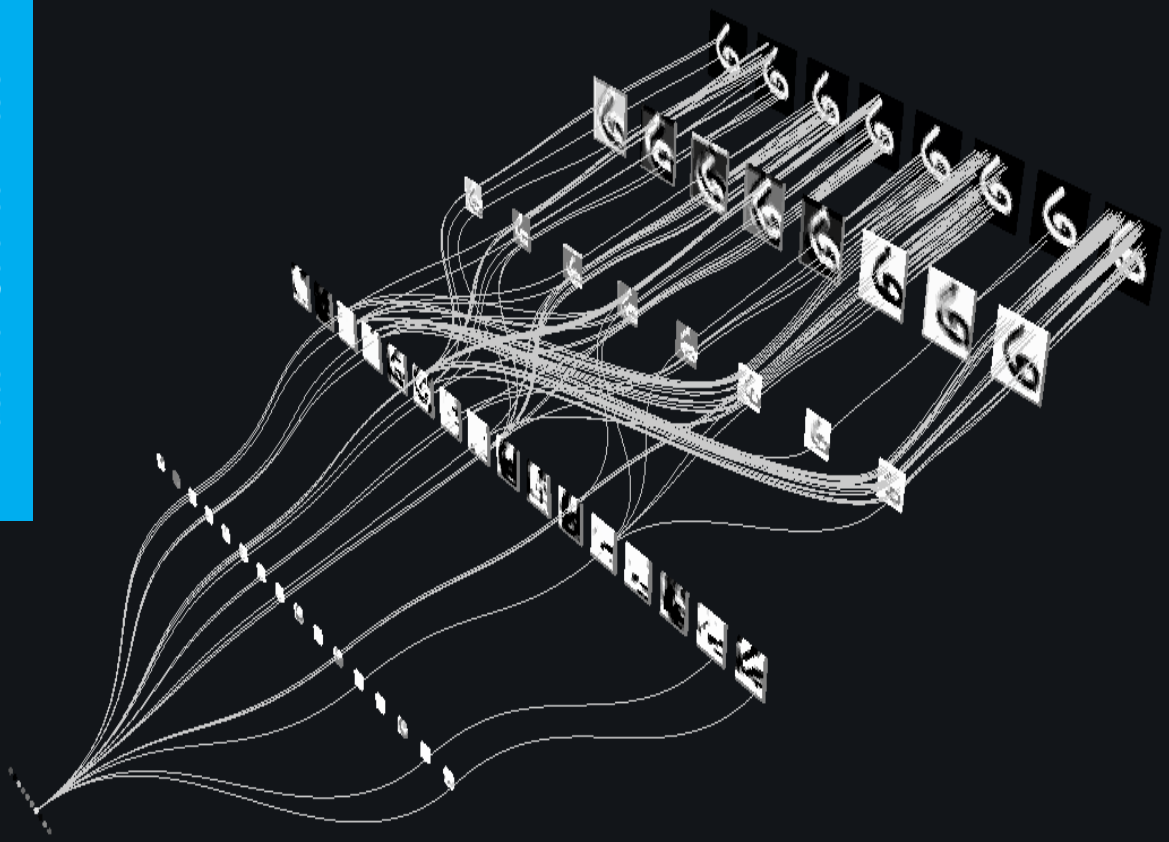
Reinforcement Learning Across Timescales

Siddharth Ravi

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Reinforcement Learning Across Timescales

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Siddharth Ravi

August 11, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Table of Contents

Glossary

List of Terms

s	State
$\mathcal{S}$	Set of all possible states
a	Action
$\mathcal{A}$	Set of all possible actions
$R_t(s_t, a_t)$	Reward at timestep t, when at state $s_t \in \mathcal{S}$ and taking action $a_t \in \mathcal{A}$
γ	Discount Factor
ϵ	Exploration factor
$P_a(s, s')$	Probability of transitioning from state s to s' when taking the action a
G_t	Returns or the accumulated discounted rewards
$Q(s, a)$	Action-value function
$Q^*(s, a)$	Optimal Action-value function
$V(s)$	Value function
$Q(s, a)$	Optimal Value function
$A(s, a)$	Advantage Function
$A^*(s, a)$	Optimal Advantage function
π	Policy
η	Advantage learning scalar multiplier
τ	Bellman Operator
τ_{AL}	Advantage Learning operator
dt	Sampling time
L_i	Loss function at iteration i
θ	Action-value network weights
θ^-	Target-network weights
α	Advantage stream weights (dueling network)
β	Value stream weights (dueling network)

Abstract

This project addresses a fundamental problem faced by many reinforcement learning agents. Commonly used reinforcement learning agents can be seen to have deteriorating performances at increasing frequencies, as they are unable to correctly learn the ordering of expected returns for actions that are applied. We call this the disappearing reinforcements problem. Moreover, truly multi-task reinforcement learning is only possible when agents are able to operate across frequencies, as different platforms operate at different frequencies. Most algorithms from control theory working on similar tasks, on the other hand, show improved performances when their operating frequencies are increased. This suggests that addressing disappearing reinforcements should enable reinforcement learning agents to have improved performance and generalization ability across timescales and tasks.

In this project, we show that disappearing reinforcements is an effect seen independent of the function approximator used in reinforcement learning, and is instead of a more fundamental nature. We explore both theoretically and empirically the relationship between agents and their performances at increasing frequencies. We show that two specific types of agents from literature address the problem, and we benchmark the agents' performances with novel benchmarking measures inspired from control theory. Finally, we create a novel agent we call the dueling advantage learner, by combining both approaches from the state-of-the-art. We then benchmark the different agents across frequencies and tasks, and our agent is seen to outperform each of the individual approaches on the majority of the tasks.

Acknowledgements

I would like to thank my supervisors Dr-Ing. Jens Kober and Ir. Tim de Bruin for their support. This thesis could never have been completed without their advice and encouragement. They have managed to cultivate in me an interest not only in the field of reinforcement learning, but also for doing research in general.

I'd also like to thank my friends and colleagues. They've been a constant source of help and support. They were excellent sounding boards for my ideas and have saved me from much trouble.

Finally, family. I owe a great deal to their endless patience and to the kindness that was extended to me during the length of this thesis, as well as throughout my life.

Delft, University of Technology
August 11, 2017

Siddharth Ravi

"Each piece, or part, of the whole of nature is always merely an approximation to the complete truth, or the complete truth so far as we know it. In fact, everything we know is only some kind of approximation, because we know that we do not know all the laws as yet. Therefore, things must be learned only to be unlearned again or, more likely, to be corrected. The test of all knowledge is experiment. Experiment is the sole judge of scientific truth"

— *Richard Feynman, The Feynman Lectures on Physics*

Chapter 1

Introduction

Machine learning is a branch of computer science that emerged as a result of our desire to understand if the computer can be programmed to learn the way humans do. It evolved from the study of computational learning theory in artificial intelligence in the early-to-mid 20th century. The field deals with the development and construction of algorithms that can learn and make predictions from data, without following explicit instructions to perform these tasks.

Reinforcement learning, the topic central to this project, is an area of machine learning that is inspired from behaviorist psychology. It bases itself on the control of human behavior in response to certain stimuli, or as a consequence of the history of the individual's stimulus-response characteristics. Reinforcement learning could be summarized as a computational approach to learning from interaction. The field concerns itself with how software agents iteratively learn to associate situations in their environment to the actions they can take, in order to maximize a certain notion of a cumulative numerical reward. Two factors, *trial and error* based learning, and the taking into account of *delayed rewards* are the defining characteristics of reinforcement learning. The agent needs to decide between *exploration* of the environment to gather more information, and *exploitation* of the best decision given its current information obtained from prior exploration [1][2].

Since its start in the 1980s as a plan to replicate biology's learning mechanisms, the field of reinforcement learning has come a long way. Although biological plausibility is still of interest, reinforcement learning research has branched off to fields as varied as the predicting of financial market movements to robotics [3][4]. It even finds its uses in areas such as information theory and game theory [5][6]. Interest in reinforcement learning has also blossomed over the previous years. One of the major breakthroughs in the field was when DeepMind created AlphaGo, a reinforcement learning algorithm that defeated the then reigning world champion Lee Sedol 4-1 at the game of Go [7]. This was a landmark victory, as Go, a game thought to require intuition, creativity and strategic thinking, was then considered a difficult problem for the field of AI, much more so than chess. Improving reinforcement learning has since been widely recognized as an important human endeavour, along the path to achieving general human-level intelligence.

That said, reinforcement learning is still far from being a solved problem, and is known to have its fair share of challenges. Our motivation behind embarking on the project lies in a handicap that affects most current reinforcement learning agents. Upon operating at higher frequencies, these agents perform increasingly worse, and eventually completely stops learning as the time elapsed between interactions with its environment approaches zero. This project will be referring to this as the problem of 'disappearing reinforcements'.

An intuitive understanding of disappearing reinforcements can be obtained when analyzing the reward gathered by the agent. A typical reinforcement learning agent obtains a rewards at each intermediate state it reaches by performing actions, before the completion of a certain task. A higher frequency of operation means that there are more intermediate states, and thereby more reward signals to be accounted for. This makes it difficult in the long run for the agent to differentiate between a high reward action from a low(-er) reward one. As the frequency reaches continuous time, individual action contributions go to zero. The problem can also be understood by analyzing the relationship between reward signals and states. As the frequency increases, the rewards, which are usually a function of the state, also move closer in value. This makes it difficult to correctly ascertain the relative importance of an action at a certain state. Most reinforcement learning algorithms are therefore currently deployed only at lower frequencies, which greatly limit their capabilities.

This project begins by investigating two important questions relating to the choice of sampling frequencies for a reinforcement learning algorithm. How does the sampling frequency affect the performance of an algorithm? Is a higher frequency always better? To answer these questions, we first look at another field closely related to reinforcement learning, *optimal control*. We do this to understand how other related fields of study decide upon an optimal frequency for sampling, and to understand theoretically defined bounds within which algorithms usually work. This is discussed in Chapter 2.

We find that alleviating the disappearing reinforcements problem is of utmost importance. The development of an improved algorithm that is effective even at higher frequencies would greatly improve the operating capabilities of the algorithm. The primary focus of this project therefore is on solving the disappearing reinforcements problem, and we do this for a specific class of reinforcement learning algorithms called *off-policy model-free value-based methods*. We explain the reasoning behind this choice, along with the fundamentals of reinforcement learning algorithms in Chapter 3.

We then move on to study the problem in detail in Chapter 4, to see exactly how performance of a reinforcement learning algorithm is affected by changes in frequency. For this we define an algorithm's performance by defining novel benchmarking criteria, taking inspiration from control theory. We create a simple task to analyze the agent, in order to see how the agent's fundamental characteristics change with respect to increasing frequency, and other factors.

Chapter 5 deals with the study of state-of-the-art agents that aim to address the disappearing reinforcements problem. We theoretically analyze agents of different kinds that appear in literature, and benchmark these with respect to the benchmarking metrics we defined in the earlier chapter. We then move on to improve on the state-of-the-art by proposing novel solutions in Chapter 6. The agent we named the dueling advantage learner is benchmarked on simple tasks with those same benchmarks defined earlier.

Chapter 7 deals with the comparison of our agent's performance to the state-of-the-art, through extensive testing on commonly seen tasks in literature. The idea behind the tests is

to simulate different real-world scenarios which the agent might face. This exercise also creates a robust benchmark of value based reinforcement learning algorithms across timescales, which is seldom seen in literature. Finally, Chapter 8 describes our conclusions and directions that can be taken in the future to improve the field of study.

It is essential for the reader to understand that the choice of sampling frequencies for operating reinforcement learning algorithms is a topic rarely discussed in literature, even though they are of great importance. Studies have been done towards the development of better agents that do well at different tasks, but it is rarely that these agents are explicitly studied across different timescales. We do so in this thesis. It is also to be understood that the algorithm's behaviour is highly dependent on many other factors including errors from the system dynamics and initialization of its constituent parameters. But reinforcement learning, as a concept, is required to generalize across tasks, achieving satisfactory levels of performances across a variety of systems operating at different frequencies. Only such a multi-task agent would be effective when operating in the complex dynamics of the real world.

Chapter 2

An Optimal Control Perspective

The field of optimal control is one that is closely related to reinforcement learning. From an optimal controller design perspective, control systems have an optimal frequency range of operation, which imposes range limits on the sampling rates for the control algorithm. These limits are mostly lower limits, as the algorithms usually improve their performances as frequencies increase. They are also mostly based on criteria such such as the *bandwidth* of control, that the user hopes to achieve. Too high a sampling frequency imposes a high load on the control system. When dealing with control systems with lower computational power, too high a sampling frequency also introduces quantization errors [8]. Design criteria also impose hard lower limits on the frequency at which sampling should be done.

Thus when designing reinforcement learning algorithms, it is important to keep in mind these frequency constraints that exist for the control system. Designers of reinforcement learning algorithms also often tend to ignore the existence of these limits. Thus it is important to look at the problem of reinforcement learning from the perspective of it being a control system.

The aim of this chapter is to familiarize the reader with how algorithms from the paradigm of optimal control, a field closely related to reinforcement learning, behave at different frequencies, and to understand how sampling frequencies are chosen by the designer according to her design criteria. Looking from this perspective is advantageous as it serves to provide the reader with insights into how optimal control algorithms behave in stark contrast to how common reinforcement learning algorithms behave, under similar operating conditions with respect to sampling frequencies.

The rest of this chapter is structured as follows. Section 2-1 gives the outline of the optimal control problem. Section 2-2 delves into the choice of the sampling frequency of optimal control algorithms. This is to make the reader understand the process of how the sampling frequency is chosen according to the design requirements of the problem in question. This is also to show how the choice of these sampling frequencies are in ranges designed according to certain rules of thumb from expert recommendations, rather than according to mathematical formulations leading to a single optimal value. Section 2-3 provides commentary on the existence of theoretical upper limits to the sampling frequency chosen.

2-1 Looking from a control theoretical perspective

Control algorithms typically exhibit better performance when operating at higher frequencies. Their operating frequencies are generally selected by rules-of-thumb based on expert recommendations, rather than by any clearly defined mathematical formulae. We shall next look at how the optimal control problem is formulated, and what these criteria for choosing sampling frequencies are.

2-1-1 Formulating the optimal control problem

The optimal control problem deals with the defining of a control law for a system so that a certain criterion for optimality is achieved [8]. For states x and actions u, the optimal control paradigm allows to determine good candidate values of the optimal feedback gain K in the control law $u = -Kx$.

Given a discrete plant $x(k+1) = \phi x(k) + \Gamma u(k)$, the optimal control task seeks to pick $u(k)$ such that a cost function defined by $\mathcal{J} = \frac{1}{2}\sum_{k=0}^{N}\left[x^T(k)Q_1x(k) + u^T(k)Q_2u(k)\right]$ is minimized. Here Q_1, Q_2 are symmetric non-negative definite weighting matrices to be selected by the designer. Various solutions to the optimal control problem exists such as by solving the Riccati equation.

2-2 Choosing sampling frequencies

Sampling continuous-time systems provide us with a discrete-time system with system matrices that depend on the sampling period. It may happen that a control system loses its observability and reachability upon choosing too low a sampling period, leading to disastrous results [9]. The sampling period's choice also depends heavily on the purpose of the system.

From a control perspective, a low sampling rate implies more time for computers to calculate. This also denotes a lower cost per function and thereby a less demand on analog to digital conversion speed for digital computers. Decisive factors when choosing a lower limit on the sampling rate are:

- *Tracking effectiveness* as measured by closed loop bandwidth, and the time response characteristics such as the rise time and settling time
- *Regulation Effectiveness* with respect to error response to random plant disturbances.
- *Measurement noise errors* and *pre-filter design methods.*

2-2-1 Tracking effectiveness

As the sampling rate is lowered, approximations taken into account during computation give rise to system instabilities. According to the Nyquist-Shannon sampling theorem [10], in order to reconstruct an unknown band-limited continuous signal from that signal, one must use a sampling rate at least twice as fast as the highest frequency contained in the unknown

signal. Translating the theorem into a design specification for the effective tracking of the command input with system bandwidth[1] as frequency, sampling should take place at twice the closed loop bandwidth of the system.

Expressing the condition in terms of the bandwidth of the closed loop control system, for the system to track input signals,

$$\omega_s \geqslant 2\omega_{BW} \tag{2-1}$$

where ω_s is the sampling frequency and the ω_{BW} is the bandwidth of the signal. This could be termed as the fundamentally lowest frequency at which a signal should be sampled. Theoretically, this limit is insufficient for efficient operation. For a rise time of approximately 1 sec ($\omega_{BW} = 0.5Hz$ [8]), typically a sampling frequency of 10-20 Hz is ideal, since this implies that the sampling frequency is around 20 to 40 times the bandwidth.

