School of Electronic Engineering and Computer Science

Final Report

Programme of study:
Computer Science

Project Title:

# RLSVI – An Introduction and Analysis in a sparse reward environment

Final Year
Undergraduate Project 2021/22

Supervisor:
Paulo Rauber

Queen Mary
University of London

Student Name:
Musa Lala - 190801410

Date: 21st March 2022

# Abstract

With recent advancements in the field of machine learning, the sub-field of Reinforcement Learning has become an ever-significant aspect of our current technological landscape as we seek to optimise our current practices beyond human limits in fields such as automation, language processing and healthcare. Consequently, a multitude of approaches have been developed with the goal of overcoming critical challenges such as the efficiency of exploration, and its optimal balance, with exploitation and generalisation.

These methods often scale in theoretical complexity, potentially leading to difficulties, for some, in the appreciation of the contributions they make. One such method is Randomised Least Squares Value Iteration (RLSVI) [Osband, Van Roy and Wen, 2016], a model-free algorithm that offers an elegant approach to learning by inducing randomness within the least-squares approximation of an action-value function to instigate efficient exploration, something that is especially important when state-action spaces scale.

Here an in-depth introduction and analysis of RLSVI is presented to build up foundational elements such that the reader can better understand the algorithms intricacies before introducing a simple, sparse reward environment in which its performance can be evaluated.

# Preface

RLSVI was initially proposed by Osband, Van Roy and Wen in 2014, an updated version of the same paper from 2016 has been credited within this paper. The algorithms behind RLSVI that are presented in Chapter 2 originate from the 2016 paper. Furthermore, some examples, equations and illustrations presented within chapter 2 are based on knowledge introduced by Sutton and Barto in the credited, 2018 version of their book.

The experiments and analysis within Chapter 3 are based upon use of the program developed for the paper titled 'Angrier Birds: Bayesian reinforcement learning' in 2016 [Ibarra, Ramos and Roemheld, 2016]. With which I have conducted a multitude of experiments with the intention of adding to the work they have conducted and provide my own perspective and analysis on results both included and not included within the paper.

# Acknowledgements

Firstly, I would like to thank my supervisor Dr Paulo Rauber for his invaluable assistance and input with my project. Despite the issues I faced over the current academic year he helped steer me towards a clearer goal for my project, assisted me in times of uncertainty and provided me with advice in regard to my research and manner of writing. Secondly, I would like to thank the senior lecturer Dr Usman Naeem for assisting me with the real-life aspects of my project (delays, claims I needed to make etc.) when I was unable to get answers from anyone else and for putting me at ease and directing me to the best courses of action multiple times. Finally, I would like to express my deepest gratitude to Dr Sandra Sullivan for her invaluable assistance and guidance during my project and with all of the support and good grace she has shown me. For which I truly cannot show my thanks enough.

# Contents

# Chapter 1: Introduction

## 1.1 Background and Motivation

Computing technology has developed and the wealth of data at our fingertips has increased astronomically, far beyond that of what humans can manage. As a result, our need for machine learning techniques has increased across many sectors, constantly pursuing improvement and optimisation [Sarker, 2021]. In terms of machine learning techniques three methods have risen in prevalence, one of which is Reinforcement Learning (RL).

In essence, RL emulates the very nature of how living beings learn through interactions with their environment. What does this mean? As humans progress through their lives, they constantly encounter circumstances without a priori knowledge. Yet, through that web of interactions, information is absorbed and used to influence future responses [Sutton and Barto, 2018]. Given our goal is to create efficient intelligent systems, it is only logical to apply the same strategy and bestow onto them the same learning ability.

Initially, in the process of researching this subject, various preconceptions were challenged, particularly the different approaches to the implementations of the general concept of reinforcement learning. The lack of readily available material that could introduce the topic in a clear and concise manner proved to be somewhat of a hindrance. Furthermore, it is vital to consider how such issues could likely serve as a deterrent to some and potentially lead to the abandonment of projects, undoubtedly a great loss from an academic perspective.

With this in mind I thought it would be interesting to create an analytical piece on a technique I enjoyed researching and sought to learn more about but felt fit the criteria outlined above. Given this specification I set out for myself I came to the desire of working further on a topic I encountered the most difficulties with when learning about, an approach that builds upon value function learning through the addition of randomisation. With a specific focus on randomised least-squares value iteration (RLSVI) [Osband, Roy and Wen, 2016].
In this report I will investigate RLSVI from the ground up, providing an analysis rooted in theory but emphasised through practical experimentation.

## 1.2 Aims and Objectives

This work aims to provide insight into RLSVI via an analysis of the algorithm's performance within a simplified setting, including the implementation of modifications to gauge their effect. It also serves to explain RLSVI from a theoretical standpoint, review its concepts and emphasise clarity whilst comprehensively explain the baseline techniques and ideas.

My main objectives are as follows:

- Research the basic concepts of machine learning and reinforcement learning, expanding on these where appropriate.
- Understand the importance of the different aspects of RL.
- Research into RLSVI from a ground level (theory) to expand my knowledge of the algorithm.
- Explain the concepts above in a clear, concise and logical manner.
- Locate a suitable model, a simple environment (for the purpose of RL) that is also easy for the reader to grasp, in which effectiveness of RLSVI can be tested.

- Breakdown the different aspects of the program to facilitate the reproducibility and replicability of methods and results.
- Test RLSVI within my selected environment, collect multiple metrics of data in regard to its performance and conduct a detailed analysis on them, linking back to the concepts that would have been explained previously.
- Provide any additional insight into the algorithms implementation based upon the results of my analysis.

## 1.3 Research Questions

Question 1 – How does the proposed version of RLSVI compare in terms of efficacy to that of other temporal difference algorithms in a simplified, sparse reward environment?

Question 2 – To what effect will alterations have on RLSVI and how will changes affect the algorithms standing when compared to other RL algorithms?

## 1.4 Report Structure

This report will be structured in a manner I believe to be logical for the purposes of learning such that each section will flow well into the next and be able to complement each other. The structure itself will be as follows:

Introduction: Here I have outlined my initial thoughts going into this project and how they relate to the problem I seek to address. Furthermore, I have given a clear outline of the aims and objectives I strive to reach during the time I spend on this project as well as an outline of the research questions I hope to address.

Literature Review: The literature analysis displays the concepts I seek to explain to the reader, this information having served as my own foundation over the course of the project. Here I shall seek to explain and expand upon key concepts such that they are explained clearly and wholly, allowing for a good understanding of RLSVI (and features of RL algorithms) to be formed. In addition to this I will provide a critical analysis of the findings, methodology and general content of research papers that have been utilised by me to better my understanding on the topic at hand, presenting conclusions about their efficacy from the standpoint of someone infantile to the topic.

Analysis of Program and research methodologies: Here I will outline my rationale when searching for and for deciding on the program in which I have conducted my experiments, the issues I have encountered during my testing (and the processes I took to overcome them) as well as the manner in which my results can be replicated for those inclined. I will also provide a detailed breakdown and explanation of important aspects of the program/program files and evaluate it (and my experience with it) as a whole.

Results and analysis: Here the data representing the results of my experiments will be outlined in the form of plots and/or tables. I will analyse the results that I have obtained and explain the significance of each whilst contextualizing everything in relation to my initial research questions.

Evaluation: This section will be used to present any conclusions that have become apparent over the course of the research and experimentation I have conducted. It will also serve as a place to discuss the difficulties I have encountered, lessons I have learned and thoughts I have at the end of my research, what I hope people will be able to derive from this paper and how it could have been further developed/improved.

References: Here I will compile the references I have used through my report, structured in according with the Harvard referencing system.

# Chapter 2: Literature Review

## 2.1 Concept Definitions

### 2.1.1 Markov Decision Processes

A Markov Process (Chain) itself is a mathematical model comprised of a state and a state transition. In accordance with certain probabilistic rules, transitions from one state to another occur. However, the probability of any particular transition to a state occurring is irrespective of any past transitions to the current state. Instead, the probability of a transition to a state in the next time interval is a function only of the current state and the state it is transitioning to [Howard,1972].

