ADAPT: Alternating Dynamics Approximation and Policy opTimization

Student Number: 16012187[1]

MSc Computational Statistics and Machine Learning

Supervisor: Prof. Carlo Ciliberto

Submission date: 9th October 2024

[1]**Disclaimer:** This report is submitted as part requirement for the MY DEGREE at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

This paper introduces ADAPT, a model-based reinforcement learning algorithm inspired by the Optimism in the Face of Uncertainty (OFU) principle and designed for sample-efficient exploration of state-action spaces. ADAPT builds on policy gradient methods and implicitly incorporates optimistic planning, allowing it to explore without the need for environment-specific subroutines. We derive a tractable expression that integrates environment dynamics with policy iteration, and we analyze the algorithm's performance in both tabular and continuous settings. Our experimental results demonstrate ADAPT's efficacy in rapidly learning optimal policies across a diverse range of environments, including classical reinforcement learning tasks, deep exploration scenarios, and continuous state-space problems. This work bridges the gap between discrete and continuous state-action settings for OFU algorithms, contributing insights to the ongoing discourse on efficient exploration strategies in realistic reinforcement learning contexts.

Contents

Chapter 1

Introduction

Intelligent systems must quickly adapt and generalize in response to change and uncertainty within their environments. The success of their adaptation and learning strategies fundamentally depends on the quality of their internal representations. As Herbert Simon[1] aptly noted, "solving a problem simply means representing it in such a way that the solution becomes transparent."

Creating effective representations remains a fundamental challenge in artificial intelligence. This thesis addresses this challenge within the context of reinforcement learning (RL), a branch of machine learning focused on sequential decision-making problems. In RL, an agent interacts with an environment: at each timestep, it observes the current state, selects an action, and receives feedback in the form of a numerical reward. The agent's objective is to maximize its expected cumulative reward over time in an unknown environment through a process of trial and error. The complexity of this task becomes apparent when considering environments that are vast and intricate, such as the game of Go.[2] In such scenarios, reward signals may be sparse, necessitating that the agent's internal representations generalize effectively across diverse observations and multiple timescales. This ability to create robust, adaptable representations is crucial for the agent to navigate complex decision spaces and achieve optimal performance.

The RL problem (sketched below), is extremely general and RL algorithms have been successfully applied across a variety of different tasks including robotics,[3] game playing,[4] recommendation systems,[5] and autonomous driving.[6]

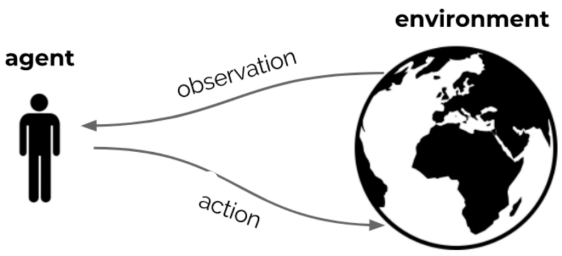


Figure 1.1: The reinforcement learning setting[7]

1.1 What to Learn, What to Approximate

In RL, there are many different choices of what to approximate- policies, value functions, dynamics models, or some combination thereof. This is in stark contrast with supervised learning, where one usually learns a mapping from inputs to outputs. RL presents a pair of orthogonal choices:

- What kind of objective to optimize? (this could be a policy, value function, or a dynamics model)
- What kind of function approximation should be used?

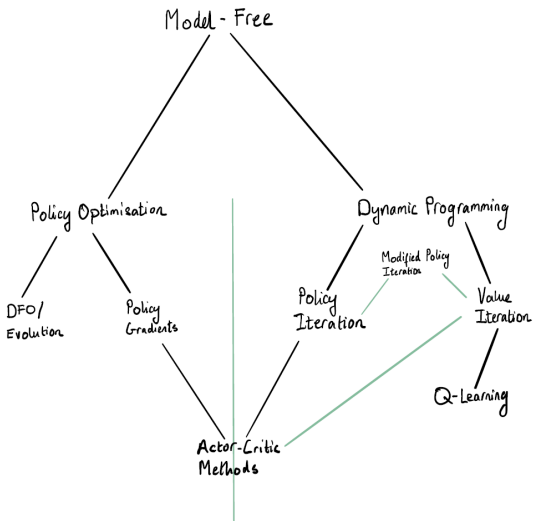


Figure 1.2: A model-free algorithm taxonomy

Model-free/ model-based algorithms are algorithms that are/ are not based on a dynamics model. In the figure above, we show a general taxonomy to describe the existing model-free algorithms. In the sections below, we develop these ideas further and discuss the challenge of drawing a similar taxonomy for model-based algorithms.

1.1.1 Model-Free Algorithms

Model-free algorithms learn policies or value functions without requiring an explicit model of the environment's dynamics.[8] These algorithms can be broadly divided into two main categories: policy optimization methods and approximate dynamic programming (ADP) methods.

Policy optimization methods focus on learning a policy, which is a function that maps an agent's state to actions. These methods view RL as an optimization problem, aiming to maximize the expected cumulative reward (or a surrogate objective) with respect to the policy parameters.[9] Two primary approaches within policy optimization are derivative-free optimization (DFO) and policy gradient methods. DFO algorithms, such as evolutionary strategies, optimize the policy by perturbing its parameters, evaluating performance, and adjusting in the direction of higher rewards.[10] Examples include the Cross-Entropy Method (CEM),[11] which uses a Gaussian distribution to

sample policy parameters and update the distribution based on high-performing samples; Natural Evolution Strategies (NES),[12] which improve search efficiency by incorporating gradient-based updates; and Covariance Matrix Adaptation Evolution Strategy (CMA-ES),[13] which adapts the covariance of the parameter distribution for better exploration in high-dimensional spaces. More advanced neuroevolution methods like HyperNEAT[14] also fall under this category, optimizing both the parameters and network topology of the policy.

Policy gradient methods, on the other hand, estimate the direction of policy improvement directly from interactions with the environment.[15] They do not require direct parameter perturbations, which allows them to optimize larger policies and handle continuous action spaces. However, they can be more challenging to implement and may struggle with long-horizon credit assignment. Popular policy gradient methods include REINFORCE,[15] which uses Monte Carlo estimates for policy gradients, and more advanced methods like Proximal Policy Optimization (PPO)[9] and Trust Region Policy Optimization (TRPO),[16] which employ surrogate objectives and trust regions, respectively, to ensure stable and efficient policy updates.

Approximate dynamic programming (ADP) methods aim to learn value functions, which estimate the cumulative reward expected from a given state or state-action pair.[17] These methods leverage the recursive structure of value functions, where the value of a state is linked to the values of subsequent states. Classic ADP algorithms include policy iteration and value iteration,[8] which alternate between evaluating a policy and improving it until convergence. A generalization of these approaches, called modified policy iteration,[18] combines elements of both.

