Temporal Credit Assignment via Traces in Reinforcement Learning

Nishanth Anand Vemgal

Computer Science
McGill University, Montreal

December 15, 2019

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science.

Abstract

Reinforcement Learning is a framework for sequential decision making which is widely used in many domains such as robotics, autonomous driving, etc. Due to the sequential nature there exists the problem of assigning the credit to the actions taken in the past. This problem in reinforcement learning is known as temporal credit assignment. The problem of temporal credit assignment lies in the core of many methods such as options, online learning, off-policy learning, etc. within reinforcement learning framework. Several problems such as high variance in the value function estimates, sub-optimal policy, high sample complexity are a consequence of improper temporal credit assignment in reinforcement learning.

In this thesis, we introduce and examine a couple of temporal credit assignment techniques. Specifically, we mitigate the problem of variance in value function by effectively assigning credit. First, we discuss the fundamental concepts of signals and reinforcement learning. Then, we introduce Recurrent Learning which smooths the value function along the trajectory. We then analyze the strengths of Recurrent Learning experimentally. Finally, we introduce filters from signal processing as a general framework for various traces in reinforcement learning. We show the effectiveness of filters with a couple of toy examples.

Résumé

L'apprentissage par renforcement est un cadre pour la prise de décision séquentielle qui est largement utilisé dans de nombreux domaines tels que la robotique et la conduite autonome. En raison de la nature séquentielle de l'apprentissage par renforcement, attribuer le crédit aux actions prises dans le passé est un problème compliqué. Ce problème est connu sous le nom d'affectation temporelle de crédits et est au cœur de nombreuses méthodes telles que les options ou l'apprentissage en ligne. Une mauvaise attribution du credit temporelle peut avoir multiple conséquence dans l'apprentissage par renforcement comme de la variance élevée dans les estimations de la fonction de valeur ou une politique sous optimale. Dans cette thèse, nous introduisons et examinons plusieurs techniques d'affectation temporelle de crédits. Plus précisément, nous atténuons le problème de la variance de la fonction de valeur en affectant efficacement le crédit. Nous présentons d'abord les concepts fondamentaux de l'apprentissage des signaux et du renforcement. Ensuite, nous introduisons l'apprentissage récurrent qui lisse la fonction de valeur le long de la trajectoire. Nous analysons ensuite les points forts de l'apprentissage récurrent de manière expérimentale. Enfin, nous introduisons les filtres, utilise pour le traitement du signal, comme cadre général pour diverses traces dans l'apprentissage par renforcement. Nous montrons l'efficacité des filtres avec quelques exemples simples.

Acknowledgements

I am extremely grateful to my supervisor Doina Precup for providing me an opportunity to work on my interests. Her constant guidance, support and feedback helped me in various aspects of my work which led to refined ideas timely. I thank my co-authors Joelle Pineau, Pierre Thodoroff, and Lucas Caccia with whom I worked on some of the ideas presented in this thesis. I would like to thank Barleen Kaur for her continuous support and debates that I had throughout my masters. I am also thankful to all the people that I interacted and discussed with, in particular, Riashat Islam, Pierre-Luc Bacon, Jayakumar Subramanian, Srinivas Venkattaramanujam, Audrey Durand and Emmanuel Bengio. Finally, the most important, my parents for their support in many aspects of my life and for making my dream of pursuing masters possible.

Contribution of Authors

This thesis introduces a couple of techniques for temporal credit assignment in reinforcement learning.

- Chapter 2 and 3 are written for this thesis to introduce the basic concepts in signal processing and reinforcement learning.
- Chapter 4 introduces Recurrent Learning and is a joint work with Pierre Thodoroff and Lucas Caccia. Both Pierre and Lucas participated in the design of the algorithms and the experiments. This material is presented at RLDM [86].
- Chapter 5 introduces a general framework for traces in reinforcement learning. The idea originated from the discussions with Pierre-Luc Bacon.

Contents

List of Figures

List of Algorithms

1

Introduction

Building artificial intelligence agents has been a long-standing problem of humankind. The attempts to make intelligent bots began as early as 1956[1] and the science community has been working on it ever since. An ideal artificial intelligence agent knows how to react to its environment and make decisions optimally. Reinforcement learning, a problem in artificial intelligence, facilitates the agent-environment interaction and learning. Reinforcement learning is a sequential decision making framework where the learner interacts with the environment at every time step to gather experience. Reinforcement Learning is inspired by the trial and error learning from psychology. In this setting, the agent gathers experience by interacting with an environment in a trial and error manner to learn the optimal behavior. The learning can proceed either by estimating the optimal behavior directly or by estimating the value function which in turn finds the optimal policy. The methods that find the optimal behavior (policy) are referred to as policy gradient methods [100, 84] whereas the methods that estimate the value function are called value based methods [79, 78].

In a different dimension, reinforcement learning algorithms can be separated either as model-free [80, 81] - where the agent learns the optimal behavior using the gathered experience or as model-based [77] - where the agent builds a model of the environment to find the optimal behavior. Model-free learning has been widely used as learning models of the environment is complicated. Model-free algorithms has shown a great

[1] first workshop on Artificial Intelligence [46]

success in many different domains such as robotics [36, 1], video games [45, 94]. It is also used in real-life applications such as autonomous driving [71], financial trading [18] and others. However, their use in real world applications is limited due to many problems such as, high variance of the estimates (value function and policy) [25], poor sample complexity [31], etc. Many of these issues can be attributed to sub-optimal temporal credit assignment in the algorithms. Temporal credit assignment deals with assigning the credit/outcome of an event to an event that happened in the past which led to the outcome. A good temporal credit assignment technique identifies a decision in the past that led to a reward in the future and updates its learning accordingly. The importance of temporal credit assignment is also argued by the founder of artificial intelligence - Marvin Minsky [41] - "In applying such methods to complex problems, one encounters a serious difficulty in distributing credit for success of a complex strategy among the many decisions that were involved."

We humans do temporal credit assignment seamlessly in our daily lives. For instance, let's say we catch a stomach flu, we immediately can associate why we caught the flu. Probably we would associate with eating unhygienic food a few days before. Another instance is a road accident, where we most likely associate it with over speeding. In fact, credit assignment is a well studied topic in economic utility theory [64] and behavioral psychology. However, the problem of credit assignment is unresolved in those fields. This problem is also prevalent in artificial intelligence where it is unclear what's the best way to assign the credit to the decision taken in the past. Many different approaches have been taken in solving this problem. Some of these solutions are inspired by human memory system [33, 29], some on computing traces [79, 70, 82]. This thesis is an attempt to address the problem of temporal credit assignment in Reinforcement Learning. In this thesis, we take the approach of traces in addressing the problem of temporal credit assignment. More precisely, we develop two different approaches to solving temporal credit assignment. The first one uses the recursive traces and the second approach, inspired by signals, use a filtering approach to address temporal credit assignment.

The rest of the thesis is outlined as below: Chapter 2 discusses the basic definitions and properties of the signals. Signals are usually represented in two different domains, namely, time domain and frequency domain. In time domain, a signal is represented as a function of time. In frequency domain, a signal can be represented either as a Fourier transform or a z-transform. In Fourier transform, a signal is represented as a sum of sinusoidals. Signals are represented as a complex exponential term in z-domain. We introduce these representations and discuss a few properties in all these domains. This chapter lays foundation for Chapter 5.

Chapter 3 introduces the fundamentals of reinforcement learning. We begin by discussing the policy evaluation case, followed by control. In policy evaluation, the goal of the agent is to predict the value function by following a policy. This problem is also termed as prediction problem in reinforcement learning. We introduce several different techniques to estimate value function. Following this, we introduce control where the goal of the agent is to find the optimal policy in an environment. Later we introduce the latest deep reinforcement methods that have shown promising performances in many difficult tasks.

Chapter 4 introduces Recurrent Learning where the value function is computed recursively as a combination of current value function and the previous value function. This give rise to a new form of traces in reinforcement learning called Recurrent Traces. We show the effectiveness of this setting in a noisy mujoco setting. We also qualitatively describe the importance of the method.

Chapter 5 introduces a generic framework for several different traces in reinforcement learning through filters. We use an IIR filter form to generalize various traces. While doing so, we use attention mechanism to set the filter coefficients. We demonstrate the ability of such traces on a Mountain car environment.

2

Signals and Systems

2.1 Introduction

Signals are a ubiquitous part of our daily lives. In one way or another, our lives are interlinked with signals. For example, communication happens through speech signals [56], images or videos are a form of signals [48], stocks represent a signal that conveys information on how the shares are doing [47], many biological phenomena such as heartbeat of a person recorded through EEG represents a signal [97]. Formally,

Definition 1. *A* ***signal*** *is defined as a function of one or more variables that conveys information on the nature of a physical phenomenon [26].*

A signal is not restricted to a single dimension. It can exist in two or more dimensions too. There is always a system associated with each signal. This system could be associated either with the generation of a signal or with the extraction of information from the signal. For example, in speech signal, the vocal tract represents a generation system whereas our ears represents a processing system. A system is formally defined as

Definition 2. *[26] An entity that manipulates one or more signals to accomplish a function, thereby yielding new signals.*

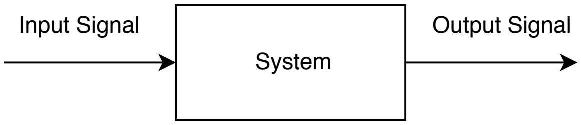


Figure 2.1: Block diagram representation of a system

Figure 2.1 schematically illustrates the interaction between a system and its associated signals. In this work, we restrict the signal-processing operations to digital domain (discrete-time). A discrete-time signal is a sequence of numbers indexed by an integer t. In other words, a discrete-time signal exists only in the countable set. These signals can be viewed as a subset of analog signals. Analog signals exist in a continuous time interval, i.e., $x(t) \in \mathbb{R}$. In practice, such signals can arise from sampling an analog signal. Note that we denote an input signal as $x(t)$ and output signal as $y(t)$ in the discussions to follow unless specified otherwise. The time, t of a signal is often referred to as an independent variable of a signal and the value of the signal, $x(t)$, is referred to as dependent variable.

2.1.1 Basic Operations on Signals

Several operations can be performed on signals. The operations can be performed on both dependent and independent variables. Some of the commonly used operations on scalar signals are as follows:

- Amplitude scaling: Let $x(t)$ denote a discrete-time signal. Then the signal $y(t)$ resulting from the amplitude scaling applied to $x(t)$ is given as $y(t) = cx(t)$, where c is the scaling factor.

- Addition: Let $x_1(t)$ and $x_2(t)$ be two different discrete-time signals. Then the signal $y(t)$ obtained by the addition of $x_1(t)$ and $x_2(t)$ is defined as $y(t) = x_1(t) + x_2(t)$.

- Multiplication: Let $x_1(t)$ and $x_2(t)$ be two different discrete-time signals. Then the signal $y(t)$ obtained by the multiplication of $x_1(t)$ and $x_2(t)$ is defined as

$y(t) = x_1(t)x_2(t)$.

The above three operations are performed on the signal itself, hence these referred to as operations on dependent variable. Now we list the operations performed on the independent variable.

- Time scaling: Let $x(t)$ denote a discrete-time signal, then the signal $y(t)$ obtained by scaling the independent variable t, by a discrete-time a is defined as $y(t) = x(at)$. If $a > 1$, then some values of the signal $y(t)$ are lost as the signal can only be represented at discrete instances of time t.
- Time shifting: Let $x(t)$ denote a discrete-time signal, then the time-shifted version of $x(t)$ is defined as $y(t) = x(t - t_0)$, where t_0 is the time-shift.

2.1.2 Systems Viewed as Interconnections of Operations

A system can be viewed as an interconnection of operations that transforms an input signal into an output signal with properties different from those of the input signal. Let's use the operator H to denote the transformation from input to output that is $y(t) = H\{x(t)\}$. H is often referred to as a transfer function. For example, consider a discrete-time system whose output signal $y(t)$ is the average of the three most recent values of the input signal $x(t)$; that is, $y(t) = \frac{1}{3}(x(t) + x(t-1) + x(t-2))$. The operator H of this system is given by, $H = \frac{1}{3}(1 + S + S^2)$, where S^k denote a system that shifts the input $x(t)$ by k time units to produce an output equal to $x(t-k)$.

2.1.3 Properties of Systems

The properties of a system describe the characteristics of the operator H representing the system.

Stability: A system is said to be *bounded-input bounded-output (BIBO) stable* if and only if every input that is bounded (less than some value) results in an output that is also bounded. The output of such a system does not diverge if the input does not diverge. Mathematically the operator H is BIBO stable if the output signal $y(t)$ satisfies the condition

$$|y(t)| \leq M_y < \infty, \forall t \tag{2.1}$$

whenever the input signal $x(t)$ satisfy the condition

$$|x(t)| \leq M_x < \infty, \forall t. \tag{2.2}$$

where M_x and M_y represent some finite positive numbers, $|.|$ denotes the absolute value of the signal. Note that stability can be measured in various other ways. Here, we restricted our discussion to BIBO stability.

Memory: A system is said to possess memory if its output signal depends on past or future values of the input signal. The temporal extent of past or future values on which the output depends defines how far the memory of the system extends into the past or future. In contrast, a system is said to be memory-less if its output signal at an instant t depends only on the input signal at instant t.

Causality: A system is said to be causal if the present value of the output signal depends only on the present or the past values of the input signal. In other words, the output signal at time t_0 depends only on the input signals at $t \leq t_0$. In contrast, non-causal system depends on one or more future values of the input signal.

Invertibility: A system is said to be invertible if the input of the system can be recovered from the output. Let H^{inv} denote the invertible system, $x(t)$ an input signal, $y(t)$ the output signal obtained after applying the operator H. If the output signal is passed to invertible system H^{inv} which is in cascade to the first operator H then,

$$H^{inv}\{y(t)\} = H^{inv}\{H\{x(t)\}\}, \tag{2.3}$$

where the two operators H and H^{inv} connected in cascade are equivalent to a single operator $H^{inv}H$. For this output signal to equal the input signal $x(t)$, we require that $H^{inv}H = I$, where I denotes the identity operator.

Time Invariance: A system is said to be time invariant if a time delay or time advance of the input signal leads to an identical time shift in the output signal. In other words, time-invariant system responds identically irrespective of when the input is applied. On the other hand, if the characteristics of the system changes with time then they are time-variant.

Linearity: A system is said to be linear in terms of the system input $x(t)$ and the output $y(t)$ if it satisfies the following two properties of superposition and homogeneity.

1. Superposition: The output of the system when a composite input $x(t) = x_1(t) + x_2(t)$ produce an output $y(t)$ which is equal to the sum of outputs of the system when each signal is passed separately, that is, $y(t) = y_1(t) + y_2(t)$.

2. Homogeneity: If the output of the system is scaled by a constant factor a to an input that is scaled by the same factor then the system is said to be homogeneous.

In other words, if $H\{x_1(t)\} = y_1(t)$ and $H\{x_2(t)\} = y_2(t)$ are the responses of the system on the application of two signal $x_1(t)$ and $x_2(t)$ separately, then system H is said to be linear if $H\{ax_1(t) + bx_2(t)\} = ay_1(t) + by_2(t)$. The systems that violate these properties are termed as non-linear.

2.1.4 Recursive Discrete-Time Computation

In recursive computation, the current value of the output signal resulting from the computation depends on two sets of quantities:

- The current and past values of the input signal

- The past values of the output signal itself.

The term *recursive* signifies the dependence of the output signal on its own past values. For example, let $x(t)$ denote the input signal and $y(t)$ denote the output signal of a system, both measured at time t. The relationship between $y(t)$ and $x(t)$ for a system can be written as $y(t) = x(t) + \rho y(t-1)$ using a first order difference equation with a linear constant coefficient. That is, ρ is a constant. Depending on the value assigned to the constant ρ, we have three cases.

1. $\rho = 1$, $y(t) = \sum_{k=0}^{\infty} x(t-k)$ is an **accumulator**, which represents the discrete-time equivalent of an integrator.

2. $|\rho| < 1$, in which case the successive contributions of the past values of the input signal $x(t)$ to the output signal $y(t)$ are continually attenuated in absolute value. Therefore, we call it as a **leaky accumulator**.

3. $|\rho| > 1$, in which case successive contributions of past values of the input signal $x(t)$ to the output signal $y(t)$ are **amplified** in absolute value as time goes on.

Note that only case 2 is a stable BIBO system whereas case 1 and 3 are unstable in the BIBO sense.

2.2 Time-Domain representation of Linear Time-Invariant Systems (LTI)

In this section, we describe both the signal and the system as a function of time, hence we refer to the representation as *time domain representation.* Methods for relating system outputs and inputs in domains other than time are described in the later sections.

LTI systems can be represented in several ways namely convolution sum, difference equation, block diagram, and state-variable description. All four of these time-domain system representations are equivalent. However, each representation relates the input to the output in a different manner. Each representation has advantages and disadvantages with respect to analyzing and implementing systems. We describe the representation using convolution sum and difference equation. Detailed discussions on the remaining representations can be found in Chapter 2 of [26].

