyond the scope of this thesis, but it is suggested as a clear avenue for future research on the tested classic POMDP problems.

6.2 Recurrent Algorithms and Flooding Evaluation

In our experimental results, the use of recurrent layers is not seen to provide a performance improvement. This is surprising, given observed performance benefits of recurrent networks on partially-observed problems in the literature. One potential explanation is the type of partial-observability of the problem domain. While both Hausknecht and Stone (2015) and Zhu et al. (2017) examine problems with occluded temporal information through flickering frames in Atari Learning Environments, the POMDP problems of Tiger and Rock Sample are partially-observable due to the possibility of imperfect sensor information. In Tag, the unobservability comes primarily from the lack of access to model dynamics [54]. Other explanations include chosen hyperparameters and features, such as the learning rate. While our selection mechanism for hyperparameters has included grid-based selection to cover a wide variety of combinations it is always a possibility that an untested set of architecture and hyperparameters may result in improved performance for recurrent networks. The use of the LSTM layers is additionally shown to converge rapidly to a fixed policy (e.g. 'always listen'), potentially due to how the TD-error is propagated through the network. This is lower than the optimal policy, and lower than the best obtained results. In this context, a primary benefit of recurrent units (i.e. fast convergence on classification problems) may be thought of as a form of overfitting that could inhibit exploration.

Flooding regularisation is not observed to positively or negatively affect algorithm performance in any significant way. As results in Table 4 show, each of the maximum score policies obtained contain no flooding value. Simple association of results did not show significant correlation between flooding values and average policy scores. We do not rule out the potential for flooding regularisation to positively affect performance in deep reinforcement learning, but would require further testing to evaluate this claim in a statistically robust manner.

6.3 Training Times

As can be seen in Table 6, the use of recurrent units leads to a substantial increase in the required training time. As an example, when training on Rock Sample (7,8) for 3,000 iterations a 1879% increase in training time between the RADQN and RADQRN models was observed.

This appears to be driven largely by the increase to the network parameters, however may also reflect the implementation of LSTM layers within Keras.

Model	**Training Time (s)**	**Score**	Parameters
DQN	829	10	14,263
ADQN	475	10	14,913
RADQN	540	10	14,963
DRQN	2,015	10	28,613
ADRQN	6,172	10	29,263
RADRQN	10,146	10	29,313

Table 6: Training times for recurrent and non-recurrent models on Rock Sample (7,8) trained for 3,000 iterations. Use of recurrent units led to a 18x increase in training time for the RADQN model, with no observed improvement in performance.

Increasing the number of training episodes is not a guarantee of improved performance. While the highest score on the Tiger problem was obtained by training a model for 8,000 episodes, the highest scores for the other test problems were each under 500. This is anomalous to literature indicating performance improvements on DRL algorithms for long training times, and it remains a possibility that further increased training may lead to shifts in performance that could lead to higher scores. This is a potential limitation of our study.

A common observation with long training periods (i.e. more than 5,000 episodes) on the tested problems was that trained policies would become fixed, as will be discussed below. In general, reinforcement learning can be considered as the combination of two separate problems: the exploration problem (collecting data), and the learning problem (performing supervised learning on this data) [62]. In the following section, we discuss the exploration problem as a potential cause of fixed policy convergence. However, we may alternatively consider long training times as subject to potential overfitting on observed experiences. Our tests on flooding regularisation were motivated by an attempt to correct for potential overfitting on long training times, however did not have an appreciable impact on our results. As a result we will need to consider other explanations for this behaviour.

6.4 Fixed Policies

In our results, we can distinguish between policies which learn a ‘fixed’ action response (e.g. ‘always-listen’ for the Tiger problem) independent of observation, and those that exhibit the more complicated behaviour required to score highly on the tested problems. The below fixed policies avoid the worst scores possible in the environment (e.g. by choosing the wrong door in Tiger to receive -100, or stepping out of bounds in Tag), however prevent the agent

from receiving higher rewards requiring more complex behaviour.

	Fixed Score	**Corresponding Policy**
Tiger	-15	Always 'listen'
Tag	-30	Never 'capture'
Rock Sample (3,1)	10	Immediately exiting
Rock Sample (7,8)	0 and 10	Stepping back and forth or immediately exiting

Table 7: Observed policies that obtain a fixed score.