In complex environments or continuous state spaces, ADP can be combined with function approximation techniques like neural networks. A prominent example is Q-Learning,[19] particularly its extension to deep learning, known as Deep Q-Networks (DQN),[20] which uses neural networks to approximate action-value functions. DQN has demonstrated success in domains such as Atari games, becoming a cornerstone for value-based RL research. Actor-critic methods combine elements of both policy optimization and ADP.[21] These methods simultaneously learn a policy (the "actor") and a value function (the "critic"). The critic aids the actor by providing feedback on the policy's performance, improving sample efficiency and reducing variance in policy gradient estimates. Advantage Actor-Critic (A2C/A3C)[22] uses an advantage function to refine policy gradient updates, while methods like Deep Deterministic Policy Gradient (DDPG)[23] extend actor-critic approaches to continuous action spaces.

1.1.2 Model-Based Algorithms

Model-based reinforcement learning (MBRL) typically approach this problem by building a "world model,"[24] which facilitates efficient learning, since the agent no longer needs to query the true environment for experience, and instead plans in the world model. MBRL present a unique challenge when it comes to creating a comprehensive taxonomy. Unlike value-based or policy gradient methods, which often have clearer categorizations, model-based reinforcement learning (MBRL) resists simple classification schemes.[25]

Firstly, these algorithms inherently involve multiple components - model learning, planning, and

often integration with model-free techniques. Each of these components can be implemented in various ways, leading to a combinatorial explosion of possible algorithm designs.[26] For example, PILCO[27] uses Gaussian processes for model learning, while PETS[28] employs ensemble dynamics models. This multifaceted nature contrasts with the more unified focus of value-based methods (primarily estimating value functions) or policy gradient approaches (directly optimizing policies).[8] Furthermore, the boundaries between these components are often blurred in practice. Many algorithms feature tight integration between model learning and planning, with each process informing and guiding the other.[26] For instance, ME-TRPO[29] uses the planning process to focus model learning on the most relevant parts of the state space, while MBPO[30] adapts its planning approach based on the current confidence in different parts of the learned model. This interplay makes it challenging to create mutually exclusive classifications.

In order to learn a world model that accurately represents the dynamics of the environment, the agent must collect data that is rich in experiences[31] . However, for faster convergence, data collection must also be performed efficiently, wasting as few samples as possible.[32] Thus, the effectiveness of MBRL algorithms hinges on the exploration-exploitation dilemma. The exploration-exploitation dilemma has been extensively studied in reinforcement learning, particularly in the context of Markov Decision Processes (MDPs) with finite states and actions. A key principle that emerged from this research is "Optimism in the Face of Uncertainty" (OFU),[33,34] which prioritizes actions with potentially high rewards and high uncertainty. This principle has been crucial in developing effective algorithms for tabular settings.[4]

However, the transition from simple tabular to more complex reinforcement learning settings has not fully capitalized on these theoretical advances. Many approaches in larger state spaces rely on simpler heuristics[35] or basic exploration strategies like epsilon-greedy.[8] This gap between theory and practice in more complex environments can be attributed to several factors.

Firstly, many OFU-based algorithms involve intricate procedures for computing and maintaining uncertainty estimates. These procedures, while theoretically sound, often become computationally intractable or lose their guarantees when scaled to high-dimensional or continuous state spaces. For instance, algorithms like UCRL2[36] require the computation of confidence bounds that become infeasible for large or continuous MDPs.

Furthermore, the direct application of OFU principles to settings with function approximation is not straightforward. The methods for quantifying uncertainty in tabular settings do not naturally extend to more complex function approximators, creating a significant challenge in adapting these algorithms to more complex environments.

The development of a unified framework that incorporates OFU principles effectively across both tabular and function approximation settings remains an open challenge in the field. While progress has been made in adapting some concepts to more complex environments,[37] a generalized approach that maintains the theoretical strengths of OFU while being computationally feasible in larger state spaces is still an active area of research. This gap presents an opportunity for future work to bridge the theoretical foundations of optimistic planning with the practical needs of modern reinforcement learning applications. Addressing this challenge could potentially lead to more efficient and

theoretically grounded exploration strategies in complex, high-dimensional environments, without necessarily requiring the full machinery of deep learning approaches.

1.2 Objectives and Outline

The main goal of this thesis is to address this gap in the literature and by developing an algorithm holistically uses the learned approximations of the environment dynamics within a tractable expression to directly optimise the the policy parameters using gradient ascent. The success of this project will hinge on our algorithms ability to learn good representations of the environment. In order to get there, Chapter 2 focusses on reviewing some technical aspects of the Markov Decision Processes (MDP) and existing work on Optimistic Planning. In Chapter 3, we introduce our own approach, the ADAPT algorithm, and in Chapter 4, empirically compare results across a range of environments.

The code for our work is available here[1].

[1] `https://github.com/shomit505/ADAPT-Alternating-Dynamics-Approximation-and-Policy-opTimization`

Chapter 2

Preliminaries

A Markov Decision Process (MDP) is a mathematical object that describes an agent interacting with a stochastic environment. It is defined by the following components:

- $\mathcal{S}$: **state space**, a set of states of the environment.
- $\mathcal{A}$: **action space**, a set of actions, which the agent selects from at each timestep.
- $\mathcal{P}(r, s' \mid s, a)$: a **transition probability distribution**. For each state s and action a, $\mathcal{P}$ specifies the probability that the environment will emit reward r and transition to state s'.

In certain problem settings, we will also be concerned with an **initial state distribution** $v_0(s)$, which is the probability distribution that the initial state s_0 is sampled from. Various different definitions of MDP are used throughout the literature. Sometimes, the reward is defined as a deterministic function $R(s)$, $R(s, a)$, or $R(s, a, s')$. These formulations are equivalent in expressive power. That is, given a deterministic-reward formulation, we can simulate a stochastic reward by lumping the reward into the state. In this work, we will define our deterministic reward function as $R(s, a)$.

The end goal is to find a **policy**, $\pi : \mathcal{S} \to Dist(\mathcal{A})$, which maps states to actions. We will consider stochastic policies, which are conditional distributions $\pi(a \mid s)$, though elsewhere in the literature, one frequently sees deterministic policies $a = \pi(s)$. For convenience, we develop our ideas assuming discrete state and action sets unless otherwise noted.

ASSUMPTION 1: The reward function is bounded:

$$\forall s \in \mathcal{S}, a \in \mathcal{A} : \max_{s,a} |r(s, a)| = \kappa < \infty \quad \text{for some scalar } \kappa.$$

We make this assumption to allow us to construct tractable algorithms based on maximising this reward.