2.2.1 The Convolution Sum

Definition 3. *An impulse signal $\delta(t)$ is a function whose value is equal to* 1 *at time* $t = 0$ *and* 0 *elsewhere.*

Definition 4. *Impulse response of the system is the output of the system when presented with an impulse signal, $\delta(t)$.*

Note that, it is possible to represent any discrete-time signal $x(t)$ as a sum of impulse functions $x(t) = \sum_{k=-\infty}^{\infty} x(k)\delta(t-k)$. Here $\delta(t-k)$ is the time-shifted version of an impulse function by k units. Let the operator H denote the system to which such an input $x(t)$ is applied. Then the output of the system is given by

$$\begin{aligned} y(t) &= H\{x(t)\} \\ &= H\left\{ \sum_{k=-\infty}^{\infty} x(k)\delta(t-k) \right\} \end{aligned} \tag{2.4}$$

Eq. (2.4) indicates that the system output is a weighted sum of the response of the system to time-shifted impulses. If we further assume that the system is time invariant, then a time shift in the input results in a time shift in the output. This relationship implies that the output due to a time-shifted impulse is a time-shifted version of the output due to an impulse, that is,

$$H\{\delta(t-k)\} = h(t-k), \tag{2.5}$$

where $h(t) = H\{\delta(t)\}$ is the impulse response of the LTI system H. Therefore, the output of the system can be written as,

$$y(t) = \sum_{k=-\infty}^{\infty} x(k)h(t-k) \tag{2.6}$$

Eq. (2.6) is termed the *convolution sum* and is denoted by the symbol $*$; that is,

$$x(t) * h(t) = \sum_{k=-\infty}^{\infty} x(k)h(t-k). \tag{2.7}$$

Note that, the convolution sum obeys distributive, associative and commutative properties. We determine $y(t)$ for all t without evaluating Eq. (2.6) at an infinite number of distinct shifts t. This is accomplished by identifying intervals of t on which $x(k)h(t-k)$ exists.

If there are two systems H_1 and H_2 that are cascaded to produce an output, then the output $y(t) = z(t) * h_2(t)$, where $z(t) = x(t) * h_1(t)$, $x(t)$ is the input signal, $h_1(t)$ and $h_2(t)$ is the impulse response of the two systems.

We now establish a relation between the LTI system properties and the impulse response. The impulse response completely characterizes the input-output behavior of an LTI system.

- Memoryless LTI system: The output of a memoryless LTI system depends only on the current input. From convolution sum (2.6) we have $y(t) = \sum_{k=-\infty}^{\infty} x(k)h(t-k)$. This system is memoryless if $y(t)$ depends only on $x(t)$. Therefore, the system's response $h(k) = 0\ \forall k \neq 0$, thus, $h(t) = c\delta(t)$, where c is an arbitrary constant.

- Causal LTI system: The output of a causal LTI system depends only on past or present values of the input. From convolution sum (2.6) the response of the system for the future values of the input is 0. In other words, $h(k) = 0\ \forall k < 0$.

- Stable LTI system: For a system to be BIBO stable, the output $y(t)$ must be bounded for all the bounded input $x(t)$. Hence for stable systems Eq. (2.6) implies,

$$\begin{aligned} |y(t)| &= |h(t) * x(t)| \\ &= \left| \sum_{k=-\infty}^{\infty} x(t-k)h(k) \right| \\ &\le \sum_{k=-\infty}^{\infty} |x(t-k)h(k)| \\ &\le \sum_{k=-\infty}^{\infty} |x(t-k)||h(k)| \\ &\le M_x \sum_{k=-\infty}^{\infty} |h(k)|. \end{aligned} \tag{2.8}$$

Hence the output is bounded if the response of the system satisfies the bound

$$\sum_{k=-\infty}^{\infty} |h(k)| \le \infty \tag{2.9}$$

2.2.2 Difference Equation Representation of LTI system

Linear constant-coefficient difference equations provide another representation for the input-output characteristics of LTI system. The general form of a linear constant-coefficient difference equation representing discrete-time signal is

$$\sum_{k=0}^{N} a_k y(t-k) = \sum_{k'=0}^{M} b'_k x(t-k') \tag{2.10}$$

The order of the difference equation, (N, M), represents the number of storage devices in the system. The output of the system at any point in time t can be obtained by rearranging the equation recursively

$$y(t) = \frac{1}{a_0} \sum_{k'=0}^{M} b'_k x(t-k') - \frac{1}{a_0} \sum_{k=1}^{N} a_k y(t-k). \tag{2.11}$$

Note that, the output of the system at time t depends on the past outputs. These are known as *initial conditions.* The initial conditions summarize the information about the system's past that is needed to determine the future outputs. Detailed discussions on the applications of difference equations and the effects of numerical round-off in signal processing can be found in [58, 49, 62].

2.3 Fourier Representation of Signals and LTI Systems

In this section we represent a signal as a superposition of complex sinusoids [20]. If such a signal is applied to an LTI system, then the system output is a weighted superposition of the system response to each complex sinusoids. This kind of a representation helps us understand the weight associated with a sinusoid of a given frequency in the overall contribution of a signal. Such an analysis is extensively studied in several works [32, 11, 51, 26]. The response of the system if an input $x(t)$ is a complex sinusoidal signal, $e^{j\Omega t}$, is known as *frequency response* where Ω is the frequency. Mathematically, it is given by

$$H(e^{j\Omega}) = \sum_{k=-\infty}^{\infty} h(k)e^{-j\Omega k}. \tag{2.12}$$

The output $y(t)$ of such a system is given by

$$y(t) = H(e^{j\Omega})e^{j\Omega t}. \tag{2.13}$$

Since the output in Eq. (2.13) is the product of system's response with a complex exponential signal $e^{j\Omega t}$, the frequency of the output sinusoid is same as that of an input but a complex scaling factor $H(e^{j\Omega})$ is multiplied.

2.3.1 Discrete-Time Fourier Series (DTFS)

Discrete-Time Fourier Series is used to represent a discrete-time periodic signal. Consider a discrete-time periodic signal $x(t)$, it can be decomposed into a sum of complex sinusoids as

$$x(t) = \sum_{k=0}^{T-1} X(k)e^{jk\Omega_0 t}. \tag{2.14}$$

where T is the fundamental period, $\Omega_0 = 2\pi/T$ is the fundamental frequency,

$$X(k) = \frac{1}{N}\sum_{t=0}^{T-1} x(t)e^{-jk\Omega_0 t} \tag{2.15}$$

are the DTFS coefficients of the signal $x(t)$. Here $x(t)$ and $X(k)$ form a DTFS pair. The coefficients $X(k)$ provides us a *frequency domain* representation of $x(t)$ as each coefficient is associated with a complex sinusoid of a different frequency. The variable k determines the frequency of the sinusoid associated with $X(k)$, so we say that $X(k)$ is a function of frequency. The magnitude of $X(k)$, denoted by $|X(k)|$ and plotted against the frequency index k, is known as the *magnitude spectrum* of $x(t)$. Similarly, the phase of $X(k)$, termed $arg\{X(k)\}$ plotted against k, is known as the *phase spectrum* of $x(t)$.

2.3.2 Discrete-Time Fourier Transform (DTFT)

The DTFT is used to represent a discrete-time non-periodic signal as a superposition of complex sinusoids. The DTFT involves a continuum of frequencies on the interval $-\pi < \Omega \leq \pi$, where Ω has units of radians. Therefore, the DTFT representation of time-domain signal involves an integral over frequency,

$$x(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\Omega}) e^{j\Omega t} \, d\Omega \tag{2.16}$$

where

$$X(e^{j\Omega}) = \sum_{t=-\infty}^{\infty} x(t) e^{-j\Omega t} \tag{2.17}$$

is the DTFT of the signal $x(t)$. The transform $X(e^{j\Omega})$ describes the signal $x(t)$ as a function of a sinusoidal frequency Ω and is termed the *frequency domain* representation of $x(t)$. The infinite sum in Eq. (2.17) converges if $x(t)$ has a finite duration and is finite valued. For other signals, the convergence can be shown in absolute terms or through a mean-square error sense. More discussions on this can be found in Chapter 3 of [26].

Note that, both the Fourier representations satisfies the linearity property. Also, convolution in time domain is equivalent to the multiplication in frequency domain. In other words, if the output $y(t)$ is obtained as a convolution between the input $x(t)$ and the system's response $h(t)$, then the output in frequency domain $Y(e^{j\Omega})$ is obtained by multiplying the frequency domain representations of $x(t)$ and $h(t)$ as, $Y(e^{j\Omega}) = X(e^{j\Omega})H(e^{j\Omega})$.

The multiplication that occurs in frequency-domain representation gives rise to the notion of *filtering*. The term filtering implies certain frequency components are eliminated while others are passed by the system. Depending upon the kind of frequencies attenuated, we have several classes of filters namely,

- low-pass filter: attenuates high-frequency components,
- high-pass filter: attenuates low-frequency components,
- band-pass filter: passes signals within a certain band of frequencies.

The band of frequencies that are passed by the system is known as *passband* and the band of frequencies that are attenuated are known as *stopband* frequencies. The range of frequencies over which the transition between passband and stopband occur is known as *transition band*. The characterization of the discrete-time filter is based on its behavior in the frequency range $-\pi < \Omega \leq \pi$ because its frequency response is 2π periodic. Therefore a high-pass discrete-time filter passes frequencies near π and attenuates frequencies near 0. The magnitude response of a filter is commonly described in units of decibels (dB) defined as

$$20 \log |H(e^{j\Omega})| \tag{2.18}$$

The magnitude response in the stopband is much smaller than that in the passband. The frequency corresponding to a magnitude response of $\frac{1}{\sqrt{2}}$ or -3dB is termed as *cutoff frequency* of the filter.

2.4 z-Transform: Complex Exponential Representation of Signals

In this section, a generalization of the complex sinusoidal representation of a discrete-time signal to a representation in terms of complex exponential signals is discussed. This generalization often helps us analyze a system easily. Moreover, unlike Fourier transforms z-transform exists for unstable systems making z-transform a useful representation. Such a representation is well studied in various works [92, 58, 50, 26].

Let us consider the output of a system when a complex exponential signal $z(t) = r^t \cos(\Omega t) + jr^t \sin(\Omega t)$ is passed. The output of a system is given by

$$y(t) = z^t \sum_{k=-\infty}^{\infty} h(k) z^{-k}. \tag{2.19}$$

We can now define a *transfer function* $H(z)$ as below,

$$H(z) = \sum_{k=-\infty}^{\infty} h(k) z^{-k}. \tag{2.20}$$

The z-Transform of an arbitrary signal $x(t)$ is given by

$$X(z) = \sum_{t=-\infty}^{\infty} x(t) z^{-t} \tag{2.21}$$

and the inverse z-transform is given by

$$x(t) = \frac{1}{2\pi j} \oint X(z) z^{t-1}\, dz. \tag{2.22}$$

where $dz = jre^{j\Omega} d\Omega$, the limits over integral is a circle of radius $|z| = r$ as Ω goes from $-\pi$ to π. Further details on inverse z-transform can be found in several works [12, 49, 50].

2.4.1 The z-Plane, Poles, and Zeros

It is convenient to represent the complex number z as a location in a complex plane termed as *z-plane*. z-plane is depicted in the Figure 2.2 where a point $z = re^{j\Omega}$ is

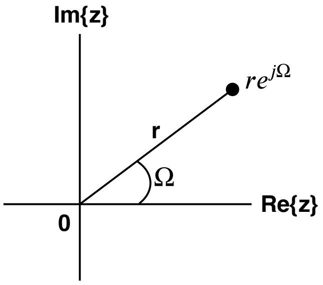


Figure 2.2: Depiction of z-plane

located at a distance r from origin at an angle Ω from the positive real axis. We define a *unit circle* as a circle in the z-plane when $r = 1$. This is interesting as this circle corresponds to the DTFT of an input $x(t)$ when it is absolutely summable. The most commonly encountered form of the z-transform is a ratio of two polynomials in z^{-1} given by

$$X(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots a_N z^{-N}} \tag{2.23}$$

We could factorize the above equation as a product of terms involving roots of the numerator and denominator as below,

$$X(z) = \frac{\tilde{b} \prod_{k=1}^{M} (1 - c_k z^{-1})}{\prod_{k'=1}^{N} (1 - d'_k z^{-1})} \tag{2.24}$$

where $\tilde{b} = b_0/a_0$. The c_k are the roots of the numerator polynomial and are termed the *zeros* of $X(z)$. The d'_k are the roots of the denominator polynomial and are termed the *poles* of $X(z)$. The locations of zeros are denoted with the "o" and the locations of the poles are denoted with "x" in the z-plane. We define the *Region of Convergence (ROC) of* $x(t)$ as the set of points in the z-plane for which the z-Transform summation converges. Mathematically,

$$ROC = \left\{ z : \left| \sum_{t=-\infty}^{\infty} x(t) z^{-t} \right| < \infty \right\}. \tag{2.25}$$

The ROC of stable systems include the unit circle in the z-plane. Hence, knowing ROC alone is enough to find whether a system is stable. In other words, a pole that is inside a unit circle in the z-plane ($|d'_k| < 1$) contributes an exponentially decaying term to the impulse response, while a pole that is outside the unit circle ($|d'_k| > 1$)

contributes an exponentially increasing term. If a system is BIBO stable, then the impulse response is absolutely summable and the DTFT of the impulse response exists. It follows that the ROC must include the unit circle in the z-plane.

2.4.2 Properties of z-Transform

1. Linearity: If $x_1(t) \leftrightarrow X_1(z)$ and $x_2(t) \leftrightarrow X_2(z)$ then $x(t) = \alpha x_1(t) + \beta x_2(t) \leftrightarrow \alpha X_1(z) + \beta X_2(z)$
2. Time shifting: If $x(t) \leftrightarrow X(z)$ then $x(t-k) \leftrightarrow z^{-k}X(z)$
3. Scaling: If $x(t) \leftrightarrow X(z)$ then $a^t x(t) \leftrightarrow X(a^{-1}z)$
4. Time reversal: If $x(t) \leftrightarrow X(z)$ then $x(-n) \leftrightarrow X(z^{-1})$
5. Differentiation: If $x(t) \leftrightarrow X(z)$ then $tx(t) \leftrightarrow -z\frac{d}{dz}X(z)$
6. Convolution: If $x_1(t) \leftrightarrow X_1(z)$ and $x_2(t) \leftrightarrow X_2(z)$ then $x(t) = x_1(t) * x_2(t) \leftrightarrow X(z) = X_1(z)\ X_2(z)$

2.5 Filters

In Section 2.3 we briefly discussed *filters* as a system that separates the unwanted frequencies from the signal by exploiting the frequency domain representation. Filters are widely popular in signal processing literature [61, 17] and therefore they are well studied in several works [4, 42, 49, 52, 60]. However [23, 38] were the first ones to describe digital filters. In this section, we discuss about design of digital filters, a class of filters for discrete-time signals based on the Chapter 8 of [26]. A system inputs a signal and outputs another signal with certain modifications to the original signal. The system is said to be *distortionless* if the output signal of the system is an exact replica of the input signal, except, possibly, for two minor modifications namely,

- A scaling in amplitude: In time domain, output signal $y(t) = Cx(t)$, where $C \neq 0$ is a constant that accounts for a change in amplitude. Similarly in frequency domain, the magnitude response $|H(e^{j\Omega})| = C$ for all frequencies of interest.

- A constant time delay: In time domain, output signal $y(t) = x(t - t_0)$, where t_0 is a constant time delay. Similarly, in frequency domain, the phase response $arg\{H(\Omega)\}$ is linear in frequency; that is, $arg\{H(\Omega)\} = -\Omega t_0$.

2.5.1 Low-Pass Filters

The ideal low-pass filter is one which transmits all the low frequencies inside the passband without any distortion and rejects all the high frequencies inside the stopband. The frequency response of an ideal low-pass filter with cutoff frequency Ω_c is given by,

$$H(j\Omega) = \begin{cases} e^{-j\Omega t_0}, & \text{if } |\Omega| \leq \Omega_c \\ 0, & \text{otherwise} \end{cases} \tag{2.26}$$

To design an ideal low-pass filter proceeds by the approximation of a prescribed frequency response by a rational transfer function which represents a system that is both causal and stable. After which the realization of the approximating transfer function is carried out. The approximation part is an optimization problem that can be solved in the context of specific criterion of optimality. The design of filters is not discussed here as it is out of scope for this work.

2.5.2 Digital Filters

Based on the duration of the impulse response, there are two classes of filters namely,

- Finite-duration impulse response (FIR) digital filters: These filters are governed by linear constant-coefficient difference equations of a non-recursive nature. As a result of the formulation, they have finite memory and are always BIBO stable.

Moreover, they can realize a desired magnitude response with an exactly linear phase response.

- Infinite-duration impulse response (IIR) digital filters: These filters are governed by a linear constant-coefficient difference equations of a recursive nature. As a result of this formulation, they have shorter filter length than the use of corresponding FIR digital filter. However, they have some phase distortion and tend to be BIBO unstable if not carefully designed.

3

Reinforcement Learning

Learning can broadly be divided into three kinds Supervised Learning, Unsupervised Learning, and Reinforcement Learning. In supervised learning, a dataset is provided apriori in the form of (x, y) pairs where x is the *input* and y is the corresponding *target*. The job of the learner is to find the mapping $h : x \rightarrow y$. Regression and classification problems fall under this setup [10]. On the other hand, unsupervised learning deals with finding the patterns in the dataset. The learner is provided only with the input data x and unlike supervised learning no information about the target is provided. The learner has to figure out the patterns with just the input data. Clustering [39], Principal Component Analysis (PCA) [54], etc are some of the learning techniques that fall under this category. By definition,

Definition 5. *[81] Reinforcement Learning is learning how to map situations to actions so as to maximize a numerical reward signal.*

The learner (agent) discovers the actions that maximize a numerical reward signal through *trial and error* search. In doing so, the agent generates its own data by interacting with the *environment*. This trial and error learning is inspired by psychology [81]. Also, the decisions that the agent makes not only has an impact immediately but also on all subsequent rewards making it challenging. Reinforcement Learning was originally inspired by the psychology of animal learning [35], however the framework is built on Markov Decision Process [9].