Note that these fixed policies avoid the observations of the problems. From this we can make a few comments. The rewards from state-action pairs selected by the fixed policy are highly certain leading to a tightly estimated Q-value. In contrast, rewards obtained on the basis of a partially-obtained state can be not only uncertain but also contradictory. This can lead to inhibition of learned behaviours during the training cycle. Consider the Tiger problem: after receiving successive observations of 'Tiger Left', the agent may correctly value the action 'Open Right' positively and choose this action. However, this may still result in a reward of -100 due to the problem dynamics. The Q-values are adjusted negatively, and the previously learned behaviour may become inhibited if now $Q(\text{'Tiger Left'}, \text{'Listen'}) \geq Q(\text{'Tiger Left'}, \text{'Open Right'})$. This is problematic with ε-greedy annealing, as the behaviour will become not only less valued but less likely to be explored.

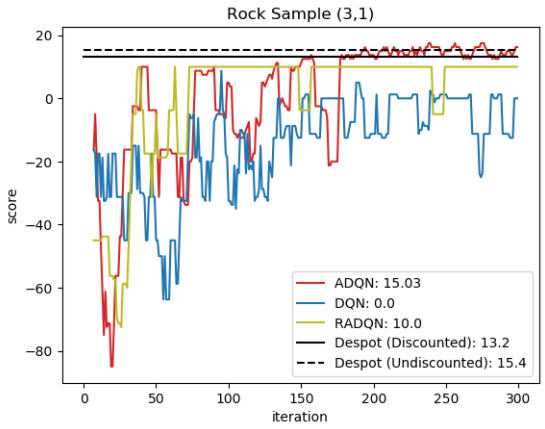


Figure 13: Convergence to different policies. The red line shows a policy converged to the optimal sequence of actions. The yellow shows a policy that exits the environment immediately for a certain 10 score. The blue shows a policy stepping back and forth for a certain 0 score.

We note here similarities to experimental results in psychology and economics. The concept of 'learned helplessness' was first described in Seligman (1967), and involves passivity in

response to repeated averse events independent of behaviour [63, 64]. The concept of *true helplessness* may be formally described using our notation as conditional independence of reward and action $P(r_{t+1}|a_t) = P(r_{t+1}|\neg a_t)$. Learned helplessness refers to situations in which rewards are not truly independent (i.e. the agent has control), but that this is not learned by the agent. Recent experiments in Maier and Seligman (2016) indicate that learned helplessness may occur as an unlearning of the agent's ability to control events in response to averse outcomes - the agent avoids actions with potentially negative outcomes and instead becomes passive (i.e. a fixed response) [63]. Lieder et al. (2013) provides a Bayesian learning simulation study investigating learned helplessness in the presence of averse outcomes, and shows that the presence of uncontrollable negative stimulus (compared to controllable negative stimulus and no negative stimulus) inhibits learning, with many more samples of relevant state-observation pairs required to overcome the contradictory information [65].

With this in mind it is possible that our models may have avoided convergence to fixed policies with an alternative exploration strategy. In particular, applying an Upper-Confidence Bound (UCB) to our exploration instead of ε-greedy annealing may have continued further exploration of behaviours in the presence of contradictory rewards. UCB selects an action at step t as:

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q_t(a, s_t) + c\sqrt{\frac{\ln(t)}{N_t(a, s_t)}} \right) \tag{9}$$

Where c is an exploration parameter, t is the current step, and $N_t(a, s_t)$ is the number of times action a has been selected for the current state. The term $\sqrt{\frac{\ln(t)}{N_t(a,s_t)}}$ accounts for uncertainty in the current estimate in state-action pairs [3]. This term reduces as the state-action pair is repeatedly sampled. As such, it provides a bound on the potential value of a, and as such will be more robust to uncertain or contradictory rewards. It should be noted that for our purposes, we may easily replace $Q_t(a, s_t)$ here with an estimated Q-value $Q_t(a, h_t, \theta)$ from a DQN where h_t refers to a history of observations and θ the model parameters. For the $N_t(a, s_t)$ term, an exact mapping of histories would suffer from the curse of dimensionality. As such an approximate mapping could be considered to allow UCB to be implemented.

6.5 POMDPX to OpenAI Gym Converter

OpenAI Gym is a well-developed API for testing reinforcement learning algorithms [22]. Environments include a variety of classic control problems (e.g. CartPole, MountainCar) from the reinforcement learning literature, along with the suite of games contained within the Atari Learning Environment introduced in Bellamare et al. (2013) [7]. The Atari environments have the advantage that there are well-developed benchmarks to the problems (e.g. Mnih et al. 2013, 2015; OpenAI Baselines) [12, 46, 66]. While all problems contained within OpenAI Gym are classed as POMDPs, once initial conditions are set some may run as if fully-observed, without the potential for measurement error or incomplete state information

that characterises the classic description of POMDPs in Åström (1965) [22, 21, 1]. To account for this and to test robustness to partial observability, authors such as Hausknecht and Stone (2015) and Zhu et al. (2017) 'flicker' observations by removing frames [8, 49].