In discounted problems, the value function of a policy π for a state s $\in \mathcal{S}$ is defined as the expected sum of discounted rewards:

$$v_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \middle| S_0 = s\right] \tag{2.1}$$

where $\gamma \in [0, 1)$ is the discount factor. In this formulation, one intuitive interpretation of the discount factor is that it helps us model problems where it may be necessary to express the fact that rewards accrued far in the future may be worth less than in the immediate time. When $\gamma = 0$ for example, the value of a state is simply the expected immediate reward.

The idea of using a discount factor to force convergence in an otherwise divergent series is generally known as *discounting* or *geometric discounting*.[8] In the context of mathematical series or sequences, this approach introduces a factor such as γ^t, where $0 \leq \gamma < 1$, to make the sum converge. This technique is often referred to as *discounted summation* or *exponential discounting*. The purpose of discounting is to weight down future values, reducing their impact on the total sum or utility over time. This concept is widely used in various fields, including economics, control theory, and reinforcement learning. In economics, terms like *discounted present value* or *present value analysis* are commonly used to describe this practice.[38] In reinforcement learning, discounting is directly applied to formulate the *discounted cumulative reward*, which helps ensure stability in learning algorithms by shaping the agent's focus between short-term and long-term rewards.[8] Thus, discounting serves both a theoretical purpose, by guaranteeing convergence of series, and a practical purpose, by influencing decision-making strategies in different contexts.

While the infinite summation in (2.1) may indicate that we are always working with problems with an *infinite horizon*, we can also think about the discounted setting as a *finite horizon* problem where the horizon length is distributed according to a geometric distribution.[39]

LEMMA 1 *The expected sum of discounted rewards is equal to the following undiscounted formulation:*

$$v_\pi(s) = \mathbb{E}\left[\mathbb{E}\left[\sum_{t=0}^{T-1} r(S_t, A_t) \middle| S_0 = s\right]\right],$$

where the length of the horizon T is a random variable, drawn from a geometric distribution with parameter γ.

PROOF:

This proof is taken from (Puterman, 1994, proposition 5.3.1).[39] Working under Assumption (1) and that $\gamma < 1$, we can interchange the order of summation in the following expression:

$$
\begin{aligned}
\mathbb{E}\left[\sum_{t=0}^{\infty} r(S_t, A_t) \middle| S_0 = s\right] &= \mathbb{E}\left[\sum_{n=0}^{\infty}(1-\gamma)\gamma^n \sum_{t=0}^{n} r(S_t, A_t) \middle| S_0 = s\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} r(S_t, A_t) \sum_{n=t}^{\infty}(1-\gamma)\gamma^n \middle| S_0 = s\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \sum_{n=0}^{\infty}(1-\gamma)\gamma^n \middle| S_0 = s\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \middle| S_0 = s\right] \\
&= v_\pi(s)
\end{aligned}
$$

□

where we interchanged the indices so as to maintain the ordering $0 \leq t \leq n-1 < \infty$. The penultimate step follows from the fact that the rightmost series is a geometric series converging to $\frac{1}{1-\gamma}$.

2.1 Bellman Expectation Equations

The Bellman equations are fundamental recursive relationships in dynamic programming and reinforcement learning. They express the value of a state (or state-action pair) as the expected return of the immediate reward plus the discounted value of the next state. We can explicitly derive them from our formulation for $v_\pi(s)$ using the law of iterated expectations.

LEMMA 2 (Law of Iterated Expectations) *Let X and Y be two random variables. Then, the expectation of X can be expressed as the expectation of the conditional expectation of X given Y:*

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$$

PROOF:

When a joint probability density function is well defined and the expectations are integrable, we write for the general case:

$$
\begin{aligned}
\mathbb{E}(X) &= \int x \Pr[X = x]\, dx \\
&= \int \left(\int x \Pr[X = x \mid Y = y]\, dx \right) \Pr[Y = y]\, dy \\
&= \int \int x \Pr[X = x, Y = y]\, dx\, dy \\
&= \mathbb{E}[\mathbb{E}[X \mid Y]]
\end{aligned}
$$

□

By unrolling the infinite sum of rewards in $v_\pi(s)$, we can obtain a recursive expression for the value of a state:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(S_t, A_t)\middle| S_0 = s\right] \\
&= \mathbb{E}\left[r(S_0, A_0) + \sum_{t=1}^{\infty}\gamma^t r(S_t, A_t)\middle| S_0 = s\right] \\
&= \mathbb{E}\left[r(S_0, A_0) + \sum_{t=0}^{\infty}\gamma^{t+1} r(S_{t+1}, A_{t+1})\middle| S_0 = s\right] \\
&= \mathbb{E}\left[r(S_0, A_0) + \gamma\sum_{t=0}^{\infty}\gamma^t r(S_{t+1}, A_{t+1})\middle| S_0 = s\right] \\
&= \mathbb{E}\left[r(S_0, A_0) + \gamma\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(S_t, A_t)\middle| S_1\right]\middle| S_0 = s\right] \\
&= \mathbb{E}[r(S_0, A_0) + \gamma v_\pi(S_1)|S_0 = s],
\end{aligned}$$

We can then expand this expectation by marginalizing over A_0 and S_1 conditioned on some initial state $S_0 = s$ and obtain the Bellman equation:

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

We use the notation $Q_\pi(s,a)$ for the expected sum of discounted rewards from a designated state and action, to obtain the state-action version of the Bellman equation, which also admits a recursive form:

$$\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(S_t, A_t)\middle| S_0 = s, A_0 = a\right] \\
&= r(s,a) + \gamma\sum_{s'}P(s'|s,a)v_\pi(s') \qquad (\star\star)
\end{aligned}$$

The Bellman expectation equations are given by equations ($\star$) and ($\star\star$). It is worth taking a moment to acknowledge the simple yet powerful implications of these two equations. Given a policy π and an MDP, the Bellman equations present a pathway to finding the association value function v_π by solving ($\star$). In order to do so, let us first rewrite ($\star$) in a more convenient matrix form by defining some useful quantities:

DEFINITION 3 (Markov Reward Process) [40]
A Markov Reward Process is a tuple $\langle \mathcal{S}, \mathcal{P}_\pi, \mathcal{R}_\pi, \gamma \rangle$*, where:*

- $\mathcal{S}$ *is a finite set of states.*
- $\mathcal{P}_\pi$ *is the state transition probability matrix, given by*

$$\mathcal{P}_\pi(s, s') = \mathbb{P}[S_{t+1} = s' \mid S_t = s] = \sum_a \pi(a \mid s) P(s' \mid s, a).$$

- $\mathcal{R}_\pi$ *is the reward function,*

$$\mathcal{R}_\pi(s) = \mathbb{E}[R_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) r(s, a)$$

- γ *is the discount factor, where* $0 \leq \gamma < 1$*.*

Firstly, let us note that $\mathcal{R}_\pi \in \mathbb{R}^{|S|}$ and $\mathcal{P}_\pi \in \mathbb{R}^{|S| \times |S|}$. These equations do not involve actions explicitly because the policy π has been marginalised out inside of the inside $\mathcal{R}_\pi$ and $\mathcal{P}_\pi$ terms. $\mathcal{P}_\pi$ is now simply defining the dynamics of a Markov chain where the actions have been abstracted away.