3.1 Markov Decision Process

A Markov Decision Process as defined by Puterman [59] consists of a tuple of 5 elements $< S, A, \mathcal{P}, r, \gamma >$.

- S is a countable non-empty set of states.
- A is a countable non-empty set of possible actions.
- $\mathcal{P}$ is the state transition probability which maps state-action pair $(s, a) \in S \times A$ to a probability measure over states i.e., $S \times \mathbb{R}$.
- r is the reward function that is obtained as a result of choosing an action $a \in A$ in a state $s \in S$.
- $\gamma \in [0, 1)$ is the discount factor.

At each point in time t, the decision maker, *agent*, interacts with the *environment* by taking an action $a_t \in A$, in a sequential manner. As a result of taking an action a_t in $s_t \in S$, the agent receives a reward r_{t+1} and makes a transition to a new state $s_{t+1} \in S$ as dictated by the the transition kernel $\mathcal{P}(s_{t+1}, r_{t+1}|s_t, a_t)$. This interaction is described in the Figure 3.1[1]. There are two settings that arise in reinforcement learning, policy evaluation and control. In summary, the agent is provided with the actions that it needs to take in policy evaluation whereas the agent discovers actions that it needs to take in control. Each aspect is discussed separately in the sections to follow.

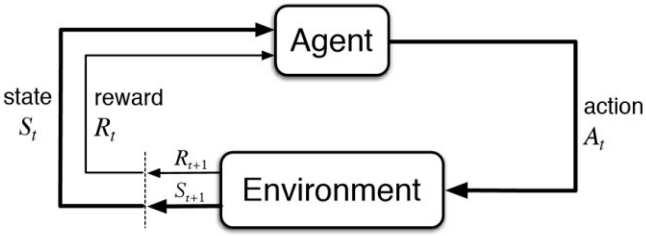


Figure 3.1: Interaction of agent-environment in MDP

[1] the picture is reproduced from [81]

3.2 Policy Evaluation

In this setting, the agent is provided with a *policy* π from which the actions a_t are sampled from. The goal of the agent is to estimate the *expected discounted cumulative rewards* for each state in the environment while following the given policy π. Formally, a *policy* π is a distribution over actions given a state, $\pi(a_t|s_t)$. We define a *value function* of a state as the *expected returns* following a policy π from that state

$$v_\pi(s) = \mathop{\mathbb{E}}_{\pi}[G_t|s] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{i=t}^{\infty}\gamma^{i-t}r_{i+1}\middle|s\right], \tag{3.1}$$

where the *return* G_t is the sum of discounted rewards, $G_t = \sum_{i=t}^{\infty}\gamma^{i-t}r_{i+1}$. The value function represents the *goodness* of a particular state. Intuitively, the higher the value of a state, the higher the expected return from that state, hence making it a good state.

3.2.1 Bellman Equation and Bellman Operator

We can express the Eq. (3.1) by unrolling the infinite sum of rewards.

$$\begin{aligned} v_\pi(s_0) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\middle|s = s_0\right] \\ &= \mathbb{E}\left[r(s_0, a_0) + \sum_{t=1}^{\infty}\gamma^t r(s_t, a_t)\middle|s = s_0\right] \\ &= \mathbb{E}\left[r(s_0, a_0) + \sum_{t=0}^{\infty}\gamma^{t+1} r(s_{t+1}, a_{t+1})\middle|s = s_0\right] \\ &= \mathbb{E}\left[r(s_0, a_0) + \gamma\sum_{t=0}^{\infty}\gamma^{t} r(s_{t+1}, a_{t+1})\middle|s = s_0\right] \\ &= \mathbb{E}\left[r(s_0, a_0) + \gamma\,\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^{t} r(s_{t+1}, a_{t+1})\middle|s = s_1\right]\middle|s = s_0\right] \\ &= \mathbb{E}[r(s_0, a_0) + \gamma v_\pi(s_1)|s = s_0] \end{aligned} \tag{3.2}$$

We can then expand the expectation by marginalizing over policy and obtain

$$v_\pi(s) = \sum_{a\in A}\pi(a|s)\left(r(s,a) + \gamma\sum_{s'}\mathcal{P}(s'|s,a)v_\pi(s')\right) \tag{3.3}$$

Eq. (3.3) is termed as *Bellman equation* for v_π. The equation expresses value of a state recursively in terms of value of its successor state. The Bellman equation can also be written in a matrix form as,

$$v_\pi = r_\pi + \gamma \mathcal{P}_\pi v_\pi, \tag{3.4}$$

where $r_\pi(s) = \sum_{a \in A} \pi(a|s) r(s,a)$, $\mathcal{P}_\pi(s,s') = \sum_{a \in A} \pi(a|s) \mathcal{P}(s'|s,a)$, $v_\pi \in \mathbb{R}^{|S|}$ the value function vector. We could simplify further and find the solution for v_π as,

$$\begin{aligned} v_\pi &= r_\pi + \gamma \mathcal{P}_\pi v_\pi \\ v_\pi - \gamma \mathcal{P}_\pi v_\pi &= r_\pi \\ (I - \gamma \mathcal{P}_\pi) v_\pi &= r_\pi \\ v_\pi &= (I - \gamma \mathcal{P}_\pi)^{-1} r_\pi \end{aligned} \tag{3.5}$$

However, inverting a matrix is expensive operation. Therefore, we estimate v_π through iterative solutions. One such solution is based on operator-theoretic point of view. We define an operator $\mathcal{T}^\pi : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ whose effect of any $v \in \mathbb{R}^{|S|}$ is the following:

$$\mathcal{T}^\pi v = r_\pi + \gamma \mathcal{P}_\pi v \tag{3.6}$$

This operator is termed as *Bellman* operator. We now show that this operator has a unique fixed point.

Theorem 1. *(Banach's fixed-point theorem) Let V be a Banach space and $\mathcal{T} : V \to V$ be a contraction mapping. Then $\mathcal{T}$ has a unique fixed point.*

Proof. Proof of this theorem is not outlined in this thesis. Details of the proof can be found in [7, 85, 5]. □

Using Theorem 1 we show that the sequence of value function by the application of Bellman operator $\mathcal{T}^\pi$ defined in Eq. (3.6) converges to a fixed point.

Lemma 1. *Bellman operator $\mathcal{T}^\pi$ defined in Eq.* (3.6) *is a contraction mapping.*

Proof. Let $v \in V$, $u \in V$ be two vectors, let $\|.\|$ denote ∞ norm of a vector.

$$\begin{aligned}
\|\mathcal{T}^\pi v - \mathcal{T}^\pi u\| &= \|r_\pi + \gamma \mathcal{P}_\pi v - (r_\pi + \gamma \mathcal{P}_\pi u)\| \\
&= \|r_\pi + \gamma \mathcal{P}_\pi v - r_\pi - \gamma \mathcal{P}_\pi u\| \\
&= \|\gamma(\mathcal{P}_\pi v - \mathcal{P}_\pi u)\| \\
&\leq \gamma \, \|\mathcal{P}_\pi\| \, \|v - u\| \\
&\leq \gamma \, \|v - u\| \qquad (3.7)
\end{aligned}$$

Hence $\mathcal{T}^\pi$ is a contraction if the discount factor $\gamma \in [0, 1)$. □

The algorithm for estimating fixed point v_π using the bellman operator is provided in the Algorithm 1.

Algorithm 1 Iterative Policy Evaluation

```
Input: $\pi$,$\gamma$
Initialize: $\mathbf{v}_\pi = 0, \mathbf{v}_\pi \in \mathbb{R}^{|S|}$
Output: $\mathbf{v}_\pi$                    ▷ Return the fixed point of the value function
Compute $\mathbf{r}_\pi$, where $\mathbf{r}_\pi(s) = \sum_{a \in A} \pi(a|s) r(s, a)$
Compute $\mathcal{P}_\pi$, where $\mathcal{P}_\pi(s, s') = \sum_{a \in A} \pi(a|s) \mathcal{P}(s'|s, a)$
repeat
    $\mathbf{v}_\pi \leftarrow \mathbf{r}_\pi + \gamma \mathcal{P}_\pi \mathbf{v}_\pi$
until convergence
```

The term $\mathbf{v}_\pi \leftarrow \mathbf{r}_\pi + \gamma \mathcal{P}_\pi \mathbf{v}_\pi$ is known as Bellman update. In practice we may not have access to transition kernel, $P(s'|s, a)$, and the rewards $r(s, a)$ could be noisy. As a result we cannot compute the quantities r_π and $\mathcal{P}_\pi$ beforehand. Hence we modify the algorithm 1 to have stochastic update as below,

$$v(s_t) \leftarrow r(s_{t+1}, a_{t+1}) + \gamma v(s_{t+1}) \qquad (3.8)$$

However, in expectation they are the same.

3.2.2 Temporal Difference Learning

Each Bellman stochastic update, Eq. (3.8), modifies the value function completely, forgetting the previous update. In doing so, previous learning is lost. Temporal difference methods [80, 78] address this issue by modifying the update equation as below

$$v(s_t) \leftarrow (1-\alpha)v(s_t) + \alpha(r(s_{t+1}, a_{t+1}) + \gamma v(s_{t+1})) \tag{3.9}$$

where $\alpha \in (0,1)$ is termed learning rate, $v(s_t)$ is the value function of state s_t, $r(s_{t+1}, a_{t+1})$ is the reward obtained for making a transition from state s_t to state s_{t+1} by taking an action $a_t \in \pi(.|s_t)$, $v(s_{t+1})$ is the value function of state s_{t+1}. As seen from Eq. (3.9) every update in temporal difference methods smooths the value function with respect to its previous estimates by taking a step in the direction rather than completely updating it. On the other hand, Bellman stochastic update Eq. (3.8) completely forgets the past estimate of the value function. The update rule given in Eq. (3.8) and Eq. (3.9) is referred to as *bootstrapping. Bootstrapping* is the idea that the value function of a state is updated based on the estimates of the value function of the successor states. We can rewrite Eq. (3.9) as,

$$v(s_t) \leftarrow v(s_t) + \alpha(r(s_{t+1}, a_{t+1}) + \gamma v(s_{t+1}) - v(s_t)) \tag{3.10}$$

where $r(s_{t+1}, a_{t+1}) + \gamma v(s_{t+1}) - v(s_t)$ is termed as *temporal difference (TD) error,* δ. Temporal difference error can further be split into two parts, first the *target*, $r(s_{t+1}, a_{t+1}) + \gamma v(s_{t+1})$, the part of the TD-error where updating is done from and second the estimate, $v(s_t)$, the one that gets updated. The target described in Eq. (3.10) is a *1-step* target. This is because we are bootstrapping from a state which is one time-step away. In fact, we have a lot of options to choose a target from. These are discussed in the following section.

3.2.2.1 n-step Targets

We saw 1-step target in the previous section when we introduced temporal difference updates. We briefly mentioned that we could have several kinds of targets in temporal difference methods. In this section we discuss other targets that are often used in reinforcement learning. Similar to a 1-step target, we could have a 2-step target which bootstraps from a state that is two time-steps away, 3-step target, or in general, a multi-step target [96]. Multi-step target accumulates the discounted rewards at each time-step and bootstraps from the state that is several time-steps in the future. Formally a general multi-step target, $n - step$ target, is defined as

$$G_{t:t+n} = \sum_{i=t}^{t+n-1} \gamma^{i-t} r_{i+1} + \gamma^n v(s_{t+n}), \tag{3.11}$$

where a simplified notation for rewards, $r_t = r(s_t, a_t)$, is used and $G_{t:t+n}$ is the n-step return. The temporal difference methods that updates a value function with n-step targets are called as TD(n) methods. The update equation for such methods is given by,

$$v(s_t) \leftarrow v(s_t) + \alpha(G_{t:t+n} - v(s_t)) \tag{3.12}$$

Algorithm for policy evaluation for TD(n) methods is given in 2.

Algorithm 2 Policy Evaluation using n-step TD

```
Input: $\pi$, $\gamma$, $\alpha$, $n$            ▷ Requires a policy, discount, learning rate, and horizon
Initialize: $v(s) = 0,\ \forall s \in S$       ▷ Initialize value function of all the states to 0
Output: $v_\pi(s),\ \forall s \in S$
repeat
    for all $s \in S$ do                         ▷ Update loop for all states
        $r = 0$
        for $i \leftarrow 1, n$ do               ▷ Loop to calculate n-step target
            Take $a$ according to $\pi$
            Observe $s_{i+1}$, $r_{i+1}$
            $r \leftarrow r + \gamma^{i-1} r_{i+1}$
        end for
        $\delta = r + \gamma^n v(s_n) - v(s)$    ▷ Compute n-step TD error
        $v(s) \leftarrow v(s) + \alpha\delta$    ▷ Update the value of a state
    end for
until Convergence
```

The convergence of these methods are discussed in [**dayan1994TD**, 15, 78] and [91] using the theory of stochastic approximation. For an *episodic environment*[2] the true return is achieved by increasing n sufficiently large. This return is called *Monte*

[2] a subset of environments where the episode is terminated when the agent reaches a goal state or after performing certain number of steps or for other reasons

Carlo return [80, 96]. The rewards that the learner receives could be stochastic in nature. As a result, increasing the horizon of the target results in an increase in variance of the returns, resulting in poor updates. On the other hand, decreasing the horizon on the target results in a bias due to the bootstrapping [83]. In practice an intermediate n target performs the best [80, 81].

3.2.2.2 Lambda Target

As discussed earlier, changing the horizon of the target can have a better performance due to bias-variance trade-off. But such methods are limited to using a particular n-step target. Watkins introduced TD(λ) methods [96] which take a geometric combination of various n-step targets. TD(λ) methods have a better bias-variance trade-off therefore yielding a better performance [80, 78, 70]. Mathematically a λ return is defined as

$$G_t^\lambda = (1-\lambda)\sum_{n=0}^{\infty}\lambda^{n-1}\left(\sum_{k=t}^{t+n-1}\gamma^{k-t}r(s_k,a_k)+\gamma^n v(s_{t+n})\right) \tag{3.13}$$

where $\lambda \in [0,1]$. When $\lambda = 0$ the target corresponds to 1-step target and when $\lambda = 1$ the target corresponds to Monte Carlo return. The TD update rule corresponding to λ target is given by

$$v(s_t) \leftarrow v(s_t) + \alpha(G_t^\lambda - v(s_t)) \tag{3.14}$$

Though TD(λ) methods are advantageous, they are limited by two problems in practice

- Computational inefficiency due to delay in updating.
- Finding the optimal λ.

Eligibility traces, is an online, TD(λ) equivalent algorithm. This method was first introduced by Sutton [79, 78]. However, the equivalence with TD(λ) methods were found much later. They exist in several different forms such as accumulating traces [79], replacing traces [75], Dutch traces [70] and others [57, 82, 40]. Accumulating version of traces is described in 3. Other versions differ either in the way they compute the eligibility trace or the setting they are employed for.

Algorithm 3 Eligibility traces, online TD(λ)

```
Input: $\pi$, $\gamma$, $\alpha$, $\lambda$
Initialize: $\mathbf{v} = 0, \mathbf{v} \in \mathbb{R}^{|S|}$
Output: $\mathbf{v}$                                  ▷ Return the fixed point of the value function
repeat
    Initialize: $\mathbf{e}_0 = 0, \mathbf{e}_0 \in \mathbb{R}^{|S|}$            ▷ Initialize eligibility trace to $\mathbf{0}$
    for $i \leftarrow 1, T$ do        ▷ Loop until the episode terminates or for a fixed steps $T$
        $\mathbf{e}_i \leftarrow \gamma\lambda\mathbf{e}_{i-1} + \mathbb{1}_{s=s_i}$                ▷ Update trace for state $s_i$
        Take $a$ according to $\pi$
        Observe $s_{i+1}$, $r_{i+1}$
        $\mathbf{v} \leftarrow \mathbf{v} + \alpha(r_{i+1} + \gamma v(s_{i+1}) - v(s_i))\mathbf{e}_i$     ▷ Update value function
    end for
until convergence
```

Online view of performing updates is often known as *backward view* in reinforcement learning literature. This is because at each point in time t, the TD error is being assigned to all the states visited in the past. This phenomenon, the mechanism of assigning error (credit) to the states visited in the past is called *Temporal Credit Assignment*. Eligibility traces assigns a credit in an exponentially decaying fashion by a factor of λ to the states in the past. The backward view in the form of eligibility traces opens up an opportunity to explore credit assignment in other forms. Chapter 4 introduces a new form of eligibility traces that is aimed at reducing a variance in the value function and Chapter 5 generalizes various forms of eligibility traces under a unifying framework - *Filters*.

As discussed earlier, setting an optimal λ is still an open question. Several works tried to answer this question There are a few properties that we look for when we are looking to choose an optimal λ. These properties are discussed in detail in several works but are limited to specific cases [34, 65, 19, 83, 74, 98]. Most recently meta-gradient approach is explored in this direction but the methodology is limited to offline view [103].

3.2.2.3 Temporally Regularized Targets

Recently, temporally regularized target which bootstraps from the previous state along with the successor state was introduced [87]. In its most general form, temporally regularized target is defined as

$$G_t^{reg} = (1 - \beta)G_t + \beta G_{past} \tag{3.15}$$

where G_t^{reg} is the temporally regularized target defined as a convex combination of the previous value function summarized in G_{past} and any of the returns (n-step, λ return), G_t, that we discussed earlier. This convex combination is governed by the parameter β which weights these targets.