2-2-2 Regulation effectiveness

Disturbance rejection is a very important criterion in deciding the sampling frequency, if not the most important. What is also crucial during design is the rejection of higher frequency components in the disturbance signal. A high sampling frequency in comparison with the frequencies in noise disturbance, implies that there is potentially no loss from digital systems when compared with a continuous controller. On the other hand, a low sampling frequency with respect to noise characteristics means that the response is similar to that without a specific control action.

Franklin et al., [8] define a rule of thumb to further define the criteria more quantitatively:

$$\omega_s \geqslant 20\omega_{BW} \tag{2-2}$$

2-2-3 Measurement noise errors and pre-filter design

Oftentimes, there is the use of an analog pre-filter between the sensor and the sampler. Designing the control system is difficult if the pre-filter has to be incorporated into the design procedure [11]. This can be eliminated by choosing the sampling frequency, $\omega_s \gg \omega_{BW}$ (the bandwidth), as there is no potential phase lag from the pre-filter. One possible metric to use is to choose the sampling frequency $\omega_s \approx$ (30 to 100) times ω_{BW}. On the other hand, including the pre-filter phase lags at system bandwidth means that one needs to include analog pre-filter characteristics during the control design phase [8].

2-3 Upper limits on the sampling frequency

When systems with low processing power (such as 8-bit microprocessors) are used for computing, the sampling frequency also has an upper limit. Introduction of quantization errors

[1] -3dB crossing frequency, obtained from closed-loop Bode plots [8]

[11] does not allow too high a sampling frequency to be used. This error is negligible when it comes to higher bit microprocessors as controllers, hence theoretically providing very high upper limits for the sampling frequency that can be used. Thus from a optimal control design perspective, these effects could hence be ignored to a large extent in such higher bit systems. This also points to the fact that a higher sampling frequency is usually beneficial for control.

2-4 Summary

In the paradigm of optimal control algorithms, sampling frequency is decided according to the control system characteristics desired by the user. It should also be understood that these choices are based on expert recommendations, and is usually a range of good frequencies which have been seen to provide good results during the control task. Most of these recommendations also do not have an upper bound, which leads to potential confusion. How high is too high? An upper limit is only theoretically provided by the computational limits of the control system, which the frequency when errors caused due to quantization appear. This is also usually left to the user to determine.

Chapter 3

Reinforcement Learning- History and Concepts

Modern reinforcement learning (RL) is a combination of two specific pre-cursory approaches that converged during the 1980s. One approach concerned understanding animal learning by trial and error. This thread can be illustrated by the masterful experiments conducted by Ivan Pavlov to demonstrate the theory of classical conditioning, or the learning occurring when a biologically potent stimulus (e.g. food) is paired with a neutral stimulus (e.g. a bell) [12]. Pavlov (1927) and Hull (1943) also serve as the basis of understanding the idea that stimuli produce after-effects in the nervous system (used to model eligibility traces, a concept often used in reinforcement learning) [13].

The other thread concerns the theory of optimal control, and how efforts were directed at using the concepts of *value functions* and *dynamic programming* to arrive at the solution. This effort was championed by Richard Bellman in the 1950s based on a theory devised by Hamilton and Jacobi in the 19^{th} century [14]. Both the theory of optimal control and the theory of RL are also very closely related, in that both address the problem of finding an optimal policy. Optimal control algorithms work with an inherent assumption of perfect knowledge about a system description in form of a model, which breaks down in the presence of computational approximations. Reinforcement learning is often more robust, because of its measured-data driven approach and its environmental-interaction based rewards system [4]. These seemingly distinct fields combined to form the basis of the theory of what we know as modern reinforcement learning.

The rest of this chapter is structured as follows. Section 3-1 gives a broad overview of the theory behind reinforcement learning algorithms. Section 3-2 explains the meaning of the elements of reinforcement learning agents that are discussed in the thesis. Section 3-3, Section 3-4 and Section 3-5 provide an overview of the different classes of reinforcement learning algorithms that exist, and also dives into some algorithms commonly seen in literature.

3-1 Theory behind reinforcement learning

The framework behind RL explains that problems of goal related learning can be reduced to three signals passing between an *agent* and the *environment* it interacts with. These are:

1. A signal to represent the choices made by the agent. (actions)
2. Another signal to represent the basis on which choices are made. (states)
3. A scalar signal to define the agent's goals. (rewards)

States, actions and associated numerical rewards vary from problem to problem, while the framework associated remains the same. The aim of the reinforcement learning agent is to maximize the rewards it obtains over a certain predefined length of time when doing the experiment. Most reinforcement learning tasks can also be neatly broken down into sequences of agent-environment interactions between *initial* and *terminal* states, and each sub-sequence of interactions between initial and terminal states is called an *episode*. The environment promptly resets to the initial state upon reaching a terminal state.

We use the cartpole balancing problem (Figure 3-1) from Sutton et al., [1] to illustrate the framework of reinforcement learning. In the problem, a pole is attached by an un-actuated joint to a cart which moves along a frictionless track. States are comprised of four parameters, namely the position of the cart from the centre, the velocity of the cart, the pole angle, and the angular velocity of the pole. The system is controlled by application of a force (the actions) of magnitude +1 or -1 units along the length of the cart. The pendulum starts upright, with the goal being to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. An episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. Thus the longer the cartpole remains upright within limits, the more rewards the agent gathers.

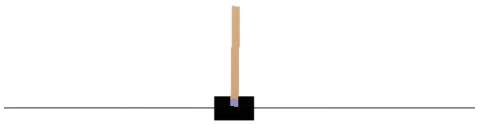


Figure 3-1: The cartpole balancing task [15]

The goal of the reinforcement learning agent can thus be defined as to maximize the rewards it obtains over the long run, by making the pole stand upright within limits for as long as is possible.

3-2 The elements of reinforcement learning

The learner and decision maker in an RL problem is called the agent. Everything outside the agent, which it interacts with is termed as the environment. They interact in discrete time

steps of $t = 0, 1, 2, ...$ wherein at each time step the agent receives a representation of the agent's environment state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is a set of all possible states. On this basis the agent selects an action, $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the set of actions that can be taken when in state s_t. As a consequence of the execution of this selected action, the agent transitions into a new state s_{t+1} for which it receives a reward R_{t+1}. The agent-environment interface is as seen in Figure 3-2.

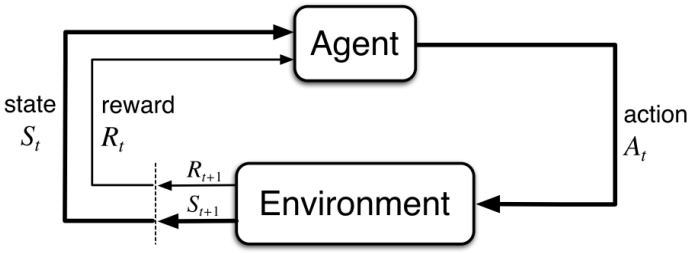


Figure 3-2: Agent-Environment interaction in RL [1]

There are a number of elements that make up reinforcement learning algorithms. Understanding these elements are necessary to understand the nature of the problem addressed by the thesis, as well as to understand the solutions to the problem. So we take the effort to explain important elements in the space of the next couple of pages.

- *Policy*: The policy, π, is a mapping from perceived states of the environment to the actions taken in those states. A policy may be as simple as a lookup table, and may also be stochastic. A policy is analogous to what is called a set of *stimulus response rules* or *association* in psychology.
- *Reward function*: A reward function, $R_t(s_t, a_t)$ defines the goal in the reinforcement learning problem. It maps each perceived state, or state-action pair to a single number that indicates the intrinsic desirability of reaching a certain state. Depending on the task, the rewards can also be sparse, in that a large reward is presented upon reaching the terminal state (specifying the goal), and be zero elsewhere.
- *Markov Decision Process (MDP)*: In the general case of a problem, the reward obtained from an action taken at a certain state depends on all previous state, action, reward combination leading to that certain state. Thus, the system dynamics can only be inferred by specifying the entire distribution, or the *history* of steps taken before reaching the current state.

  $$Pr\left\{s_{t+1} = s', R_{t+1} = R' \mid s_0, a_0, R_1 \cdots R_t, s_t, a_t\right\}$$

  For a reinforcement learning algorithm, it is ideal to have a state signal that summarizes past operations compactly while retaining all the relevant information. A state that successfully retains all the relevant information is said to have the *Markov property.*

  If the Markov property holds, then given state s and action a, the probability of of each possible pair of next state and reward, s', r is given by

  $$Pr\left\{s_{t+1} = s', R_{t+1} = R' \mid s_0, a_0, R_1 \cdots R_t, s_t, a_t\right\} = Pr\left\{s_{t+1} = s', R_{t+1} = r \mid s_t, a_t\right\},$$

The above equation being true for all s_t, a_t, s', R. If the Markov property for the state holds, then the environment and task as a whole is termed as an MDP. The one-step dynamics of such a problem enables us to predict the next state and expected reward from the next state just by knowing the current state and action.

An MDP can be defined as a 5-tuple $<S,A,P_a, R_a, \gamma>$ with:

- S - a set of states
- A - a set of actions.
- $P_a(s, s')$ - the probability of the system in state s at time t will reach state s' at time $t + 1$ when an action a is taken.
- $R_a(s, s')$ - the immediate reward obtained after transitioning from state s to s'.
- γ - a discount factor that prioritizes the importance of immediate and future rewards. [1]

Transition graphs such as that seen in Figure 3-3 are used to summarize the dynamics of a (finite) MDP.

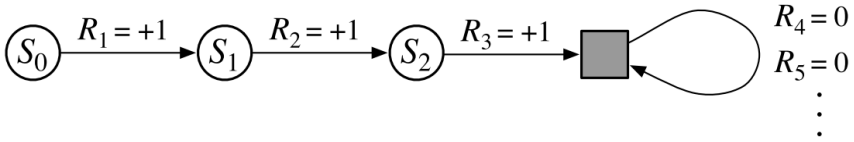


Figure 3-3: State transition diagram in reinforcement learning. The agent moves from state to state for which the environment provides them a numerical reward. The gray box signifies the end of the episode.

MDPs can also be generalized into Partially Observable MDPs (or POMDPs) in which the system dynamics are directly observed by the agent, but the underlying states cannot be directly observed.

- *Value function*: Value of a state is the expected value of the cumulative discounted reward an agent can hope to accumulate over the future, when following a certain policy starting from that state. Values correspond to the long term desirability of states, by taking into account the accumulation of rewards from the states that are likely to follow. In effect value functions help the agent prioritize long term reward collection. The value function at a state s when following a policy π is defined as -

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|s_t = s] \tag{3-1}$$

where G_t represents the gain or the return, representing the discounted rewards accumulated over time, and $\mathbb{E}_\pi[\cdot]$ refers to the expected value of a random variable, given that the agent follows the policy π, with t being the time step. The discount rate for rewards, represented as $\gamma \in [0, 1]$, determines the present value of future rewards. A discount rate close to zero signifies *myopic* behaviour, in which only rewards in the near

future are considered. A discount rate closer to one on the other hand signifies a longer lookahead, meaning that rewards further into the future are also taken into account.

A fundamental property of value functions in reinforcement learning is their inherently recursive nature. For a policy, π, and any state, s, the value function from (3-1) is defined with its possible successor states in mind as a recursive rule given by:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t|s_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t = s\right] \\ &= \mathbb{E}_\pi\left[\left(R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}\right) \middle| s_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} P_a(s,s') \left[R + \gamma V_\pi(s')\right] \end{aligned} \tag{3-2}$$

- *Action-value function*: Q-values or action values are estimates of the returns expected for each action taken from a non-terminal state, and following the policy thereafter. The value of taking an action a in a state s, and thereafter following a policy π, is represented as:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|s_t = s, a_t = a] \tag{3-3}$$

  Here Q_π is defined as the action-value function following policy π. Q-values also possess recursive properties similar to the value function.

- *Advantage function* The concept of the advantage function is defined to bring forward the differences between the rewards expected from taking different actions from the same state [16][17]. The advantage function is therefore a measure by which the expected value of taking a certain action is different from the expected value of taking the action that is currently considered optimal. Mathematically:

$$\begin{aligned} A(s,a) &= (Q(s,a) - V(s)) \\ &= \left(Q(s,a) - \max_{a'} Q(s,a')\right) \end{aligned} \tag{3-4}$$

  By definition, at a specific state, the advantage value of taking the optimal action from a state is zero. if $a = \text{argmax}_a A^*(s,a)$, with $A^*(s,a)$ being the optimal advantage function at s, a:

$$\begin{aligned} A^*(s,a) &= \left(Q^*(s,a) - \max_a Q^*(s,a)\right) \\ &= \left(\max_a Q^*(s,a) - \max_a Q^*(s,a)\right) \\ &= 0 \end{aligned}$$

  Here $Q^*(s,a)$ refers to the optimal action-value function at s, a. The advantage function was first used by Baird, (1993) in the *advantage updating* algorithm [16]. Advantage

functions can be thought of as an ordering of actions based on their inherent profitability, with the advantage value of an optimal action taken from a particular state equal to zero.

Due to this property of being a timescale-agnostic ordering of actions, Advantage functions have seen a surge in popularity over the recent years. Table 3-1 shows some of the algorithms that have made use of advantage functions to achieve superior results.

Algorithm	Authors
Advantage Updating	L.C Baird (1995) [16]
Natural Actor-Critic	Peters and Schaal (2008)[18]
A3C	Mnih et al., (2016) [19]
Dueling Network Architectures	Wang et al., (2016) [20]
ACER	Wang et al., (2016) [21]
Reactor	Gruslys et al. (2017), [22]

Table 3-1: Algorithms that utilize advantage functions

- *Bellman operator*: When it comes to value-iteration algorithms such as Q-learning (discussed in 3-4-1), the Bellman operator τ serves as a simple scheme to indicate iteration, without going through the hassle of defining subscripts for the iteration number, meaning that $Q_i = \tau Q_{i-1}$ with the subscript i indicating the iteration number.

  The Bellman operator is a contraction mapping in the supremum norm, and converges to a single value according to the Banach fixed point theorem [23].

  The Bellman operator $\tau Q \rightarrow Q$ is defined pointwise as:

$$\tau Q(s,a) = R(s,a) + \gamma \mathbb{E}_{P_a} \max_{a' \in \mathcal{A}} Q(s',a') \tag{3-5}$$

  with the term $\mathbb{E}_{P_a}[\cdot]$ referring to the expectation over the transition probability function $P_a(s,s')$ from s to s' when taking action a.

  It is proven [24] that this operator converges to a unique *fixed point* for which $\tau Q^* = Q^*$:

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{P_a} \max_{a' \in \mathcal{A}} Q^*(s',a') \tag{3-6}$$

  which in turn induces the optimal policy given by π^*:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^*(s,a) \quad \forall s \in \mathcal{S} \tag{3-7}$$

  Convergence to optimal values for the Bellman operator is dependent on certain conditions being satisfied by the agent employing the operator for learning. The conditions are -

  – Infinite exploration i.e., each state-action pair is visited infinitely often by the agent.

  – Discount factor $\gamma < 1$.

– *Robbins-Monro* conditions - $\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$, α_t being the learning rate at time step t.

As the expectations in (3-5) cannot be directly observed, estimates of Q-values are iterated towards optimal values by a learning rule given by $Q_i(s,a) \leftarrow (1-\alpha)Q_{i-1}(s,a) + \alpha\tau Q_{i-1}(s,a)$, with the left arrow signifying an update.

- *Action-gap*: The action-gap $g_Q^*(s)$ is defined as the difference in Q-values between the optimal action a_1 and the nearest sub-optimal action a_2 for that state. The *action-gap* for the system is mathematically expressed as:

$$g_Q^*(s) = |Q^*(s,a_1) - Q^*(s,a_2)|$$

The action-gap can also be explained as a special case of the advantage function between best and second best actions at a certain state, after convergence to optimality.

$$g_Q^*(s) = |A^*(s,a_2)|$$

with the value a_2 being the second-best action at a certain state s.

Remark: In this thesis we show that smaller action gaps are related to the disappearing reinforcements problem we address here. This relationship between the action-gap for Bellman operator based algorithms and performances at different frequencies are further explored in Chapter 4. Increasing the action-gap also leads to better performances at higher frequencies, and this relationship is explored in Chapter 5 and Chapter 6.

Function Approximation

The simplest way of representing a value function is by use of a lookup table, with the values of each state-action pair stored.

When the state-action spaces are large, storing and retrieving values become a problem, as it takes up large amounts of computational resources. To solve this problm, function approximators can also be used instead of a lookup table for representing value functions, thereby limiting the memory being used and speeding up the learning process.

$$\hat{v}(s,w) \approx V_\pi(s)$$
$$\hat{q}(s,a,w) \approx Q_\pi(s,a)$$

where w is a certain parameter vector for the function approximators given by $\hat{v}$ and $\hat{q}$. This formulation helps in the process of generalizing states, and the extra parameter w is updated by a learning algorithm such as the *Monte-Carlo* method or the *Temporal Difference* learning method, both discussed in Section 3-3. The function approximator introduced can of course be of varying complexity, a few examples being linear approximators and neural networks. Mathematically a linear value function approximator is written as:

$$\hat{v}(s,w) = x(s)^T w = \sum_{j=1}^{n} x_j(s) w_j \tag{3-8}$$

with $x(s) = (x_1(s)...x_n(s))^T$ being the states represented as a feature vector.