Markov Decision Processes (MDPs) take the philosophy of Markov Chains for use as decision making processes within a stochastic (described by a random probability distribution, outcome has a level of uncertainty) environment in a sequential manner. Fundamentally, they are classified in accordance with the mathematical properties of a state and action spaces, a time at which decisions are made (epochs), the length of time over which decisions are made and an optimality policy [Heyman and Sobel, 1990].

### 2.1.2 Environment

In Reinforcement Learning, an environment is a space in which an agent (learner and decision maker) resides and is able to interact. The definition any respective environment pertains toward the rules and dynamics of space for example, the specification of actions that are available to agents, the rewards and penalties that are present and what states the environment can be in. These base rules are such that they are unaffected by an agent's actions in the same way humans cannot affect the rotation of the earth or the laws of physics but their actions and lives are confined such that they must abide by them.

### 2.1.3 State space $\boldsymbol{s}$

The state is the observational information in a given environment that will be received by an agent, providing them with a view of the world at that point. The key factor of this is the word "observational". For a humanoid robot that makes tea, the state may include the location of a cup, tea bags, a kettle and water in addition to the relative locations of its own appendages. In an instance of playing a card game the agent would be given information in regard to its own hand but wouldn't have knowledge of other players cards or any future cards that could be drawn from the deck, thus the state available to it would only present the agent with partial information.

### 2.1.4 Actions $\boldsymbol{\alpha}$

Actions are anything that an agent is allowed to do in relation to interacting with its environment, they can range from simple directional inputs akin to games like Pac-man, thus limiting the choices an agent can make, to more complex action sets. This can be best understood by applying a human perspective to the concept for example if one were to play an MMORPG game, any movements you would like to make, NPCs you speak to, things you fight, loot or interact with and any general interaction you could have with the world and interfaces available to you would be classed as actions (or a set of actions).

### 2.1.5 Transition Probabilities $\mathbf{P_t}(.|\mathbf{s},\mathbf{a})$

The probabilities that determine the state $s'$ of a system in the next time step given an action $a$ in state $s$ at the current epoch $t$. This transition model links back to the Markov Process referenced earlier whereby the probability of transition is independent of any prior states when at the current state and is instead reliant upon the actions taken in the current epoch within the current state.

### 2.1.6 The notion of Regret

To regret, in the context of regretting an action one has undertaken, is to know of a possible action with more benefits to the one undertaken and hence have a feeling of disappointment proportional to how much better said action would have been.

Mathematically, regret can be defined as the difference between the reward of an action that has been taken and another possible action. It follows that the "design of an optimal algorithm is then equivalent to the design of an algorithm that minimizes regret." [Tranos and Proutiere, 2021] when this is applied to reinforcement learning, it leads to the logical implication that the cumulative regret over a course of decisions is something one would want to minimise in an attempt to optimise the decision-making processes of a learning algorithm.

Agents in a Markov Decision Process adhere to this in that their goal is to maximise their cumulative rewards, meaning a precedent of rewards and regret being somewhat inverse to each other can be set, therefore they must seek to minimise their cumulative regret.

### 2.1.7 Expected and Discounted Returns

To explain the expected return, a return itself must be defined. As mentioned previously when thinking about agents in an MDP, the main goal of an agent would be to maximise the reward they'd receive at any given time step. In this process the return itself is what the agent is going to gain from committing to an action, hence what reward it is going to get. Importantly the reward the agent is trying to maximise is not exclusive to the next time step but rather as a cumulative reward for both the action the agent is taking now and the sum the rewards it would obtain in the future. This sum of future rewards can be seen as our expected return and it is denoted by:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

Where $G_t$ denotes the return following time $t$, $R_t$ is our reward at time $t$ and $T$ is the final time step.
There is an issue however with this concept, namely that our final time step could in fact be infinite and if that were the case calculating an expected return would take infinitely long for every possible action, thus would not be feasible to compute.
To get around this the concept of a discounted return is used. To begin, the discounted return is denoted by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \end{aligned}$$

Where $\gamma$ is the discount rate which can be between 0 and 1. If you look at the sigma notation you will see that for $k$ future returns within the equation the discounted return is multiplied to the power of $k$. Due to 0 < $\gamma$ < 1, when multiplied in this way $\gamma$ will become smaller for every subsequent iteration.

This process would make it so that subsequent rewards that are multiplied by $\gamma^k$ are weighted lower in terms of the overall expected return allowing for more immediate rewards to have a higher amount of influence on the agent's decision making:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notice that though the return is a "sum of an infinite number of terms, it is still finite if the reward is nonzero and constant—if < 1" [Sutton and Barto, 2018] and can thus be calculated via the reward at time $t+1$ and the discounted return at time $t+1$.

### 2.1.8 Policy $\boldsymbol{\pi}$

A policy defines the strategy that dictates the behaviour of an agent at any given time. Due to the Markovian nature that underpins this type of learning, this strategy works such that there is one answer given per an agent's current state as opposed to a plan that built up over a culmination of the agent's current and previous states. Formally it would be said that a policy is a mapping of states to the corresponding probabilities of selecting every possible action [Sutton and Barto, 2018]. Mathematically a probability within this strategy can be seen as $\pi(a|s)$. Here $\pi$ denotes a policy being followed at time t, $a = A_t$ and $s = S_t$ where $A_t$ is the current action and $S_t$ is the current state.
To note, $\pi$ is usually to denote a policy that is stochastic in nature. If the policy is deterministic, it will usually be denoted by $\mu$.

### 2.1.9 Finite-horizon MDP

Finite-horizon MDP
This article is constrained to the definition of a finite-horizon MDP $\mathrm{M} = (\mathrm{S}, \mathrm{A}, \mathrm{H}, \mathrm{P}, \mathrm{R}, \pi)$ in accordance with the outline presented by [Osband, Roy and Wen, 2016].
Here $S$ is a finite state space, $A$ is a finite action space, $H$ is the horizon over which the agent will act (thereby the length of an episode), $P$ encodes transition probabilities, R encodes the distributions for the returns and $\pi$ is a state distribution in accordance with the policy.
In each episode, $S_0$ is sampled from $\pi$ as our initial state. Next for the selected state $s_h$ and action $a_h$, such that $h$ is a period where $h = 0, 1, \ldots, H-1$, the next state $s_{h+1}$ can be sampled from the transition probability function $P_h(.|s_h, a_h)$. A return $r_h$ is then sampled from $R_h(\cdot|s_h, a_h, s_{h+1})$. The episode ends when period $h$ concludes such that the final state $s_H$ is sampled and a terminal reward is sampled from $R_H(\cdot|s_H)$ [Osband, Van Roy and Wen, 2016].
To note, a center-dot (interpunct, ·) is used to represent that any value can be inserted as the value of the function allowing us to write functions without needless introduction of extra variables.

### 2.1.10 Value Functions

Value functions are functions of states/state-action pairs that estimate the expected return of an agents for performing an action within a given state. Given that the expected return is hinged upon the strategies implored by an agent within a state, value functions are defined with respect to the type of policy that is implemented.

On-policy State Value Function:

The state value function denotes the expected return when starting in a state $s$ and following a policy $\pi$ thereafter. For MDPs it can be defined formally by

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right],$$

Where $\mathbb{E}_\pi$ denotes the expected value when an agent acts in accordance with policy $\pi$, and $\mathbf{t}$ is any time step. Thus $v_\pi(s)$ denotes our expected return for state $s$ under policy $\pi$ [Sutton and Barto, 2018].

On-Policy Action-value function:

The action-value function is similar in terms of mathematical structure to the state value function however, it incorporates the inclusion of an action being taken when starting in state $s$ and thereafter following a policy $\pi$. The formal definition of an action-value function is as follows:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s,\ A_t = a\right].$$

Here, the action-value function for policy $\pi$ itself is denoted by $q_\pi$.
To note, as value functions are defined with respect to strategies/actions that can be implored, they may change within the same environment in accordance with a change in policy [Naeem, Rizvi and Coronato, 2020].