3.2.3 Beyond Tabular Setting

The algorithms described so far scale linearly with respect to the number of states in an MDP. This is because the value function of each state is represented uniquely as a component in the value function vector. This is however unfeasible as the state space in many of the real world problems are exponentially huge. Also, we may have access only to the features describing the states rather than the state itself. In such setting, we use a function approximator parameterized by θ to approximate the value function. The aim in this setting is to find an optimal parameters θ^* that minimizes the following error,

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\pi, s \in S}[(V^\pi - V_\theta)^2] \tag{3.16}$$

where V^π is the optimal value function, V_θ is the value function estimated by the function approximator. However, we may not have access to V^π and therefore we use any of the targets described previously as below,

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\pi, s \in S}[(G_t - V_\theta)^2]. \tag{3.17}$$

Minimizing this update equation yields the following update rule,

$$\theta_{t+1} \leftarrow \theta_t + \alpha(G_t - V_{\theta_t}(s_t))\nabla_{\theta_t} V_{\theta_t}(s_t) \tag{3.18}$$

where α is the learning rate, $\nabla_{\theta_t} V_{\theta_t}(s_t)$ is the gradient with respect to the estimate $V_{\theta_t}(s_t)$. Note that the gradient in the update equation (3.18) is only with respect to the estimate, hence it is termed as semi-gradient [81].
Tabular setting that we discussed earlier is a special case of function approximation where each state is represented as a one hot vector. Also, in the case of linear function approximation, the gradient $\nabla_{\theta_t} V_{\theta_t}(s_t)$ reduces to a feature vector, ϕ, as the value function is approximated by the function $\theta^T \phi$.

3.3 Control

So far we have seen methods to estimate a value function when a policy π is given to us. Many of the tasks in real world concerns with finding a policy that leads to the highest expected discounted rewards. The problem of finding such policies is termed as *Control* in reinforcement learning and the policy that achieves the highest expected discounted rewards is called an *optimal policy.* In this section we will discuss some of the methods to find an optimal policy.

3.3.1 Bellman Optimality Equation

Similar to an operator in the policy evaluation case, we define an operator for an optimal value function, v^*, for an optimal policy, π^*, as below

$$T^* v^* = \max_{\pi} [r_\pi + \gamma \mathcal{P}_\pi v^*]. \tag{3.19}$$

In practice, we can find an optimal value function, v^*, and an optimal policy, π^*, using either by *value iteration* [9] where the value function is updated for all the states before updating the policy greedily or *policy iteration* [28] where a policy is evaluated completely before updating it greedily with respect to the value function. However, these algorithms are limited to the cases where a transition kernel is available to us.

Also, such algorithm doesn't scale up with the number of states and actions. To circumvent this problem action-value function(Q-value function) is introduced.

3.3.2 Q value function

Q-value function or an action-value function [80] is defined as the expected discounted returns starting at a state, picking an action and following a policy later on. Mathematically an action-value function is given by

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\middle| s_0 = s, a_0 = a\right]. \tag{3.20}$$

where $q_\pi(s,a)$ is an action-value function for a state s, an action a. Similar to Section 3.2 we can unroll Eq. 3.20.

$$\begin{aligned}
q_\pi(s,a) &= \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a)\middle| s_0 = s, a_0 = a\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[r(s,a) + \sum_{t=1}^{\infty}\gamma^t r(s_t,a_t)\middle| s_0 = s, a_0 = a\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[r(s,a) + \sum_{t=0}^{\infty}\gamma^{t+1} r(s_{t+1},a_{t+1})\middle| s_0 = s, a_0 = a\right] \\
&= \mathbb{E}[r(s,a) + \gamma v_\pi(s_1)|s_0 = s, a_0 = a] \\
&= \mathop{\mathbb{E}}_{s}[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')|s_0 = s, a_0 = a].
\end{aligned} \tag{3.21}$$

In this regard there are two algorithms to find action-values, Q-learning and SARSA learning. In Q-learning we take the highest action-value of the next state to bootstrap from where as in SARSA learning we bootstrap from an action-value that is sampled according to a greedy policy. The complete algorithms of Q-learning and SARSA are given in 4 and 5.

Algorithm 4 Q learning

```
Input: $\pi$, $\gamma$, $\alpha$
Initialize: $q(s, a) = 0 \ \forall s \in S, \ \forall a \in A$
Output: $q*$                                   ▷ Return the optimal action-value function
repeat
    for $i \leftarrow 1, T$ do         ▷ Loop until the episode terminates or for a fixed steps $T$
        Choose $a$ from $s$ using a policy derived from $q$
        Take $a$, observe $r$ and $s'$                   ▷ Observe next state and reward
        $q(s, a) \leftarrow q(s, a) + \alpha[r + \gamma \text{max}_{a'} q(s', a') - q(s, a)]$
        $s \leftarrow s'$
    end for
until convergence
```

Algorithm 5 SARSA learning

```
Input: $\pi$, $\gamma$, $\alpha$
Initialize: $q(s, a) = 0 \ \forall s \in S, \ \forall a \in A$
Output: $q*$                                   ▷ Return the optimal action-value function
repeat
    Choose $a$ from $s$ using a policy derived from $q$
    for $i \leftarrow 1, T$ do         ▷ Loop until the episode terminates or for a fixed steps $T$
        Take $a$, observe $r$ and $s'$                   ▷ Observe next state and reward
        Sample $a'$ from $s'$ using a policy derived from $q$
        $q(s, a) \leftarrow q(s, a) + \alpha[r + \gamma q(s', a') - q(s, a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    end for
until convergence
```

The algorithms presented here use one step TD update. These algorithms can be extended to incorporate various targets such as n-step, λ target and regularized target as discussed in 3.2.2. Also, these algorithms can be extended to a function approximation setting in a similar fashion to that of policy evaluation case [24, 44, 76]. However, these algorithms find the optimal policy that are limited to discrete action space. Moreover, the optimal policy is found indirectly by finding the values first.

3.3.3 Policy Gradient Methods

Policy gradient methods are a family of reinforcement learning algorithms that maximizes the objective of an agent by directly mapping the states to a policy [8, 101, 99]. In these methods, a set of parameters θ map the states to the policy. These parameters are trained to maximize the below objective function

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right], \tag{3.22}$$

where $J(\theta)$ is the objective function of the agent. Once the objective function has been calculated, the parameters θ are updated in a direction that maximizes the objective function

$$\theta \leftarrow \theta + \alpha \nabla J(\theta), \tag{3.23}$$

where α is the learning rate, $\nabla J(\theta)$ is the gradient of the objective function with respect to the parameters θ. For a discrete states and actions, the gradient, $\nabla J(\theta)$, can be expressed as [84]

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a). \tag{3.24}$$

where $\mu(s)$ is the state distribution according to the policy π. Substituting Eq. (3.24) in the update equation yields

$$\theta \leftarrow \theta + \alpha \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a). \tag{3.25}$$

It is also possible to update the parameters using the Monte Carlo return. REINFORCE [99, 100] uses a Monte Carlo return to update the parameters as below.

$$\theta \leftarrow \theta + \alpha G_t \frac{\nabla_\theta \pi(a|s)}{\pi(a|s)}, \tag{3.26}$$

where G_t is Monte Carlo return. As discussed earlier, Monte Carlo return has a high variance resulting in poor updates. To avoid this problem, a REINFORCE with baseline method was introduced [99, 100]. These methods subtract a state-dependent baseline from the returns to reduce the variance. This idea is often referred to as control variate in statistics.

$$\theta \leftarrow \theta + \alpha(G_t - b(s))\frac{\nabla_\theta \pi(a|s)}{\pi(a|s)}. \tag{3.27}$$

where $b(s)$ is a state-dependent baseline. Several works introduced various techniques to set the baseline to reduce variance of the return G_t without incurring additional bias [25, 88].

3.3.4 Actor-Critic Methods

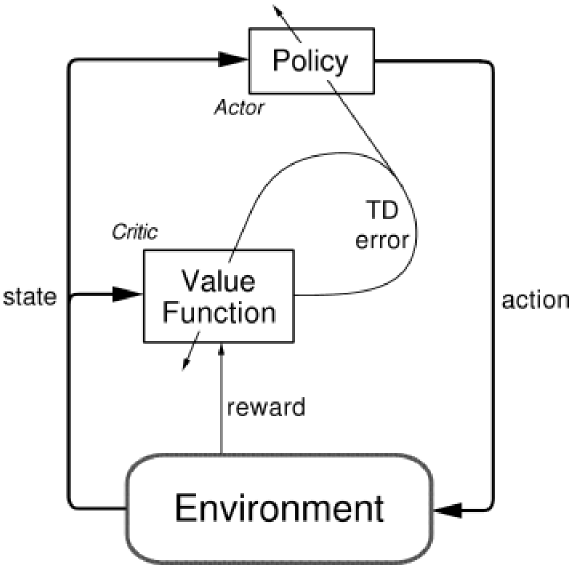


Figure 3.2: Actor-Critic Methods

Monte Carlo methods tend to have a poor performance due to the variance they introduce. Also, those methods are not suited for online learning or continual learning. Actor-Critic methods incorporate the strengths of TD methods (multi-step methods, bootstrapping) in policy gradient methods thereby not using a Monte Carlo return. The update for one-step actor-critic method looks as follow:

$$\begin{aligned} \theta &\leftarrow \theta + \alpha(G_t - v(s_t))\frac{\nabla_\theta \pi(a|s)}{\pi(a|s)} \\ \theta &\leftarrow \theta + \alpha\delta_t\frac{\nabla_\theta \pi(a|s)}{\pi(a|s)}. \end{aligned} \tag{3.28}$$

where $\delta = r + \gamma v(s_{t+1}) - v(s_t)$ is the TD-error. Note that we replace the Monte Carlo return in the REINFORCE with value function baseline with TD error. Typically the value function is estimated by a different function approximator parameterized by w. At each point in time, both value approximator (critic) and policy approximator (actor) are updated. This is illustrated in the Figure 3.2.[3] Note that the update in Eq. (3.28) uses one step update. The updates can easily be extended to incorporate eligibility traces both in actor and critic.

3.3.5 Deep Reinforcement Learning

In this section we discuss some of the latest works in deep reinforcement learning. All the methods discussed so far are restricted to small state space. Most of the real-world problems have exponentially large state spaces. For example, the game of go has more states than the total number of atoms in the universe. To tackle such high dimensional problems deep learning architectures along with reinforcement learning has been effective [44]. However, deep learning architectures along with reinforcement learning often has many problems [30]. One key issue is because of the high correlation between the samples which deep learning architectures cannot handle. Several methods were introduced to reduce the correlation between the samples. Some of the methods use experience replay buffer [44, 66, 3] to store the samples and and other methods use multiple agents to collect the data [43]. In this section we discuss some of the popular methods in policy gradient literature that work well with deep learning.

3.3.5.1 A3C, A2C, and PPO

Asynchronous Advantage Actor-Critic, A3C, methods [43] are a class of actor-critic methods where an emphasis is made on parallel training. In A3C, the critics learn the value function while multiple actors are trained in parallel and get synced with global parameters. As a result of parallel agents, the data collected is diverse and also

[3] the picture is reproduced from [5]

uncorrelated which yields better updates. Also using advantage [95], $A = q(s, a) - v(s)$, allows the actor to determine how good the action it picked was with respect to the expected value. [43] showed that they could achieve better results with this setting. However, in A3C, each actor running in parallel interacts with the global parameters independently. Due to asynchronous interaction, it could happen that various agents are running a different version of the policy therefore the aggregated update is not optimal. To resolve this issue, A2C was introduced [43]. A2C methods is similar to A3C methods except a coordinator is introduced to sync the updates. This method resulted in faster training and convergence speeds due to the synchronized updates. The Figure 3.3 [4] shows the architectural difference between A2C and A3C. Training stability can be improved by restricting the updates that change the policy too much at one step. Trust Region Policy Optimization (TRPO) [69] enforces a constraint on the size of policy update at each iteration.

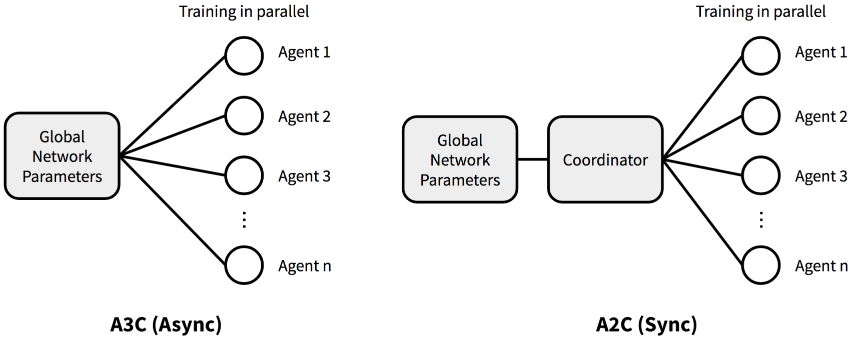


Figure 3.3: Architecture of A3C versus A2C

$$J(\theta) = \mathbb{E}[G_t] + \beta \mathrm{KL}(\pi_{\mathrm{old}} || \pi_{\mathrm{new}}). \tag{3.29}$$

The constraint defined in Eq. (3.29) forces the old and new policies to not diverge too much while still guaranteeing monotonic policy improvements. However, this constraint is complicated to use in practice. Proximal Policy Optimization (PPO) [68] simplifies the constraint by using a clipped surrogate objective while retaining the

[4] reproduced from https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#policy-gradient-theorem with permission

performance. Clipping the objective in a region restricts the extreme policy updates. Trust Region methods can be viewed as a regularization in the policy space. PPO has been the state of the art algorithm in model-free reinforcement learning and has achieved impressive results across various domains. For this reason, PPO is widely popular to benchmark various methods. In the next chapter we introduce Recurrent Temporal Difference which uses a gated exponential value function to estimate the value of a state. We use PPO and A2C algorithms to benchmark the results.

4

Recurrent traces

In the previous chapter we saw various Temporal Difference methods. Though these methods are widely popular [36, 1, 94, 45, 43], its use in the real-world remains limited due, in part, to the high variance of value function estimates [25], leading to poor sample complexity [22, 31]. This phenomenon is exacerbated by the noisy conditions of the real-world [21, 55]. Real-world applications remain challenging as they often involve noisy data such as sensor noise and partially observable environments.

The problem of disentangling signal from noise in sequential domains is not specific to Reinforcement Learning and has been extensively studied in the Supervised Learning literature. In this work, we leverage ideas from time series literature and Recurrent Neural Networks to address the robustness of value functions in Reinforcement Learning. We propose Recurrent Value Functions (RVFs): an exponential smoothing of the value function. The value function of the current state is defined as an exponential smoothing of the values of states visited along the trajectory where the value function of past states are summarized by the previous RVF.

However, exponential smoothing along the trajectory can result in a bias when the value function changes dramatically through the trajectory (non-stationarity). This bias could be a problem if the environment encounters sharp changes, such as falling of a cliff, and the estimates are heavily smoothed. To alleviate this issue, we propose to use exponential smoothing on value functions using a trainable state-dependent emphasis function which controls the smoothing coefficients. Intuitively, the emphasis function adapts the amount of emphasis required on the current value function and the past RVF to reduce bias with respect to the optimal value estimate. In other words, the emphasis function identifies important states in the environment. An important state can be defined as one where *its value differs significantly from the previous values along the trajectory.* For example, when falling off a cliff, the value estimate changes dramatically, making states around the cliff more salient. This emphasis function serves a similar purpose to a gating mechanism in a Long Short Term Memory cell of a Recurrent Neural Network [27].

To summarize, in this chapter, we introduce RVFs to estimate the value function of a state by exponentially smoothing the value estimates along the trajectory. RVF formulation leads to a natural way of learning an emphasis function which mitigates the bias induced by smoothing. First, we perform a set of experiments to demonstrate the robustness of RVFs with respect to noise in continuous control tasks and then provide a qualitative analysis of the learned emphasis function which provides interpretable insights into the structure of the solution.

4.1 Recurrent Value Functions (RVFs)

As mentioned earlier, performance of value-based methods are often heavily impacted by the quality of the data obtained [21, 55]. For example, in robotics, noisy sensors are common and can significantly hinder performance of popular methods [63]. In this work, we propose a method to improve the robustness of value functions by estimating the value of a state s_t using the estimate at time step t and the estimates of previously visited states s_i where $i < t$. Mathematically, the Recurrent Value Function (RVF) of a state s at time step t is given by:

$$V^{\beta}(s_t) = \beta(s_t)V(s_t) + (1 - \beta(s_t))V^{\beta}(s_{t-1}), \tag{4.1}$$

where $\beta(s_t) \in [0, 1]$. V^β estimates the value of a state s_t as a convex combination of current estimate $V(s_t)$ and previous estimate $V^\beta(s_{t-1})$. $V^\beta(s_{t-1})$ can be recursively expanded further, hence the name Recurrent Value Function. β is the emphasis function which updates the recurrent value estimate.

In contrast to traditional methods that attempt to minimize Eq. 3.16, the goal here is to find a set of parameters θ, ω that minimize the following error:

$$\begin{aligned} \mathcal{L}(\theta, \omega) &= \mathop{\mathbb{E}}_{\pi}[(V^\pi - V_\theta^\beta)^2], \\ V_\theta^\beta(s_t) &= \beta_\omega(s_t)V_\theta(s_t) + (1 - \beta_\omega(s_t))(V_\theta^\beta(s_{t-1})), \end{aligned} \tag{4.2}$$

where V_θ is a function parametrized by θ, and β_ω is a function parametrized by ω. This error is similar to the traditional error in Eq. 3.16, but we replace the value function with V_θ^β. In practice, V^π can be any target such as TD(0), TD(n), TD(λ) or Monte Carlo [80] which is used in Reinforcement Learning.