As well as enabling access to benchmark problems, the OpenAI Gym API also enables access to efficient DRL algorithm implementations through the OpenAI Baselines suite [66]. The baselines are stable implementations of high-performance recent algorithms, such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), and Soft Actor Critic (SAC). These policies are known to perform well on Atari benchmarks and represent active advances in research performance.

Along with the standard suite of problems in OpenAI Gym, it is possible to encode POMDPs as custom gym environments through a simple wrapper. As part of this thesis a POMDPX to OpenAI Gym conversion wrapper was constructed. Full integration with the OpenAI Baselines modules was not finalised, but remains a natural extension for this project. This would enable direct comparison on leading DRL algorithms on the POMDPX tested environments.

7 Conclusion

To summarise, this thesis has examined a number of DRL algorithms on classic POMDP problems. Results for some policies obtained on the small problems of Tiger and Rock Sample (3,1) exceed tested human policies, and are comparable to planning algorithms such as DESPOT. However, this performance is inconsistently obtained, with repeated training of policies receiving different final scores. Performance on larger problems such as Tag and Rock Sample (7,8) is far below literature results for planning algorithms. The use of expert buffers and prioritised experience replay is shown to increase performance, with more complex behaviours observed in final policies using these methods. Flooding regularisation is not shown to have impact on the tested algorithms. Potential causes for fixed policy convergence have been discussed, including the chosen exploration strategy. Analogous results in psychology and economics such as learned helplessness indicate the potential use of POMDP models as models for testing behavioural theories. Immediately suggested expansions for our results include the use of simulation and Monte-Carlo Tree search within the DRL algorithm, and the testing of exploration strategies such as Upper Confidence Bound to improve performance on these POMDP problems.

Our testing has included DRL algorithms noted for performance on fully and partially-observed environments, including an extension to ADQRN that includes a concatenation of the reward in the observation history. Evaluation policies for algorithms on the same hyperparameters show a deal of variability between individual training, making it difficult to claim that any individual algorithm has improved performance on the tested environments. Nonetheless, we note that the best results on each problem were obtained on the ADQN and RADQN models. In addition, we note that recurrent networks (ADRQN, DRQN, RADRQN) appear to converge to a fixed policy quickly during the training process. This may inhibit exploration required for convergence properties of DRL algorithms, reducing the best score obtained using these methods to a local optima.

This thesis also contributes an extension of an existing basic POMDPX parser and simulator in Python to handle keyword and wildcard terms, enabling the testing of DRL algorithms on benchmark POMDP environments specified in the literature (including Tiger, Rock Sample, and Tag). In addition, a POMDPX to OpenAI Gym environment converter was constructed for this thesis. Full integration with OpenAI Baselines was not achieved, meaning that direct comparison with further literature algorithms (such as Proximal Policy Optimization) on these problems remains an open extension to this project. It is hypothesised that in absence of incorporating a form of Monte Carlo simulation or optimistic exploration strategies such as UCB, these algorithms may still be limited in comparison to solvers such as SARSOP and DESPOT due to limited sampling of the potential policy space.

References

[1] K. J. Astrom, "Optimal control of markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Adaptive computation and machine learning series, Cambridge, Massachusetts: The MIT Press, second edition ed., 2018.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Higher Ed, 2016.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[6] M. Kempka, M. Wydmuch, G. Runc, *et al.*, "Vizdoom: A Doom-based AI research platform for visual reinforcement learning," 2016.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, *et al.*, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, p. 253–279, 2013.

[8] M. J. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," *Computing Research Repository*, 2015.

[9] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP Planning with Regularization," *Journal of Artificial Intelligence Research*, vol. 58, pp. 231–266, 2017.

[10] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *Advances in Neural Information Processing Systems 23*, pp. 2164–2172, 2010.

[11] D. Poole and A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. Cambridge, UK: Cambridge University Press, 2 ed., 2017.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," 2013.

[13] A. P. Badia, B. Piot, S. Kapturowski, *et al.*, "Agent57: Outperforming the atari human benchmark," 2020.

[14] T. Lane and S. William, "Why (PO)MDPs lose for spatial tasks and what to do about it," 2005.

[15] S. C. W. Ong, "POMDPs for robotic tasks with mixed observability," p. 8, 2009.

[16] T. Lane and L. P. Kaelbling, "Nearly deterministic abstractions of markov decision processes," in *Eighteenth National Conference on Artificial Intelligence*, (USA), p. 260–266, American Association for Artificial Intelligence, 2002.