The Bellman expectation equation ($\star$) now amounts to the statement:

$$v_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi v_\pi,$$

where the value function for a policy is now seen as a vector $v_\pi \in \mathbb{R}^{|S|}$. Rearranging, we can obtain an equivalent expression:

$$v_\pi = r_\pi + \gamma \mathcal{P}_\pi v_\pi \iff v_\pi - \gamma \mathcal{P}_\pi v_\pi = \mathcal{R}_\pi \iff (I - \gamma \mathcal{P}_\pi) v_\pi = \mathcal{R}_\pi.$$

We observe that this describes a linear system of equations "$Ax = b$". When solving the policy evaluation problem, v_π is seen as our unknown "x" which we solve for by forming:

$$v_\pi = (I - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi.$$

The existence of a solution to the policy evaluation problem thus hinges on the non-singularity of matrix $I - \gamma P_\pi$.

DEFINITION 4 (Spectral Radius) *Let $\mathbf{A} \in \mathbb{C}^{n\times n}$ be a square matrix with eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$. The spectral radius of $\mathbf{A}$, denoted $\rho(\mathbf{A})$, is defined as:*

$$\rho(\mathbf{A}) = \max_{1\leq i\leq n} |\lambda_i|.$$

Equivalently, by Gelfand's formula,[41] the spectral radius can be expressed as:

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \|\mathbf{A}^k\|^{1/k},$$

where $\|\cdot\|$ denotes any matrix norm.

We observe that for any sub-multiplicative matrix norm, we have $\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$ for matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$. Applying this recursively:

$$\|\mathbf{A}^k\| \leq \|\mathbf{A}\|^k$$

Gelfand's formula states that $\rho(\mathbf{A}) = \lim_{k\to\infty} \|\mathbf{A}^k\|^{1/k}$. Therefore:

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \|\mathbf{A}^k\|^{1/k} \leq \lim_{k\to\infty} (\|\mathbf{A}\|^k)^{1/k} = \|\mathbf{A}\|$$

LEMMA 5 *When the discount factor satisfies $0 \leq \gamma < 1$, the inverse of $\mathbb{I} - \gamma\mathcal{P}_\pi$ exists and:*

$$(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1} = \sum_{t=0}^{\infty} (\gamma\mathcal{P}_\pi)^t. \tag{2.2}$$

PROOF:

> This follows directly from Corollary C.4 in Putterman (1994)[39] which shows that $(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1}$ exists only when $\rho(\gamma\mathcal{P}_\pi) < 1$. Because $\mathcal{P}_\pi$ is a stochastic matrix,[42] $\sum_{s'} \mathcal{P}_\pi(s, s') = 1$ for any state s so $\rho(\gamma\mathcal{P}_\pi) \leq \|\gamma\mathcal{P}_\pi\| < 1$. □

LEMMA 6 (Inverse Sum Lemma) *For a stochastic matrix $\mathcal{P}_\pi$ and discount factor $\gamma \in [0, 1)$, the following equality holds:*

$$(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1} = \sum_{t=0}^{\infty} (\gamma\mathcal{P}_\pi)^t. \tag{2.3}$$

PROOF:

> We begin by establishing the existence of $(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1}$. According to Corollary C.4 in Putterman (1994),[39] this inverse exists if and only if $\rho(\gamma\mathcal{P}_\pi) < 1$, where $\rho(\cdot)$ denotes the spectral radius. Since $\mathcal{P}_\pi$ is a stochastic matrix,[42] we have
>
> $$\sum_{s'\in\mathcal{S}} \mathcal{P}_\pi(s, s') = 1 \quad \text{for any state } s,$$

implying $|\mathcal{P}_\pi|_\infty = 1$. Consequently, $\rho(\gamma\mathcal{P}_\pi) \leq |\gamma\mathcal{P}_\pi|_\infty = \gamma|\mathcal{P}_\pi|_\infty = \gamma < 1$.[43] Thus, $(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1}$ exists.

Next, we consider the convergence of $\sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t$. This geometric series of matrices converges if $|\gamma\mathcal{P}_\pi| < 1$ for any matrix norm.[44] We have already shown that $|\gamma\mathcal{P}_\pi|_\infty = \gamma < 1$, ensuring the sum's convergence.

To complete the proof, we demonstrate that $\mathbf{S} = \sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t$ satisfies $(\mathbb{I} - \gamma\mathcal{P}_\pi)\mathbf{S} = \mathbb{I}$:

$$
\begin{aligned}
(\mathbb{I} - \gamma\mathcal{P}_\pi)\mathbf{S} &= (\mathbb{I} - \gamma\mathcal{P}_\pi)\sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t && \text{(Expanding } \mathbf{S}\text{)} \\
&= \sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t - \gamma\mathcal{P}_\pi\sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t && \text{(Distributing } (\mathbb{I} - \gamma\mathcal{P}_\pi)\text{)} \\
&= \sum_{t=0}^{\infty}(\gamma\mathcal{P}_\pi)^t - \sum_{t=1}^{\infty}(\gamma\mathcal{P}_\pi)^t && \text{(Shifting index in second sum)} \\
&= \mathbb{I} + \sum_{t=1}^{\infty}(\gamma\mathcal{P}_\pi)^t - \sum_{t=1}^{\infty}(\gamma\mathcal{P}_\pi)^t && \text{(Separating } \mathbb{I} \text{ from first sum)} \\
&= \mathbb{I} && \text{(Cancelling remaining terms)}
\end{aligned}
$$

This confirms that $\mathbf{S}$ is indeed $(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1}$, concluding the proof. □

In this expression, each row of $\mathcal{P}_\pi^t$ (the t-th power of $\mathcal{P}_\pi$) contains the distribution over the next states t steps in the future. In other words, if we choose a row s and column s', then $\mathcal{P}_\pi^t(s, s') \triangleq P(S_t = s'|S_0 = s)$: the probability that the process is in state s' t steps into the future if it started from state s. Hence, when taking the sum to infinity in (2.3), we are effectively marginalizing over all possible paths between any two states. The entries of $(I - \gamma\mathcal{P}_\pi)^{-1}$ then have the following meaning:

$$
(I - \gamma\mathcal{P}_\pi)^{-1}(s, s') = \sum_{t=0}^{\infty}\gamma^t\mathcal{P}_\pi^t(S_t = s'|S_0 = s). \tag{2.4}
$$