We minimize Eq. 4.2 by updating θ and ω using the semi-gradient technique which results in the following update rule:

$$\begin{aligned} \theta &= \theta + \alpha\delta_t\nabla_\theta V_\theta^\beta(s_t), \\ \omega &= \omega + \alpha\delta_t\nabla_\omega V_\theta^\beta(s_t), \end{aligned} \tag{4.3}$$

where $\delta_t = V^\pi(s_t) - V_\theta^\beta(s_t)$ is the TD error with RVF in the place of the usual value function. The complete algorithm using the above update rules can be found in Algorithm 6.

Algorithm 6 Recurrent Temporal Difference(0)

```
Input: $\pi$,$\gamma$,$\theta$,$\omega$
Initialize: $V_\theta^\beta(s_0) = V_\theta(s_0)$
Output: $\theta, \omega$                                    ▷ Return the learned parameters
for t do
    Take action $a \sim \pi(s_t)$ , observe $r(s_t), s_{t+1}$
    $V_\theta^\beta(s_t) = \beta_\omega(s_t)V_\theta(s_t) + (1 - \beta_\omega(s_t))V_\theta^\beta(s_{t-1})$        ▷ Compute the RVF
    $\delta_t = V^\pi(s_t) - V_\theta^\beta(s_t)$                 ▷ Compute TD error with respect to RVF
    $\theta = \theta + \alpha\delta_t\nabla_\theta V_\theta^\beta(s_t)$          ▷ Update parameters of the value function $\theta$
    $\omega = \omega + \alpha\delta_t\nabla_\omega V_\theta^\beta(s_t)$      ▷ Update parameters of the emphasis function $\omega$
end for
```

As discussed earlier, β_ω learns to identify states whose value significantly differs from previous estimates. While optimizing for the loss function described in Eq. 4.2, the $\beta_\omega(s_t)$ learns to bring the RVF V_θ^β closer to the target V^π. It does so by placing greater emphasis on whichever is closer to the target, either $V_\theta(s_t)$ or $V_\theta^\beta(s_{t-1})$. Concisely, the updated behaviour can be split into four scenarios. A detailed description of these behaviours is provided in Table 4.1. Intuitively, if the past is not aligned with the future, β will emphasize the present. Likewise, if the past is aligned with the future, then β will place less emphasis on the present. This behaviour is further explored in the experimental section.

Table 4.1: Behaviour of β based on the loss

	$V^\pi(s_t) > V_\theta^\beta(s_t)$	$V^\pi(s_t) < V_\theta^\beta(s_t)$
$V_\theta(s_t) > V_\theta^\beta(s_{t-1})$	$\beta \uparrow$	$\beta \downarrow$
$V_\theta(s_t) < V_\theta^\beta(s_{t-1})$	$\beta \downarrow$	$\beta \uparrow$

Note that, the gradients of V_θ^β take a recursive form (gradient through time) as shown in Eq. 4.4. The gradient form is similar to LSTM [27], and GRU [13] where β acts as a gating mechanism that controls the flow of gradient. LSTM uses a gated exponential smoothing function on the hidden representation to assign credit more effectively. In contrast, we propose to exponentially smooth the outputs (value functions) directly rather than the hidden state. This gradient can be estimated using backpropagation through time by recursively applying the chain rule where:

$$\nabla_\theta V_\theta^\beta(s_t) = \beta_\omega(s_t)\nabla_\theta V_\theta(s_t) + (1 - \beta_\omega(s_t)).\nabla_\theta V_\theta^\beta(s_{t-1}) \tag{4.4}$$

However, this can become computationally expensive in environments with a large episodic length, such as continual learning. Therefore, we could approximate the gradient $\nabla_\theta V_\theta^\beta(s_t)$ using a recursive *eligibility* trace as a function of θ:

$$e_t = \beta_\omega(s_t)\nabla_\theta V_\theta(s_t) + (1 - \beta_\omega(s_t))e_{t-1}. \tag{4.5}$$

4.2 Experiments

In this section, we perform experiments on various tasks to demonstrate the effectiveness of RVF. First, we explore RVF robustness to partial observability on a synthetic domain. We then showcase RVF's robustness to noise on several complex continuous control tasks from the Mujoco suite [90]. An example of policy evaluation is also provided in Section 4.4.1 as a further reading.

4.2.1 Partially observable multi-chain domain

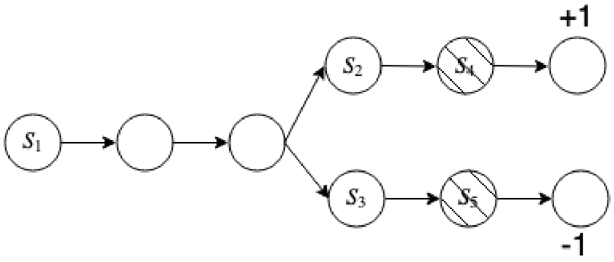


Figure 4.1: Simple chain MDP.

We consider partial observability as a first issue to study because RVFs can carry forward the value function from the past. Therefore instead of computing the value function of a state which is not completely observable, it uses the value of the past state as a proxy for the value of current state. To show this we consider the simple chain MDP described in Figure 4.1. This MDP has three chains connected together to form a Y. Each of the three chains (right of S_1, right of S_2, right of S_3) is made up of a sequence of states. The agent starts at S_1 and navigates through the chain. At the intersection, there is a 0.5 probability to go up or down. The chain on top receives a reward of $+1$ while the one at the bottom receives a reward of -1. Every other transition has a reward of 0, unless specified otherwise. We explore the capacity of recurrent learning to solve a partially observable task in the Y chain. In particular, we consider the case where some states are aliased (share a common representation). The representation of the states S_4 and S_5 in Figure 4.1 are aliased. The goal of this environment is to

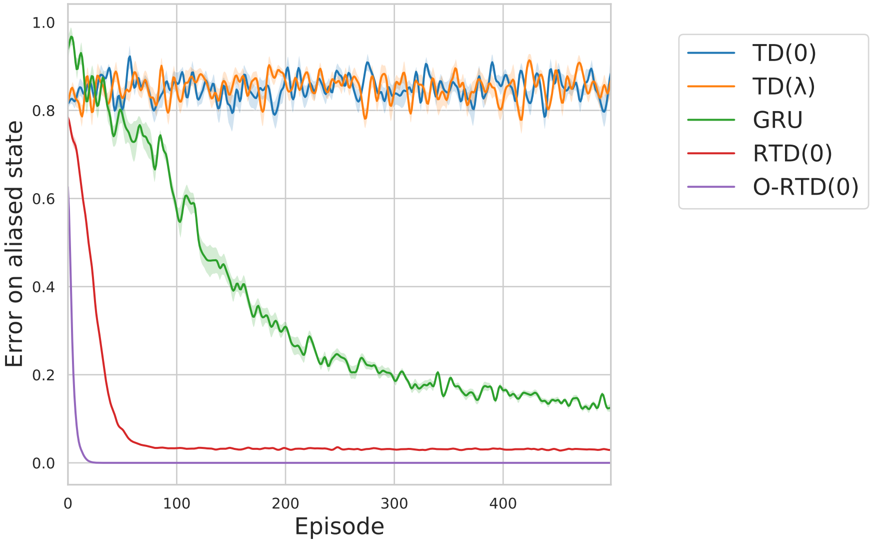


Figure 4.2: Results on the aliased Y-chain.

correctly estimate the value of the aliased state $V^{\pi}(S_4) = 0.9, V^{\pi}(S_5) = -0.9$ (due to the discount factor(0.9) and the length of each chain being 3). When TD methods such as TD(0) or TD(λ) are used, the values of the aliased states S_4 and S_5 are close to 0 as the reward at the end of the chain is +1 and −1. However, when learning β (emphasis function β is modelled using a sigmoid function), Recurrent Value Functions achieve almost no error in their estimate of the aliased states as illustrated in Figure 4.2. This can be explained by observing that $\beta \to 0$ on the aliased state due to the fact that the previous values along the trajectory are better estimates of the future than those of the aliased state. As $\beta \to 0$, $V_{\theta}^{\beta}(S_4)$ and $V_{\theta}^{\beta}(S_5)$ tend to rely on their more accurate previous estimates, $V_{\theta}^{\beta}(S_2)$ and $V_{\theta}^{\beta}(S_3)$. We see that learning to ignore certain states can at times be sufficient to solve an aliased task. We also compare the proposed method with a optimal recurrent temporal difference (O-RTD) which uses optimal values of β. O-RTD is exactly the same as proposed method except it uses optimal β values instead of learning them. In this setting, $\beta(S_1) = \beta(S_2) = \beta(S_3) = 1$ and other states have $\beta = 0$. Another interesting observation is with respect to Recurrent Neural Networks. RNNs are known to solve tasks which have partial observability by inferring the underlying state. LSTM and GRU have many parameters that are used to infer the hidden state. Correctly learning to keep the hidden state intact can be sample-inefficient. In comparison, β can estimate whether or not to put emphasis (confidence) on a state value using a single parameter. This is illustrated in Figure 4.2 where RNNs take 10 times more episodes to learn the optimal value when compared to RVF. This illustrates a case where learning to ignore a state is easier than inferring its hidden representation. The results displayed in Figure 4.2 are averaged over 20 random seeds. For every method, a hyperparameter search is done to obtain their optimal value. These can be found in Section 4.4.2. We noticed that the emphasis function is easier to learn if the horizon of the target is longer, since a longer horizon provides a better prediction of the future. To account for this, we use λ-return as a target.

4.2.2 Deep Reinforcement Learning

Next, we test RVF on several environments of the Mujoco suite [90]. We also evaluate the robustness of different algorithms by adding ϵ sensor noise (drawn from a normal distribution $\epsilon \sim N(0, 1)$) to the observations as presented in [105]. We modify the critic of A2C [102] (R-A2C) and Proximal Policy Optimization (R-PPO) [68] to estimate the recurrent value function parametrized by θ. We parametrize β using a seperate network with the same architecture as the value function (parametrized by ω). We minimize the loss mentioned in Eq. 4.2 but replace the target with generalized advantage function (V_θ^λ) [67] for PPO and TD(n) for A2C. Using an automatic differentiation library (Pytorch [53]), we differentiate the loss through the modified estimate to learn θ and ω. The default optimal hyperparameters of PPO and A2C are used. Due to the batch nature of PPO, obtaining the trajectory information to create the computational graph can be costly. In this regard, we cut the backpropagation after N timesteps in a similar manner to truncated backpropagation through time. The number of backpropagation steps is obtained using hyperparameter search. Details can be found in Section 4.4.3.

We use a truncated backprop of $N = 5$ in our experiments as we found no empirical improvements for $N = 10$. For a fair comparison in the noisy case, we also compare the performance of two versions of PPO with an LSTM. The first version processes one trajectory every update. The second uses a buffer in a similar manner to PPO, but the gradient is cut after 5 steps as the computation overhead from building the graph every time is too large. The performance reported is averaged over 20 different random seeds with a confidence interval of 68% displayed for visual clarity. [1]

4.2.2.1 Performance

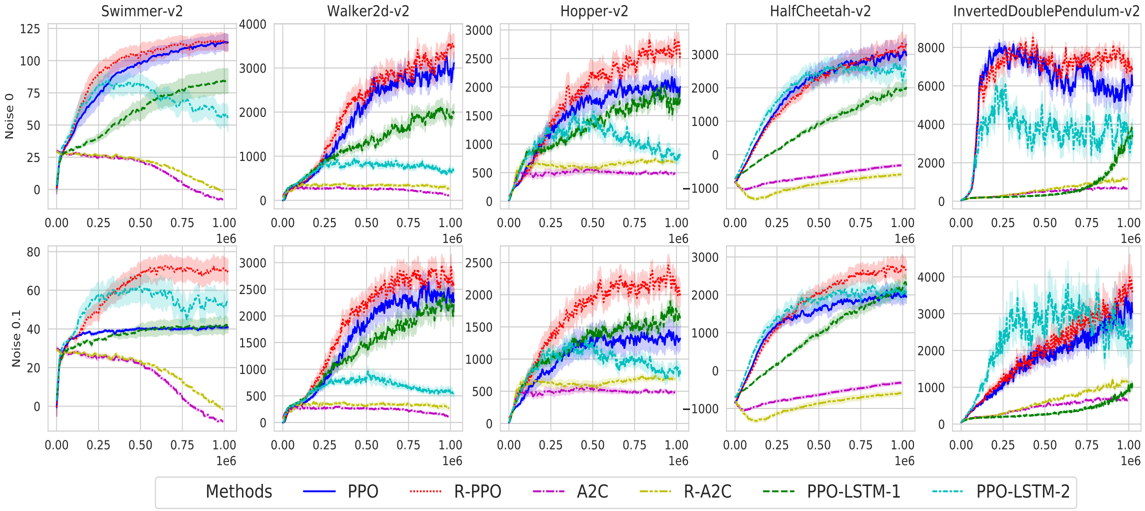


Figure 4.3: Performance on Mujoco tasks with and without a Gaussian noise ($\epsilon \sim \mathcal{N}(0, 0.1)$) in the sensor inputs.

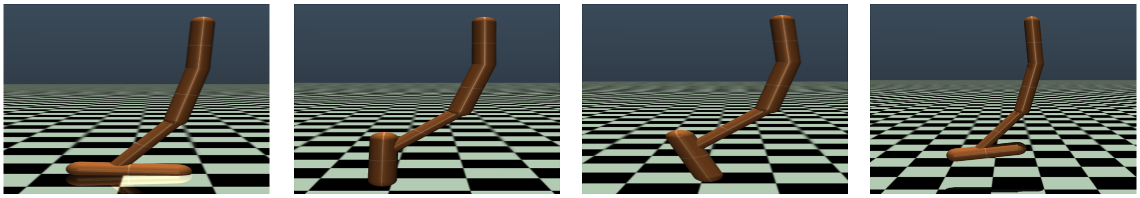
As demonstrated in Figure 4.3, we observe a marginal increase in performance on several tasks such as Swimmer, Walker, Hopper, Half Cheetah and Double Inverted Pendulum in the fully observable setting. However, severe drops in performance were observed in the vanilla PPO when we induced partial observability by introducing a Gaussian noise to the observations. On the other hand, R-PPO (PPO with RVF) was found to be robust to the noise, achieving significantly higher performance in all the tasks considered. In both cases, R-PPO outperforms the partially observable

[1] The base code used to develop this algorithm can be found here [37]. The code for Recurrent PPO can be found in the supplementary material.

models (LSTM). The mean and standard deviation of the emphasis function for both noiseless and noisy versions can be found in Section(4.7, 4.8). At the same time, A2C's performance on both vanilla and recurrent versions (referred as R-A2C) was found to be poor. We increased the training steps on both versions and noticed the same observations as mentioned above once A2C started to learn the task. The performance plots, along with the mean and standard deviation of the emphasis function during training, can be found in Section (4.9, 4.10, 4.11, 4.12).

4.2.2.2 Qualitative interpretation of the emphasis function β

Hopper: At the end of training, we can qualitatively analyze the emphasis function (β) through the trajectory. We observe cyclical behaviour shown in Figure 4.5, where different colours describe various stages of the cycle. The emphasis function learned to identify *important states* and to ignore the others. One intuitive way to look at the emphasis function(β) is: *If I were to give a different value to a state, would that alter my policy significantly?* We observe an increase in the value of the emphasis function (β) when the agent must make an important decision, such as jumping or landing. We see a decrease in the value of the emphasis function (β) when the agent must perform a trivial action. This pattern is illustrated in Figure 4.4 and 4.5. This behaviour is cyclic and repetitive, a video of which can be found in the following link[2].



Phase 1: high β Phase 2: low β Phase 3: high β Phase 4: low β

Figure 4.4: Cyclical behaviour of β on Hopper.

[2] https://youtu.be/0bzEcrxNwRw

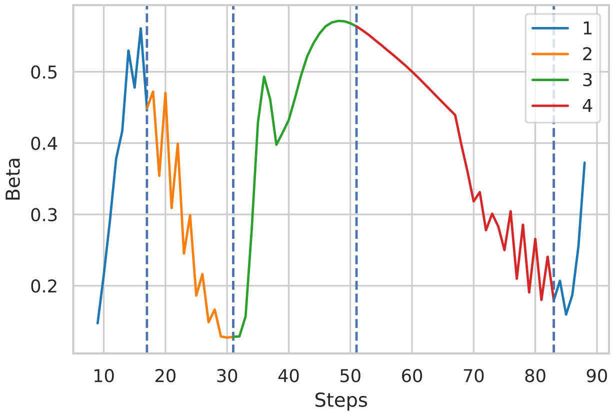


Figure 4.5: Behaviour of β through the trajectory.

4.3 Conclusion

In this chapter we propose Recurrent Value Functions to address variance issues in Model-free Reinforcement Learning. First, we demonstrate the robustness of RVF to noise and partial observability in a synthetic example and on several tasks from the Mujoco suite. Then, we describe the behaviour of the emphasis function qualitatively.