The goal of training is to find a parameter vector w that minimizes the mean square error between approximator's value $\hat{v}(s,w)$ and the actual value $V_\pi(s)$

Mathematically the cost function to be minimized can be expressed as:

$$J(s,w) = \mathbb{E}_\pi[(V_\pi(s) - \hat{v}(s,w))^2] \tag{3-9}$$

Gradient descent is a method that is commonly used to find a local minimum. The parameter vector update term in the case of a gradient descent algorithm is expressed as follows

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w) = \alpha \mathbb{E}_\pi \left[(V_\pi(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w) \right] \tag{3-10}$$

where$\nabla_w J(w)$ is the gradient of $J(w)$. Mathematically, the update is expressed as -

$$\Delta w = \alpha (V_\pi(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w) \tag{3-11}$$

3-3 Classifying RL - Model-based versus model-free

There are various broad categorizations for reinforcement learning algorithms. One of these classifications is as model-based or model-free, based on whether the algorithm relies on an environment model during operation.

3-3-1 Model-based reinforcement learning

Model-based reinforcement learning systems collect information about transitions from experience, which is then used to generate a model. The transition model is then used to plan future steps. The effectiveness of the model lies in the ability of the agent to perform virtual searches through the state-action space, to decide upon a seemingly optimal course of action from the current state of the agent. A disadvantage of the model-based approach is that such methods typically have a large computational complexity. Learning accurate models is also a challenging task in some cases. Using a model is also disastrous in the case when inaccurate models are learnt by the system, leading to impaired performances.

3-3-2 Model-free learning

Model-free approaches do not explicitly need to learn a model in order to generate experiences, and also have their primary goal of learning by improving a behavioral policy that maximizes a numerical reward signal. These approaches are lesser computationally intensive when compared with model-based methods, since they do not have an intermediate step

that requires them to calculate virtual experiences. Model-free approaches also prevents the hassle of learning models that capture the environment's dynamics well enough to prove useful. Model-free approaches like Monte-Carlo Learning and Temporal-Difference learning learn directly from episodes of experience.

Monte-Carlo learning

Monte-Carlo (MC) based methods learn directly from *complete* episodes of experience. They have a caveat that they can only be applied to *episodic* tasks (in which there exist a clearly defined terminal state, reaching which would initiate a reset of the task), as in order for the algorithm to work, episodes need to terminate. MC methods typically have high variances, zero-bias and theoretically assured convergence properties. They are also not very sensitive to initialization, and are simple to understand and use. An incremental MC algorithm updates value function (expected return) $V(s_t)$ towards the actual return G_t.

Temporal-difference (TD) learning

Temporal-difference learning is a concept central to the formulation of reinforcement learning algorithms. They can be classified as a combination of the concepts from Monte Carlo methods and dynamic programming. Temporal-difference learning schemes learn from incomplete episodes by *bootstrapping* existing estimates, updating a guess towards a guess [25]. TD methods can learn before knowing the final outcome, and can learn online after every step. They can also learn without the final outcome, from incomplete sequences as opposed to Monte-Carlo learning. TD methods can thus also work for non-terminating environments. They typically also have low-variance and some bias, are usually statistically more efficient than MC methods, and are also more sensitive to initial values. The simplest form of TD learning algorithm is the TD(0) algorithm, which works by updating the current estimate of the value function $V_t(s_t)$ towards the sum of the observed return R_{t+1} and the discounted estimated value function at the next step $V_t(s_{t+1})$ by using the following rule.

$$V_{t+1}(s_t) \leftarrow V_t(s_t) + \alpha \left(R_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right) \tag{3-12}$$

The value $G_t^{(1)} = R_{t+1} + \gamma V_t(s_{t+1})$ is called the one-step target, while the value $\delta(t) = (R_{t+1} + \gamma V(s_{t+1}) - V(s_t))$ is termed as the *one-step* TD error, as the update is happening towards the target obtained from taking a single step. Methods in which the temporal difference extends over n steps are called n-step TD methods, and the updates are in the direction of a target which is the discounted return after taking n steps. The n-step returns are then considered as approximations for the total return over the experiment. Mathematically the n-step returns are given by

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdot + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(s_{t+n}), \quad n \geq 1,\ 0 \leq t < T - n$$

Algorithms such as Q-learning [26] (discussed in Section 3-4) and SARSA [27] are some examples of TD learning algorithms frequently seen in literature.

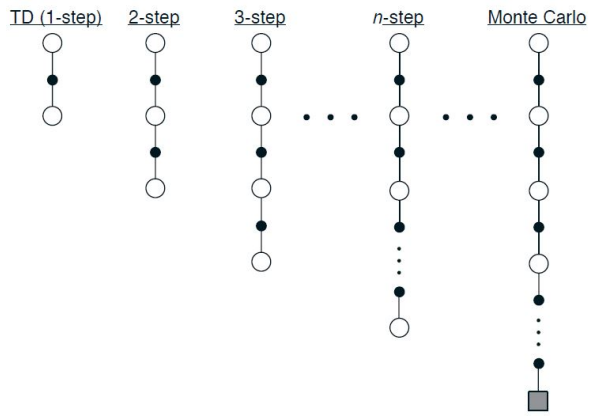


Figure 3-4: TD versus MC learning - spectrum ranges from one-step backups in TD learning and up to the until-termination backups of the MC learning [1]

Remark: Model based methods are known to be tricky to train. They tend to be sample efficient, but the policies learned only turn out to be as good as the learned model. Also for real world tasks, it is often easier to learn a good policy than a good model [28]. Because we are tackling problems related to controlling abstractions of real world systems, we decide to focus on model-free learning schemes. Moreover, we also preferred to focus on temporal difference methods over Monte-Carlo learning schemes, as TD learning methods can be applied to continuing (non-episodic) tasks.

3-4 Classifying RL - Value-based/Policy-based/Actor-Critic

Reinforcement learning algorithms can be placed in categories based on them being a value function based approach (or a *critic-only* approach) wherein they optimize an explicit representation of the value function, a policy based approach (an *actor-only* approach) which stores and optimizes a representation of the policy, or whether they are an actor-critic based approach, using representations of both the value and policy.

3-4-1 Value-based RL

Action-value methods or value-based approaches rely on optimizing for a specific value function that is dependent on the states of the environment. The value function for a particular state s, when following policy π is $V^{\pi}(s)$, and is defined as in (3-1). The state-action value function $Q^{\pi}(s,a)$, which explicitly includes the information about the effects of taking a particular action from a state is defined as in (3-3). Optimizing for a better value function with an iterative scheme implicitly produces a better policy for the agent to follow.

Value functions can be represented as a lookup table, with every state having an entry $V(s)$, or every state-action pair having an entry $Q(s,a)$. A problem with this approach is that when dealing with large MDPs with many states, there are often too many states and/or actions to commit to memory. It also makes the process too slow to learn each state value. In these cases

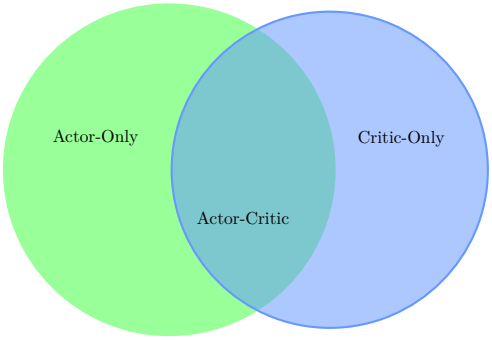


Figure 3-5: Categorizing reinforcement learning - The critic represents the value function and the actor, the policy

Some of the most popular action-value learning algorithms are value iteration, Q-learning and also its variant, the deep Q-network (DQN) which is a Q-learning agent with a neural network as its Q function approximator.

Q-Learning

Q-Learning as introduced by Watkins, (1989) [26] is a simple model-free value iteration algorithm based on the *Bellman updates* on Q-values. For a state s, and action a, and a constant learning rate per episode α, for a discount factor γ, and a policy π, the update rule for the Q-learning algorithm is defined as:

$$Q^{\pi}(s,a) \leftarrow (1-\alpha)Q^{\pi}(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q^{\pi}(s',a')\right) \tag{3-13}$$

s' being the state that the environment transitions to.

This is as a consequence of using the Bellman operator τ on the Q-values.

$$\tau Q(s,a) = R(s,a) + \gamma \mathbb{E}_{P_a} \max_{a' \in \mathcal{A}} Q(s',a')$$

As we cannot apply the Bellman operator exactly for Q-values for most problems because of the need of the correct estimate for the transition probabilities, we approximate it using the Q-learning updates seen in (3-13).

Q-learning can be used not only with a look-up table based representation of Q(s,a) but also with other function approximator based representations like radial basis functions and neural networks. Some of the popular algorithms utilizing the Bellman update rule can be seen in Table 3-2.

Q-Learning with a tabular representation of Q-values is also proven to converge to the optimal policy when every state-action pair s and a is tried infinite number of times, when the rewards R_t and learning rate for the n^{th} episode α_n are bounded [24].

Algorithm	Authors
Q-Learning	C Watkins (1995) [16]
Deep Q-networks	Mnih et al., [19]
Gorila	Nair et al., [29]

Table 3-2: Popular agents that utilize Bellman update rule

Mathematically, Given

$$\begin{aligned} R_t &\leq \mathbb{R} \\ 0 \leq &\alpha \leq 1 \\ \sum_{i=1}^{\infty} \alpha_{n^i}(s,a) = \infty, &\sum_{i=1}^{\infty} \left[\alpha_{n^i}(s,a)\right]^2 < \infty \forall s,a \end{aligned}$$

then as $n \to \infty$,

$$Q_n(s,a) \to Q^*(s,a) \forall s,a; \tag{3-14}$$

With Q_n being the Q-value for the n^{th} episode, and Q^* being the optimal value.

The algorithm for one-step Q-learning in procedural form is as seen in Algorithm 1 in the appendix.

Deep Q-Networks

Deep Q-networks (DQN) are Q-learning agents which uses deep neural networks as a function approximator for the Q-function. Neural networks are prone to divergences when a nonlinear function approximator is used. The divergences are usually attributed to

1. Correlations present in the sequences of observations.
2. Correlations between the action-values $Q(s,a)$ and the target values $r + \gamma \max_{a'} Q(s',a')$ [30].

Experience replay is a technique employed to remove the temporal correlations in the observation sequence and to smooth over the data distribution [31]. For this, experiences from transition steps are stored in a replay buffer. These experiences are uniformly sampled from the buffer and used in the update steps. The iterative update that updates the Q-values towards the target values are also only periodically updated to reduce correlations with the target. This is seen to increase the stability of the agent.

Mnih et al.,[30] implemented the DQN architecture to achieve human level performances on 2D Atari games. Deep convolutional neural networks(CNNs) were preferred for approximating the Q-function in the paper. This is because of the fact that a vision based approach was used, in which the pixel positions from frames captured from the Atari games were used as the inputs to the system, and CNNs are known as an effective solution for these types of approaches.

The algorithm used for DQN can be seen in Algorithm 2 in the appendix.

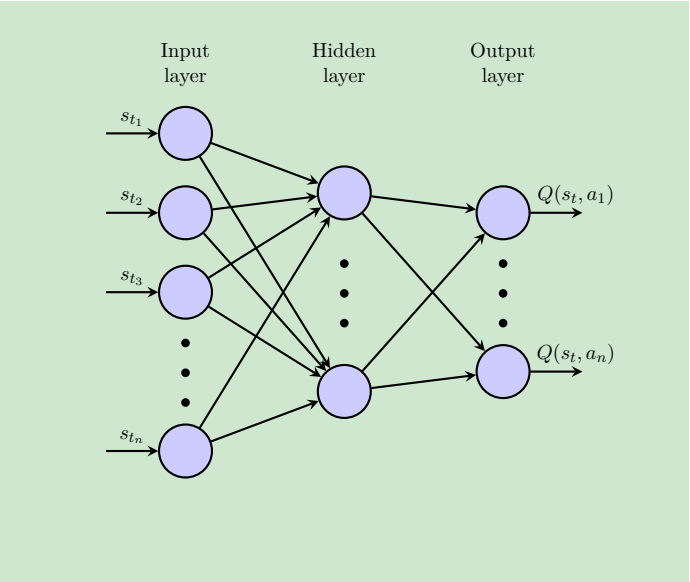


Figure 3-6: DQN with the inputs as state variables and the outputs as Q-values

3-4-2 Policy-based RL

Policy based RL approaches in contrast to value based approaches work by explicitly representing a policy with its own weights, independent of a value function. Parametrization of the policy is the main focus of the method, with the parametrized policy given by:

$$\pi_\theta(s, a) = \mathbb{P}[a|s, \theta] \tag{3-15}$$

with π_θ being the policy to be represented and the $\mathbb{P}$ being its parametrization. The goal of the policy based method is to find the best parameters θ for the given policy. The problem can be phrased as an optimization problem, with the average reward per time-step $\bar{R}$ to be optimized

The advantages of policy based RL techniques are that they are effective in high dimensional or continuous action-spaces, and also possess the ability to learn stochastic policies. But on the other hand, policy based RL techniques exhibit tendencies to converge to local optima, rather than global ones. Evaluating a policy is also typically inefficient and has higher variance [1].

3-4-3 Actor-critic

Actor critic algorithms maintain two specific sets of parameters, thereby separating the action-value function and the policy. The role of the *actor* is to update and maintain an action selection policy, while the role of the *critic* is to estimate value functions associated with the actor's policy.

The critic evaluates how effective the policy π_θ is for current parameters θ. In effect, it calculates the expected reward, while at a state, and following a certain control policy.

Actor-critic algorithms require minimal computation for action selection. Because the policy is separately stored, this eliminates the need to search in order to pick actions during each step.

Remark: For this thesis, we shall be focusing on (action-)value based methods that focus on discrete actions. This is partly due to the reason that they are generally regarded as better understood than their policy-based and actor-critic counterparts [1]. Also action-value methods applied to control problems are rarely seen in literature, so benchmarking these would be a contribution to the existing body of knowledge.

3-5 Classifying RL - On and off-policy learning

RL algorithms are also categorized on the basis of whether they are *on-policy* or *off-policy* learning methods.

- *On-policy learning* - An on-policy method requires the algorithm to learn the policy π from the experience sampled from π. This means that exploration needs to be built into the policy, and that determines the speed of policy improvements [4].
- *Off-policy learning* - An off-policy learning method tries to learn the policy from experience sampled from $\mu(a|s)$, a *behavior* policy that the agent can follow, different from the desired final *target* policy. This is important since the agent can utilize different policies at the same time, say one for exploration and another for learning. The agent is thereby able to learn about the optimal policy during its exploratory phases, with off-policy learning.

  Off-policy algorithms utilizing function approximators for their internal representations of (action)-value functions or policy currently have no theoretical convergence guarantees [1]. They are more general in nature than their on-policy counterparts. They include on-policy methods as the special case in which the target and behavior policies are the same.

Remark: Because of their more general nature and because they aid exploration, we primarily focus on off-policy methods for this thesis.

3-6 Summary

Reinforcement learning is a biologically plausible computational learning scheme for solving Markov decision problems, wherein the learning agent interacts with the environment, and adjusts its long term behavior according to the rewards it obtains. There are also multiple classes of reinforcement learning algorithms, each having their own advantages and shortcomings. Selecting an algorithm thus depends on what goals the designer chooses to achieve. We focus on solving the disappearing reinforcements problem for model-free, off-policy, value-based methods in this thesis.

Chapter 4

Defining the Problem

As opposed to algorithms from the optimal control paradigm discussed in Chapter 2 that improve their performances with increasing sampling frequency, commonly used Bellman operator based reinforcement learning algorithms deteriorates in performances. This is as a result of what we called the disappearing reinforcements problem in Chapter 1.

It is essential for the reader to clearly understand the problem from both a theoretical and an empirical perspective. We show that the *action-gap* from Section 3-2 is a concept that is central to understanding the nature of the problem. We also illustrate the effects of the disappearing reinforcements problem by simulating the returns gathered by algorithms on a simple MDP based on the motion of a point-mass on a 1D line.

The rest of this chapter is structured as follows. Section 4-1 deals with theoretically defining the nature of the problem. It also deals with empirically showcasing the effects of disappearing reinforcements, and the possible implications of the problem in real-world scenarios. Section 4-2 clarifies the effect of disappearing reinforcements on an algorithm with the help of an easy-to-understand MDP based on the motion of a 1D point mass. In this section, we also introduce novel benchmarking criteria for algorithms, inspired from concepts in control theory. Section 4-3 explores the importance of an agent's robustness to noise, which explores how robust the agent is when dealing with noise during updates.