2.1.1 Solving the Bellman Expectation Equations Iteratively

Our analysis in the previous section showed that when the discount factor is strictly less than 1, the value function of a policy is given by

$$
v_\pi = (I - \gamma\mathcal{P}_\pi)^{-1}\mathcal{R}_\pi.
$$

A direct approach for computing this expression may consist in first inverting the matrix explicitly and then computing the matrix-vector product with $\mathcal{R}_\pi$. This approach, however, is traditionally avoided in practice because of numerical instabilities.[45] A better solution would be to solve for v_π through a QR factorization[45,46]) of $I - \gamma\mathcal{P}_\pi$ without ever computing the inverse explicitly. While gaining in numerical stability, we note that going through the QR factorization step does not lead to substantive computational savings since it would require $O(|S|^3)$ in time, as is also the case for matrix inversion.

An alternative to the direct approach is to iteratively improve an approximation to the solution based on an operator-theoretic point of view on the Bellman expectation equations ($\star$) and ($\star\star$). In fact, we can define a linear operator $T_\pi : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ whose effect on any $v \in \mathbb{R}^{|S|}$ is the following:

$$T_\pi v = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi v.$$

The value function v_π corresponding to the policy π then satisfies $T_\pi v_\pi = v_\pi$; i.e., v_π is the fixed point of T_π. In our previous discussion, we saw that a solution to the Bellman expectation equations exist when $\rho(\gamma \mathcal{P}_\pi) < 1$. Under the operator point of view, this discussion now translates into an inquiry on the convergence of T_π to a fixed-point, which can be shown using the Banach fixed-point theorem.[47]

DEFINITION 7 (Contraction mapping) *Let $\mathcal{U}$ be a Banach space. The operator $T : \mathcal{U} \to \mathcal{U}$ is a contraction mapping if for any $u, v \in \mathcal{U}$,*

$$\|Tu - Tv\| \leq \alpha \|u - v\|$$

for some $0 < \alpha < 1$.

DEFINITION 8 (Uniform Convergence) *A sequence $\{v_k\}$ is said to converge to v if:*

$$\lim_{k \to \infty} \|v_k - v\| = 0.$$

THEOREM 9 (Banach Fixed-Point Theorem) *Let $\mathcal{U}$ be a Banach space. If $T : \mathcal{U} \to \mathcal{U}$ is a contraction mapping on $\mathcal{U}$, then:*

1. *There exists a unique fixed point $v^* \in \mathcal{U}$ such that $Tv^* = v^*$.*
2. *v^* can be found in the limit of the sequence defined by $v_{k+1} = Tv_k$, where v_0 is arbitrary.*

The Bellman expectation equations allow us to evaluate the value of a policy (i.e. policy evaluation). Casting the policy evaluation problem as an instance of fixed-point methods has the obvious advantage of providing us with a template for designing new iterative policy evaluation algorithms. Indeed, the second statement of the theorem suggests that the repeated application of the policy evaluation operator T_π converges to the value function v_π for any initial guess $v_\pi^{(0)}$. But in order for all of this to hold, we need to establish that T_π is indeed a contraction mapping.

LEMMA 10 *T_π is a contraction mapping.*

PROOF:

Applying the definition of a contraction and because $\mathcal{P}_\pi$ is stochastic:

$$\|T_\pi v - T_\pi v'\| = \|\gamma \mathcal{P}_\pi (v - v')\| \leq \gamma \|\mathcal{P}_\pi\| \|v - v'\| = \gamma \|v - v'\|,$$

Hence, T_π is a contraction if the discount factor is $0 \leq \gamma < 1$. □

Algorithm 1 Iterative Policy Evaluation

Require: A stationary policy π (deterministic or randomized), and discounted MDP
Pre-compute:
$r_\pi \in \mathbb{R}^{|S|}$, where $r_\pi(s) = \sum_a \pi(a|s) r(s,a)$
$\mathcal{P}_\pi \in \mathbb{R}^{|S| \times |S|}$, where $\mathcal{P}_\pi(s,s') = \sum_a P(s'|s,a)\pi(a|s)$
Initialize:
$v \leftarrow 0 \in \mathbb{R}^{|S|}$
while a fixed number of steps is not reached, or other exit condition (e.g., using a bound) **do**
    $v \leftarrow r_\pi + \gamma \mathcal{P}_\pi v$
end while
return v

The fixed-point iteration scheme based on (T_π) can be efficiently implemented using the vector form of the policy evaluation equations, as illustrated in Algorithm 1. This approach leverages dense matrix operations, which are particularly well-suited for modern computer architectures and GPU hardware.[48–50] Such vector-based implementations tend to yield superior performance compared to their scalar counterparts, due to the ability of modern processors to exploit data parallelism and memory hierarchy effectively.[51]

2.2 Bellman Optimality Equations

DEFINITION 11 (Optimal Policy) *A policy π^* is optimal when it means its value function is such that for any other $\pi \neq \pi^*$ and for any state $s \in \mathcal{S}$, $v_{\pi^*}(s) \geq v_\pi(s)$.*

The optimal value function v^* (the value function associated with an optimal policy) satisfies a nonlinear system of equations that we called the Bellman Optimality Equations:

$$v^* = \max_{\pi \in \Pi^D} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v^*),$$

where Π^D denotes the set of deterministic Markov policies. We can also write these equations in component form:

$$\begin{aligned} v^*(s) &= \max_{\pi \in \Pi^D} \left(r(s,\pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) v^*(s') \right) \\ &= \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) v^*(s') \right). \end{aligned}$$

It has been shown by Putterman (1994)[39] that in a finite discounted MDP, there exists at least one optimal stationary deterministic policy that satisfies the Bellman optimality equations. A natural question one might have after looking at this is that are we "losing information" in some sense by restricting our search for optimal policies to the set of deterministic Markov policies; more precisely: is the maximum value achievable by stationary deterministic policies the same as the maximum value achievable by stationary randomized policies?

The following proposition (adapted from Puterman, 1994, proposition 6.2.1)[39] shows us that this statement is in fact true.

PROPOSITION 12

$$\max_{\pi \in \Pi^D} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v) = \max_{\pi \in \Pi^R} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v),$$

where Π^D *and* Π^R *denote the class of stationary deterministic and randomized policies respectively.*

PROOF:

Pick any randomized policy $\pi \in \Pi^{\mathrm{MR}}$. Because $\pi(\cdot \mid s)$ is a conditional distribution over actions:

$$\begin{aligned}
&\max_a \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v(s') \right) && \text{Maximum value achievable} \\
&= \sum_a \pi(a \mid s) \max_a \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v(s') \right) && \text{Weighted sum of maximum} \\
&\geq \sum_a \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v(s') \right) && \text{Jensen's inequality}
\end{aligned}$$

The first equality holds because $\sum_a \pi(a \mid s) = 1$ for any probability distribution. The inequality follows from the fact that the maximum is always greater than or equal to any specific value.