4.4 Additonal Details

4.4.1 Policy Evaluation

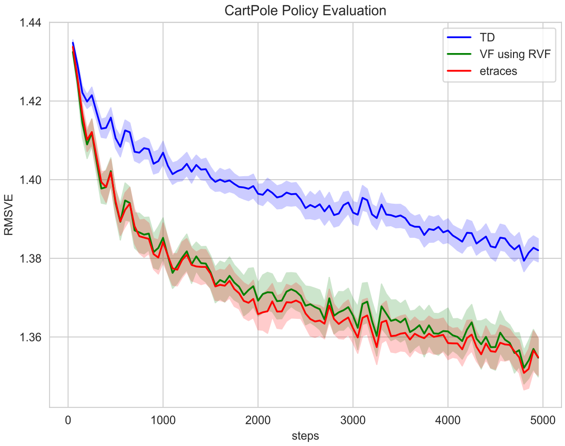


Figure 4.6: Policy Evaluation using RVF

In this experiment, we perform a policy evaluation on CartPole task. In this environment the agent has to balance the pole on a cart. The agent can take one of the two actions available which would move the cart to either left or right. A reward of +1 is obtained for every time step and an episode is terminated if the cart moves too far from the center or the pole dips below a certain angle.

In this task, we use a pretrained network to obtain the features that represent the underlying state. We train a linear function approximator using those features to estimate the value function. This experimental setup is similar to [14] but the features in our case are obtained through a pretrained network instead of training a separate network. The samples are generated following a fixed policy and the same samples were used across all the methods to estimate value function. Each sample consists of 5000 transitions and the results are averaged over 40 such samples. We calculated the optimal value V_π using a Monte Carlo estimate for 2000 random states following the same policy. We use the trained linear network to predict value function on these 2000 states. Once we get the predictions we calculate their Root Mean Square Value Error (RMSVE). The best hyperparameters were obtained through hyperparameter search for each method separately. The optimal learning rate was found to be 0.005 for TD and RVF while 0.0005 for eligibility traces when we did a search in $\{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05\}$. The optimal β learning rate was found to be 0.005 when we searched in $\{0.001, 0.005, 0.0001, 0.00050.01\}$ and the optimal lambda for eligibility traces was found to be 0.9 when searched in $\{0.1, 0.4, 0.7, 0.9\}$. The RMSVE on various methods such as TD, eligibility traces - online TD(λ) and the value functions of the state obtained using RVF algorithm are reported in Figure 4.6. We notice that the the value function learned through RVF algorithm has approximately the same error as the value function learned through eligibility traces. Both RVF and eligibility traces outperform TD methods.

4.4.2 Hyperparameters Toy MDP

For every method the learning rate and λ was tuned for optimal performance in the range $[0, 1]$. For RTD a learning rate of 0.5 for the value function and 1 for the beta function was found to be optimal with a lambda of 0.9. For the GRU model we explored different amount of cell($\{1, 5, 10, 15, 20, 25\}$) to vary the capacity of the model. The optimal number of hidden cell we found is 10, learning rate 0.5 and lambda 0.9.

4.4.3 Deep Reinforcement Learning

The best hyperparameters were selected on 10 random seeds.

PPO: The following values were considered for choosing the best learning rate $\{3E-05, 6E-05, 9E-05, 3E-04, 6E-04\}$ and $N = \{2, 5, 10\}$. The optimal values for learning rate is the same one obtained in the original PPO paper - $3E-04$ and $N = 5$. We also compare with a larger network for PPO to adjust for the additional parameter of β the performance of vanilla PPO were found to be similar. In terms of computational cost, RVF introduce a computational overhead slowing down training by a factor of 2 on a CPU(Intel Skylake cores 2.4GHz, AVX512) compared to PPO. The results are reported on 20 random seeds and a confidence interval of 68% is displayed.

A2C: The following values were considered for the learning rate $\{1E-05, 1E-04, 1E-03, 1E-02\}$ and the best learning rate was found to be $1E-04$. We tested bootstrapping on $5, 10, 15$ steps, we found no empirical improvements between 5 and 10 but noticed a significant drop in performance for bootstrapping > 10. We believe that, it is due to the increase in variance of the target as we increase the steps of the bootstrapping. The best hyperparameters were found by averaging across 10 random seeds. The results reported are averaged across 20 random seeds. The confidence interval of 95% is displayed.

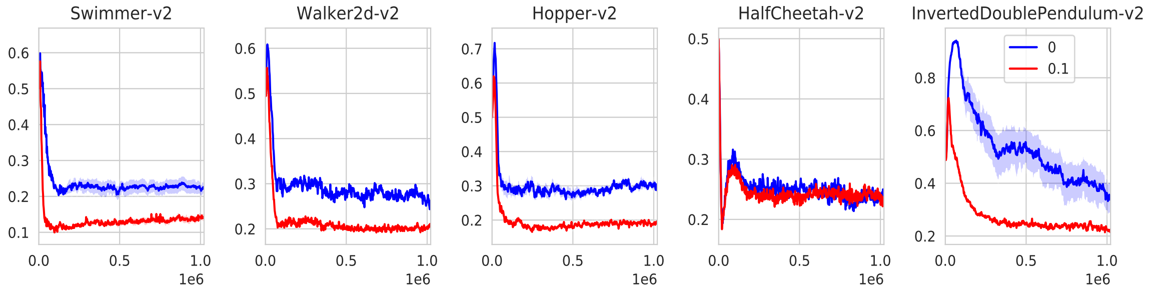


Figure 4.7: Mean beta values using recurrent PPO on Mujoco domains. X-axis is the number of steps and Y-axis is mean value of β. The two lines corresponds to different noise levels in the observation space are shown in the legend.

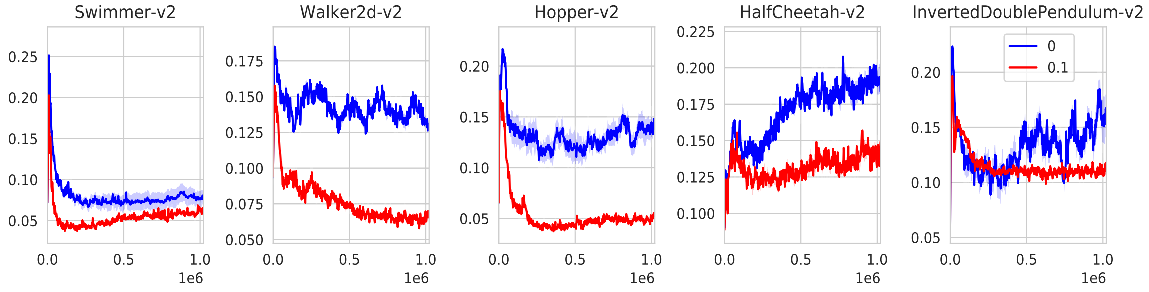


Figure 4.8: Standard deviation of beta using recurrent PPO on Mujoco domains. X-axis is the number of steps and Y-axis is standard deviation of β. The two lines corresponds to different noise levels in the observation space are shown in the legend.

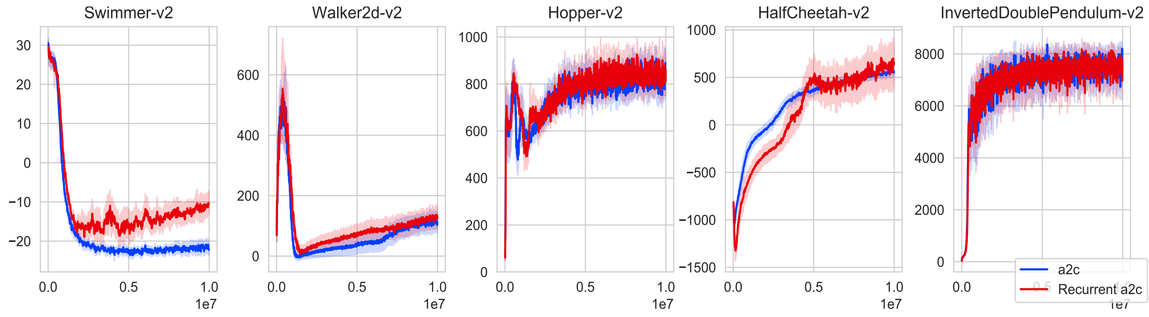


Figure 4.9: Performance of A2C and recurrent A2C on Mujoco tasks without noise in observations for 10M steps

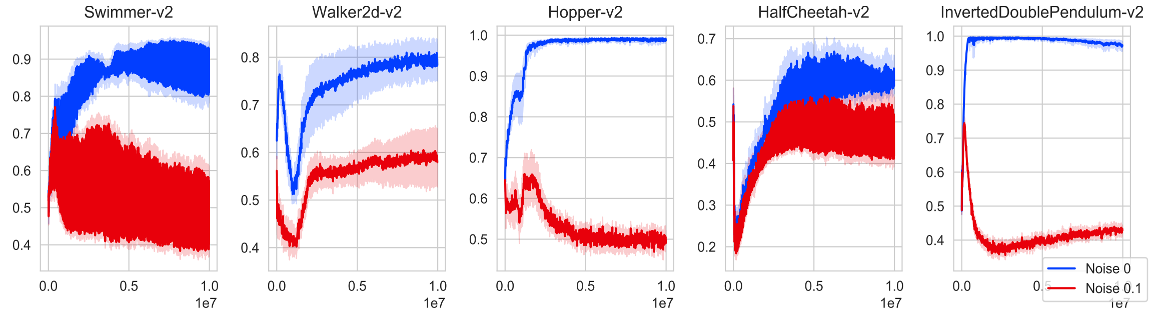


Figure 4.11: The mean of the emphasis function on various Mujoco tasks with and without noise plotted against the number of updates. X-axis is the number of steps and Y-axis is mean value of β. The two lines corresponds to different noise levels in the observation space are shown in the legend.

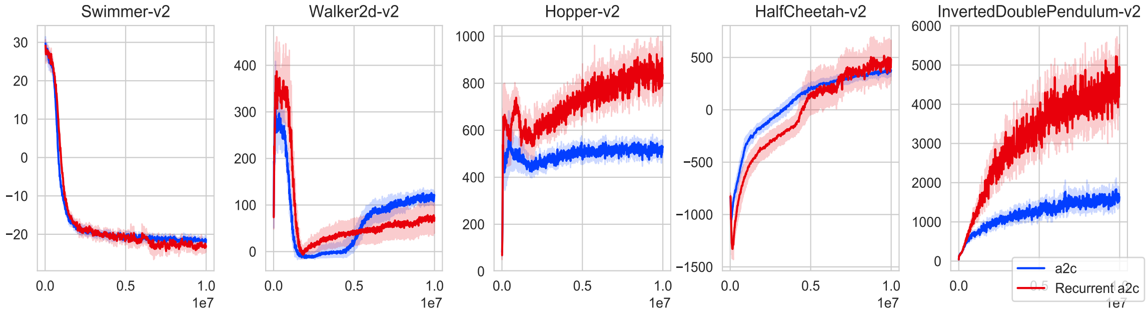


Figure 4.10: Performance of A2C and recurrent A2C on Mujoco tasks with a Gaussian noise(0.1) in observations for 10M steps. X-axis is the number of steps and Y-axis is mean performance score.

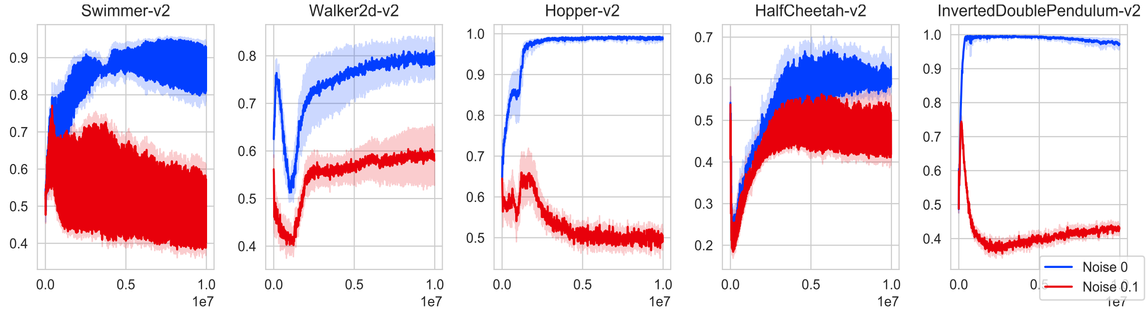


Figure 4.12: The standard deviation of the emphasis function on various Mujoco tasks with and without noise plotted against the number of updates. X-axis is the number of steps and Y-axis is standard deviation of β. The two lines corresponds to different noise levels in the observation space are shown in the legend.

5

Filter traces

In the previous chapters, we saw several trace-based methods that try to solve temporal credit assignment problem in reinforcement learning. The first trace-based method is accumulating eligibility traces [79]. Other trace based methods [75, 70] were built on top of accumulating eligibility traces. These algorithms were aimed at estimating TD(λ) return in an online fashion. Another method of interesting traces is emphatic trace [40, 82] which uses the interest function of a state to update the trace value. More recently, recurrent traces were proposed (Chapter 4), [86] which is obtained by modifying the temporal difference error. In this work, we use IIR filters to compute trace in reinforcement learning. Mathematically, general form of the trace is $e_t = \sum_{i=1}^{n} \alpha_i e_{t+1-i} + \sum_{j=1}^{k} \beta_j x_{t+1-i}$, where x_i are the input signal to eligibility trace (indicator function in the case of tabular, gradients in the case of function approximator), α_i, β_j are filter coefficients. This version of traces is inspired by Filter traces [2] which uses IIR filter traces for credit assignment. We use the same formulation to estimate traces but the main difference between this work and the Filter traces is that, firstly, we show how various traces can be viewed as an IIR filter, then use this information to unify them. Also unlike the Filter traces, we do not have fixed filter coefficients. Instead, we use attention mechanism to learn the filter coefficients.

Filters are a common signal processing tool that is used in many applications. They possess a property of blocking certain components of signals while passing other components through them. The passage and blockage of the signal can be controlled by setting the filter coefficients appropriately. However, in reinforcement learning, unlike signal processing tasks, no information about the properties of components that we want to filter out is available. This makes it challenging to set the coefficients of the filter. We propose a method to circumvent this problem in this chapter.

To summarize, in this chapter, we unify various credit assignment techniques mentioned earlier under filters framework. Later, we explore the application of filters for credit assignment on a synthetic MDP. To test the effectiveness of filters, we apply the algorithm on Mountain Car with a tile coding function approximator. While doing so, we use attention mechanism to set the filter coefficients.

5.1 Accumulating traces as Filters:

We limit our discussions to Infinite Impulse Response (IIR) filters. In particular, as discussed in Chapter 2, IIR filters have an infinite response when an *impulse signal* is presented at the input. The output of such a system never dies. These filters typically have a feedback connection from the output which acts as an input to the system at later points in time. These filters are mathematically given by,

$$y_t = \sum_{i=1}^{n} \alpha_i y_{t+1-i} + \sum_{j=1}^{k} \beta_j x_{t+1-i} \tag{5.1}$$

where, y_i is the output of the filter at time step i, n is the total memory units of the filter corresponding to the input, k is the total memory units of the filter corresponding to the output, α_i, β_i are the filter coefficients, x_i is the input to the system at time step i.

Before we move on to unify various credit assignment techniques using filters, we first show that accumulating traces obey the properties listed in section 2.1.3. In particular, we show that accumulating traces obey linearity and time-invariance (shift-invariance) properties. We first prove the following lemma before the final proof.

Lemma 2. *In the linear function approximation case where the input to the eligibility traces are the features ϕ_t, the accumulating traces can be written as a convolution product of input ϕ_t with the system's response. Additionally, the system's response is given by $\gamma\lambda$, $e_t = \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\phi_k$.*

Proof. We begin by unrolling the trace equation $e_t = \gamma\lambda e_{t-1} + \phi_t$ over time,

$$\begin{aligned}
e_1 &= \gamma\lambda e_0 + \phi_1 = \phi_1 \\
e_2 &= \gamma\lambda e_1 + \phi_2 = \gamma\lambda\phi_1 + \phi_2 \\
e_3 &= \gamma\lambda e_2 + \phi_3 = (\gamma\lambda)^2\phi_1 + \gamma\lambda\phi_2 + \phi_3
\end{aligned} \tag{5.2}$$

In general, for any time t, eligibility traces can be written as a convolution product of $\gamma\lambda$ and the input. Also, $\gamma\lambda$ is the response of the system to the input.

$$e_t = \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\phi_k \tag{5.3}$$

□

We now verify the linearity and the shift-invariance properties of the accumulating traces.

Theorem 2. *Accumulating traces are a Linear Shift Invariant systems.*

Proof. **Linearity:** Lets consider two eligibility traces, e' and e'' such that e' is the eligibility trace for the input ϕ_t and e'' is the eligibility trace for the input ψ_t.

From convolution product (5.3), we can write e' and e'' as $e'_t = \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\phi_k$ and $e''_t = \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\psi_k$.

Now, consider a new input η_t which is a linear combination of ϕ_t and ψ_t. That is, $\eta_t = \alpha\phi_t + \beta\psi_t$ where α and β are some constants.

$$\begin{aligned}
e_t &= \sum_{k=1}^{t}(\gamma\lambda)^{t-k}[\alpha\phi_k + \beta\psi_k] = \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\alpha\phi_k + \sum_{k=1}^{t}(\gamma\lambda)^{t-k}\beta\psi_k \\
e_t &= \alpha\sum_{k=1}^{t}(\gamma\lambda)^{t-k}\phi_k + \beta\sum_{k=1}^{t}(\gamma\lambda)^{t-k}\psi_k
\end{aligned} \tag{5.4}$$

Therefore, $e_t = \alpha e_t{}' + \beta e_t{}''$. This shows that changing the input in a linear fashion changes the output of the system linearly. In other words, if we were to modify the traces using some operations (addition or multiplication) then that modified trace corresponds to the trace of modified states (similar operation).