4-1 Problem definition

This section deals with defining the problem from both a theoretical as well as from an empirical standpoint.

4-1-1 From a theoretical standpoint

To explain the problem for action-value methods updated by the Bellman operator, there are two important properties of the algorithms to be understood -

1. Decreasing action-gaps at increasing frequencies.
2. Estimation errors in the Q-values for common algorithms utilizing Bellman updates.

In this thesis, we assert that a combination of both these factors is what leads to decreasing performances at increasing frequencies, the effect we term disappearing reinforcements.

Decreasing action-gaps

To explain the problem, we take into account a simple off-policy action-value based algorithm utilizing the Bellman update rule, the Q-learning algorithm [26] explained in Section 3-4-1. The algorithm seeks to maximize the Q-value $Q(s, a)$, which is a measure of the expected discounted return over all the steps when executing action a from the current step s, and following the policy π, thereafter.

Mathematically, Q-values are defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a\right] \tag{4-1}$$

where $\mathbb{E}_{\pi}$ denotes the expected value given that the agent follows policy π, and R_{t+k+1} is the reward obtained at any time step $t + k + 1$.

When the sampling frequency for the algorithm is increased, more number of actions are executed within a similar duration. Alternatively, states at which actions are executed at also move closer in values. For all non-terminal states, as the length of a time step $dt \to 0$, the distance between consecutive states become negligible, and the Q-values of two different actions from the same state become similar. This is because rewards obtained are dependent on the states and action, and because $s_1, s_2 \to s$, this implies that for most commonly defined reward functions, $R_{t+1} \to R'_{t+1}$, with R'_{t+1} and R'_{t+1} being the reward upon transitioning when taking an optimal and a sub-optimal action respectively from current state s.

Therefore -

$$\begin{aligned}
\lim_{dt\to 0} Q^{\pi}(s,a_1) &= \lim_{dt\to 0} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t = s, a_t = a_1\right] \\
&= \lim_{dt\to 0} \mathbb{E}_{\pi}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \infty \middle| s_t = s, a_t = a_1\right] \\
&\approx \lim_{dt\to 0} \mathbb{E}_{\pi}\left[R'_{t+1} + \gamma R'_{t+2} + \gamma^2 R'_{t+3} + \cdots \infty \middle| s_t = s, a_t = a_2\right] \\
&= \lim_{dt\to 0} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R'_{t+k+1} \middle| s_t = s, a_t = a_2\right] \\
&= \lim_{dt\to 0} Q^{\pi}(s,a_2)
\end{aligned}$$

In other words,

$$\forall a_1, a_2 \lim_{dt \to 0} (Q^{\pi}(s, a_1) - Q^{\pi}(s, a_2)) = 0 \tag{4-2}$$

where a_1, a_2 are an optimal and a sub-optimal action respectively, to take from the state s. As explained in Section 3-2 the *action-gap* for the state is defined as-

$$g^*_Q(s) = |Q^*(s, a_1) - Q^*(s, a_2)|$$

Farahmand (2011) [32] also proved that the concept of the action-gap is closely linked to the performances achieved by the system.

Estimation errors from Bellman updates

The overestimation of Q-values by the Bellman operator based Q-learning algorithm has been explored multiple times in literature. Before 2016, it was believed that the overestimation of Q-values was linked to the errors caused due to insufficiently flexible function approximations (Thrun and Schwartz, 1993 [33]) and also from noise in observations and update steps [34]. Hasselt et al., (2016) [35] showed that overestimation of Q-values is possible irrespective of the presence of noise and approximation error, as was previously believed.

We assert that smaller action-gaps coupled with inaccurate estimates of action-values for the Bellman operator-based agents are responsible for disappearing reinforcements. A favourable magnitude of action-gap would intuitively lead to more robustness to errors and noise. When the values of $Q^{\pi}(s, a_1)$ and $Q^{\pi}(s, a_2)$ are similar, it implies that there is less penalty for taking a sub-optimal action after a string of nearly-optimal action sequences (choosing sub-optimal actions during training can be due to factors such as approximation errors, noisy observations, improper initialization or from the use of an exploratory policy). Improperly estimating the values of sub-optimal actions as higher than that of optimal actions will in turn lead to the creation of sub-optimal policies. We assert that at higher frequencies, the policy becomes increasingly prone to approximation errors and noise, as the Q-values gather closer together, thereby causing disappearing reinforcements. Here, a small amount of noise might lead to a reordering of action-values due to their closeness.

Another way to look at the problem is by looking at the actions themselves. As the operating frequency increases, the number of actions that are required to be taken by the agent before termination increases. This decreases the importance of a singular action in a sequence of actions, as the returns from from the use of different actions turn out to be increasingly similar in value. At continuous time, i.e when $dt = 0$, the algorithm stops learning completely. When reinforcement learning algorithms such as Q-learning are therefore used on control tasks, this effect implies that the algorithm performs increasingly poorly when their operating frequencies are raised. The problem is not as a result of any function approximation used, but is instead inherent in the definition of Q-values itself [16]. This increased sensitivity to noise also implies that the Bellman operator-based algorithm can only be used on platforms and tasks which operate at lower frequencies, as those algorithms operating at higher frequencies are required to be more robust towards approximation errors and noise.

Different environments also work optimally at different frequencies. The formulation of an algorithm which scales its performance appropriately with the frequency it operates at is imperative for effective control. Only then will *multi-task control*, the possibility of the agent working satisfactorily across environments, be truly possible.

4-2 The point mass MDP

To illustrate the effects of action-gaps at higher frequencies, we create a simple MDP that simulates the manipulation of a point mass on a frictionless horizontal line segment. The point mass is manipulated by forces of equal magnitude towards the left and the right. The goal of the MDP is to move the point mass from the start state which at one end of the line segment towards the terminal state at the other end. Rewards to the agent $R(s, a)$, are defined as a function of the distance travelled from the initial state, or $R(s, a) = K \times d$, with k being an arbitrary scaling constant (we used $k = 0.001$), and terminal state is 5 units away from the initial state. We used a Bellman operator based Q-Learning algorithm as the agent for solving the task.

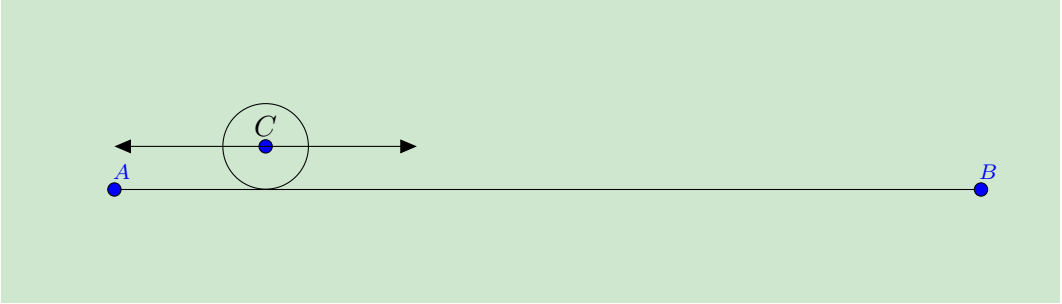


Figure 4-1: A simple MDP with the objective of moving an arbitrary point mass C from starting state A towards terminal state B. The arrows represent actions of equal magnitudes that can be taken in either direction

Results

The results from the experiment can be seen in Table 4-2. The experiment is conducted for the same number of episodes (40), across frequencies. As expected, the action-gaps at higher frequencies decrease in magnitude, when the Q-values for different actions at the same state move towards each other. Thus the presence of noise in the update steps, from improper initialization or from errors due to function approximation increasingly upsets the ordering of action-values when frequencies are increased, thereby leading to the apparent drop in performances. The Q-learning algorithm was implemented with tilecoded Q-values, with 100 tilings used. The parameters used for the implementation can be seen in Table 4-1.

Parameter	Value
α	0.00025
γ	0.99
Policy	ϵ-greedy
$\epsilon_{\text{initial}}$	1
ϵ_{Final}	0.1
Decay rate	0.99 (exponential decay at every step)

Table 4-1: Q-Learning - Parameters of choice

Algorithm	f_x	Action Gap	Performance
Q-Learning	1	0.0002	50.7 ± 1.4
	10	0.00015	19.4 ± 8.1

Table 4-2: Comparing action-gaps on the point mass MDP with a base frequency of 50Hz. The term f_x refers to the frequency multiplier.

4-2-1 Empirical analysis and benchmarking

In this section, we further empirically analyze and show the effects of disappearing reinforcements. Experiments are conducted using two different variants of the Q-learning algorithm. We employ the algorithms at different frequencies to benchmark their performances. Different metrics based on the total normalized un-discounted episodic returns are devised.

The normalized un-discounted return $\bar{R}_i$ for episode i is defined by:

$$\bar{R}_i = \frac{1}{f_x R_{\max}} \sum_{t=1}^{N} R_t(s_t, a_t) \tag{4-3}$$

with $R_{\max}$ defined as the upper limit of returns achievable for the length of an episode (in case one such conditions exists- if not, we assume this value to be unity), f_x is the frequency multiplier with respect to the base frequency associated with the experiment, $R_t(s_t, a_t)$, the rewards obtained at each non-terminal state and action for the length of the episode, and N being the length of the experiment in episodes. The resulting returns are normalized within the range of $[0, 1]$, so as to easily compare the learning rates of algorithms for a similar number of steps, across frequencies.

With the normalized un-discounted returns over episodes defined, the following metrics are put forward for analyzing the performance of algorithms across frequencies:

- *Maximum returns* over 100-contiguous episodes: This is used as a measure to understand highest performance that the algorithm can achieve. A sliding window with a length of 100 is employed to analyze the highest contiguous returns over the length of the experiment.
- *Mean returns* over all episodes: Coupled with the previous metric, these give an insight into whether the algorithm being benchmarked learns to our satisfaction.
- *Step response metrics* -
  - *Rise Time, τ_{rise}*: We define the rise time in the number of episodes required for the algorithm's performance to rise from 10% to the 90% of the *steady-state* value $\bar{R}_f$ during the duration of the experiment, the performance here being the total un-discounted returns gathered over the length of the episode. We chose this definition of a rise time so that it also is relatable to the definition of rise times found in control theory.

$$\tau_{\text{rise}} = \tau_{90} - \tau_{10}$$

– *Settling Time,* τ_s: The settling time in episodes is defined as the number of episodes after which the learning curve enters and remains within a certain bound ϵ of the final value.

$$\tau_s = i \leftrightarrow \underset{i}{\text{argmax}} \left| \bar{R}_f - \bar{R}_i \right| \leq \epsilon \bar{R}_f \; \forall i \in [0, N]$$

We chose the final steady-state value of epsilon to be 0.05. The steady-state value $\bar{R}_f$, mentioned above represents the final value after the algorithm converges, which we calculate as the average normalized undiscounted returns over the last c episodes.

$$\bar{R}_f = \frac{1}{c} \sum_{j=N-c+1}^{N} \bar{R}_j$$

The value of c can be arbitrarily chosen, the value for our experiments being 100.

For the sake of simplicity and ease of tuning, the algorithms were simulated on a test MDP setup, namely the cartpole [1]. The OpenAI gym [15] provided the platform, and it is commonly seen in reinforcement learning literature as a universal benchmark.

The cartpole problem

The cartpole, briefly discussed in Section 3-1 is a fundamental and oft-used test-bed when it comes to employing reinforcement learning algorithms. For this problem, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pole starts upright, and the goal is to prevent it from falling over. The episode ends when the pole is more than 15 degrees from vertical, or when the cart moves more than 2.4 units of distance from the centre. In this version of the task, discrete actions of magnitude [-1,1] are allowed along a single dimension [36]

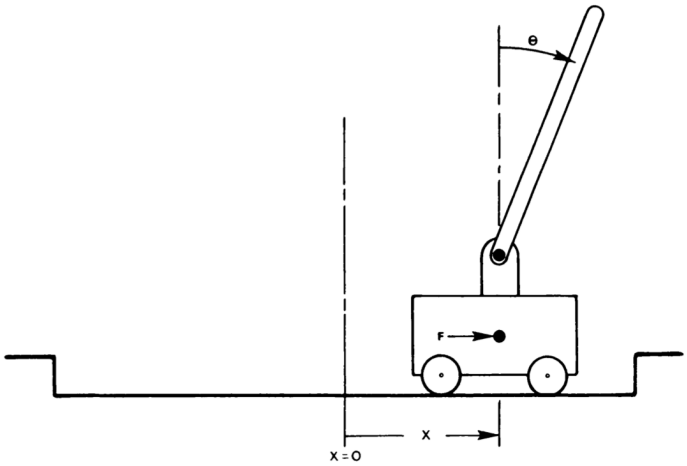


Figure 4-2: Visualizing the cartpole state parameters [36]

The reward function presented a unit reward for every step taken in the episode before termination. A large negative reward for episode termination was also programmed into

the algorithm, as this was seen to improve performances to an extent. The objective of the problem is therefore to maximize the time for which the cartpole is balanced in an upright position.

Results

We simulate two different algorithms on the cartpole, namely the standard Q-learning with a tile-coding based function approximator with 100 tiles for the action-values [37], and a fully-connected neural network based deep Q-learning algorithm [30] across different frequencies. This is done to show that the disappearing reinforcements problem exists across different algorithms that make use of Bellman updates, independent of the function approximator in use. An ϵ-greedy exploratory policy was chosen which exponentially decayed the value of the exploration rate ϵ at every episode until a value of 0.1 was reached. Upon reaching a state the epsilon-greedy approach chooses a random action with probability ϵ, and chooses the highest valued action according to the current policy otherwise. The parameters chosen to implement the Q-learning algorithm can be seen in Table 4-1. The parameters chosen for the DQN can be seen in Table 4-3. The neural network used in the implementation was the same as used by Mnih et al., [30] with the exception of using fully connected neural networks instead of CNNs.

Parameter	Value
γ	0.99
Mini-batch size	32
Replay Memory	1000000
Error	Mean Absolute Error
Target update frequency	4
Policy	ϵ-greedy
$\epsilon_{\text{initial}}$	1
ϵ_{Final}	0.1
Decay rate	0.99 (exponential decay at every step)
Optimizer	Adam (α=0.0002, β_1=0.9, β_2=0.999, $\hat{\epsilon}$=$1e-8$, decay=0)

Table 4-3: DQN - Parameters of choice

The results from the experiment can be seen in Table 4-4. It can be seen that the metrics of mean returns and max returns employed decrease consistently in magnitude when the sampling frequency is increased by a factor of ten from a base of 50 Hz. Thus it can be said that as theorized, the experiments reveal that the learning is reduced while sampling frequencies are increased.

The step response indicators indicate that the rise times at higher frequencies are faster, but the returns it obtains at these frequencies are lower because of disappearing reinforcements. The Q-learning algorithm also fails to converge at 10x frequency, and hence the settling time is indicated with a blank. The DQN, as expected, has an improved performance when compared with Q-learning on the same task, but the max and mean returns achieved still takes a hit when operating at 10x frequencies. Also, as expected, the algorithm converges slower judging by the settling time, which indicates a slower learning rate.

The learning curves for Q-learning and DQN can be seen in Figure 4-3 and Figure 4-4.