Optimal Value & Action-value functions:

In reinforcement learning our goal is to find a policy that maximises our cumulative reward over an extended period. For finite MDPs an optimal policy $\pi_*$ can be defined such that the expected return of the policy $\pi$ in question is greater than or equal to all other policies $\pi'$ for all states, the existence of at least one such policy is always definite. As depicted in [Sutton and Barto, 2018], optimal policies share the same state-value function $v_*$ and action-value function $q_*$ such that,

$$v_*(s) \doteq \max_\pi v_\pi(s), \qquad \text{for all s} \in \text{S and,}$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \qquad \text{for all s} \in \text{S and a} \in \text{A(s).}$$

Bellman:

Bellman Equations are self-consistency equations that illustrate the relationship between the value of a state and the value of its successors. The rule behind them states that the value of our starting state is equal to the discounted value of the expected next state and the rewards we expect along the way [Sutton and Barto, 2018].
The Bellman equations for the on-policy value functions are

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_\pi(s')],$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(s',a') q(s',a')\right].$$

Where it is implied that $a$ is taken from set $A(s)$, the next states $s'$ are taken from the set $S$ and the rewards $r$ are taken from the set $R$.
And the Bellman equations for the optimal value functions are

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_*(s')],$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right].$$

The difference between the two being the inclusion of $max$ which forces an agent to pick the action that leads to the highest return.

### 2.1.11 Exploration vs Exploitation

The concepts of exploration and exploitation have particular importance in reinforcement learning, more specifically attempting to find a balance between the two such that learning can be optimized over an extended period.

To define these concepts and better illustrate this dilemma we begin with a theoretical scenario. You are wanting to read a book and so move over to your imaginary bookshelf in search of a piece of literature to immerse yourself in. On the bookshelf you see your 3 favorite books amongst many unfamiliar books, simply there to fill the space. From here you have a choice, you can pick up one of your favorite books exploiting the certainty you have of finding a fair amount of enjoyment when rereading it due to your previous reading experience and the knowledge you gained at that point. Alternatively, you can take a risk and choose to explore the selection of books you are yet to read with no certainty of the outcome, you could find "the book to end all books" and come away with an evening full of enjoyment and a new favorite book to add to your list or you could stumble upon something you dislike, left feeling as though you had wasted your time as the evening wanes away. The choices you end up making here would depend on a multitude of factors in relation to how inclined they are to try something new, their general mood, how appealing the other books look etc.

Recontextualising this, similarly to how we can choose to read our favorite books, literature which is familiar to us, agents have a suite of actions they have already tried and know the effectiveness of (in terms of rewards to be gained). They also have a multitude of "books they have not read", the actions they have not selected, and encounter the same level of uncertainty as us when picking up something unfamiliar. The catch in both scenarios is to consider how familiar articles were available to be exploited in the first place. To have the option of exploitation, one must have already gained information about their environment through exploration.

The dilemma then is to balance the two in a way such that neither the same actions are endlessness repeated due to the security they hold nor is information gathered endlessly which may cause us to stray from maximizing the possible cumulative reward. It is important to note that a string of actions from time t that seem to be the most rewarding could in fact be worse, cumulatively, than a set of actions that starts of as less rewarding but leads to a better payoff after multiple time steps.

Strategies exist that attempt to solve this dilemma, an interesting yet basic study of which can be seen in [Coggan, 2004]. However, the core of this dilemma has been studied for by mathematicians for decades, yet it remains unsolved [Sutton and Barto, 2018].

### 2.1.12 Linear Function Approximation

As reinforcement learning problems scale so does the number of states needing to be represented. As such a representation that is able to hold these states whilst remaining memory efficient is required. In machine learning the standard approach is to use a linear function approximator to represent values.
A linear function approximator is a function that is linear in weights but not necessarily linear for an input $x$. The function is of the form $y = f(x, w)$ where $f$ is a function that can be either linear or non-linear, $w$ is our weight and $x$ is our input, $x, y \,\&\, z$ can be vectors.
Here we introduce the value function approximation:

$$v(s) \approx v_\theta = \theta^T \varphi(s).$$

Where $\varphi(s)$ is a feature vector capable of holding linear basis functions and $\theta^T$ is a parameter vector used to align functions in that it is a transpose of matrix weights intended for application with our features. The dimensionality of the parameter vector $\theta \in \mathbb{R}^K$ is equal to the number of features $K$ for the function $\varphi(s)$ [Kunz, 2000].

### 2.1.13 Temporal Difference Learning

As explained in [Sutton and Barto, 2018], temporal-difference (TD) learning is a rather novel yet important idea to reinforcement learning. The approach itself combines ideas from both Monte Carlo and dynamic programming (DP).

Monte Carlo - Put simply, Monte Carlo methods focus on allowing agents to learn about their environment through interactions. Agents generate state-action-reward samples through their experiences to approximate future returns by averaging said returns. It's important to note that for Monte Carlo methods, updates to state values occur at the end of each episode.

Dynamic Programming - First proposed by [Bellman, 1954] DP refers to the simplification of a complex problem by recursively breaking it down into subproblems. The collection of algorithms under DP can be used to calculate optimal policies given a perfect model of an environment [Sutton and Barto, 2018].

Temporal Difference - TD methods allow for learning to occur through experiences without a model of an environment's dynamics, akin to Monte Carlo methods. Whilst also allowing for estimates to be updated incrementally based in part on other estimates (bootstrapping), without waiting for a final outcome, akin to TD [Sutton and Barto, 2018]. The updates TD methods compute are of the form

$$New\ Estimate \leftarrow Old\ Estimate + \alpha[Target - Old\ Estimate]$$

$$= (1-\alpha)\ Old\ Estimate + \alpha\ Target,$$

Where $\alpha$ is the learning rate between zero and one and the part within the square brackets denotes an estimation error $\delta_t$ [Naeem, Rizvi and Coronato, 2020]. This structure allows for variance in the degree of consideration an agent has for rewards based $\alpha$, whereby for $\alpha = 1$ the most recent reward will be fully taken into consideration and at $\alpha = 0$ it will be disregarded.

As previously mentioned, Monte Carlo methods only update state values after an episode has ended. A notable example is pointed out in [Sutton and Barto, 2018] that illustrates this interaction and its relationship with TD. To summarise it, a theoretical scenario we are wanting to predict how long it will take one to travel from their workplace to home, the time of this journey is influenced by many external factors (traffic, weather conditions etc.) hence variance in one's travel is to be expected. When taking the Monte Carlo approach we would undertake our journey, encountering events at each step (states), and only after we have reached home do we calculate our total travel time (return) and go back to each of our previous states, updating the prediction of our journey time for each. The TD approach differs to this as, as opposed to waiting until the journey has ended, we update our predictions for each state during our journey. Meaning that we actively correct our estimates using the experiences we receive as we progress. In RL this is referred to as On-Policy learning.

An issue that arises from this updating process is that in TD, all of the rewards are not available to us, and we instead have access to the rewards at t+1 and estimations [Naeem, Rizvi and Coronato, 2020]. Here we can introduce the concept that allows us to generate new estimations based on our current estimations, bootstrapping. Bootstrapping denotes the behaviour of comparing the prediction of the current and next state and using the estimation error $\delta_t$ that results from this comparison to alter our prediction.