Time (Shift) Invariance: The trace vector e_t for input ϕ_t according to convolution product (5.3) is given by $e_t = \sum_{k=0}^{t} (\gamma\lambda)^{t-k}\phi_k$. The shifted version of the trace by n units is given by,

$$e = \sum_{k=n}^{t+n} (\gamma\lambda)^{t+n-k}\phi_{k+n} = e_{t+n} \tag{5.5}$$

Note that $\phi_k = 0 \; \forall \; k < 0$. Hence, the convolution starts at n. Since ϕ_{k+n} is the shifted version of ϕ_k the input $\phi_{k+n} = 0 \; \forall \; k < n$. Similarly, the convolution terminates at t for the original input ϕ_k, which means the convolution for the shifted input ϕ_{k+n} terminates at $k + n$. □

In terms of reinforcement learning, the shifted version of the trace corresponds to the trace whose input is shifted by the same amount. This property may be used in the long-term credit assignment. By shifting the trace by certain amount, we could pretend as if some state in the past has been observed recently. As a result of these two properties, accumulating traces are an LSI system and in particular a first order IIR filter as their response $(\gamma\lambda)$ never dies.

5.2 Filters as a Unifying Framework

In the previous section, we showed that accumulating version of trace is an IIR filter. In this section, we generalize other variants of traces as IIR filter. In particular, we consider accumulating traces[79], emphatic traces[40, 82], and recurrent traces[86] as IIR Filters. Before doing that, we first decompose the update equation in each of the cases into the *Credit* and *Assignment* parts as discussed in 3.2.2.2. To recap, *Credit* is the error calculated by the agent, *Assignment* is how this credit is used to update the states/state-actions followed in the past.

Table 5.1: Decomposition of various traces into credit and assignment parts.

Method	**Credit**	**Assignment**
Accumulating trace	$r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$	$\gamma\lambda e_{t-1} + \nabla v(s_t)$
Emphatic trace	$r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$	$\gamma_t\lambda_t e_{t-1} + ((1-\lambda_t)\gamma_t F_{t-1} + i_t)\nabla v(s_t)$
Recurrent trace	$G_t - v_\beta$	$(1-\beta_t)e_{t-1} + \beta_t\nabla v(s_t)$

We see from the table 5.1 that accumulating, emphatic and filter traces use one step TD error for calculating credit but the assignment is done in a different manner. Accumulating traces use an exponential weighing to assign the credit. Emphatic traces are also exponential weighing, however the parameters are state dependent. Moreover, they have an additional component in their formulation (interest function). Note that we limit our discussions to the on-policy version of emphatic traces and hence we discard the importance sampling ratio term from the traces. Recurrent traces, on the other hand, modify the TD error to include the trajectory value to estimate the credit. As a result of this, they have a convex version of traces.

Table 5.2: Realization of various assignment techniques as IIR filter. The state dependence is shown using the superscript t.

Method	**Filter coefficient (α)**	**Filter coefficient (β)**
Accumulating trace	$\alpha_1 = \gamma\lambda$, $\alpha_i = 0\ \forall i > 1$	$\beta_1 = 1$, $\beta_j = 0\ \forall j > 1$
Emphatic trace	$\alpha_1^t = \gamma_t\lambda_t$, $\alpha_i^t = 0\ \forall i > 1$	$\beta_1^t = ((1-\lambda_t)\gamma_t F_{t-1} + i_t), \beta_j^t = 0\ \forall j > 1$
Recurrent trace	$\alpha_1^t = (1-\beta_t), \alpha_i^t = 0\ \forall i > 1$	$\beta_1^t = \beta_t, \beta_j^t = 0\ \forall j > 1$

The table 5.2 shows the realization of various credit assignment techniques using IIR filter. Similar proof as that of Theorem 2 can be done on all these versions. Note that none of the parameters are time dependent hence they can be shown to obey shift invariance property. Also, all the traces are first order traces, that is, they have feedback only from one time step before. On the other hand, there is no limit on the number of feedbacks one can have in IIR traces. Hence, it opens up door to think of credit assignment beyond a single feedback system. More about this is discussed in the later sections of the paper.

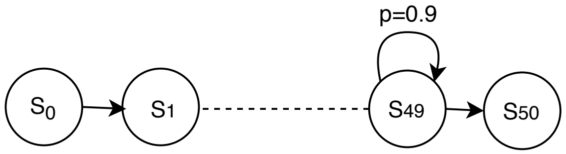


Figure 5.1: Loopy Chain MDP

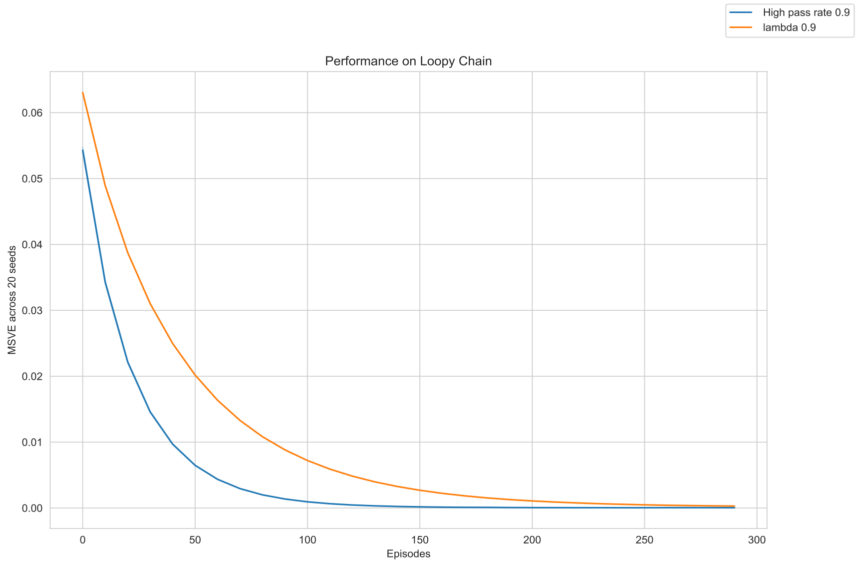


Figure 5.2: Performance of Filters against TD(λ)

5.3 Experiment: Toy MDP

We now demonstrate the application of filters in a toy domain. Consider a toy MDP as shown in Figure 5.1. This MDP consists of a sequence of 50 states at the end of which the agent receives a reward of +1. The agent transitions along the chain in all the states except for the penultimate loopy state where the agent is transitioned back to the same state with a probability p and transitioned to the last state with the probability $1 - p$. In this experiment we fix $p = 0.9$. The goal of this task is to estimate the value function v_π. We compare accumulating traces - an online version of TD(λ) method against "out-of-the-box" high-pass filter. The reason for using high-pass filter is that the state representation change rapidly in the tabular setting as each state is represented uniquely as a one-hot encoding vector. We calculate the filter traces using the filter, $e_t = a_0(e_{t-1} + |x_t - x_{t-1}|)$ where a_0 is the filter coefficient, e_t is the trace at time t, $|x_t - x_{t-1}|$ is the input to the filter. In this example, $|x_t - x_{t-1}|$ corresponds to the absolute difference between the components of two one-hot encoded state vectors. Since a negative component doesn't correspond to any state, we add an absolute value to the input. Since the chain is linear, we set a fixed learning rate for both filters and eligibility traces at 0.1 for a fair comparison. Best values of λ and filter coefficients were found on a set of values $\{0, 0.1, 0.4, 0.7, 0.9, 1.0\}$. The best parameter for both filter traces and λ traces are found to be 0.9. The results are averaged across 20 different random seeds. The performance is reported in Figure 5.2 where a Mean Squared Value Error (MSVE) is plotted against the episodes. The optimal value function, v_π is found using the bellman equation, $v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$ where P_π, r_π are transition probability and reward vector following the policy π. A discount factor, γ, is fixed at 0.99. From the Figure 5.2, we see that the filters learn the value v_π in less number of episodes than eligibility traces. This is because the filter doesn't update the trace when the agent is looping continuously in the loopy state. This is because the filter is a high pass filter and the signal is constant when the agent is looping in the same state. Also, at each point in time other than the loopy state, the trace is updated for states where the agent currently is and also where the agent was in the past state. As a result, both the past states and current states are updated at each point in time. Now we move on to discuss a method to learn filter coefficients.

5.3.1 Setting the filter coefficients

Unlike signal processing tasks, no prior information about the components of the input to filter out is provided in reinforcement learning. Filter traces [2] use the prior information about the environment to set the coefficients of the filter. This property is not desired in reinforcement learning as the agent has no knowledge about the environment apriori. Hence, setting the coefficients of the filter is challenging. Another challenge is the stability of the filters as discussed in Chapter 2. Most of the feedback systems (IIR filters) tend to be unstable if the coefficients are not set appropriately. Keeping these two challenges in mind, we propose to use two levels of attention mechanism [6] to set the coefficients of the filter. In the first level, we set the coefficients of the feedback (past traces) and in the second level we set the coefficients of the inputs. To set the coefficients of past traces/feedback, a *softmax* similarity score is obtained between the the past traces in the memory and the next state. A similar score is obtained to set the coefficients of the inputs separately. Mathematically, the filter coefficients are given by,

$$\alpha_i = \frac{e_{t-i}^T \phi_{s_{t+1}}}{\sum_{j=1}^{n} e_{t-j}^T \phi_{s_{t+1}} + \sum_{k=0}^{m} \phi_{t-k}^T \phi_{s_{t+1}}}, \ \forall i \in n \tag{5.6}$$

$$\beta_i = \frac{\phi_{t-i}^T \phi_{s_{t+1}}}{\sum_{k=0}^{m} \phi_{t-k}^T \phi_{s_{t+1}}}, \ \forall i \in m \tag{5.7}$$

where α and β are attention coefficients, n is the number of recursive memory units in the filter, m is the number of memory units to store the input. This is a stable filter as all the coefficients are less than 1. A similar technique through attention is also used in sparse attentive backtracking [33]. However their proposed framework doesn't compute a trace, instead they store all the transitions in the memory. Complete algorithm for a linear function approximation case is provided in Algorithm 7.

Algorithm 7 IIR credit assignment with attention for policy evaluation

```
Input: $\pi$,$n$,$m$,$\gamma$,$\theta$
Initialize: $e_{0:-n} = 0$, $x_{0:-m} = 0$
Output: $\theta$                                    ▷ Return the learned parameters
Append $\phi(s_0) \rightarrow x$ and Remove the oldest element in $x$
for t do
    Choose $a \sim \pi(s_t)$, observe $r(s_t), s_{t+1}$
    Compute filter coefficients $\alpha$ and $\beta$ using Eq. (5.6) and Eq. (5.7)
    $e_t = \sum_{i=1}^{n} \alpha_{i-1} e_{t-i} + \sum_{j=0}^{m} \beta_j x_{t-j}$      ▷ Update the trace using the filter
    $\delta_t = r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$                 ▷ Compute TD error
    $\theta = \theta + \alpha \delta_t e_t$                 ▷ Update parameters of the value function
    Append $e_t \rightarrow e$ and Remove the oldest element in $e$
    Append $\phi(s_{t+1}) \rightarrow x$ and Remove the oldest element in $x$
end for
```

5.4 Experiment: Mountain Car

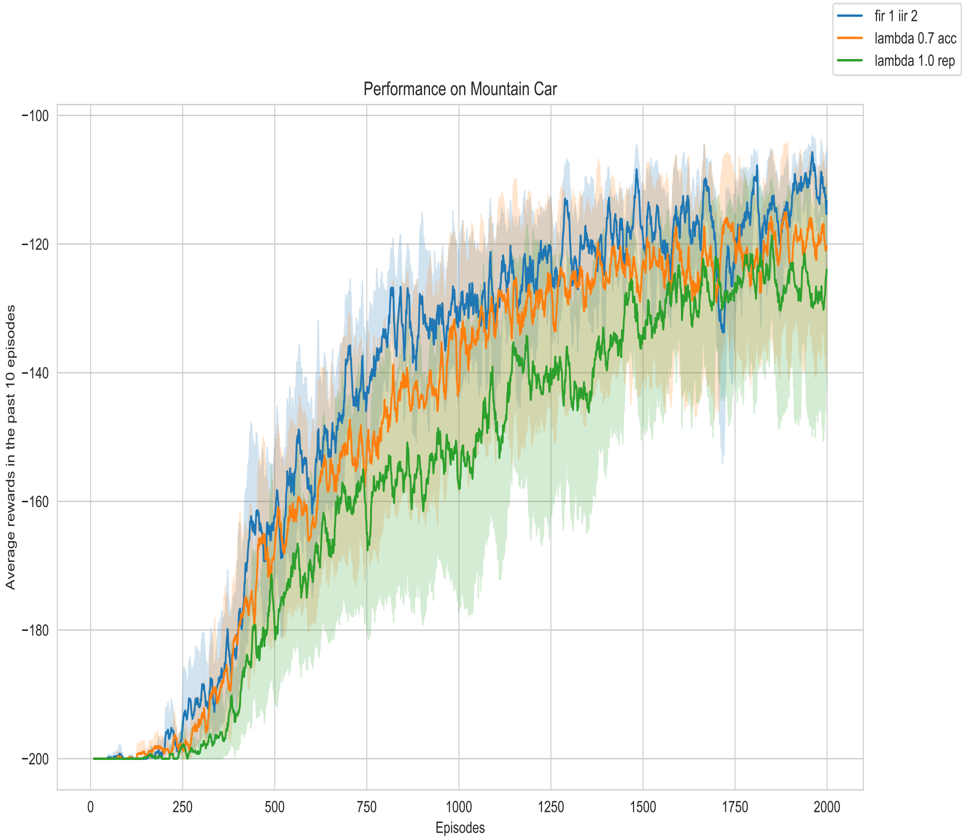


Figure 5.3: Performance of Filter traces against Accumulating traces and Replacing traces

In this case, we consider Mountain Car environment with a tile coding [72] function approximator in a control case to test the performance of the Filter traces. We set the total number of features to 3000 and number of tiles to 8. We considered various configurations, (m, n), of the filters and the coefficients of the filters were set as described in the section 5.3.1. The best filter configuration was found to be $(1, 2)$ among the ones tested from $\{(1, 1), (1, 0), (2, 1), (1, 2), (2, 2)\}$. We tested the performance on a range of learning rates in $\{0.05, 0.005, 0.01, 0.1, 0.01\}$ and we found that 0.05 performed the best over 10 different seeds. We also did hyperparameter tuning to find the best value of λ for the accumulating and replacing traces. We considered the values $\{0, 0.2, 0.4, 0.7, 1\}$ for λ and we found that $\lambda = 0.7$ performed the best for accumulating traces over 10 different seeds and $\lambda = 1$ performed the best for replacing traces. The optimal learning rate was found to be 0.05 in this case as well after testing it on the set of values mentioned earlier. The final performances were averaged over 10 different seeds and the mean of last 10 episodes are plotted. We use a confidence interval of 0.95 to plot the standard deviation in the returns. We used the same algorithm as discussed in Algorithm 7 but translated it to control case with SARSA learning.

As seen from the figure 5.3, we see an increase in the performance when we use filter traces with attention. This improvement is significant when compared against replacing traces and accumulating version. The reason for this is two fold. Firstly, it is due to the state dependent parameter which helps assign the credit efficiently. As we see from Figure 5.4 for a $(1, 2)$ filter, the filter coefficient α in the beginning is low forcing a multi-step bootstrapping. As the training proceeds the value increases for certain states forcing a TD style update. In the beginning the values are all noisy and hence a multi-step updating is desired. As the agent trains more, the estimates of all the states are learnt well, as a result, the agent could bootstrap from a value which is just 1 time-step ahead. Second reason for an increase in performance is due to the configuration of the filter. We have two feedbacks from the past in the filter traces. As a result, we are smoothing out the noise in the past trace which leads to better updates at the beginning of the training. As the training proceeds, the trace learns to pay more attention to the trace that is two time steps behind rather than the immediate past trace as opposed to accumulating traces and replacing traces where the immediate past trace is paid all the emphasis. More experiments similar to this can be performed but it is not considered for this thesis.