[17] H. B. McMahan and G. J. Gordon, "A unification of extensive-form games and markov decision processes," in *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1*, p. 86–93, 2007.

[18] M. Lanctot, E. Lockhart, J.-B. Lespiau, *et al.*, "Openspiel: A framework for reinforcement learning in games," 2019.

[19] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic programming for partially observable stochastic games," in *Proceedings of the 19th National Conference on Artifical Intelligence*, p. 709–715, 2004.

[20] V. Kovařík, M. Schmid, N. Burch, *et al.*, "Rethinking formal models of partially observable multiagent decision making," 2019.

[21] S. Whiteson, B. Tanner, M. E. Taylor, *et al.*, "Protecting against evaluation overfitting in empirical reinforcement learning," in *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 120–127, 2011.

[22] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "OpenAI Gym," 2016.

[23] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[24] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[25] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, p. 58–68, 1995.

[26] S. Russell, *Human Compatible: AI and the Problem of Control*. Penguin UK, 2019.

[27] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[28] J. Pinel and S. Barnes, *Biopsychology*. Pearson Education, 2017.

[29] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[30] F. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65 6, pp. 386–408, 1958.

[31] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2nd ed., 2010.

[32] R. H. R. Hahnloser and H. S. Seung, "Permitted and forbidden sets in symmetric threshold-linear networks," in *Advances in Neural Information Processing Systems 13*, pp. 217–223, MIT Press, 2001.

[33] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[34] E. R. Kandel, *In Search of Memory: The Emergence of a New Science of Mind*. W. W. Norton & Company, 2007.

[35] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[36] J. Peter, "Did Frank Rosenblatt invent deep learning in 1962? – UMass Computational Phonology."

[37] F. Rosenblatt, *Principles of neurodynamics : perceptrons and the theory of brain mechanisms.* Washington: Spartan Books, 1962.

[38] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[40] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," 2020.

[41] C. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[42] C. Watkins and P. Dayan, "Q-Learning," in *Machine Learning*, pp. 279–292, 1992.

[43] H. V. Hasselt, "Double Q-Learning," in *Advances in Neural Information Processing Systems 23*, pp. 2613–2621, 2010.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.

[45] M. Hessel, J. Modayil, H. van Hasselt, *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[47] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for POMDPs," 2018.

[50] D. Takeshi, "Understanding Prioritized Experience Replay," 2019.

[51] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3–4, p. 293–321, 1992.

[52] T. Ishida, I. Yamane, T. Sakai, G. Niu, and M. Sugiyama, "Do we need zero training loss after achieving zero training error?," 2020.

[53] "Home Page - Approximate POMDP Planning Software."

[54] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," in *Proceedings of Robotics: Science and Systems*, 2008.

[55] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, p. 520–527, AUAI Press, 2004.

[56] J. Pineau, G. J. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps.," in *International Joint Conference on Artificial Intelligence*, pp. 1025–1032, 2003.

[57] S. Vladimir and P. Frano, "POMDPx Parser Python Module," 2017.

[58] "POMDPX File Format - Approximate POMDP Planning Software."

[59] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," 2018.

[60] T. Hester, M. Vecerik, O. Pietquin, *et al.*, "Deep Q-Learning from demonstrations," 2017.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[62] D. Seita, "Reinforcement learning is supervised learning on optimized data," 2020.

[63] S. F. Maier and M. E. Seligman, "Learned helplessness at fifty: Insights from neuroscience," *Psychological Review*, vol. 123, no. 4, pp. 349–367, 2016.

[64] M. E. Seligman and S. F. Maier, "Failure to escape traumatic shocks," *Journal of experimental psychology*, vol. 74, no. 1, pp. 1–9, 1967.

[65] F. Lieder, N. Goodman, and Q. Huys, "Learned helplessness and generalization," pp. 900–905, 2013.

[66] P. Dhariwal, C. Hesse, O. Klimov, *et al.*, "OpenAI Baselines." `https://github.com/openai/baselines`, 2017.