This implies that:

$$\max_{\pi \in \Pi^{\mathrm{D}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v) \geq \max_{\pi \in \Pi^{\mathrm{R}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v)$$

The inequality in the other direction follows from the fact that the class of randomized policies is strictly larger than the class of deterministic ones:

$$\max_{\pi \in \Pi^{\mathrm{D}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v) \leq \max_{\pi \in \Pi^{\mathrm{R}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v)$$

Combining these inequalities, we conclude:

$$\max_{\pi \in \Pi^{\mathrm{D}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v) = \max_{\pi \in \Pi^{\mathrm{R}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v)$$

Thus, searching in the space of deterministic policies does not involve any loss of optimality. □

Thus, the restriction to deterministic policies does not affect the question of optimality and simply eases the development of control algorithms: algorithms for finding optimal policies.

2.2.1 Solving the Bellman Optimality Equations Iteratively

The stationary deterministic policy π^* that selects actions according to:

$$\pi^*(s) := \arg\max_a \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) v^*(s') \right),$$

is in fact called the *greedy policy* and can be obtained after having found v^* as the fixed point of the Bellman optimality operator T^*:

$$T^* v := \max_{\pi \in \Pi^{\mathrm{D}}} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v).$$

The optimal value function satisfies $T^* v^* = v^*$. After having established that T^* is a contraction mapping, we will be able to leverage the Banach Fixed Point Theorem to find v^* as the fixed point in $\lim_{t \to \infty} (T^*)^t v = v^*$. The resulting fixed-point iteration algorithm based on T^* is called *value iteration*.[52]

LEMMA 13 (Bellman Optimality operator is a contraction mapping) *For any two value functions v and v':*

$$\|T^* v - T^* v'\| \leq \gamma \|v - v'\|,$$

where T^* is the Bellman optimality operator and γ<1

PROOF:

We note that for any $Q : S \times A \to \mathbb{R}$ associated with a value function $v : S \to \mathbb{R}$ and for any $Q' : S \times A \to \mathbb{R}$ corresponding to a $v' : S \to \mathbb{R}$:

$$\left| \max_a Q(s,a) - \max_a Q'(s,a) \right| \leq \max_a |Q(s,a) - Q'(s,a)|$$

$$= \gamma \max_a \sum_{s'} P(s' \mid s,a) \, |v(s') - v'(s')|$$

$$\leq \gamma \max_s |v(s) - v'(s)| \, .$$

By the definition of a contraction, it follows under the infinity norm that:

$$\|T^* v - T^* v'\| = \max_s \left| \max_a Q(s,a) - \max_a Q'(s,a) \right| \leq \gamma \|v - v'\|.$$

□

The value iteration algorithm follows the same template as the iterative policy evaluation procedure previously shown in Algorithm 1. However, this time we are only given an MDP as an input because our goal is to output an optimal policy.

Algorithm 2 Value Iteration

Require: A discounted MDP
Ensure: The optimal state-action value function Q^* for this MDP.
1: **Initialize:**
2: $Q^* \leftarrow 0, Q^* \in \mathbb{R}^{|S|\times|A|}$
3: **repeat**
4: **for all** $s \in S, a \in A$ **do**
5: $Q^*(s,a) \leftarrow r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_a Q^*(s',a)$
6: **end for**
7: **until** a fixed number of steps is reached, or other exit condition (Puterman, 1994)

The key observation is that this algorithm gives us the optimal state-action values associated with the optimal policy π^* but not the optimal policy itself. Rather than propagating the value only for the next states, the policy iteration algorithm solves the Bellman expectation equations for the current candidate optimal policy before proceeding to a maximization step (a greedification step).[8] In policy iteration, the role of the value function is to support the search of an optimal policy.

Algorithm 3 Policy Iteration

Require: A discounted MDP
Ensure: An optimal deterministic stationary policy
1: **Initialize:** π_0 arbitrarily
2: $k \leftarrow 0$
3: **repeat**
4: **Policy Evaluation:**
5: $v_{\pi_k} \leftarrow (I - \gamma \mathcal{P}_{\pi_k})^{-1} \mathcal{R}_{\pi_k}$
6: **Policy Improvement:**
7: $\pi_{k+1} \leftarrow_{\pi \in \Pi^D} (\mathcal{R}_\pi + \gamma \mathcal{P}_\pi v_{\pi_k})$
8: $k \leftarrow k + 1$
9: **until** $\pi_{k+1} = \pi_k$

2.3 Policy Gradient Methods

Policy gradient methods are analogous to policy iteration in the Bellman expectation equations setting as in both cases the search for an optimal policy π^* is guided by a policy evaluation procedure. The fundamental difference is that instead of representing the optimal deterministic (greedy) policy thorugh a value function, policy gradient methods operate within a designated family of policies. The advantage of this approach is the additional flexibility available to us in the modelling effort, it allows the practitioner to express prior knowledge about the optimal policy. Naturally, the trade-off is that the policy may be misspecified; i.e. the optimal policy is not contained in the choice of parameterised policy. Although both policy gradient methods and policy iteration use a value function to move towards the optimal policy, policy gradient methods perform their improvement step using the gradient of some objective w.r.t. to the parameters of the policy.

2.3.1 Vanilla Policy Gradients (REINFORCE)

Informally, policy gradient methods aim to optimize a policy by iteratively estimating the gradient of its expected performance with respect to its parameters. A common approach is to use a score function gradient estimator. Let $\mathbf{x}$ be a random variable with probability density $p(\mathbf{x} \mid \boldsymbol{\theta})$, and let $f(\mathbf{x})$ be a scalar function (e.g., a reward). To compute $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})]$, we derive:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})] &= \nabla_{\boldsymbol{\theta}} \int d\mathbf{x}\, p(\mathbf{x} \mid \boldsymbol{\theta}) f(\mathbf{x}) \\
&= \int d\mathbf{x}\, p(\mathbf{x} \mid \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x} \mid \boldsymbol{\theta}) f(\mathbf{x}) \\
&= \mathbb{E}_{\mathbf{x}} \left[f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x} \mid \boldsymbol{\theta}) \right].
\end{aligned}$$

This estimator suggests sampling $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \sim p(\mathbf{x} \mid \boldsymbol{\theta})$ and estimating the gradient $\hat{\mathbf{g}}$ as:

$$\hat{\mathbf{g}} = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}_i \mid \boldsymbol{\theta}) f(\mathbf{x}_i).$$

To apply this, we require a stochastic policy that, at each state s, specifies a distribution over actions, $\pi(\mathbf{a} \mid s)$. Including the parameter vector $\boldsymbol{\theta}$, we denote this distribution as $\pi_{\boldsymbol{\theta}}(\mathbf{a} \mid s)$. A *trajectory* $\tau = (s_0, \mathbf{a}_0, s_1, \mathbf{a}_1, \ldots, s_T)$ refers to a sequence of state-action pairs. The probability of τ under the policy parameters $\boldsymbol{\theta}$ is $p(\tau \mid \boldsymbol{\theta})$, and its total reward is $R(\tau)$.