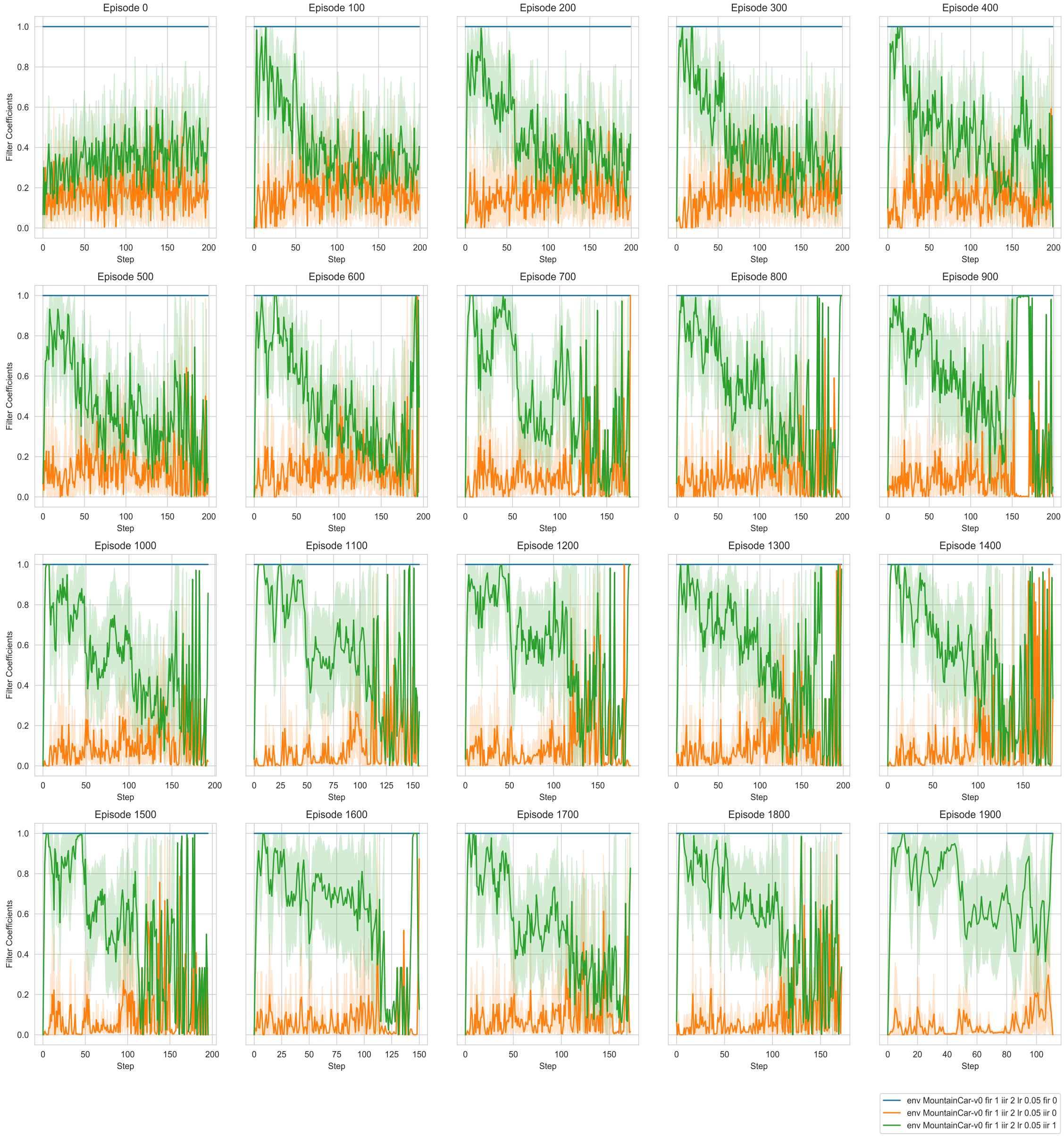


Figure 5.4: Behaviour of the filter coefficients over the episodes for (1,2) filter

5.5 Conclusion

In this chapter we unified various popular traces in reinforcement learning using IIR filters. We experimentally verified the effectiveness of such a unification on a toy chain and a mountain car. Finally, we proposed a method to find the filter coefficients and verified the application of that method experimentally. The experiments demonstrate the applicability of filters in reinforcement learning. Furthermore, several extensions can be made in reinforcement learning to include filters in different ways. These details are discussed in the next chapter.

6 Conclusion

Temporal credit assignment is one of the oldest and toughest problems in reinforcement learning. Temporal credit assignment is also one of the most important problems in several methods of reinforcement learning. In this thesis we introduced a couple of techniques that are aimed at assigning credit in a better manner. Specifically, we first introduced recurrent learning where the value function is computed recursively to reduce the variance in the estimates. Due to its formulation recurrent learning leads to a recursive traces that assign credit to states in the past. Later, we unify popular trace algorithms in reinforcement learning using filters. We show the importance of such a unification experimentally on a couple of environments.

Both the methods introduced can be explored further in several ways. Recurrent Learning has a state dependent parameter β which learns to emphasize a certain state. A state is emphasized if there is temporal inconsistency between the states in the past and the states in the future. In Reinforcement Learning, having access to a function quantifying the *interest* [40] of a state can be helpful. For example, one could decide to explore from those states, prioritize experience replay based on those states, and use β to set the λ to bootstrap from interesting states. We also believe β to be related to the concepts of bottleneck states [89] and reversibility.

Partial observability is common in real world. As demonstrated in Chapter 4, RVFs are able to correctly estimate the value of an aliased/noisy state using the trajectory's estimate. We believe that this is a promising area to explore because, as the experiments suggest, ignoring an uninformative state can sometimes suffice to learn its value function. This is in contrast to traditional POMDP methods which attempt to infer the belief state. Smoothing the output with a gating mechanism could also be useful for sequential tasks in Supervised Learning, such as regression or classification.

Natural Value Approximator(NVA) [104] shares a similarity with Recurrent Value Functions(RVF) in its formulation. RVFs are aimed to smooth the value function along the trajectory using time series techniques. However, NVAs are aimed to capture the sudden change in the value function along the trajectory. We believe many more methods could be explored in this space. One could find an algorithm which switches between RVF and NVA depending on the scenario.

Filters Chapter 5, on the other hand, opens up an opportunity to explore a range of algorithms in reinforcement learning. Specifically, one could use advanced filters to smooth out the value function. Smoothing operation achieved by RVFs can be viewed as having a low pass filter on value function, whereas capturing the sharpness in value function through NVAs can be viewed as having a high pass filter on the value function. One could use an adaptive filter on the value function to smooth the value function or to pass the sharp changes (high frequency component) when required.

Filters can also be used to update weights of a function approximator. We can maintain two sets of weights, one that is updated by a low-pass trace and the other updated by a high-pass trace. These weights correspond to having permanent and transient memory in reinforcement learning [73] where the long-term trends is captured by a low-pass weights (permanent weights) and a short-term trends or immediate changes are captured by a high-pass weights (transient weights). This setting could be useful in continual learning or life-long learning where the agent has to adapt to the rapid changes in environment.

The notion of having two sets of weights also relate to temporal hierarchy similar to Feudal learning in reinforcement learning [16, 93] where a master is operated at a longer time scale and a student is operated at a shorter time scale. The weights learned through a low-pass trace can be viewed as a master which operates at a longer time scale, whereas the weights learned through high-pass trace can be viewed as a student that is operated at a shorter time scale.

Overall, we believe the ideas presented in this thesis could give rise to many interesting research ideas in the future. The experiments presented in the thesis demonstrate the importance of temporal credit assignment in reinforcement learning. Also, the results indicate the effectiveness of these ideas. We hope that it motivates further work in this area.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. “Autonomous helicopter aerobatics through apprenticeship learning”. In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.

[2] Douglas Aberdeen. “Filtered reinforcement learning”. In: *European Conference on Machine Learning*. Springer. 2004, pp. 27–38.

[3] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[4] A Antonion. “Digital Filters: Analysis, Design and Application”. In: *McGraw Hill, New York* 5 (1993), p. 886896.

[5] Pierre-Luc Bacon. “Temporal Representation Learning”. PhD thesis. McGill University Libraries, 2018.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[7] Stefan Banach. “On operations in abstract sets and their application to ”. In: *Fund. math* 3.1 (1922), 133 to 181.

[8] Andrew G Barto, Richard S Sutton, and Charles W Anderson. “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[9] Richard Bellman. “A Markovian decision process”. In: *Journal of Mathematics and Mechanics* (1957), pp. 679–684.

[10] Christopher M Bishop. “Machine learning and pattern recognition”. In: *Information science and statistics. Springer, Heidelberg* (2006).

[11] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York, 1986.

[12] James Ward Brown et al. “Complex variables and applications.” In: (2009).

[13] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[14] Wesley Chung et al. “Two-Timescale Networks for Nonlinear Value Function Approximation”. In: (2018).

[15] Peter Dayan. “The convergence of TD (λ) for general λ”. In: *Machine learning* 8.3-4 (1992), pp. 341–362.

[16] Peter Dayan and Geoffrey E Hinton. “Feudal reinforcement learning”. In: *Advances in neural information processing systems*. 1993, pp. 271–278.

[17] John R Deller, John HL Hansen, and John G Proakis. “Discrete-time processing of speech signals”. In: (2000).

[18] Yue Deng et al. “Deep direct reinforcement learning for financial signal representation and trading”. In: *IEEE transactions on neural networks and learning systems* 28.3 (2016), pp. 653–664.

[19] Carlton Downey, Scott Sanner, et al. “Temporal Difference Bayesian Model Averaging: A Bayesian Perspective on Adapting Lambda.” In: *ICML*. Citeseer. 2010, pp. 311–318.

[20] Joseph Fourier. *Analytical Theory of Heat, by M. Fourier*. At Firmin Didot, mother and son, 1822.

[21] Roy Fox, Ari Pakman, and Naftali Tishby. “Taming the noise in reinforcement learning via soft updates”. In: *arXiv preprint arXiv:1512.08562* (2015).

[22] Jan Gläscher et al. “States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning”. In: *Neuron* 66.4 (2010), pp. 585–595.

[23] Bernard Gold and Charles M Rader. *Digital processing of signals*. Krieger Publishing Co., Inc., 1983.

[24] Geoffrey J Gordon. *Approximate solutions to Markov decision processes*. Tech. rep. Carneigie mellon university, Pittsburgh, School of computer science, 1999.

[25] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. “Variance reduction techniques for gradient estimates in reinforcement learning”. In: *Journal of Machine Learning Research* 5.Nov (2004), pp. 1471–1530.

[26] Simon Haykin and Barry Van Veen. *Signals and systems*. John Wiley & Sons, 2007.

[27] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Ronald A Howard. “Dynamic programming and markov processes.” In: (1960).

[29] Chia-Chun Hung et al. “Optimizing agent behavior over long time scales by transporting value”. In: *arXiv preprint arXiv:1810.06721* (2018).

[30] Andrew Ilyas et al. “Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?” In: *arXiv preprint arXiv:1811.02553* (2018).

[31] Sham Machandranath Kakade et al. “On the sample complexity of reinforcement learning”. PhD thesis. 2003.

[32] David W Kammler. *A first course in Fourier analysis*. Cambridge University Press, 2007.

[33] Nan Rosemary Ke et al. “Sparse attentive backtracking: Temporal credit assignment through reminding”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 7640–7651.

[34] Michael J Kearns and Satinder P Singh. “Bias-Variance Error Bounds for Temporal Difference Updates.” In: *COLT*. Citeseer. 2000, pp. 142–147.

[35] A Harry Klopf. *Brain function and adaptive systems: a heterostatic theory*. Tech. rep. Air force cambridge research labs hanscon AFB, MA, 1972.

[36] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[37] Ilya Kostrikov. *PyTorch Implementations of Reinforcement Learning Algorithms*. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr`. 2018.

[38] F.F. Kuo and J.F. Kaiser. *System Analysis by Digital Computer*. Wiley, 1966.

[39] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[40] A Rupam Mahmood et al. “Emphatic temporal-difference learning”. In: *arXiv preprint arXiv:1507.01569* (2015).

[41] Marvin Minsky. “Steps toward artificial intelligence”. In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30.

[42] Sanjit Kumar Mitra and Yonghong Kuo. *Digital signal processing: a computer-based approach*. Vol. 2. McGraw-Hill New York, 2006.

[43] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[44] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[45] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[46] James Moor. “The Dartmouth College artificial intelligence conference: The next fifty years”. In: *Ai Magazine* 27.4 (2006), pp. 87–87.

[47] Binoy B Nair et al. “Application of hybrid adaptive filters for stock market prediction”. In: *2010 International Conference on Communication and Computational Intelligence (INCOCCI)*. IEEE. 2010, pp. 443–447.

[48] Wayne Niblack et al. *An introduction to digital image processing*. Vol. 34. Prentice-Hall Englewood Cliffs, 1986.

[49] Alan V Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.

[50] Alan V Oppenheim and Ronald W Schafer. “Digital signal processing(Book)”. In: *Englewood Cliffs, N. J., Prentice-Hall, Inc., 1975. 598 p* (1975).

[51] Athanasios Papoulis. *The Fourier integral and its applications*. McGraw-Hill, 1962.

[52] TW Parks and CS Burrus. *Digital Filter Design. Topics in Digital Signal Processing*. John Wiley & Sons, New York, 1987.

[53] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: *NIPS Workshop*. 2017.

[54] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[55] Mark D Pendrith. *On reinforcement learning of control actions in noisy and non-Markovian domains.* citeseer.

[56] John Robinson Pierce and A Michael Noll. *Signals: The science of telecommunications.* Scientific American Library, 1990.

[57] Doina Precup. “Eligibility traces for off-policy policy evaluation”. In: *Computer Science Department Faculty Publication Series* (2000), p. 80.

[58] John G Proakis. *Digital signal processing: principles algorithms and applications.* Pearson Education India, 2001.

[59] Martin L Puterman. “Markov decision processes”. In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[60] Lawrence R Rabiner, Bernard Gold, and CK Yuen. *Theory and application of digital signal processing.* Prentice-Hall, 2007.

[61] Lawrence R Rabiner and Ronald W Schafer. *Digital processing of speech signals.* Vol. 100. Prentice-hall Englewood Cliffs, NJ, 1978.

[62] Richard A. Roberts and Clifford T. Mullis. *Digital Signal Processing.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[63] Joshua Romoff et al. “Reward estimation for variance reduction in deep reinforcement learning”. In: *arXiv preprint arXiv:1805.03359* (2018).

[64] Paul A Samuelson. “A note on measurement of utility”. In: *The review of economic studies* 4.2 (1937), pp. 155–161.

[65] Robert E Schapire and Manfred K Warmuth. “On the worst-case analysis of temporal-difference learning algorithms”. In: *Machine Learning* 22.1-3 (1996), pp. 95–121.

[66] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[67] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).

[68] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[69] John Schulman et al. “Trust region policy optimization”. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[70] Harm Seijen and Rich Sutton. “True online TD (lambda)”. In: *International Conference on Machine Learning*. 2014, pp. 692–700.

[71] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “Safe, multi-agent, reinforcement learning for autonomous driving”. In: *arXiv preprint arXiv:1610.03295* (2016).

[72] Alexander A Sherstov and Peter Stone. “Function approximation via tile coding: Automating parameter choice”. In: *International Symposium on Abstraction, Reformulation, and Approximation*. Springer. 2005, pp. 194–205.

[73] David Silver, Richard S Sutton, and Martin Müller. “Sample-based learning and search with permanent and transient memories”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 968–975.

[74] Satinder P Singh and Peter Dayan. “Analytical mean squared error curves in temporal difference learning”. In: *Advances in Neural Information Processing Systems*. 1997, pp. 1054–1060.

[75] Satinder P Singh and Richard S Sutton. “Reinforcement learning with replacing eligibility traces”. In: *Machine learning* 22.1-3 (1996), pp. 123–158.

[76] Richard S Sutton. “Generalization in reinforcement learning: Successful examples using sparse coarse coding”. In: *Advances in neural information processing systems*. 1996, pp. 1038–1044.

[77] Richard S Sutton. “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming”. In: *Machine Learning Proceedings 1990*. Elsevier, 1990, pp. 216–224.

[78] Richard S Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3.1 (1988), pp. 9–44.

[79] Richard S Sutton. “Temporal Credit Assignment in Reinforcement Learning.” In: (1985).

[80] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge, 1998.

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

[82] Richard S Sutton, A Rupam Mahmood, and Martha White. “An emphatic approach to the problem of off-policy temporal-difference learning”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2603–2631.

[83] Richard S Sutton and Satinder P Singh. “On step-size and bias in temporal-difference learning”. In: *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*. Citeseer. 1994, pp. 91–96.

[84] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[85] Csaba Szepesvári. “Algorithms for reinforcement learning”. In: *Synthesis lectures on artificial intelligence and machine learning* 4.1 (2010), pp. 1–103.

[86] Pierre Thodoroff et al. “Recurrent Value Functions”. In: *arXiv preprint arXiv:1905.09562* (2019).

[87] Pierre Thodoroff et al. “Temporal Regularization for Markov Decision Process”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 1782–1792.

[88] Philip S Thomas and Emma Brunskill. “Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines”. In: *arXiv preprint arXiv:1706.06643* (2017).

[89] Naftali Tishby and Daniel Polani. “Information theory of decisions and actions”. In: (2011), pp. 601–636.

[90] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033.

[91] John N Tsitsiklis. “Asynchronous stochastic approximation and Q-learning”. In: *Machine learning* 16.3 (1994), pp. 185–202.

[92] Grant V. Welland. “Z Transform Theory and Applications (Robert Vich)”. In: *Siam Review - SIAM REV* 31 (Mar. 1989).

[93] Alexander Sasha Vezhnevets et al. “Feudal networks for hierarchical reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3540–3549.

[94] Oriol Vinyals et al. “Starcraft II: A new challenge for reinforcement learning”. In: *arXiv preprint arXiv:1708.04782* (2017).

[95] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *arXiv preprint arXiv:1511.06581* (2015).

[96] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

[97] Rolf Weitkunat. *Digital biosignal processing*. Elsevier, 1991.

[98] Martha White and Adam White. “A greedy approach to adapting the trace parameter for temporal difference learning”. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2016, pp. 557–565.

[99] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[100] Ronald J Williams and David Zipser. “Gradient-based learning algorithms for recurrent networks and their computational complexity”. In: *Backpropagation: Theory, architectures, and applications* 1 (1995), pp. 433–486.

[101] RJ Willianms. “Toward a theory of reinforcement-learning connectionist systems”. In: *Technical Report NU-CCS-88-3, Northeastern University* (1988).

[102] Yuhuai Wu et al. “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation”. In: *Advances in neural information processing systems*. 2017, pp. 5279–5288.

[103] Zhongwen Xu, Hado van Hasselt, and David Silver. “Meta-Gradient Reinforcement Learning”. In: *arXiv preprint arXiv:1805.09801* (2018).

[104] Zhongwen Xu et al. “Natural Value Approximators: Learning when to Trust Past Estimates”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 2120–2128.

[105] Amy Zhang, Nicolas Ballas, and Joelle Pineau. “A dissection of overfitting and generalization in continuous reinforcement learning”. In: *arXiv preprint arXiv:1806.07937* (2018).