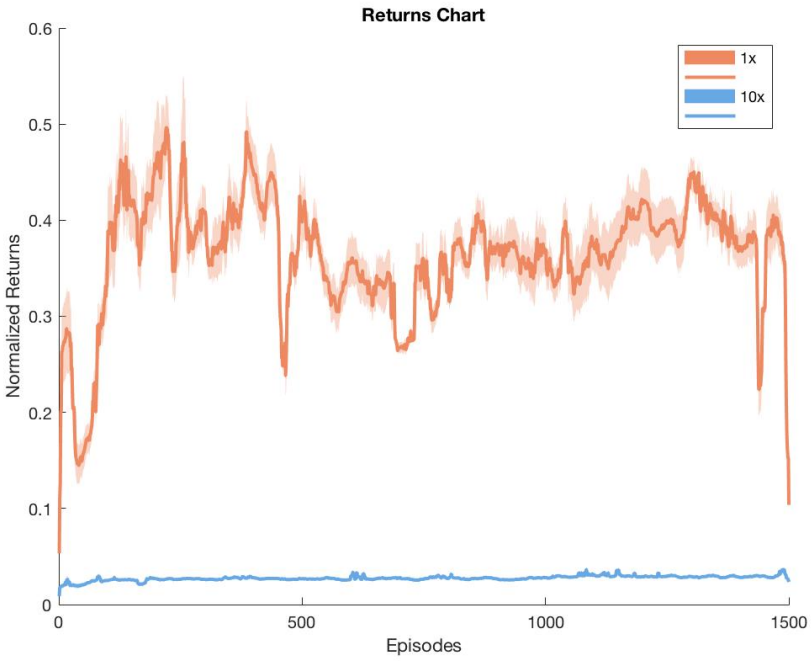


Figure 4-3: Q-Learning - Returns per episode across frequencies on Cartpole

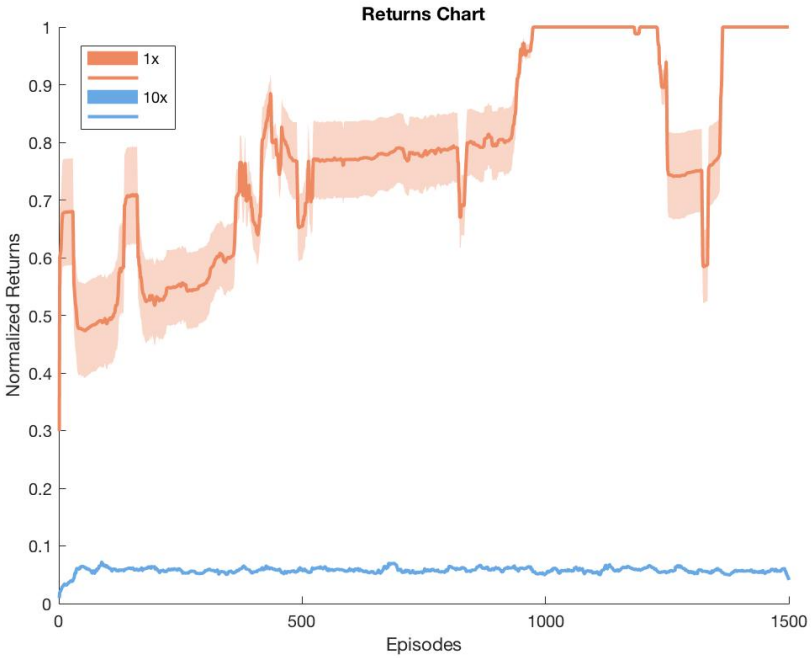


Figure 4-4: DQN - Returns per episode across frequencies on Cartpole

Algorithm	f_x	Mean Return	Max Return	Rise Time	Settling Time
	1	0.78	1	7900	24820
Q-Learning	10	0.028	0.049	93	-
	1	0.82	1	435	1317
DQN	10	0.035	0.17	59	2479

Table 4-4: Comparison chart. Base frequency= 50 Hz. Frequency multiplier f_x shows by how much the base frequency is multiplied, and the results are averaged over 5 randomized trial runs.

4-3 Robustness to noise

As explained, the nature of the disappearing reinforcements problem lies in the increased sensitivity of the algorithm to errors and noise at higher frequencies. In this section, we associate and quantify the relationship between noise and how it affects the performance of the algorithm.

As our aim is to understand and interpret results, we conduct experiments with a Bellman operator based algorithm on the point particle MDP explained in Section 4-2. We simulate noise in the system by adding a normally distributed zero mean pseudorandom signal to the observations (state values). For simplicity's sake, we conduct experiments with a Q-learning algorithm, with a tabular representation of action-values, and operating with an epsilon-greedy policy. The standard deviation of the noise added was selected to be the arbitrary small values of 0.001, and 0.01. As expected, the Bellman operator based Q-learning algorithm performs increasingly worse with the addition of noise, with the drop in performance becoming increasingly apparent at higher frequencies. The performance (in mean returns averaged over 5 random runs) of Q-learning with the added noise can be seen in Table 4-5.

Noise	f_x	Action Gap	Performance
	1	0.00017	31.1 ± 2.3
0.01	10	0.0001	8.9 ± 3.9
	1	0.00018	48.3 ± 2.1
0.001	10	0.00013	12.6 ± 6.2

Table 4-5: Comparing the effects of noise on the Bellman Updates.

4-4 Summary

Disappearing reinforcements lead to negative effects when the algorithms are deployed in real-life scenarios. On the one hand, this effect renders reinforcement learning algorithms useless in environments that require operating at higher frequencies. There is also the loss in resolution when reinforcement learning algorithms are employed at low frequencies on platforms as a result. Events that happen on a timescale lower than that of the current sampling time get ignored due to the coarse representations, leading to a sub-optimal policy. The effects of this can be seen from the sharp dip in mean/max returns for the tilecoded Q-learner as seen in Table 4-4.

In this chapter, we defined the nature of the problem as due to estimation errors and decreasing action gaps in Section 4-1, and analyzed it in both theoretical and empirical standpoints. We established the connection between the Bellman operator and disappearing reinforcements. The decrease in action-gaps for the Bellman operator based Q-learning algorithm at higher frequencies was empirically shown with the help of a point mass based MDP in Section 4-2. We then illustrated the effects of disappearing reinforcements with experiments conducted on the cartpole MDP, with different novel metrics to indicate performance measures across frequencies. We also showed how common Bellman operator based algorithms are affected by noise.

Chapter 5

Exploring The State Of The Art

Truly multi-task reinforcement learning is achieved only when the algorithms are able to work across frequencies. This is because different platforms have different optimal frequency ranges of operation. Disappearing reinforcements prevent Bellman operator based algorithms from being used at higher frequencies, thus preventing them from being used for control. This chapter looks into the state-of-the-art and identifies solutions to disappearing reinforcements, while comparing and contrasting these methods on the basis of their advantages and shortcomings. We identify two different methods, each attacking a cause of the problem we identified in the previous chapter, namely the estimation errors in action values and the decreased action-gaps at higher frequencies.

The rest of this chapter is structured as follows. We first provide a theorem from literature to identify optimality-preserving, gap-increasing (OP/GI) operators, which we show to be useful. We then move forward to identify operators that satisfy the conditions in the theorem defined in Section 5-2. One operator satisfying the conditions discussed is the advantage learning operator. Another approach to the problem is to leverage a different neural network architectures to create more robust estimates of the Q-values themselves, thereby alleviating the problem. The approach discussed in section Section 5-4 makes use of adapting the neural function approximators to form different streams, each exhibiting different properties that are beneficial f operation.

5-1 On the properties of operators

The Bellman operator, as discussed in Chapter 3 is a contraction mapping in the supremum norm. Bellemare et al., 2016 [38], proved that for an algorithm to be robust towards errors and noise, the operator based on which an algorithm works should have two properties, the *optimality-preserving* property, and the *gap-increasing* property. The Bellman operator creates small action gaps, thereby making it prone to approximation and estimation errors [38]. This can be seen from the experiments on the point particle MDP discussed in the previous chapter.

We argue that this increased sensitivity to errors also leads to worsening performances at higher frequencies, when the optimal action-values move closer together.

Bellemare et al., (2016) [38] define the conditions for an operator to be optimality-preserving and gap-increasing (OP/GI), and a theorem based on these conditions to check if an operator possesses these properties. The conditions for optimality preservation is defined as follows.

Condition 5-1.1. ***Optimality-preserving***
For the MDP M, operator τ' is optimality preserving, if $\forall Q_0 \in \mathcal{Q}$ and $s \in \mathcal{S}$, letting $Q_{k+1} = \tau' Q_k$:

$$\tilde{V}(s) = \lim_{k \to \infty} \max_{a \in \mathcal{A}} Q_k(s, a)$$

exists, is unique, $\tilde{V}(s) = V^*(s)$ and $\forall a \in \mathcal{A}$,

$$Q^*(s, a) < V^*(s) \Rightarrow \lim_{k \to \infty} \sup Q_k(s, a) < V^*(s)$$

Simply put, under an optimality preserving operator, an optimal action remains optimal, and sub-optimal actions remain sub-optimal.

The condition for an operator to possess a gap-increasing property is defined as follows.

Condition 5-1.2. ***Gap-increasing***
An operator τ' for MDP M is gap-increasing if $\forall Q_0 \in \mathcal{Q}$, $s \in \mathcal{S}$ and $a \in \mathcal{A}$, letting $Q_{k+1} := \tau' Q_k$ and $V_k(s) = \max_b Q_k(s, b)$, then,

$$\lim_{k \to \infty} \inf \left[V_k(s) - Q_k(s, a) \right] \geq V^*(s) - Q^*(s, a)$$

For gap-increasing operators, even the least action-gap for a policy remains greater than or equal to the action gap for the Bellman operator. This means that the operator possesses gap-increasing properties when compared with the Bellman operator.

5-2 OP/GI operators

Bellemare et al., (2016) defined a theorem based on the conditions 5-1.1 and 5-1.2 for the operator in question to be checked against. The theorem is as follows:

Theorem 5-2.1. *Let* $\tau Q(s,a) = R(s,a) + \gamma \mathbb{E}_{Pa} \max_{a \in \mathcal{A}} Q(s',a)$.
Let τ' *be an operator with the property that there exists an* $\eta \in [0,1)$ *such that* $\forall Q \in \mathcal{Q}, s \in \mathcal{S}$ *and* $a \in \mathcal{A}$, *if*

1. $\tau' Q(s,a) \leq \tau Q(s,a)$
2. $\tau' Q(s,a) \geq \tau Q(s,a) - \eta \left[V(s) - Q(s,a) \right]$

Then τ' *is both optimality-preserving and gap-increasing*

5-3 Advantage Learning

As explained in Section 4-1-1, for Bellman update based algorithms, Q-values at a particular state tend to re-order themselves in the presence of noise from the approximators and estimated values. This is increasingly so at higher frequencies when the action-gaps reduce, thereby making it easier for the noise to upset the ordering of actions.

Our interest lies in the selection of operators that adheres to both Condition 5-1.1, for preserving optimality, and Condition 5-1.2 which ensures a gap-increasing property. We assert that defining such an operator gives rise to a robust algorithm that is able to operate satisfactorily across frequencies. Bellemare et al.,(2016) [38] go on to define several such operators. One such operator defined by the authors was the *advantage learning* operator, modified from the original *advantage learning algorithm* defined by Baird, (1995) [17], so as to possess the same fixed point and at the same time be more stable in practice. We shall be referring to this operator with the symbol τ_{AL}. Bellemare et al., (2016) redefined the operator linked to the original advantage learning algorithm can be redefined with the same fixed point as equivalent to the Bellman operator fitted with a correction term which is a scaled value of the advantage function:

$$\tau_{\text{AL}} Q(s,a) = \tau Q(s,a) + \eta \left[\underbrace{Q(s,a) - V(s)}_{\text{advantage function}} \right] \tag{5-1}$$

with τ being the Bellman operator, and $\eta = (1 - K)$, K proportional to the sampling time dt. Bellemare et al., [38] explain that the form as seen in (5-1) is more stable in practice than the original version by Baird, as it prevents the multiplication by the K^{-1} term. It also allows for easy verification to see that it adheres to the conditions seen in Theorem 5-2.1.

The update rule for the *advantage learning* algorithm, which can be accurately labelled as a corrected Q-learner, is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left\{ R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a) + \eta \left[Q(s,a) - \max_{b \in \mathcal{A}} Q(s,b) \right] \right\} \quad (5\text{-}2)$$

The term α refers to a learning rate, while the γ refers to a discount factor. Advantage learning, similar to Q-learning is also an off-policy, model-free, critic-only algorithm.

To illustrate the gap increasing properties of the advantage learning operator, we carry out experiments on the point particle MDP formulated in Section 4-2. To recap, a point mass is moved along a one-dimensional line segment, with the goal of the MDP being to move the point mass from one end of the segment to the other. Discrete forces can be applied on the point particle to move it in either directions, while the particle receives a reward at each state according to its distance from the terminal state at one end of the line segment.

We calculate the action-gap for states over episodes for an advantage learner using a tilecoded state-action value representation on the point particle MDP. The results can be seen in Table 5-1, along with the performances achieved with respect to the mean returns averaged over 5 consecutive runs.

On the point mass MDP, the action-gaps were seen to decrease accordingly at higher frequencies for the Q-learning agent when the states become closer, as can be seen in Table 4-2. The advantage learning agent on the other hand, because of its gap-increasing property has an overall higher action-gap, this becoming more apparent at higher frequencies. The results from the experiment can be seen in Table 5-1. Also, judging from the non-decreasing performance, the advantage learning operator can be identified to be timescale-agnostic, implying its ability to work satisfactorily across frequencies.

Algorithm	f_x	Action Gap	Performance
	1	0.0038	112.79 ± 5.0
Advantage Learning	10	0.022	119.91 ± 21.1

Table 5-1: Comparing average action-gaps and performances (mean returns over five random runs) across timescales on the point mass MDP

5-3-1 Robustness to noise of OP/GI operators

We also check how robust the OP/GI operators are when faced with noise. Theoretically, as the action gaps are larger for the advantage learning operator, the algorithm becomes resistant to noise. We test this hypothesis by the addition of two sets of zero mean Gaussian noise to the observed values of states, each with different values of standard deviation. The test is run with five trials of 40 episodes each on the point mass MDP, and the results from doing the experiment are seen as in Table 5-2.

The results indicate that advantage learning is more robust towards noise than the Bellman operator based Q-learning. The action gaps are larger when in comparison with Q-learning, and the algorithm also is more robust towards noise. This indicates the correctness of our

f_x	Noise	Action-gap	Performance
1	0.01	0.0029	102.4 ± 5.5
10		0.018	101.1 ± 7.1
1	0.001	0.0036	107.3 ± 1.2
10		0.021	108.0 ± 3.9

Table 5-2: Comparing noisy readings with performances across frequencies for advantage learning.

hypothesis, that the action gaps and estimation errors are related to disappearing reinforcements. It also shows that the problem is one that can be alleviated by the use of similar approaches to the one discussed here.

Convergence

Bellemare et.al, (2016) [38] theoretically proved the convergence of the advantage learning operator. Further studies of convergences of advantage learning based agents coupled with action-value function approximators have not been conducted. It can however be seen empirically that the advantage learning operator converges in different setups.

Shortcomings

Advantage learning is also known to be very difficult to scale to MDPs involving large action spaces, being an action-value method. This is due to the curse of dimensionality.

Remark: The advantage learner can be thought of a modified Q-learning agent. At its core, similar to Q-learning, advantage learning is also a model-free, off-policy, value-based (critic-only) algorithm like Q-learning, but it converges to values different from Q-learning which in turn improves performances at higher frequencies because of increased action-gaps.

5-4 Dueling Network Architectures

As disappearing reinforcements were shown by us to be caused by the estimation errors of the Q-values at higher frequencies, another way to improve performances at finer timescales is to improve the estimates of Q-values themselves.

Dueling network architectures are a state-of-the-art approach that are known to achieve impressive performances on Atari games [20]. This approach by Wang et al., improves on the performances achieved by other architectures such as the DQN [30] by leveraging changes to the neural architecture that approximates the Q-functions. The premise behind the construction of the dueling architecture is simple. A Q-function can be written as the sum of the value function and the advantage function, with both terms having significantly different properties.

$$Q(s,a) = V(s) + A(s,a)$$

The value function represents the expected long-term discounted returns starting from a certain state.

While the DQN uses a standard neural network (usually convolutional or fully connected) taking in state values to produce Q-values, the dueling architecture (seen in Figure 5-1) splits the penultimate layer to have one part of the split layer act as an approximator for the value-function $V(s)$ and the other part act as an approximator for the advantage function $A(s, a)$. The sum of the outputs of both these layers in-turn give the action-value function $Q(s, a)$. The network is trained using Bellman updates, similar to the DQN.