The derivation of the score function gradient estimator tells us that

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[R(\tau)] = \mathbb{E}_{\tau}[\nabla_{\boldsymbol{\theta}} \log p(\tau \mid \boldsymbol{\theta}) R(\tau)].$$

Next, we need to expand the quantity $\log p(\tau|\theta)$ to derive a practical formula. Using the chain rule of probabilities, we obtain:

$$p(\tau|\theta) = \mu(s_0)\pi(a_0|s_0,\theta)P(s_1,r_0|s_0,a_0)\pi(a_1|s_1,\theta)P(s_2,r_1|s_1,a_1)\ldots\pi(a_{T-1}|s_{T-1},\theta)P(s_T,r_{T-1}|s_{T-1},a_{T-1}),$$

where μ is the initial state distribution. When we take the logarithm, the product turns into a sum, and when we differentiate with respect to θ, the terms $P(s_t|s_{t-1},a_{t-1})$ terms drop out as does $\mu(s_0)$. We obtain:

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau\left[\sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|s_t,\theta)R(\tau)\right]$$

The key observation about this result is that the policy gradient can be computed without knowing anything about the system's dynamics function. An intuitive interpretation is that we collect a trajectory, and then increase its log-probability proportionally to its goodness. That is, if the reward $R(\tau)$ is very high, we ought to move in the the direction in parameter space that increases $\log p(\tau|\theta)$.

We can derive versions of this formula that eliminate terms to reduce variance.

First, we can apply the above argument to compute the gradient for a single reward term:

$$\nabla_\theta E_\tau[r_t] = E_\tau\left[\sum_{t'=0}^{t}\nabla_\theta \log\pi(a_{t'}|s_{t'},\theta)r_t\right]$$

Note that the sum goes up to t, because the expectation over r_t can be written in terms of actions $a_{t'}$ with $t' \leq t$. Summing over time (taking $\sum_{t=0}^{T-1}$ of the above equation), we get

$$\begin{aligned}\nabla_\theta E_\tau[R(\tau)] &= E_\tau\left[\sum_{t=0}^{T-1} r_t \sum_{t'=0}^{t}\nabla_\theta \log\pi(a_{t'}|s_{t'},\theta)\right] \\ &= E_\tau\left[\sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|s_t,\theta)\sum_{t'=t}^{T-1} r_{t'}\right]. \qquad (1)\end{aligned}$$

The second formula (Equation (1)) results from the first formula by reordering the summation. We will mostly work with the second formula, as it is more convenient for numerical implementation.

We can further reduce the variance of the policy gradient estimator by using a baseline: that is, we subtract a function $b(s_t)$ from the empirical returns, giving us the following formula for the policy gradient:

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau\left[\sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|s_t,\theta)\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t)\right)\right] \qquad (2)$$

This equality holds for arbitrary baseline functions b. To derive it, we'll show that the added terms $b(s_t)$ have no effect on the expectation, i.e., that $E_\tau[\nabla_\theta \log\pi(a_{t'}|s_{t'},\theta)b(s_t)] = 0$. To show this, split up the expectation over whole trajectories $E_\tau[\ldots]$ into an expectation over all variables before a_t, and all variables after and including it.

$$
\begin{aligned}
& E_\tau[\nabla_\theta \log \pi(a_t|s_t,\theta) b(s_t)] \\
&= E_{s_0,a_{0:t-1}}\left[E_{s_{t+1:T},a_{t:T-1}}[\nabla_\theta \log \pi(a_t|s_t,\theta) b(s_t)]\right] && \text{(break up expectation)} \\
&= E_{s_0,a_{0:t-1}}\left[b(s_t) E_{s_{t+1:T},a_{t:T-1}}[\nabla_\theta \log \pi(a_t|s_t,\theta)]\right] && \text{(pull baseline term out)} \\
&= E_{s_0,a_{0:t-1}}\left[b(s_t) E_{a_t}[\nabla_\theta \log \pi(a_t|s_t,\theta)]\right] \\
&= E_{s_0,a_{0:t-1}}[b(s_t)\cdot 0]
\end{aligned}
$$

The last equation follows because $E_{a_t}[\nabla_\theta \log \pi(a_t|s_t,\theta)] = \nabla_\theta E_{a_t}[1] = 0$ by the definition of the score function gradient estimator.

A near-optimal choice of baseline is the state-value function,

$$
V^\pi(s) = E\left[r_t + r_{t+1} + \cdots + r_{T-1} | s_t = s, a_{t:T-1} \sim \pi\right]
$$

In practice, we will generally choose the baseline to approximate the value function, $b(s) \approx V^\pi(s)$. We can intuitively justify the choice $b(s) \approx V^\pi(s)$ as follows. Suppose we collect a trajectory and compute a noisy gradient estimate

$$
\hat{g} = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta) \sum_{t'=t}^{T-1} r_{t'}
$$

which we will use to update our policy $\theta \to \theta + \epsilon \hat{g}$. This update increases the log-probability of a_t proportionally to the sum of rewards $r_t + r_{t+1} + \cdots + r_{T-1}$ following that action. In other words, if the sum of rewards is high, then the action was probably good, so we increase its probability. To get a better estimate of whether the action was good, we should check to see if the returns were better than expected. Before taking the action, the expected returns were $V^\pi(s_t)$. Thus, the difference $\sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is an approximate estimate of the goodness of action a_t—Chapter 4 discusses in a more precise way how it is an estimate of the advantage function. Including the baseline in our policy gradient estimator, we get

$$
\hat{g} = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right),
$$

which increases the probability of the actions that we infer to be good—meaning that the estimated advantage $\hat{A}_t = \sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is positive.