To explain this better we will begin by expanding on the TD update rule mentioned previously. Given the methodology for updates in TD, our target refers to our ability to sample one of the rewards available to us and the next value/action-value function. This could be achieved by using $R_{t+1} + \gamma v_\pi(S_{t+1})$ which would allow us to estimate $v_\pi(S_t)$. However, as mentioned previously we are having to base our estimations off of an estimation as we do not have access to $v_\pi(S_t)$ and so we instead use our current estimate of this, denoted as $V_t(S_t)$ [Garcia, 2019]. Thus, giving us a target of,

$$Target = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots \gamma^{k-1} R_{t+k+1})] = \mathbb{E}[R_{t+1} + \gamma V_t(S_{t+1})],$$

where $V_t$ is our estimate of $v_\pi$ at time t. Using this we can write our final update rule

$$V_{t+1}(S_t) \leftarrow V_t(S_t) + \alpha[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)],$$

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t),$$

The first equation/method outlined above is referred to as TD(0) or one step TD and it is the simplest TD method whereby a target is immediately formed at the next timestep and an update is made using the observed reward $R_{t+1}$ and estimation $V_t(S_{t+1})$ [Sutton and Barto, 2018]. Furthermore, the equations provide a mathematical illustration of bootstrapping through our estimation error $\delta_t$ (also referred to as the temporal difference) where the estimations for the next state $V_t(S_{t+1})$ are used when updating our prediction itself.

### 2.1.14 SARSA

Modified Connectionist Q-Learning is an on-policy TD control method proposed by [Rummery, 1994]. The design of MCQ-L is akin to all on-policy methods whereby the action-value function $q_\pi(s, a)$ is continually estimated for a behaviour policy $\pi$ and $\pi$ is changed towards greediness with respect to $q_\pi$ [Sutton and Barto, 2018]. The method itself has an update of the form

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

whereby, in an initial state $S_t$, an agent will take an action $a \in A(s)$ and observe their reward after moving forward to $S_{t+1}$, therefore having a new state and action. Our update method is the implored to update the action-value function and the policy is updated by selecting the action with the highest Q value. In the case where $S_{t+1}$ is the terminal state however, $Q(S_{t+1}, A_{t+1})$ will instead be defined as zero.

As you can see, our update rule utilises the quintuple of elements $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, thus the alternate name SARSA (State-Action-Reward-State-Action) was used (and suggested) by [Sutton and Barto, 2018]. For SARSA, as long as all of the state-action pairs are observed an infinite number of times, the algorithm is guaranteed to converge implying that the loss will reach a point within our range of error around a final value such that no further training will lead to improvement.

### 2.1.15 Q-learning

Quality (Q) learning is an off-policy TD control method outlined in [Watkins, 1989], off-policy being the notion that an optimal policy is sought/found independent to an agent's strategy and the actions they implore. The update method for Q-learning is defined as

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

In Q-learning, when not in the terminal state, our action-value functions are updated using the action-value function of the next state $\boldsymbol{S_{t+1}}$ and a greedy action $\boldsymbol{a \in A(s)}$, thus implying a direct approximation of $\boldsymbol{q_*}$ independent of our policy.

### 2.1.16 RLSVI

Randomised Least-Squares Value Iteration (RLSVI) was proposed in [Osband, Van Roy and Wen, 2016] by Osband et al. as an exploration strategy inspired by Thompson Sampling [Russon, Van Roy, Kazerouni, Osband and Wen, 2018] and has been described as an approach that shares the spirit of methods such as SARSA and TD.

In RLSVI, Bayesian Linear Regression [Minka, 2000] obtains a distribution, thought of as a posterior distribution, over plausible value functions which is then sampled. Actions are then selected such that they optimise said samples [Osband, Van Roy and Wen, 2016]. RLSVI stores full history to derive optimal policy.

The update method for RLSVI is defined by

$$Q_w(S_t, A_t) = Reward(S_t, A_t) + \gamma \max_{a_{t+1}} Q_w(Successor(S_t, A_t), a_{t+1}),$$

where $\boldsymbol{\gamma}$ is an arbitrary discount factor of the form, $\boldsymbol{v \sim N(0, \sigma^2)}$ where sigma is a hyperparameter [Ibarra, Ramos and Roemheld, 2016].

## 2.2 Critical Literature Analysis

### 2.2.1 A brief review of Ian Osband, Benjamin Van Roy, Daniel J. Russo and Zheng Wen, 2019, ‘Deep Exploration via Randomized Value Functions’ (Found at: https://arxiv.org/pdf/1703.07608.pdf)

The abstract clearly sets out the two main intentions of the paper, in both the presentation and analysis of methods that leverage randomised value functions and their analysis in regard to establishing and providing a regret bound that establishes statistical efficiency however, it slightly lacks in detail as it doesn’t entirely flow with the structure of the paper.

Explanations into each function are detailed with diagrams to explain key concepts however, certain aspects feel slightly underdeveloped due to a slight lack of both depth within the definitions in prior sections. The introduction in particular introduces certain concepts without fully explaining them damaging the flow of the piece.

The paper considers each problem as though the environment is unknown; thus no environmental based analysis is given and whilst a good comparison of efficacy is given, it’s difficult to visualise without the context of an environment to work in.

The conclusion reaffirms that their goal was to introduce RVFs as concepts and demonstrate their efficacy whilst outlining how certain results could differ in real life scenarios.

### 2.2.2 A brief review of Daniel Russo, 2019, ‘Worst-Case Regret Bounds for Exploration via Randomized Value Functions’ (Found at: https://proceedings.neurips.cc/paper/2019/file/451ae86722d26a608c2e174b2b2773f1-Paper.pdf)

The piece as a whole flows very well, particularly the transition from the introduction to the rest of the paper whereby concepts have been explained thoroughly in a well order manner, provided extra context and/or background where appropriate.

The paper only looks into RLSVI, comparing its efficacy to epsilon greedy and Boltzmann exploration and concluding RLSVI offers a fundamentally different, more sophisticated form of exploration than the latter two and that RLSVI is the most reliable of the three options. Whilst it does a good job of this by outlining comparison clearly in a well-ordered manner, the comparisons lack numerical data to illustrate the differences better with explanations instead being rooted firmly on the side of theory.

I feel as though the title and abstract could have been expanded on slightly to encompass the singular focus of the paper even more so though they do provide a generally good outlook of the paper overall.

The conclusion is left quite open, cementing that the intention of this paper was to serve as a foundation for future work and it proposes a lot of questions which is very fitting for the paper considering how Russo has conducted his explanations and analysis thought.

### 2.2.3 A brief review of Muddasar Naeem, Syed Tahir Hussain Rizvi and Antonio Coronato, ‘A Gentle Introduction to Reinforcement Learning and Its Application in Different Fields’ (Found at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9261348)

The piece is structured in a logical manner such that most concepts are only introduced when enough of their background has been covered. In regard to the explanations themselves, I found that the preliminary explanations about MDPs were conducted well, providing the reader with a sufficient amount of detail in a clear, concise manner. However, some explanations (such as those on rewards, bellman functions and value function) tended to be too simplistic such that it lead to the development of questions and confusion than would be expected for an informative piece.

When explaining temporal difference methods, too much focus was placed on the improvement of the methods themselves without first clearly explaining each. Furthermore, the background of the improvements mentioned had not been fully explained creating more elements of confusion.

Later aspects of the paper including a discussion piece and literature review were conducted very nicely, giving those interested in RL applications within certain fields a multitude of sources through which they could both broaden and deepen their knowledge on a particular subject.

Given its intention as an informative piece, I believe that it has been written well with a particular emphasis on the first and last third of the paper. I do however believe that for more complex topics, greater detail is required to ensure the reader understands everything.

### 2.2.4 My thoughts on the literature and rationale

Whilst grounded in older concepts, the research on the use of randomised value functions (specifically RLSVI) generally appears to be quite new. With this in mind whilst the outlining of key concepts within a multitude of papers in regard to this topic appears to be fairly consistent this could be attributed to a lack of material of which to build off of and a potential degree of bias amongst certain schools.

A general theme amongst papers appears to be prioritising explanations and comparisons through the use of theory as opposed to test data in smaller, more controlled environments and most of the work seems to be intended for use as a foundation on which other research can be conducted which only a small subset of papers seem to achieve well with others requiring a significant amount of additional context to be understood.

My initial aim when choosing a project was to pick a field that I was interested in and home in on one aspect of it in an attempt to strengthen and refine my knowledge on the topic. I thoroughly enjoyed reading about the concepts that were explained in certain papers to where I genuinely would like to conduct research on the implementation of RLSVI.