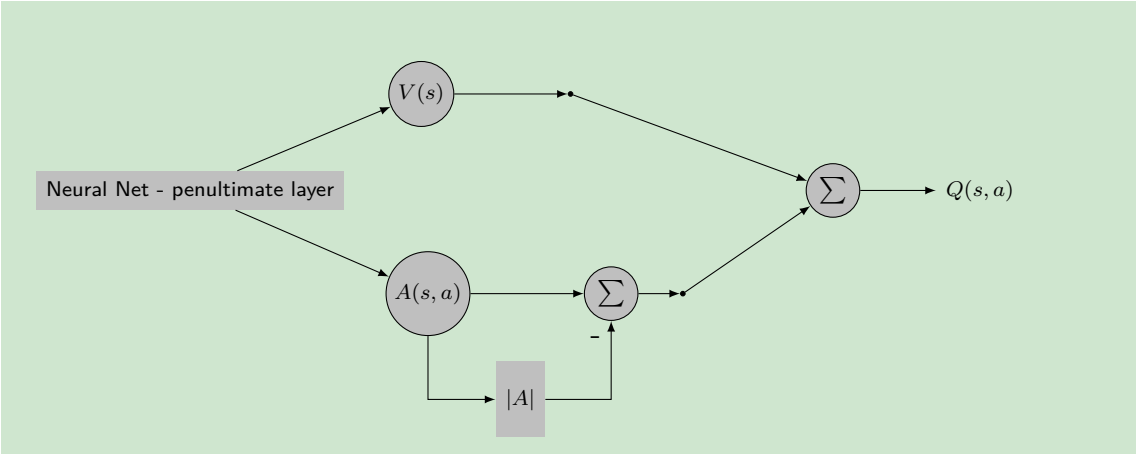


Figure 5-1: The dueling network architecture

The architecture enforces the Advantage function approximation layer to be zero for the chosen action, so that the other layer is forced to converge to the value function. This is done at the penultimate layer of the dueling architecture, by subtracting the maximum advantage over all actions from the advantage function layer. This step also serves to increase identifiability, so that we are be able to recover the value function and advantage function uniquely, from the action-value function output by the network. The equation for the dueling network can be expressed as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s,\theta,\beta) + \underbrace{\left[A(s,a,\theta,\alpha) - \max_{a' \in \mathcal{A}} A(s,a';\theta,\alpha)\right]}_{\text{advantage function}} \tag{5-3}$$

θ being the parameters of the layers of the neural network, α and β being the parameters of the value and advantage layers, respectively. Wang et al.,[20] argue that subtracting the average of the advantage function at the penultimate layer instead of a maximum provides better performances, and serves to increase the stability of the optimization as the advantages need only change as fast as the mean. Thus the overall equation for the network output turns out to be:

$$Q(s,a;\theta,\alpha,\beta) = V(s,\theta,\beta) + \left[A(s,a,\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right] \tag{5-4}$$

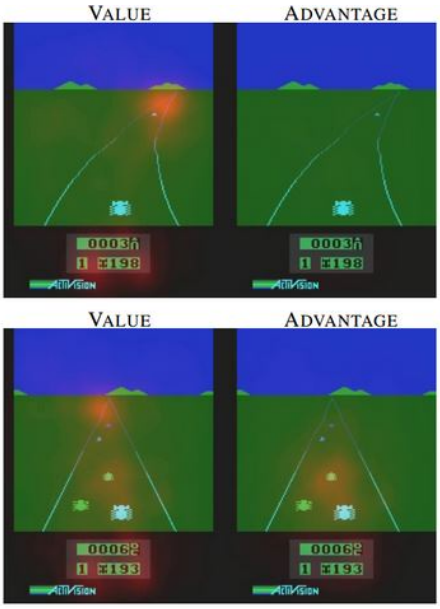


Figure 5-2: Saliency map - Advantage learning v/s value function. [20]

The enduro game requires the gamer to control a motorcycle to avoid obstacles. The orange patch on the image refers to the area where the agent gives the most attention to.

The dueling network architecture provides a better estimate of the action-value function, and this makes it more robust towards noise in the environment. The robustness of the estimates of Q-values are due to the creation of the two channels mentioned, a value function channel that creates a measure of how good the long-term returns are when starting from the state and following a certain policy, and the advantage function, which is an indicator of how useful specific actions are, when at a certain state. As the maximum advantage values are forced to move to zero at each states, the value function stream also moves towards its correct values, and a subsequent sum of these would create robust estimates of the Q-values themselves. This makes the dueling network robust against disappearing reinforcements at higher frequencies.

Looking at the saliency map of the value function channel (see Figure 5-2), the agent playing the game Enduro can be seen to have its attention tuned more towards the pixels at the horizon, and less towards the pixels immediately in front of the car. On the other hand, the advantage function represents a relative ordering of actions according to their profitability. The advantage function channel can be seen to be only concerned with obstacles in close proximity to the car, and therefore its attention is tuned that way. Its saliency map to measure channel attention can be seen to be as in the right hand side of Figure 5-2 [20], where the orange patch indicates the pixels in the image with the most attention given to by a certain channel.

The dueling network is updated in the same way as the deep Q-network (DQN) described by Mnih et al., [30] using a Bellman operator based update rule. To summarize, the network parameters are separately stored as target values and the updates are used to reduce the difference between the target and the Q-values output by the network. The loss function L_i for the algorithm at iteration i is given by:

$$L_i(\theta_i) = \mathbb{E}_{Pa}\left[\left|R(s,a) + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right|\right] \tag{5-5}$$

with θ_i^- representing the target value function and θ_i denoting the parameters being used in the estimation of value functions. The loss represents the temporal difference error term in the Q-learning update rule. A gradient descent step is then done on the loss function with respect to the parameter θ. Kingma and Ba (2015) devised Adam [39], which is a gradient based optimization algorithm as an alternative to other popular gradient descent algorithms such as Stochastic Gradient Descent [40] and RMSProp [41]. For our implementations, we used Adam instead of RMSProp used originally in the paper by Mnih et al. for practical reasons. We also decided to update the target network with the concept of soft updates as described by Lillicrap et al., [42] for the providing increased stability for the algorithm.

To keep our implementation simple, we used a standard experience replay based update rule similar to the DQN on the dueling network architecture. Experience replay updates, as explained in Chapter 3, make it possible to break temporal correlation between updates, thereby improving performances. This also makes it possible for rare experiences to be considered for more than a single update, thereby improving the policies used. As we were dealing with direct state inputs from the environment (and not images), we used 3 fully connected deep neural layers for the architecture with 32, 64 and 32 neuron units respectively, The penultimate layer with the value and advantage streams have 512 units each, with the value stream having one output and the advantage stream having the same number of outputs as there are valid actions. All hidden units have rectified linear units for activations.

The details of the implementation are as shown in Table 5-3, and the results from simulating the agent on the cartpole can be seen in Table 5-4.

Parameter	Value
γ	0.99
Mini-batch size	32
Replay Memory	1000000
Error	Mean Absolute Error
Target update	0.01 (Soft Updates)
Policy	ϵ-greedy
$\epsilon_{\text{initial}}$	1
ϵ_{Final}	0.1
Decay rate	0.99 (exponential decay at every step)
Optimizer	Adam (α=0.0002, β_1=0.9, β_2=0.999, $\hat{\epsilon}$=$1e-8$, decay=0)

Table 5-3: Dueling Network - Parameters of choice

Convergence

There exist no theoretical convergence guarantees for the agent due to its neural architecture. The learning rule, however is based on the Bellman operator, which is proven to converge in tabular settings.

Shortcomings

Dueling networks have a neural architecture as a function approximation scheme which increases the computational complexity of the algorithm. Moreover, as previously explained, there are no theoretical convergence guarantees for the algorithm. Being a value-based method, dueling networks are also difficult to scale to large action spaces due to the curse of dimensionality. The technique is better suitable for working with discretized action-spaces.

5-5 Results

In this section we compare the performances of the two algorithms on the cartpole problem over the same number of episodes, which can be seen in Table 5-4. The mean and max returns are normalized to be between 0 and 1, with 1 being the maximum reward that is achievable in an episode of the algorithm. For the purposes of fair comparison, we created a *deep* version of the advantage learning agent, which uses the deep Q-network framework for representing action-values, but is instead updated against the modified target from the advantage learning operator. The new target value is of the form $R(s,a)+\gamma \max_{a'} Q(s',a';\theta_i^-)+\eta\left(Q(s,a;\theta_i^-)-\max_a Q(s,a;\theta_i^-)\right)$. The parameters for the agent we used were the same as that seen in Table 4-3.

Agent	f_x	Rise Time	Settling Time	Mean Return	Max Return
Advantage Learning	1	189	610	0.786	1
	10	248	724	0.814	1
Dueling Network	1	386	591	0.848	1
	10	455	531	0.841	1

Table 5-4: Benchmarking state-of-the-art agents across timescales on the cartpole problem

The results are reported over 5 random trials. From the results, it can be seen that both the deep advantage learner and the dueling network perform well across frequencies, indicating their timescale agnostic nature. Rise times for the dueling network is seen to be slightly longer than the deep advantage learner, but this is compensated by the settling times, which are faster for the advantage learner. These results indicate that both agents are timescale-agnostic, and perform satisfactorily.

The performance curves of both agents on the cartpole can be seen in Figure 5-3 and Figure 5-4.

5-6 Summary

In this chapter, we argued the importance of the OP/GI properties of operators for tackling disappearing reinforcements at higher frequencies. We explained theorems to select such an operator. We also empirically showed that algorithms making use of operators which possess these properties do indeed perform satisfactorily at higher frequencies, and that this performance is correspondent to their larger action-gaps. We also show that the action-gaps

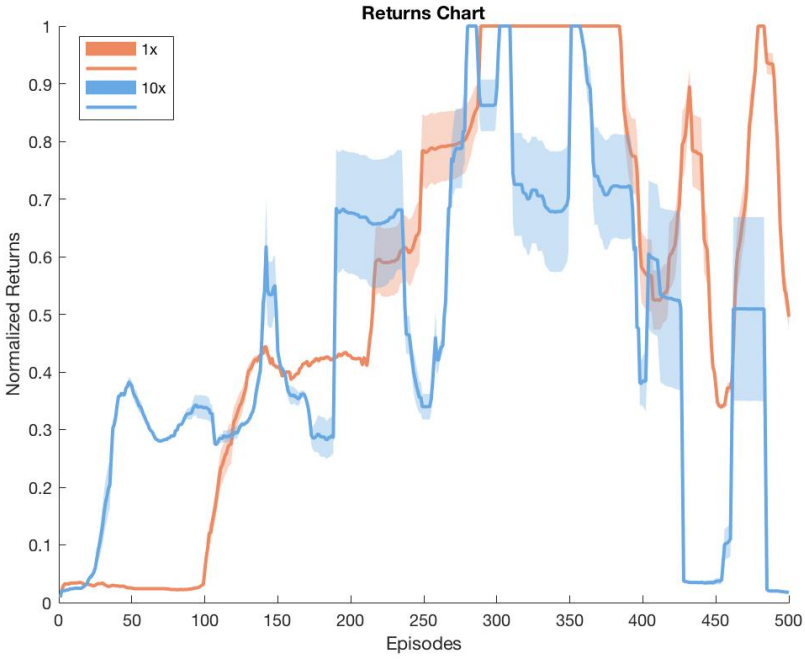


Figure 5-3: Advantage Learning - Returns per episode for different frequencies on Cartpole

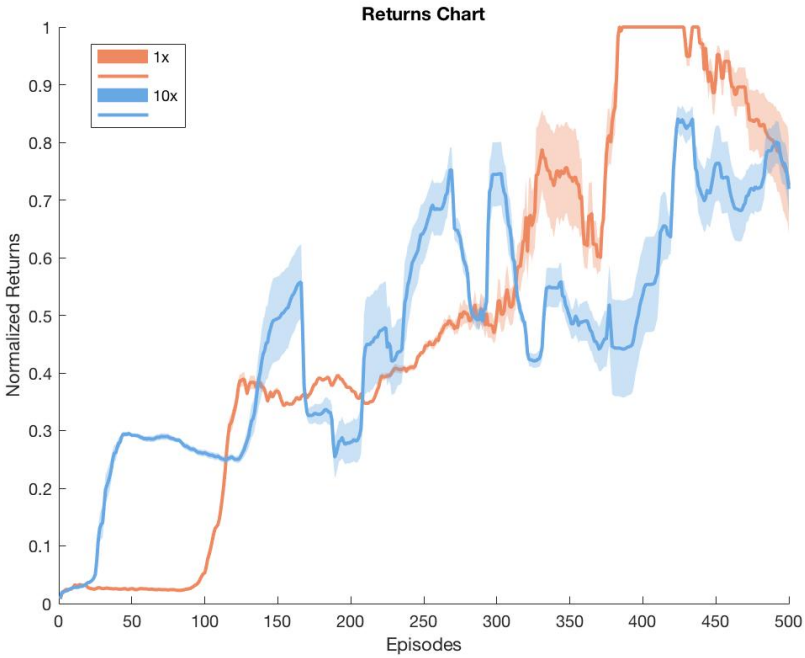


Figure 5-4: Dueling Network - Returns per episode for different frequencies on Cartpole

and therefore the performance is unaffected by noisy updates across frequencies, thereby showing that the advantage learning operator as an example of an OP/GI operator based agent is timescale-agnostic.

We also illustrated a second approach to solve the disappearing reinforcements problem, utilizing the dueling network architecture to improve the estimation of the action-values themselves. We established the importance of a clearly defined advantage and value function stream. Convergence guarantees and shortcomings of both approaches were also discussed. We then went forward and benchmarked the performance of both timescale agnostic approaches on the cartpole problem, for which the results seen were promising.

Chapter 6

Extending Solutions

We explored two approaches to alleviate disappearing reinforcements in the previous chapter, namely by creating operators that increase action-gaps, and by improving the robustness of the Bellman operator itself so it produces better estimates of the Q-values. In this chapter we put forward our extensions to the state-of-the-art. We formulate a novel reinforcement learning agent, the dueling advantage learner, that mixes both the approaches from the state-of-the-art explained in the previous chapter.

The rest of the chapter is structured as follows. Section 6-1 introduces the concept of the dueling advantage learner. Section 6-1-1 delves into a sensitivity analysis of a correction-term parameter η with respect to the performance of the algorithm on the cartpole balancing task. Section 6-1-2 benchmarks the algorithm on the cartpole task by using criteria discussed in Section 4-2-1. Finally Section 6-1-3 and Section 6-1-4 deal with the convergence guarantees and the disadvantages of employing the agent, respectively.

6-1 Dueling Advantage Learners

The first approach to solving disappearing reinforcements which we explored in Chapter 5 dealt with the use of optimality-preserving and gap-increasing operators. The concept of discounting sub-optimal actions come out to be highly beneficial to our cause as the advantage learning operator has illustrated. In the previous chapter, we created a deep advantage learning agent, which used the framework of the DQN for benchmarking purposes. The framework of the DQN utilized the concept of splitting target networks and action-value networks. The DQN utilized experience replay based mini-batch updates which were explained in Section 3-4-1, in which gradient descent is performed on a squared temporal difference (TD) error $\Delta Q(s,a)^2$. The TD error $\Delta Q(s,a)$ is defined as:

$$\Delta Q(s,a) = R(s,a) + \gamma V(s') - Q(s,a) \tag{6-1}$$

for the Bellman operator given by $\tau Q = R(s,a) + \gamma \mathbb{E}_{Pa} \max_{a'} Q(s',a')$. As mentioned in Section 5-3, the advantage learning operator, which is a modified version of the Bellman operator is defined as:

$$\tau_{AL} Q(s,a) = \tau Q(s,a) - \eta \left[V(s) - Q(s,a) \right] = \tau Q(s,a) + \eta A(s,a) \tag{6-2}$$

For advantage learning, the temporal difference error can be modified to resemble the Q-learning TD-error with a correction term:

$$\begin{aligned} \Delta_{AL} Q(s,a) &= \Delta Q(s,a) + \eta \left[Q(s,a) - V(s) \right] \\ &= \Delta Q(s,a) + \eta A(s,a) \end{aligned} \tag{6-3}$$

With a neural network as the function approximator, the value of $Q(s,a)$ is approximated to be $Q(s,a,\theta)$, with θ being the parameters of the Q-network function approximator. Using the framework of the DQN, the loss function (considering a mean absolute loss function) to be minimized for advantage learning thus becomes:

$$L_i(\theta_i) = \mathbb{E}_{Pa} \left[\left| R(s,a) + \gamma \max_{a'} Q(s',a';\theta_i^-) + \eta \left(Q(s,a;\theta_i^-) - \max_{a} Q(s,a;\theta_i^-) \right) - Q(s,a;\theta_i) \right| \right] \tag{6-4}$$

with θ_i being the parameters for the action-value network and θ_i^- are the target network's parameters at iteration i.

The formulation of the advantage learning operator requires the parameter $\eta \in [0,1)$ used to be a function of the sampling time dt. The exact formulation of the term according to Bellemare et al. [38], is $\eta = (1 - Cdt)$ with C being an arbitrary scaling constant for dt, the sampling time. The term alludes to the strength of our belief in the ordering of action values, with a value close to one indicating a strong belief in the ordering due to the small amount of corrections required during the update, and a value close to zero indicating a weak belief due to the larger magnitude of corrections. The advantage learning operator, as explained in Chapter 5 creates an optimality-preserving and gap-increasing operator, and this improves performances at higher frequencies.

The second approach we explored concerned the use of a dueling network architecture that helps in better estimating the Q-values themselves. As explained in Section 5-4, the dueling architecture provides a convenient way to separate out the value function from the advantage function, thereby creating separate channels in the neural architecture that behave with distinct behavioral characteristics [20]. The value function channel of the dueling network exhibits a tendency to look towards the long term horizon, as the value function is representative of the long term rewards gathered by the agent. The dueling network architecture forces the value function and advantage functions to be formed, thereby creating better estimates of the Q-values themselves when both channels are combined. This makes it robust towards noise in the updates, which becomes especially apparent at higher frequencies when there are large numbers of updates before the terminal state is reached.