A List of Key Deep Learning Algorithms

Deep Q-Network (DQN) Key paper: Mnih et al. 2015
Deep Recurrent Q-Network (DRQN) Key paper: Hausknecht and Stone, 2015
Action-Specific Deep Recurrent Q-Network (ADRQN) Key paper: Zhu et al., 2018
Deep Variational Reinforcement Learning (DVRL) Key paper: Igl et al., 2018
Determinized Sparse Partially Observable Tree (DESPOT) Key paper: Ye et al. (2017)
C51 (Distributional DQN) Key paper: Bellamare et al. (2017)
Vanilla Policy Gradient (VPG) Key papers: Sutton et al. 2000, Schulman 2016(a), Duan et al. 2016, Schulman et al. 2016(b)
OpenAI implementation: link Trust Region Policy Optimization (TRPO) Key papers: Schulman et al. 2015, Schulman et al. 2016, Kakade and Langford 2002 OpenAI implementation: link
Proximal Policy Optimization (PPO) Key papers: Schulman et al. 2017, Schulman et al. 2016, Heess et al. 2017 OpenAI implementation: link
Deep Deterministic Policy Gradient (DDPG) Key papers: Silver et al. 2014, Lillicrap et al. 2016 OpenAI implementation: link
Twin Delayed DDPG (TD3) Key papers: Fujimoto et al, 2018 OpenAI implementation: link
Soft Actor-Critic (SAC) Key papers: Haarnoja et al, 2018(a), Haarnoja et al, 2018(b), Haarnoja et al, 2018(c) OpenAI implementation: link
AlphaZero Key paper: Silver et al, 2017
Generalized Advantage Estimation (GAE) Key paper: Schulman et al. 2015
Discriminative Particle Filter Reinforcement Learning (DPFRL) Key paper: Ma et al. 2020
Long-Short Term Memory (LSTM) Key paper: Hochreiter et al., 1997
SARSA Key paper: Rummery & Niranjan (1994)

B Comparative Results

Rock Sample (7,8)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	21.47	0.04
Ye et al. (2017)	SARSOP	21.47	0.04
Silver and Veness (2011)	HSVI-BFS	21.46	0.22
Ong et al. (2009)	SARSOP	21.39	0.01
Silver and Veness (2011)	SARSOP	21.39	0.01
Silver and Veness (2011)	AEMS2	21.37	0.22
Kurniawati et al (2008)	SARSOP	21.27	0.13
Kurniawati et al (2008)	HSVI2	21.27	0.09
Ye et al. (2017)	DESPOT	20.93	0.3
Ye et al. (2017)	DESPOT (unregularised)	20.9	0.3
Ye et al. (2017)	AEMS2	20.89	0.3
Silver and Veness (2011)	POMCP	20.71	0.21
Ye et al. (2017)	POMCP	16.8	0.3
Silver and Veness (2011)	Rollout	9.46	0.27
Rock Sample (10,10)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	21.47	0.04
Ong et al. (2009)	SARSOP	21.47	0.11
Rock Sample (11,11)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	21.8	0.04
Ye et al. (2017)	DESPOT	21.75	0.3
Ye et al. (2017)	DESPOT (unregularised)	21.75	0.3
Ong et al. (2009)	SARSOP	21.56	0.11
Silver and Veness (2011)	SARSOP	21.56	0.11
Ye et al. (2017)	SARSOP	21.56	0.11
Silver and Veness (2011)	POMCP	20.01	0.23
Ye et al. (2017)	POMCP	18.1	0.36
Silver and Veness (2011)	Rollout	8.7	0.29
Ye et al. (2017)	AEMS2	-	-

Rock Sample (15,15)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ye et al. (2017)	DESPOT	18.64	0.28
Ye et al. (2017)	DESPOT (unregularised)	18.15	0.29
Silver and Veness (2011)	POMCP	15.32	0.28
Ye et al. (2017)	POMCP	12.23	0.32
Silver and Veness (2011)	Rollout	7.56	0.25
Silver and Veness (2011)	SARSOP	-	-
Ye et al. (2017)	SARSOP	-	-
Ye et al. (2017)	AEMS2	-	-
Tag (29)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	-6.03	0.04
Ong et al. (2009)	SARSOP	-6.03	0.12
Ye et al. (2017)	SARSOP	-6.03	0.12
Kurniawati et al (2008)	SARSOP	-6.13	0.12
Ye et al. (2017)	DESPOT	-6.23	0.26
Ye et al. (2017)	AEMS2	-6.41	0.28
Ye et al. (2017)	DESPOT (unregularised)	-6.48	0.26
Ye et al. (2017)	POMCP	-7.14	0.28
Kurniawati et al (2008)	HSVI2	-7.43	0.11
Tag (55)			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	-9.9	0.11
Ong et al. (2009)	SARSOP	-9.9	0.12
AUV Navigation			
Citation	**Algorithm**	**Score**	**"+/-"**
Ong et al. (2009)	MOMDP	1020	8.5
Ong et al. (2009)	SARSOP	1019.8	9.7
Kurniawati et al (2008)	SARSOP	722.59	1.3
Kurniawati et al (2008)	HSVI2	721.45	0.75