Furthermore, having noticed the lack of small-scale comparative pieces, that focus on every aspect of each implementation of RLSVI and their efficacy against each other, I believe that there is a need to delve into this topic further through this slightly different outlook in an attempt to compliment the other foundational work that has been created and create a potentially clearer, introductory piece to which others can build upon.

# Chapter 3: Analysis of Program and research methodologies

## 3.1 Background & Methodology

The following details my rationale behind the selection of the program I have conducted the experimentation aspect of my project with and the problem solving I had to carry out across a 3-4 week period in order to get the program to function.

### 3.1.1 Rationale

When initially planning my preliminary research, I set out with a clear criteria that I wanted my methodology to adhere to.

The research methodology should adhere to clear criteria in order to create a clean and easily understood working environment. Given that this project is both an introductory essay piece and an analytic test on RLSVI, it is important to avoid too much uncertainty and confusion that could prevent a clear understanding of its underpinnings.

Furthermore, I needed to create and/or use a program that was structured clearly for the purposes of explanation. This feeds into the aforementioned point but focuses on the code and overall program's structure as opposed to its output. Regardless of how well a program may carry out a task, presenting something that is poorly structured and difficult to understand in the context of education would have a negative effect on the quality of one's (a reader's) knowledge when coming away from the paper, potentially causing them to develop poor practices and/or suffer from an understanding standpoint.

I required a sufficiently large, Sparse Reward Environment in which I could experiment with RLSVI. Such that I could present the algorithms performance based on a fair environment that would be sufficiently challenging for a range of RL algorithms. Moreover, I sought the option of having access to comparative baselines (a measure of the performance of algorithms such as Q-Learning) within the space to compare the results of RLSVI to. As well as a program with the potential for modification such that I could making direct alterations to the implemented state of RLSVI and analyse the effect of these changes over a period of learning.

Open was the options of either finding existing projects that fit my criteria or implementing something myself. Each of which had their own merits with the latter being costly from a time perspective but would ensure adherence with the requirements I had outlined, whilst the former had the potential to take up a lot of time for research, potential debugging and modifications and any other unforeseen issues. Finding an existing project however, would potentially allow me to experiment in a space outside of the scope of my project itself given the time available to me.

After researching, keeping the prospect of both possibilities open, I found that existing projects with implementations of RLSVI were few and far between and those that were available would often be inaccessible for most from an educational standpoint with them not adhering to my criteria such as those used for [Zanette, Brandfonbrener, Brunskill, Pirotta and Lazaric, 2020] and [Osband, Van Roy and Wen, 2016].

After much deliberation over a lengthy period, I happened upon a program used within [Ibarra, Ramos and Roemheld, 2016] that appeared to provide a reasonably complex sparse reward environment that would aid in my explanations and analysis to a high degree, with the potential for adaptability. Given I had found something of this nature I decided it best for me to use this existing project as opposed to manually creating and implementing an environment and algorithms (RLSVI), potentially opening up options for me to expand on aspects of it as my work progressed.

### 3.1.2 Problems Encountered

Initially I attempted to run the program (using AngryBirds.py as __main__) on windows 10, in Visual Studio Code using Python 3.10.4. Furthermore, I installed the latest versions of the dependencies I had found when looking through the code myself as no supporting documentation in regard to what was required to run the program had been provided. These initial trials resulted in errors pertaining to the dependency Pymunk, the 2D physics library Chipmunk and general syntax errors within the code itself.

Given these errors I decided to conduct a more thorough investigation into the files attributed with the program to find any notable comments, instructions or general guidelines but was unable to find anything of note. I did however find the original python code that the project's environment was based upon, along with a text file detailing the requirements for the code (Pygame 2.0.1 and Pymunk 6.0.0), instructions of how to run the program and a video illustrating the (original) program being executed and running as expected.

I then attempted to run the original version of the program on my machine to see if the faults I was experiencing were only on my end but it worked perfectly well, thus I thought that my issues were solved and replicated the manner in which I ran the original program with the RL project I was having issues with. Whilst this appeared to fix the Pymunk and Chipmunk errors I had experienced, the program continued to throw numerous syntax errors that no amount of troubleshooting seemed to fix, to specify these syntax errors were in relation to line of Pygame code that were present.

Rather than continuously attempting to debug the program I decided to conduct further research into Pygame and Chipmunk themselves to see if they detailed any such issues. I also contacted one of the developers of the RL program and some third parties who worked with Pygame commercially but, as expected from a project created seven years ago, I did not receive any response from the former and the latter was unable to assist as they too were unsure as to what the issue was. When reading into the documentation of Pygame and Chipmunk I found that recent versions of Pygame included support for
Chipmunk on windows from Pymunk 5.0.0 onwards and that earlier versions did not include support for it. However, at the time I believed that I required Pymunk 6.0.0, thus did not see this as an issue.

This was until I noticed, on the GitHub for the original program, an edit had been made 18 months ago with the original requirements listing and other supporting documentation being extremely vague and not detailing the versions of the dependencies that were required other than a compatibility with Python 2.0 and 3.0. Given this development I listed the dates at which the RL project had been created and located corresponding versions for any programs, dependencies or etc. I required; namely, Python versions 2.7.10 to 3.5.1, Pygame ver. 1.9.1 and Pymunk ver. 4.0.0.

When conducting further research on Pymunk ver. 4.0.0 I found that, in accordance with a blog post made in 2013 by the developer, it did not support 64 bit windows and thus getting Chipmunk to work on such a platform would be quite an ordeal, no substantial instructions were given. From this point onward I then had two options, either to attempt to update the entirety of the RL algorithm I intended to work with such that it would work with the most recent versions of Pymunk and Pygame on windows 10 or to attempt to run the program on a corresponding (to the date of the program's creation) version of Ubuntu to which I chose the latter.

### 3.1.3 Program Requirements

The following outlines the conditions observed in order to get Angry Birds, Reinforcement Learning project to function:

- Ubuntu 18.04.5 LTS (older versions are no longer supported by Canonical)
- Python 3.7.8 (upon further testing, this version of Python appeared to be compatible with the program thus a downgrade was not necessary).
- Pymunk 4.0.0
- Pygame 2.1.2 (most recent, appeared to be compatible with the program when running on Linux)
- The removal of the "pymunk-4.0.0" folder (caused issues such as attempted overwrites and unwanted installations/reinstallation attempts) as well as modifications to aspects of the source code ("src") to reflect this.
- The existence of (based upon the GitHub page for the RL project) the "src" (folder and contents), "resources" (folder and contents), "results" (folder), "plots" (folder). In its current state the program does not create a new folder if one is not present and will instead throw an error. This is simple to fix but in its current iteration having the knowledge that these folders are required is useful. If Graphical plots already exist for a combination of variables when experimenting, they will be overwritten.

## 3.2 Explanation of Program

### 3.2.1 Overview Of Program

Angry Birds is a popular physics-based puzzle game in which players slingshot projectiles at targets for points. The projectiles are, as the name suggests, enraged birds who aim to breach forts constructed out of wooden beams and columns to kill pigs. The program utilises a simplified recreation of the game by [estevaofon, 2015] consisting of 12 levels, one type of bird (the red, standard bird), one type of target (standard pig models) and one material for beams and columns (wood).

A level is completed by hitting all of the targets on screen such that they disappear, this can either be done via direct collisions with the birds that are slingshot or impactful enough collisions with the wooden beams and columns surrounding them. A level can also fail if the player runs out of birds with targets remaining on the screen. In the case of a completion the game will move on to the next level. In the case of failure for a human player the current level would reset however, in our model if an agent fails they will be sent back to the first level. Finally, in the case of a successful completion a score will be calculated based on the number of unfired projectiles, beams and columns broken, and targets destroyed.

The code for the program is located in "src" [imanolarrieta and larsroemheld, 2015] with resources necessary for its execution located in "resources". It should be assumed that all Python files mentioned end with the PY file extension.