We combine this approach of an advantage learning agent with the architecture of the dueling network (explained in Chapter 5) to create a novel *dueling advantage learner*. We harness the property of creating more robust Q-value estimates from the dueling network architecture and

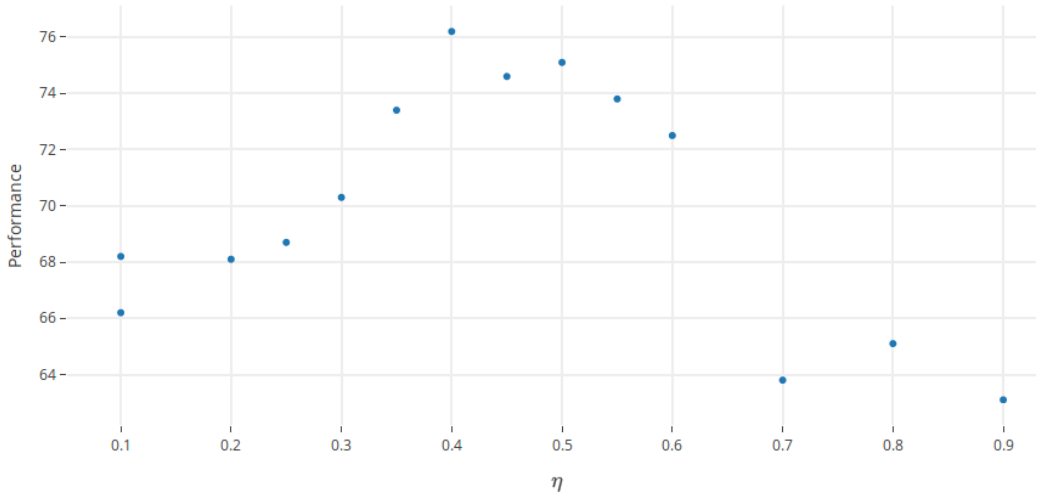


Figure 6-1: Scatterplot - Performance (in mean returns) versus η

the OP/GI property of the advantage learning operator, to formulate the dueling advantage learning agent. For this, we utilize the dueling architecture for representing Q-values and utilize the advantage learning operator for the update steps. What this ensures is that there are increased values of the action-gaps for the Q-values represented, and the values themselves have improved estimates. For implementing the dueling advantage learner, we chose to implement the advantage channel subtracted by the average values over all actions for practical purposes. We also tried using a double Q-learning [35] version of the advantage learning update rule, but we discarded this as we did not see too much improvements from doing this over the standard update rule during our experiments. We also used the same structure of the dueling network as discussed in Section 5-4.

6-1-1 Parametric analysis

We perform a parametric analysis to understand how the performance is affected by the value η of the dueling advantage operator. To create this, we run the agent on the cartpole-balancing task and store the average returns over an entire trial run for a frequency multiplier of 1x (50Hz). To specifically get the effect of changing values of η, we decided to keep the rest of the variables as constant. The results from the experiment can be seen in the scatter-plot as seen in Figure 6-1.

From the results seen in the figure, we chose a value of η to be 0.4 for the rest of the experiments, as it provided the best mean returns overall. The experiment was run for a length of 150 episodes, with the values represented in the scatter-plot being the average mean returns over five trial runs of the experiment.

6-1-2 Results

We test the dueling advantage learner on the cartpole agent and estimate the efficacy of our agent, with the benchmarks described in Section 4-2-1. The results from simulating the

dueling advantage learner over 5 random runs for 1000 episodes on the cartpole can be seen in Table 6-1, and the performance curves across frequencies can be seen in Figure 6-2.

Agent	f_x	Rise Time	Settling Time	Mean Return	Max Return
	1	186.58	368	0.8664	1
Dueling Advantage Learner	10	228	592	0.899	1

Table 6-1: Results from simulating the Dueling Advantage Learner on the cartpole environment

For the benchmarking experiments, we chose the values for the parameters seen in Table 6-2. The results indicate that the dueling advantage learner has improved performances when compared to the individual agents we combined to create it (the results of which can be seen in Table 5-4). The results also indicate that the dueling advantage learner is timescale agnostic, as expected.

Parameter	Value
γ	0.99
η	0.4
Mini-batch size	32
Replay Memory	1000000
Error	Mean Absolute Error
Target update	0.01 (Soft Updates)
Policy	ϵ-greedy
$\epsilon_{\text{initial}}$	1
ϵ_{Final}	0.1
Decay rate	0.99, exponential over every step.
Optimizer	Adam (α=0.0002, β_1=0.9, β_2=0.999, $\hat{\epsilon}$=$1e-8$, decay=0.0)

Table 6-2: Dueling Advantage Learner - Parameters of choice

6-1-3 Convergence guarantees

Using neural function approximation on reinforcement learning algorithms removes any theoretical guarantees of convergence. The dueling advantage learner is no exception. That said, Bellemare et al., [38] proved that the operator on which the agent is based on, the advantage learning operator, has theoretically assured guarantees of convergence. Empirically, it can also be seen that the dueling advantage learner converges in a finite amount of time, under favourable initialization of the agent's parameters.

6-1-4 Shortcomings

The complex neural architecture of the dueling advantage learner introduces a level of computational complexity to the task at hand. But this can be overlooked, as the increase in complexity is accompanied by an increased learning rate accompanied by lower estimation errors at higher frequencies. The agent also has no theoretically-assured properties of convergence, even though it can be empirically shown to converge.

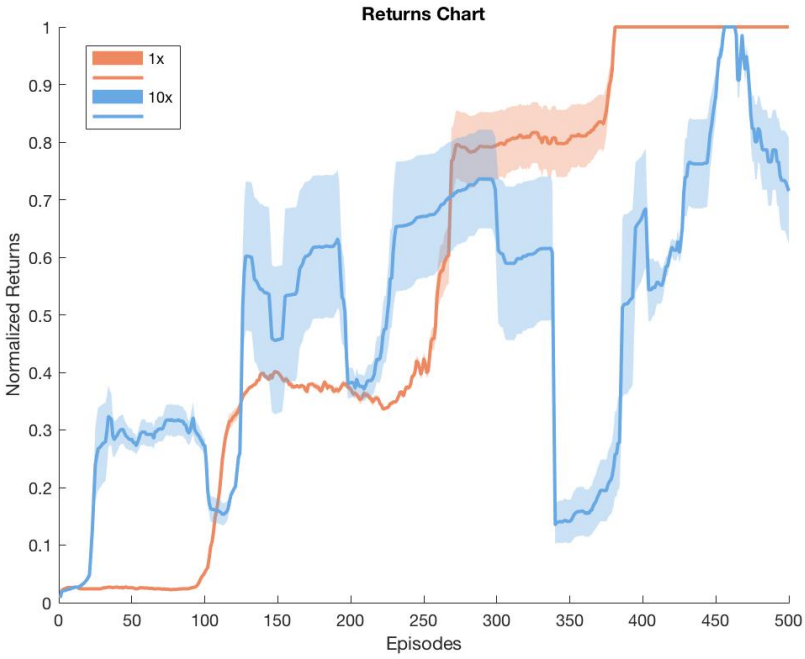


Figure 6-2: Dueling Advantage Learning - Returns per episode for different frequencies on Cartpole

6-2 Summary

We propose the dueling advantage learner as a novel agent that achieves improved performances at higher frequencies. Formulation of such an agent requires a combination of the ideas of the dueling network architecture and the advantage learning operator. The agent is seen to be timescale agnostic when it comes to its performance, indicating that it alleviates the effects of disappearing reinforcements. This makes it one of the most attractive agents out of the ones studied for multi-task reinforcement learning.

Chapter 7

Benchmarking and Testing

Benchmarking algorithms is a difficult task. It can be inferred from literature that creating consistently reproducible statistically significant results under similar conditions is difficult, with various sources at times seen citing different results [43]. In this chapter we compare the performance of different agents that are potential solutions to the disappearing reinforcements problem. We compare the performance achieved by the algorithms across frequencies on different environments, by taking the mean returns gathered over a similar number of episodes as a benchmark. This is scientifically important, as studies of value-based methods on control tasks, and studies of agents across frequencies are seldom seen in literature.

7-1 Benchmarking solutions

In order to understand the efficacy of the different solutions, we benchmark the 2 state-of-the-art agents along with the dueling advantage learner across different control-based test scenarios and frequencies. The deep version of the advantage learning agent discussed in Chapter 5 is used for the advantage learning agent.

We use the mean returns gathered over episodes, which is a criterion frequently seen in literature. Although we created benchmarks that can be used to produce more meaningful results than just the mean returns, our task is made difficult by the large number of results that we'd have to convey, as our benchmarks span frequency ranges, in addition to different types of tasks. Hence we decided to focus on mean returns for this exercise. The results we gather can be seen in Table 7-1. The testing was done with the agents using the parameters mentioned in Chapter 5 and Chapter 6.

The testing was done for three classic control tasks seen in literature, namely the cartpole balancing task which was explained in previous chapters, along with the acrobot swing up task [44] and the mountain car problem [45]. Variants of the tasks are also considered, namely with added observation noise, with delayed actions, and also as a Partially Observable MDP task, with the dynamics of the system changing at each step of the task.

The results can be seen in Table 7-1, and the results in bold indicate a statistically significant improvement. This we ascertained by checking the p-values of the sets of results obtained at a certain frequency on a certain task, with a statistically significant result indicated by a p-value of around 0.1 or lower. In case of a tie, all best results were marked in bold.

The details of the tasks are as follows. All agents were simulated for 150 episodes, with the results reported being the mean returns and standard deviations averaged over five random trials.

Cartpole balancing

The cartpole balancing task explained in Section 4-2-1 is chosen as one of our benchmark tasks. The base frequency at which the cartpole operates at is chosen as 50Hz.

Acrobot swing-up

The acrobot swing-up task requires the agent to swing the lower link of a two-link two-joint under-actuated robotic arm over a specified line, when the links initially hang downwards. The setup can be seen in Figure 7-1.

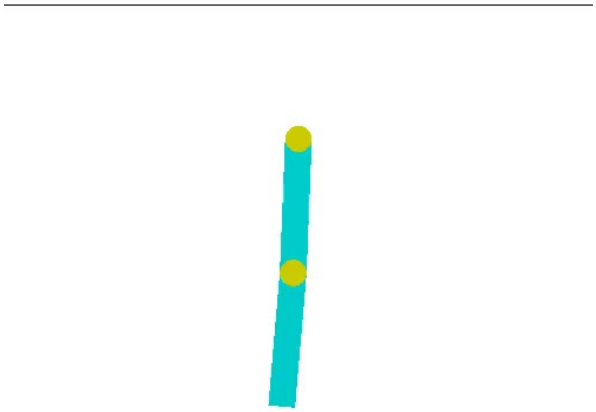


Figure 7-1: Acrobot swing-up task. The horizontal line signifies the level that the lower link needs to swing up over.

The rewards from the environments are a value of -1 for every step if the lower link has not reached the terminal state. The objective of the task is to execute the swinging-up action quickly. The base frequency for the operation of the cartpole is chosen as 5Hz. There are three discrete actions for the agent to choose from, a positive torque, negative torque and no torque.

Mountain-car climb-up

The mountain car climb-up task, seen in Figure 7-2, is a popular task used to benchmark control algorithms. The objective of the task is to drive a car on a one-dimensional track up a mountain. The engine of a car is also not powerful enough to make it traverse the mountain in one pass, so it needs to drive back-and-forth in order to build momentum to scale the height. The mountain-car problem has a negative reward structure similar to the acrobot, which is obtained at each step before reaching the terminal state denoted by the flag atop the mountain. Hence the objective of the task is to scale the height of the mountain as quickly as is possible. The base frequency at which the mountain-car operates at is chosen to be 10Hz. The agent also can take one of 3 discrete actions at every step which correspondingly allow it to accelerate, decelerate or do nothing.

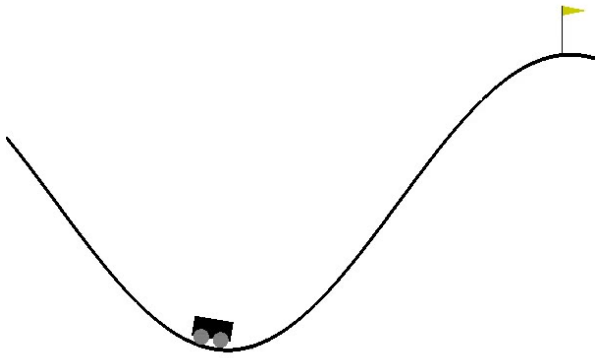


Figure 7-2: MountainCar climb-up task. The flag at the top of the mountain signifies the level that the car needs to cross.

7-1-1 Standard tasks

From the results on the standard version of tasks described above (seen in Table 7-1 and Figure 7-3), it can be understood that the dueling advantage learning agent performs on average better than the other two agents. The performance of the advantage learner can also be seen to be considerably worse than the other two agents on the acrobot environment. It can also be inferred from the results that the dueling network is the second-best performing agent on the standard tasks. All agents are also seen to be relatively timescale agnostic, with some seen to improve performances with increasing frequencies.

7-1-2 Observation noise and action delays

We use an additive Gaussian random noise with zero mean and an arbitrary standard deviation of 0.01 to the observed states of agents, so as to simulate how the agents behave when faced with noisy sensors in the real world. Along with this, the execution of the actions are delayed by a value of 0.01s so as to simulate physical delays during real-world operation. The results from this section indicate that the performance of the agents are relatively robust to noise and action-delays, even though there is a slight decrease in the overall performance

when in comparison with the standard versions of the tasks. The agents are also seen to be relatively timescale agnostic. Here as well, the advantage learning agent is seen to perform relatively worse when compared to the other two agents on the acrobot environment. Overall, the dueling advantage learner is seen to outperform other agents on the majority of the tasks.

7-1-3 System identification

The system identification task deals with uncertainty in the task we simulated on. We inject an additive Gaussian random noise of zero mean and 0.001 standard deviation to the dynamics of the environment, so the behavior of the model becomes slightly different at every step of the experiment. This aspect of modelling is useful as it provides insight into how the agent would work when operating in a real-world environment, which could be dynamic. The results still portray timescale agnostic performances for all agents, with the dueling advantage learner performing especially well at the 5x frequencies when in comparison with other agents. The dueling advantage learner is also on average better performing than the other agents.

Environment	f_x	Duel Net	AL	DAL
Cartpole	1	**70.07 ± 1.9**	**65.4 ± 4.4**	**66.2 ± 1.4**
	2	**140.2 ± 9.3**	105.2 ± 12.1	**138.4 ± 3.2**
	5	149.1 ± 12.9	143.5 ± 32.3	**164.9 ± 15.3**
Acrobot	1	-133.9 ± 6.775	-266.18 ±2.7	**-125.1 ± 1.2**
	2	**-123.9 ± 0.3**	-256.2 ± 5.2	**-126.3 ± 8.2**
	5	**-145.3 ± 7.27**	-237.5 ± 23.3	**-132.3 ± 49.9**
Mountain Car	1	**-192.5 ± 0.6**	**-192.4 ± 2.2**	**-192.1 ± 3.1**
	2	-170.7 ± 6.82	-170.4 ± 2.5	**-147.4 ± 3.4**
	5	**-98.8 ± 2.2**	**-96.2 ± 1.3**	**-101.8 ± 4.6**
Cartpole (NO/DA)	1	69.9 ± 10.8	59.6 ± 7.7	**75.4 ± 8.2**
	2	94.5 ± 23.1	105.8 ± 8.37	**124.2 ± 7.7**
	5	116.3 ± 13.7	**144.8 ± 15.2**	**142.9 ± 34.4**
Acrobot (NO/DA)	1	-141.4 ± 3.4	-258.3 ± 17.2	**-117.9 ± 4.7**
	2	-142.5 ± 25.3	-236.3 ± 6.7	**-129.5 ± 3.6**
	5	-168.7 ± 32.7	-228.1 ± 10.2	**-136.7 ± 7.2**
Mountain Car (NO/DA)	1	-195.7 ± 1.78	-196.3 ± 2.5	**-188.3 ± 1.5**
	2	**-184.5 ± 14.5**	**-182.2 ± 9.9**	**-181.6 ± 11.2**
	5	-170.5 ± 4.9	**-164.6 ± 5.2**	**-159.4 ± 23.5**
Cartpole (SI)	1	**65.3 ± 21.8**	**61.7 ± 5.5**	**71.7 ± 15.9**
	2	91.8 ± 6.9	**109.5 ± 14.9**	**123.61 ± 10.3**
	5	103.6 ± 18.2	121.5 ± 7.6	**138.3 ± 13.6**
Acrobot (SI)	1	-127.0 ± 1.41	-273.2 ± 4.4	**-117.6 ± 9.2**
	2	**-135.5 ± 11.3**	-265.4 ± 8.2	**-134.6 ± 27.6**
	5	-209.6 ± 51.02	-243.4 ± 36.5	**-166.2 ± 26.4**
Mountain Car (SI)	1	**-194.6 ± 3.09**	**-191.3 ± 5.8**	**-191.7 ± 2.5**
	2	**-180.8 ± 12.0**	-221.6 ± 3.4	**-184.0 ± 21.75**
	5	-165.2 ± 9.4	-203.4 ± 12.6	**-134.6 ± 17.52**

Table 7-1: Benchmarking algorithms over different frequencies wrt mean returns/standard deviations - NO/DA refers to the inclusion of noisy observations and delayed actions, and SI refers to the system identification tasks. Duel Net refers to the dueling network architecture updated by Bellman updates, AL refers to the advantage learning operator using the deep Q-network framework, and DAL refers to the dueling advantage learner.