### 3.2.2 Environment

The environment for the program consists of four files: "polygon", "level", "characters" and AngryBirds. "polygon" defines the characteristics for the beams and columns and allows for the retrieval of the position, radius, vertices and velocity of any respective beams/columns object. "characters" defines the characteristics of the bird and pig objects and the retrieval of their position, radius and velocity, for bird object's direction, age and status. Following this, "levels" utilises "polygon" and "character" to create each of the 12 levels available, creating and denoting the initial object locations for each level respectively.

"AngryBirds" defines the games features in an object-oriented setting, using the syntax Pygame. Here basic methods pertaining to the games features are defined in manner that allows for feature extraction. AngryBirdsGame.__init__ initialises the game's display window, foundation of our physics engine and collision handlers for all of our objects. It also loads the resources for the game and creates a static 'floor' such that our objects cannot fall through the bottom of the screen and will instead react to it as they would with any solid surface. The characteristics of the slingshot with which projectiles (birds) are launched is defined by "sling action", this denotes how a bird object interacts with the sling and the effect the sling will have on a bird in relation to the resulting angle of impulse when the sling is interacted with. Finally, Mechanics such as game state alteration handling (on level completion, failure and restart) and collisions between objects, and the games event handling and updates (rules per frame) are also defined. When "AngryBirds" is run as main, the game will be initialised in a state intended for human/immediate-user interaction.

With our environment created, "abAPI" can now be used to define an approximate MDP for our RL algorithms to solve (attempt to). The class AngryBirdsMDP initialises our game and defines methods for the initial states, current state, actions and rewards. The GameState class defines the condition of the of a state in relation to the current level, the amount of each respective object left in the simulation and what condition this leaves the game in (checks if an episode has ended and computes if it has resulted in a win or loss).

### 3.2.3 Reinforcement Learning Algorithms

"util" acts as a utility file, containing the basic tools required to run episodic simulations of the following RL algorithms.

#### Q-Learning

"Q-Learner" contains the Q-Learning algorithm to be run in a simulation of the MDP, defined by the class QLearningAlgorithm. Alternatively, it will be asked for an action to perform in a given state and will use the feedback from said action to inform subsequent decision making. The QLearningAlgorithm.getAction method first returns an action in accordance with a simple $\varepsilon$-greedy policy, the result is handled by the QLearningAlgorithm.incorporateFeedback method which will take a $(s, a, r, s')$ tuple and update our weights using the standard Q-Learning update function. Finally, the algorithm incorporates the function approximation $Q_w(s, a) = w^T \phi(s, a)$ where $Q_w$ is self.getQ, $w$ is denoted by self.weights and $\phi$ is the feature extractor function.

#### RLSVI

Adapted from the RLSVI algorithm used in [Osband, Van Roy and Wen, 2016], "denseLearnerRLSVI" begins by defining our RLSVI agent in the class RLSVI. The class has been created such that the memory is a list of lists that can store a full history of $(s, a, r, s')$ tuples to derive an optimal policy. RLSVI.update_obs takes in transitions and adds them to memory after which RLSVI.update_policy recomputes and, by simulating a gaussian mixture with the assumption of independence for our sampling, conducts an in place update of $\theta$ parameters via a planning step. Finally, RLSVI.pick_action returns an action based on a greedy policy with respect to a sampled policy $\theta$.

The RLSVI_wrapper class is utilises the methods in RLSVI in a similar structure to "Q-Learner" such that it is compatible with the tuple-based features that have been implemented. The above classes mentioned have been implemented such that LSVI can also be simulated given that $\varepsilon \neq 0$. "sparseLearnerRLSVI" is implemented identically to "denseLearnerRLSVI" bar the treatment of sparce matrix operations (results in a computational complexity equal to the number of nonzero elements in the matrix) at all steps.

### 3.2.4 GameAgent and evaluation

"GameAgent" provides the framework for the application of our RL algorithms on our game. The class is split into two main aspects, angryAgent.getAngryBirdsActions returns a list of the possible actions an agent can make. These actions consist of the angle at which the agent can aim the slingshot and the distance the slingshot can be pulled back (launch power). The rest of the class consists of the feature extractors PP, NPP, NPPS & NPPO, all of which are explained thoroughly in [Ibarra, Ramos and Roemheld, 2016]. When run as main, a demonstration of a selected RL algorithms performance will be carried out, the appropriate function needing to be uncommented as per the user's intention.

The final class "evaluation" allows for the generation of results and graphs for the implemented RL algorithms with each type of feature extractor. This can be accomplished by uncommenting the required evaluator function for a particular algorithm and feature extractor combination.

# Chapter 4: Results and Analysis

## 4.1 Results

### 4.1.1 Q-Learning Performance

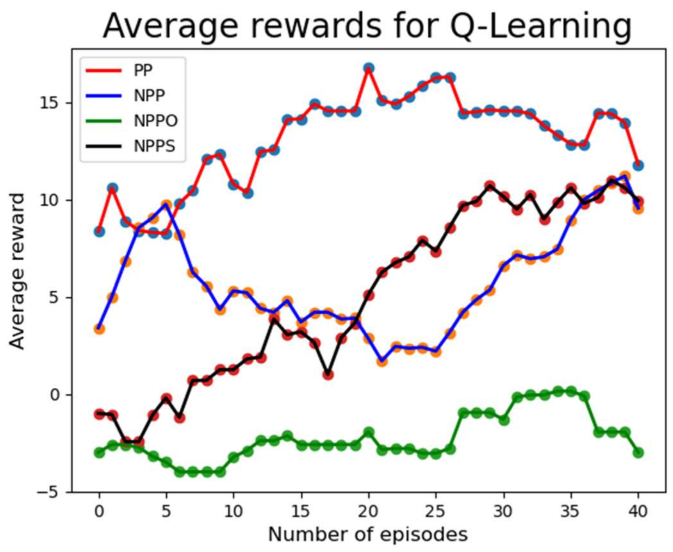

Figure 1. Moving average for the feature extractors PP, NPP, NPPO & NPPS for the Q-Learning Algorithm. The moving average is calculated of the next ten attempts after a number of attempts have been made (where an attempt is defined by any length of trial until failure is met and a restart occurs). The y-axis is in units of $10^4$ points.

### 4.1.2 LSVI Performance

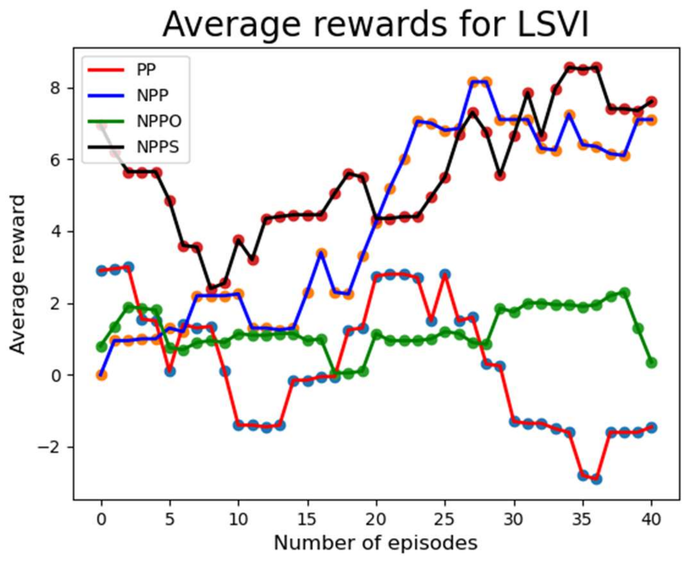

Figure 2. Moving average for the feature extractors PP, NPP, NPPO & NPPS for the LSVI Algorithm. The moving average is calculated of the next ten attempts after a number of attempts have been made (where an attempt is defined by any length of trial until failure is met and a restart occurs). The y-axis is in units of $10^4$ points.

### 4.1.3 Dense RLSVI Performance

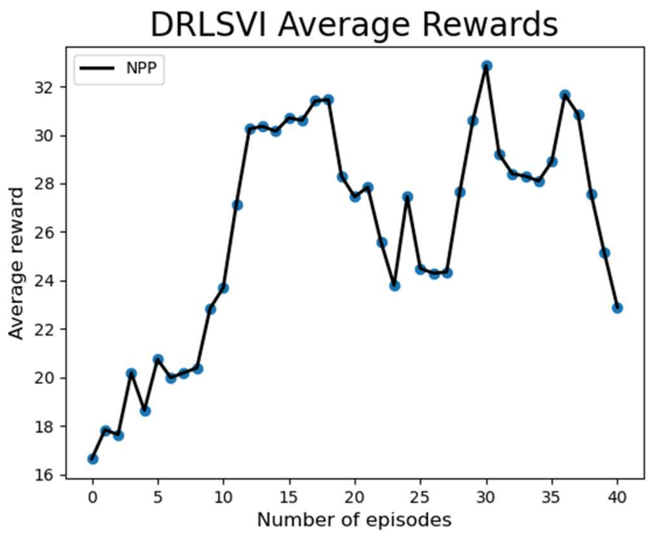

Figure 3. Moving average for the feature extractor NPP for the Dense RLSVI Algorithm. The moving average is calculated of the next ten attempts after a number of attempts have been made (where an attempt is defined by any length of trial until failure is met and a restart occurs). Feature extractors PP, NPPO & NPPS were experimented with however, the algorithm was not able to converge during testing. The y-axis is in units of $10^4$ points.

### 4.1.4 Sparse RLSVI Performance

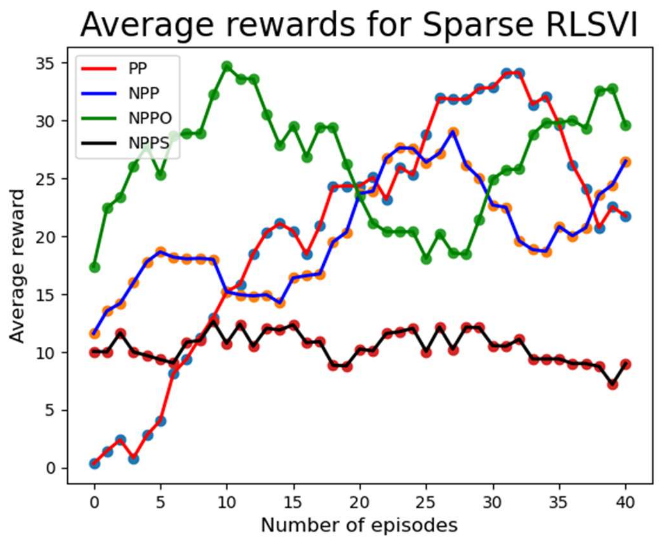

Figure 4. Moving average for the feature extractors PP, NPP, NPPO & NPPS for the Sparse RLSVI (referred to as SRLSVI going forward) Algorithm. The moving average is calculated of the next 10 attempts after a number of attempts have been made (where an attempt is defined by any length of trial until failure is met and a restart occurs). The y-axis is in units of $10^4$ points.

### 4.1.5 Comparison of rewards for NPP features

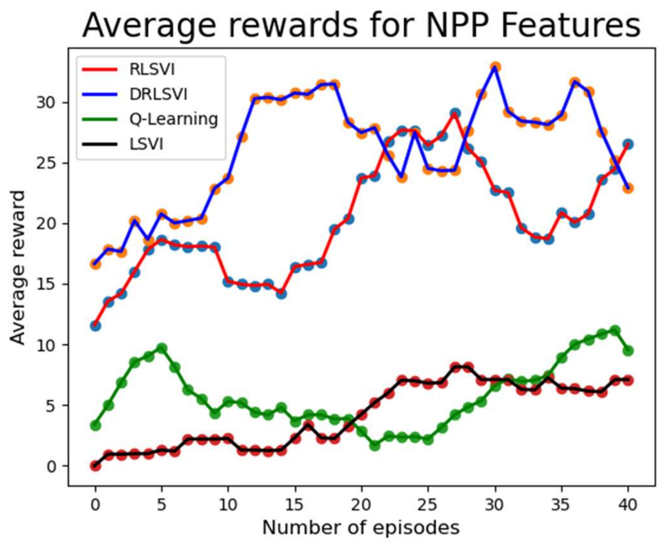

**Figure 5.** Moving average for the Algorithms SRLSVI, DRLSVI, Q-Learning and LSVI for the feature extractor NPP. Both DRLSVI and SRLSVI were significantly faster and achieved more rewards than Q-Learning and LSVI. The performance of Q-Learning was marginally better than that of LSVI. The performances of DRLSVI and SRLSVI were similar however, DRLSVI was faster initially. The moving average is calculated of the next 10 attempts after a number of attempts have been made (where an attempt is defined by any length of trial until failure is met and a restart occurs). The y-axis is in units of $10^4$ points.

### 4.1.6 Average Trials to complete a level

| Model | | Number of trials to complete a level | | | |
|---|---|---|---|---|---|
| Algorithm | Features | Level 0 | Level 3 | Level 6 | Level 9 |
| Human | -- | 1 | 2 | 5 | 5 |
| Q-Learning | NPP | 1 | 4 | 14 | - |
| LSVI | NPP | 2 | 7 | - | - |
| DRLSVI | NPP | 1 | 2 | 4 | 8 |
| SRLSVI | NPP | 1 | 2 | 6 | 7 |

**Table 1.** Average number of trials needed to finish levels, SRLSVI and DRLSVI learnt at an impressive rate in mid-late levels, achieving comparable results to and even surpassing our human baseline at a point.

### 4.1.7 Maximum Performance

| Model | | Highest Recorded Performance | |
|---|---|---|---|
| Algorithm | Features | Score | Level |
| Human | -- | 435000 | 11 |
| Q-Learning | NPP | 375000 | 6 |
| LSVI | NPP | 170000 | 4 |
| DRLSVI | NPP | 615000 | 9 |
| SRLSVI | NPP | 650000 | 9 |

**Table 2.** Maximum level and scores achieved. In terms of score per levels completed, Q-Learning, DRLSVI and SRLSVI outperformed our human player. For levels completed, DRLSVI and SRLSVI did the best of the algorithms evaluated, followed by Q-Learning and finally LSVI.

## 4.2 Analysis

### 4.2.1 Comparison of feature extractors

We attempted to compare our four feature extractors' performances on each of the algorithms and were successful in three instances as displayed in figures 1, 2 and 4.

For Q-Learning and LSVI, NPPO started from a low baseline and remained fairly stagnant, getting stuck in the earlier levels. In contrast to this NPPO did well for RLSVI starting off with a high baseline but showing substantial signs of improvement over time though, as evident in figure 4 NPPO did present a noticeable amount of variance between its peaks and troughs.

PP started off with mid to high baselines in figures 1 and 2 but stagnated in a comparable manner to that of NPPO. Figure 4 however illustrated that with RLSVI, PP started off low but showed constant signs of improvement across out tests.

Though to a lesser degree in figure 1, NPP displayed an average to high learning rate across the board when compared to the other feature extractors for each respective algorithm.

Finally, NPPS shows a similar performance to NPP in figures 1 and 2 however, it underperforms in figure 4, remaining low and stagnant. Given RLSVI had the greatest amount of variability between feature extractors given its scale, it can be seen that NPPs performance was inadequate.

### 4.2.2 RLSVI vs LSVI

As evident in figures 2 and 4, RLSVI vastly outperformed LSVI for all feature extractors bar NPPS. When comparing the two algorithms using NPP in figure 5 it can be seen that RLSVI had both a higher base line and rate of improvement than that of LSVI. Furthermore, tables 1 and 2 illustrate that within my experiments LSVI was only able to perform at just under half of the standard of RLSVI.