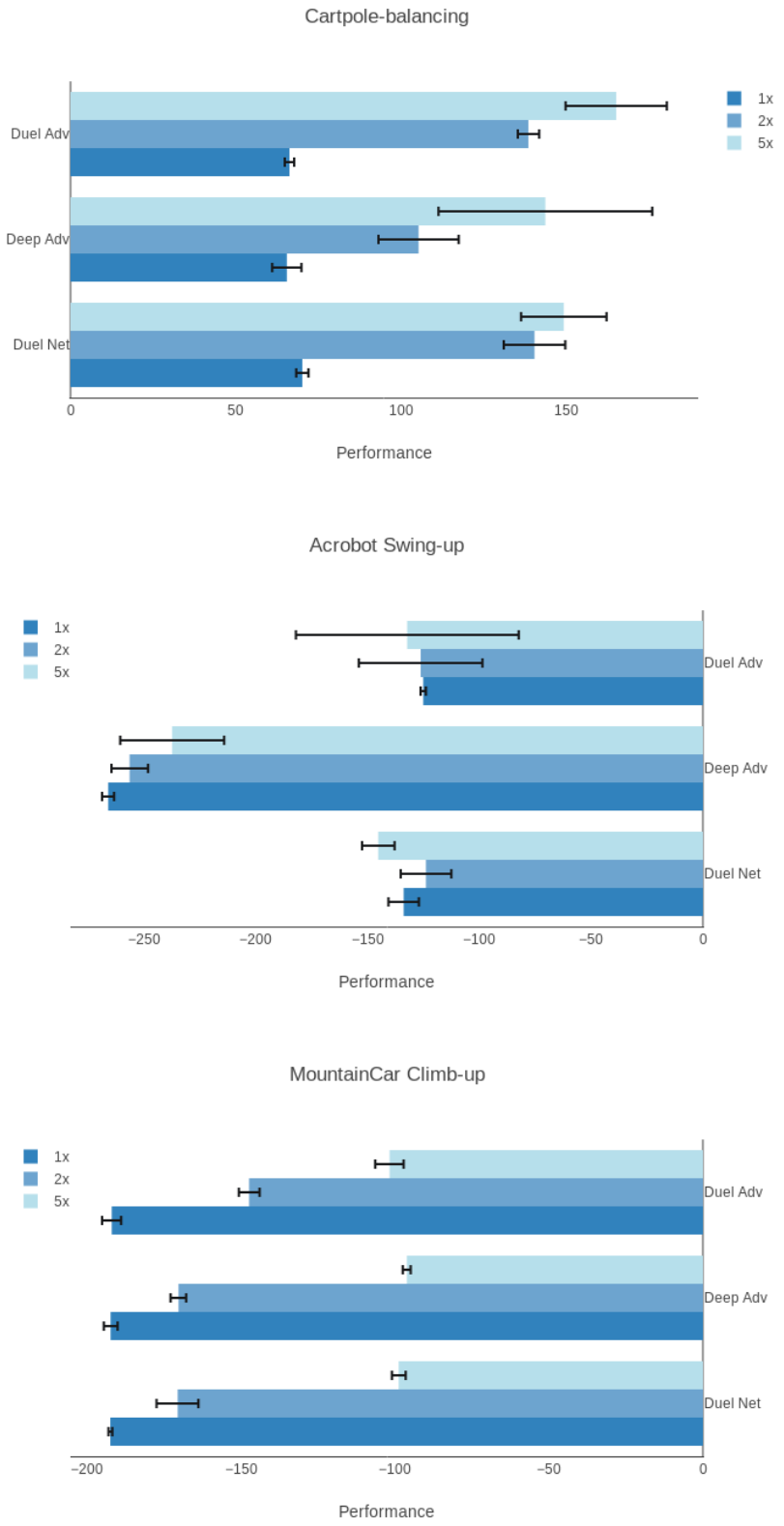


Figure 7-3: Performance (in mean returns) on the standard cartpole, mountain-car and acrobot environments

Figure 7-4: Performance (in mean returns) on the cartpole, mountain-car and acrobot environments, with noisy observations and delayed actions.

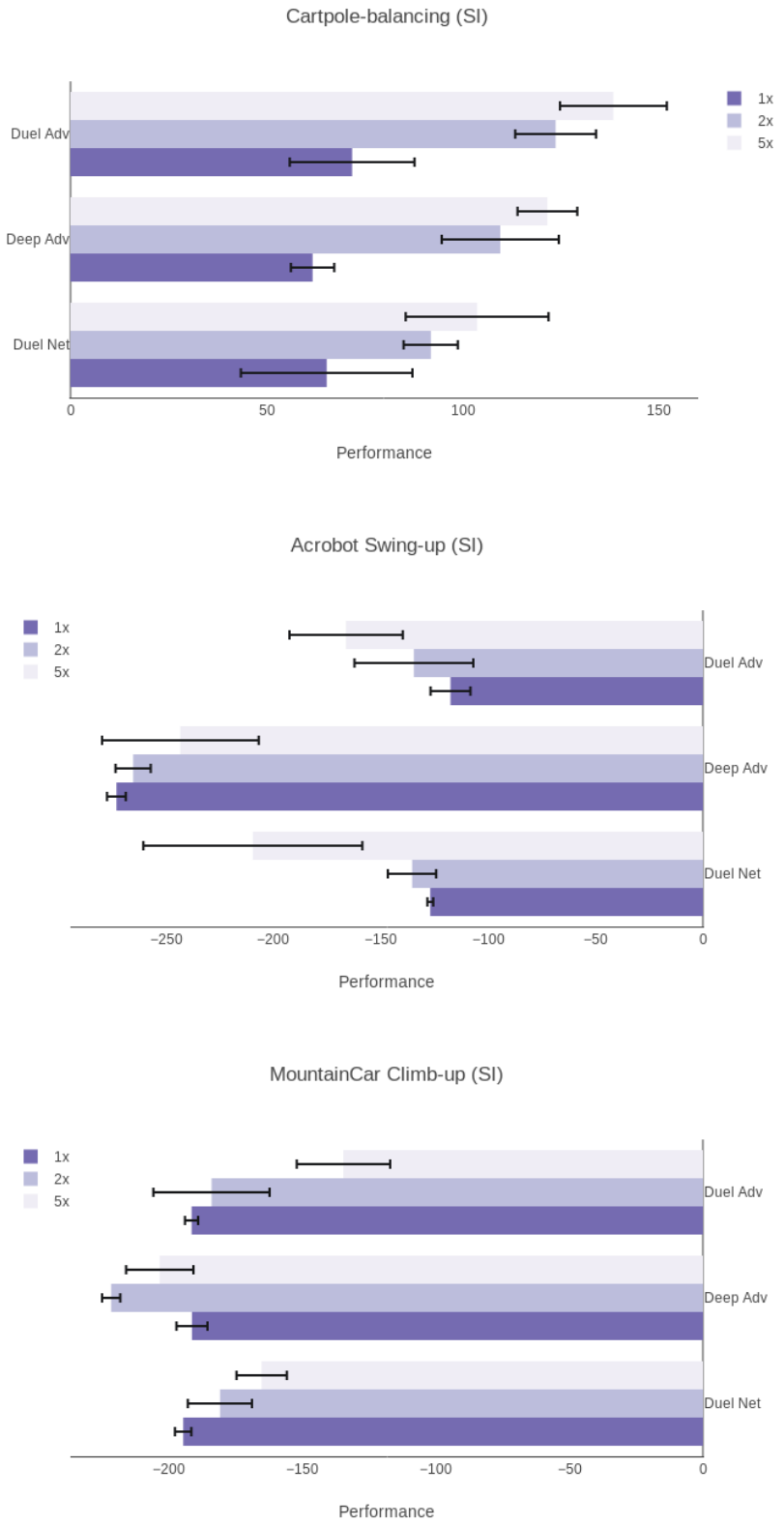


Figure 7-5: Performance (in mean returns) on the cartpole, mountain-car and acrobot environments as system identification tasks.

Chapter 8

Conclusions and Future Work

In this thesis we put forward the idea that operating reinforcement algorithms at higher frequencies is useful, by looking at general rules from the field of optimal control. We then established the relation between small action-gaps, estimation errors of Q-values and the decreasing performance of common reinforcement learning algorithms at increasing frequencies. This we called the disappearing reinforcements problem. We explored two approaches from the state-of-the-art aimed at alleviating the problem. The first approach concerned the use of agents updated by the advantage learning operator, which is an optimality-preserving and action gap-increasing operator. The second approach concerned the use of a dueling network architecture for deep reinforcement learning agents, which creates more robust estimates of action-values. We also defined novel benchmarks inspired from control theory to benchmark the performance of the agents.

Finally, we put forward our agent that combines both approaches and improves on the state-of-the-art. This we called the dueling advantage learning agent. We then proceeded to benchmark the agent on three different tasks, namely on the cartpole balancing task, the acrobot swing-up task and the mountain-car climb-up task. The agents were also tested on variants of these tasks to simulate real-world scenarios. We added observation noise and action-delays to the agent, as well as created variants of the tasks with the underlying dynamics of the task subject to change at every step. The dueling advantage learner was seen to outperform the other agents on a large number of the tasks.

The concept of reinforcement learning as control algorithms operating across frequencies has been rarely explored in literature. Explanations as to why agents behave in a certain way on a specific platform have almost always been geared towards other parts of the task, such as the complexity of the task itself. While this may be true in many cases, it is important to understand that many of the popular tasks that are used as benchmarks operate at different frequencies. This thesis hopefully has brought to light the importance of the looking at the reinforcement learner in a new light, as a control system that operates at different frequencies.

It is also untrue to be stating that the disappearing reinforcements problem is one that is solved. Although the approaches detailed in this thesis address the issue to a large extent,

there is still much work to be done to improve the agent's performance at higher frequencies. But the results from this thesis is testament to the fact that the combination of older and simpler agents and ideas can give rise to superior agents that perform well.

8-1 Future Work

In this section we detail other approaches which can be utilized to create more robust agents for alleviating the disappearing reinforcements problem. Moreover, we also detail some open research questions to be explored in the future.

- **Persistent Dueling Advantage Learner**: Bellemare et al., (2016) [38] devised the Persistent Advantage Learning agent as an improvement to the advantage learning, while preserving the original operator's optimality-preserving and gap-increasing properties. The algorithm makes use of a max operator to select the TD error term. The Persistent Advantage Learning operator is expressed as:

$$\tau_{PAL}Q(s,a) = \max\left\{\tau_{AL}Q(s,a), R(s,a) + \gamma\mathbb{E}_{Pa}Q(s,a)\right\} \tag{8-1}$$

  The persistent advantage learning agent was introduced as a measure to encourage repeated actions [38]. This operator can also be used on the dueling network architecture, similar to the dueling advantage learner we devised.

- **Prioritized experience replay**: For the purpose of facilitating a fair and easy-to-understand comparison of agents, we decided to forgo complexity in favour of simplicity to the agents being compared in this thesis. There have been other techniques devised in literature to improve results and generalization ability of deep reinforcement learning agents. The concept of the prioritized experience replay [46] is one such method, and this can also be used in place of the experience replay being utilized for the updates.

- **Other operators for reinforcement learning**: Although research has pointed to the efficacy of alternative operators to the Bellman operator (such as the advantage learning operator), there still is no proof for the existence of a maximally efficient operator. Studies about the statistical efficiency of different operators in use are also rare [38].

- **Theoretical criteria for convergence**: Although it can be seen that the agents converge empirically, a theoretical guarantee for the convergence of neural reinforcement learning agents is rarely seen in literature, and ought to be studied in the future.

Appendix A

Implementation details

All algorithms run on the point-mass MDP including the physics simulator itself were custom built and run on azure cloud notebooks[1]. For the deep reinforcement learning agents, a computer with 16 GB of RAM, and a GPU with 640 parallel processing cores and 4GB of GDDR5 SDRAM was used. All simulations with the exception of the point-mass MDP were run with the OpenAI gym [15] framework. The deep reinforcement learning agents were created in Keras [47], running on a tensorflow backend [48].

[1] https://notebooks.azure.com/

Appendix B

Algorithms

Algorithm 1 One-Step Q-Learning [1]

Initialize $Q(s,a), \forall s \in \mathbb{S}, a \in \mathbb{A}$ arbitrarily, and $Q(terminal-state,\cdot)=0$
repeat(for each episode):
    Initialize s
    repeat(for each step of the episode):
        Choose a from s using policy derived from Q
        Take action a, observe $R(s,a), s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
        $s \leftarrow s'$
    until s is terminal

Algorithm 2 Deep Q-learning [30]

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = 0$
for episode= 1, M **do**
    Initialize state sequence
    for $t = 1$, T **do**
        With probability ϵ select a random action a_t
        Otherwise select $a_t = \text{argmax}_a \, Q(s, a; \theta)$
        Execute action a_t in emulator and observe reward R_t and state x_{t+1}
        $s_{t+1} = s_t, a_t, x_{t+1}$
        Store transition (s_t, a_t, R_t, s_{t+1}) in D
        Set y_j as

$$y_j = \begin{cases} R_j & \text{if episode terminates at step} j+1 \\ R_j + \gamma \max'_a \hat{Q}(s_{j+1}, a'; \theta') & \text{otherwise} \end{cases} \tag{B-1}$$

        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to network parameters θ
        Every C steps reset $\hat{Q} = Q$
    end for
end for

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[2] M. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, Learning, and Optimization*, vol. 12, 2012.

[3] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, no. 56, pp. 441–470, 1998.

[4] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Reinforcement Learning: State of the Art*, pp. 579–610, Springer, 2012.

[5] B. Awerbuch, D. Holmer, and H. Rubens, "Provably secure competitive routing against proactive byzantine adversaries via reinforcement learning," *John Hopkins University, Tech. Rep*, 2003.

[6] I. Erev and A. E. Roth, "Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria," *American economic review*, pp. 848–881, 1998.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, jan 2016.

[8] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 1997.

[9] G. Kreisselmeier, "On sampling without loss of observability/controllability," *IEEE Transactions on Automatic Control*, vol. 44, no. 5, pp. 1021–1025, 1999.

[10] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[11] K. J. Astrom and B. Wittenmark, *Computer-controlled Systems (3rd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[12] I. P. Pavlov and G. V. Anrep, *Conditioned Reflexes; an Investigation of the Physiological Activity of the Cerebral Cortex*. London: Oxford Univ. Press, 1927.

[13] C. Hull, *Principles of Behavior*. New York, USA: Appleton-Century-Crofts, 1943.

[14] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 11 1954.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[16] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," 1994.

[17] L. Baird *et al.*, "Residual algorithms: Reinforcement learning with function approximation," in *Proceedings of the twelfth international conference on machine learning*, pp. 30–37, 1995.

[18] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.

[20] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[21] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[22] A. Gruslys, M. G. Azar, M. G. Bellemare, and R. Munos, "The reactor: A sample-efficient actor-critic architecture," *arXiv preprint arXiv:1704.04651*, 2017.

[23] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: an overview," in *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, vol. 1, pp. 560–564, IEEE, 1995.

[24] M. L. Littman, "Algorithms for sequential decision making," 1996.

[25] "David Silver's lectures at UCL." http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching_files/control.pdf. Accessed: 2016-11-02.

[26] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[27] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[28] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, pp. 2829–2838, 2016.

[29] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] H. Modares, F. L. Lewis, and M.-B. Naghibi-Sistani, "Integral reinforcement learning and experience replay for adaptive optimal control of partially-unknown constrained-input continuous-time systems," *Automatica*, vol. 50, no. 1, pp. 193–202, 2014.

[32] A.-m. Farahmand, "Action-gap phenomenon in reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 172–180, 2011.

[33] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[34] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

[35] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning.," in *AAAI*, pp. 2094–2100, 2016.

[36] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[37] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *International Symposium on Abstraction, Reformulation, and Approximation*, pp. 194–205, Springer, 2005.

[38] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. Munos, "Increasing the action gap: New operators for reinforcement learning.," in *AAAI*, pp. 1476–1483, 2016.

[39] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[41] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[42] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[43] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," 2017.

[44] A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How, "Rlpy: a value-function-based reinforcement learning framework for education and research.," *Journal of Machine Learning Research*, vol. 16, pp. 1573–1578, 2015.

[45] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in neural information processing systems*, pp. 369–376, 1995.

[46] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[47] F. Chollet *et al.*, "Keras," 2015.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.