### 4.2.3 RLSVI vs Q-Learning

Overall RLSVI performed better that Q-Learning, being able to perform significantly better with 3/4 feature extractors (figures 1 and 4) and the increased capability RLSVI presented in tables 1 and 2 whereby it was able to learn faster as levels progressed and reach a further point than that of the Q-Learner. This being said, when comparing scores to the number of levels completed in table 2, the Q-Learner was able to achieve a relatively high score, higher than that of our human baseline relative to the number of levels completed.
It should be noted that the memory requirements of RLSVI were significantly higher than that of the Q-Learning algorithm due to the requirement of storing an history of observed features [Ibarra, Ramos and Roemheld, 2016].

### 4.2.4 RLSVI vs DRLSVI

When using feature extractor NPP, RLSVI and DRLSVI performed relatively similar with each other with DRLSVI maintaining a slightly higher reward count throughout (figure 5). In accordance with tables 1 and 2 RLSVI and DRLSVI's performance was such that they could be considered equal with each other within a small degree of error.
It is worth noting that due to its high memory requirements, DRLSVI was unable to converge for PP, NPPO & NPPS. It is worth noting that simulations using DRLSVI took significantly longer to carry out than that of RLSVI.

# Chapter 5: Evaluation

## 5.1 Future Developments

Over the course of this project, I experienced multiple (real world) setbacks that hindered my progress with this project greatly and lead to some of my project goals to being simplified for the sake of time. Following these areas of improvement have been identified that expand this specific project and other that build on top of the RLSVI algorithm itself.

In regard to the explanations of the RLSVI algorithm itself, I believe them to be lacking despite the background knowledge provided being substantial. I believe it would have been beneficial to conduct a more in-depth mathematical analysis of the algorithm as well as present it in the for of pseudocode with a breakdown of exactly how each aspect of the algorithm works.

After using the Angry Birds RL program to obtain my results I believe that a restructuring of the program would have been beneficial for the purposes of education, many methods and functionalities of the program have been left unclear by it's creators and thus a redesign would allow for other experiments to be conducted with a greater level of ease.

Though the sparse matrices alteration was evaluated and proved to be beneficial in terms of hardware-based performance. It would have been interesting to make substantial changes to the algorithm and evaluate the effects and interactions that result from this. Particular examples include the potential implementation of deep exploration methods to allow the algorithm to create a state representation based off of the data it is learning from as opposed to the hand-design representation that RLSVI currently requires. As well as the exploration of different distributions such as Beta and Dirichlet Optimism, observing the impact of sampling from these other distributions.

## 5.2 Final Words

I was able to partially meet my goals for this project, having created a fairly comprehensive introduction to RLSVI and conducted an analysis of the algorithms efficacy in a simplified, sparse reward environment in comparison to LSVI and a Q-Learning algorithm with function approximation. Furthermore, I believe that I conducted this experimentation and analysis in a manner than enhances the readers practical understanding on the subject at a greater level than the research papers I encountered. However, I was not able to delve as deep as I would have liked to for the latter portion of the concepts, I explained such that a greater foundation and level of understanding for RLSVI could have been formed (from example an explanation of Thompson sampling and an introduction to Bayesian Linear regression). Furthermore, I was not able to alter the algorithm I was working with such that I could evaluate the effect of multiple alterations to the algorithm in a controlled environment. Overall, I am still pleased with the work I have carried out though I know it could have been improved upon vastly and believe this will assist those new to the subject (as I was) to gain an overall understanding of the algorithms in question, to a sufficient standard through one comprehensive source.

# Appendix A – Project Risk Analysis

| Risk Description | Impact on Project | Likelihood | Impact Rating | Preventative Measures |
|---|---|---|---|---|
| Environment is unsuitable for RL implementation | As the groundwork for the practical aspect of the project, if this is done incorrectly nothing can function/be tested. | Low | High | Do sufficient research, test implementations, and consult supervisor for his feedback on the environment. |
| RVFs implemented incorrectly | Incorrect methodology will invalidate the purpose of my testing and comparisons. | Medium | High | Do sufficient research, test implementations, and consult supervisor in any issues in my understanding occur. |
| Misunderstanding of foundational concepts | May cause fundamental mistake in terms of implementation and testing/evaluation. | Medium | High | Thoroughly study and make use of the material I have gathered for my project. |
| Inability of agents to learn due to mistakes | Results from testing will be incorrect. | Medium | High | Be extra mindful about my implementation of things, clearly comment all areas to be able to pinpoint items of interest/concern |
| Not enough testing (volume) | Not enough data to compare different methodologies reliably. | Medium | High | Set aside sufficient time for testing as I have in my Gantt chart. |
| Testing lacks detail (written) | Failure to convey research in a research-based project. | Low | High | Thoroughly review my results and set aside a sufficient amount of time for a write up both during and after practical testing. |
| Loss of work | Will result in both time and work quality complications. | Low | High | Back up my work at multiple stages to a few cloud based platforms. |
| External Time Constraints | Will cause delays and negatively affect work quality. | Medium | Medium | Manage my time better and adhere to the schedule I've set for myself. |
| Illness | Delays as I would be unable to complete certain tasks both for the project and for other university work. | Medium – High | Medium | Inform external circumstances team (if necessary) and supervisor for assistance. |
| More unforeseen Circumstances | I could be put in another situation where external factors lead to delays as a whole. | Medium | Medium | Inform external circumstances team (if necessary) and supervisor for assistance. |

# References

Sarker, I.H., 2021. Machine learning: Algorithms, real-world applications and research directions. SN Computer Science, 2(3), pp.1-21.

Howard, R.A. (1972). Dynamic programming and Markov processes. Cambridge, Mass.: M.I.T. Press.

Heyman, D.P. and Sobel, M.J. (1990). Stochastic models. Amsterdam: North-Holland; Elsevier.

Tranos, D. and Proutiere, A., 2021. Regret Analysis in Deterministic Reinforcement Learning. arXiv preprint arXiv:2106.14338.

Osband, I., Van Roy, B. and Wen, Z., 2016, June. Generalization and exploration via randomized value functions. In International Conference on Machine Learning (pp. 2377-2386). PMLR.

Sutton, R.S. and Barto, A. (2018). Reinforcement learning : an introduction. Cambridge, Ma ; Lodon: The Mit Press.

M. Naeem, S. T. H. Rizvi and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," in IEEE Access, vol. 8, pp. 209320-209344, 2020, doi: 10.1109/ACCESS.2020.3038605.

Coggan, M., 2004. Exploration and exploitation in reinforcement learning. Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University.

Kunz, F., 2000. An introduction to temporal difference learning. In Seminar on autonomous learning systems (pp. 21-22).

Bellman, R., 1954. The theory of dynamic programming. Bulletin of the American Mathematical Society, 60(6), pp.503-515.

Garcia, J.F.H., 2019. Unifying n-Step Temporal-Difference Action-Value Methods.

Rummery, G.A. and Niranjan, M., 1994. On-line Q-learning using connectionist systems (Vol. 37, p. 14). Cambridge, UK: University of Cambridge, Department of Engineering.

Watkins, C.J.C.H., 1989. Learning from delayed rewards.

Russo, D.J., Van Roy, B., Kazerouni, A., Osband, I. and Wen, Z., 2018. A tutorial on thompson sampling. Foundations and Trends® in Machine Learning, 11(1), pp.1-96.

Minka, T., 2000. Bayesian linear regression. Technical report, MIT.

Ibarra, I.A., Ramos, B. and Roemheld, L., 2016. Angrier birds: Bayesian reinforcement learning. arXiv preprint arXiv:1601.01297.

Zanette, A., Brandfonbrener, D., Brunskill, E., Pirotta, M. and Lazaric, A., 2020, June. Frequentist regret bounds for randomized least-squares value iteration. In International Conference on Artificial Intelligence and Statistics (pp. 1954-1964). PMLR.

estevaofon. 2015, January. “Angry birds in python”, [online at https://github.com/estevaofon/angry-birds-python/tree/develop, accessed 5 Jun 2022] imanolarrieta. and larsroemheld., 2015, December. “angrybirds”. [online at https://github.com/imanolarrieta/angrybirds, accessed 19 April 2022]