If the trajectories are very long (i.e., T is high), then the preceding formula will have excessive variance. Thus, practitioners generally use a discount factor, which reduces variance at the cost of some bias. The following expression gives a biased estimator of the policy gradient.

$$
\hat{g} = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right)
$$

To reduce variance in this biased estimator, we should choose $b(s_t)$ to optimally estimate the discounted sum of rewards,

$$b(s) \approx V^{\pi,\gamma}(s) = E\left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \middle| s_t = s; a_{t:(T-1)} \sim \pi\right]$$

Intuitively, the discount makes us pretend that the action a_t has no effect on the reward $r_{t'}$ for t' sufficiently far in the future, i.e., we are downweighting delayed effects by a factor of $\gamma^{t'-t}$. By adding up a series with coefficients $1, \gamma, \gamma^2, \ldots$, we are effectively including $1/(1-\gamma)$ timesteps in the sum.

The policy gradient formulas given above can be used in a practical algorithm for optimizing policies.

Algorithm 4 Vanilla Policy Gradient (VPG)

Initialize policy parameter θ, baseline b
for iteration=1,2,... do **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
    the return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
    the advantage estimate $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
    summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
    which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$
end for

In the algorithm above, the policy update can be performed with stochastic gradient ascent, $\theta \to \theta + \epsilon\hat{g}$, or one can use a more sophisticated method such as Adam (Adaptive Moment Estimation).[53] Adam optimizes parameters by maintaining exponential moving averages of both gradients (first moment) and squared gradients (second moment) for each parameter, effectively incorporating momentum. These moments are used to compute adaptive learning rates, with bias correction applied to improve the estimates. This allows Adam to adapt to both the scale and rate of change of gradients. Empirically, practitioners have found these features to make Adam particularly well-suited for the challenges inherent in RL problems, such as sparse gradients, non-stationary objectives, and parameters of varying scales.[54]

To numerically compute the policy gradient estimate using automatic differentiation software (autograd),[55] we swap the sum with the expectation in the policy gradient estimator:

$$\begin{aligned}\hat{g} &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t)\right) \\ &= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\hat{A}_t\end{aligned}$$

Hence, one can construct the scalar quantity $\sum_t \log \pi_\theta(a_t|s_t)\hat{A}_t$ and differentiate it to obtain the policy gradient.

The vanilla policy gradient method has been well-known for decades, with seminal works dating back to the 1990s.[15,56] Despite its theoretical elegance, it was often considered a poor choice for

many problems due to its high sample complexity.[16] This inefficiency stems from the need for numerous samples to estimate the gradient accurately, especially in high-dimensional action spaces. Several practical difficulties have hindered the widespread adoption of vanilla policy gradients. One key challenge is selecting an appropriate stepsize that remains effective throughout the optimization process. This difficulty arises because the statistics of states and rewards evolve as the policy improves, leading to non-stationarity in the optimization landscape.[57] While adaptive optimization methods like Adam[53] have been proposed to mitigate this issue, they do not fully resolve it. Another significant problem is premature convergence to a nearly-deterministic policy that exhibits suboptimal behavior.[58] This issue is particularly acute in environments with sparse rewards or long-horizon tasks. Simple remedies, such as adding an entropy bonus to encourage exploration,[59] often prove insufficient. While entropy regularization can help maintain policy stochasticity, it doesn't guarantee finding the optimal policy and can sometimes lead to overly random behavior.[22]

2.3.2 Trust Region Policy Optimisation

Trust Region Policy Optimization (TRPO)[16] is an on-policy reinforcement learning algorithm that addresses key limitations of vanilla policy gradient methods. TRPO updates policies by maximizing performance improvement while constraining the divergence between new and old policies. This constraint is expressed using KL-Divergence, which measures the distance between probability distributions. Unlike standard policy gradient methods that maintain proximity in parameter space, TRPO enforces closeness in the space of policies. This distinction is crucial, as small changes in parameters can lead to large differences in policy behavior, potentially causing performance collapse with large step sizes in vanilla methods. The theoretical TRPO update is formulated as:

$$\theta_{k+1} = \arg\max_{\theta} L(\theta_k, \theta) \tag{2.4}$$

$$\text{s.t.} \quad \bar{D}_{KL}(\theta|\theta_k) \leq \delta \tag{2.5}$$

Here, $L(\theta_k, \theta)$ is the surrogate advantage, quantifying how the new policy π_θ performs relative to the old policy π_{θ_k} using data from the old policy:

$$L(\theta_k, \theta) = \mathbb{E}s, a \sim \pi\theta_k \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) \right] \tag{2.6}$$

The constraint $\bar{D}_{KL}(\theta|\theta_k)$ represents the average KL-divergence between policies across states visited by the old policy:

$$\bar{D}KL(\theta|\theta_k) = \mathbb{E}s \sim \pi_{\theta_k}[D_{KL}(\pi_\theta(\cdot|s)|\pi_{\theta_k}(\cdot|s))] \tag{2.7}$$

This formulation allows TRPO to take the largest possible improvement step while ensuring that the new policy doesn't deviate too far from the old one, promoting stable and monotonic performance improvements.

2.3.3 Proximal Policy Optimization

Proximal Policy Optimisation (PPO)[9] addresses the critical challenge of policy update stability. While Trust Region Policy Optimization (TRPO) tackled this issue using complex second-order methods, PPO achieves similar or superior performance through first-order optimization techniques, offering a more computationally efficient and implementationally straightforward approach. The core innovation of PPO lies in its objective function design, which implicitly constrains policy updates without the need for explicit KL-divergence calculations. We focus on the PPO-Clip variant, which has gained widespread adoption due to its simplicity and effectiveness. The PPO-Clip algorithm iteratively updates the policy parameters θ according to:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}s, a \sim \pi\theta_k[L_{\text{CLIP}}(s, a, \theta, \theta_k)], \tag{2.8}$$

where the clipped surrogate objective L_{CLIP} is defined as:

$$L_{\text{CLIP}}(s, a, \theta, \theta_k) = \min\left(r_t(\theta)A^{\pi_{\theta_k}}(s, a), (r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_k}}(s, a)\right). \tag{2.9}$$

Here, $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is the probability ratio between the new and old policies, $A^{\pi_{\theta_k}}(s, a)$ is the advantage function estimate, and ϵ is a hyperparameter controlling the clip range. This formulation balances exploration and exploitation. When the advantage is positive, the objective incentivizes probability ratio increases up to $1 + \epsilon$, promoting exploitation of beneficial actions. Conversely, for negative advantages, it allows ratio decreases to $1 - \epsilon$, facilitating exploration by reducing the probability of detrimental actions. A key insight of PPO-Clip is the simplification of the clipping mechanism, which can be expressed as:

$$L_{\text{CLIP}}(s, a, \theta, \theta_k) = \min\left(r_t(\theta)A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right), \tag{2.10}$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \ (1-\epsilon)A \\ A < 0. \end{cases} \tag{2.11}$$

This formulation provides a computationally efficient way to constrain policy updates, ensuring stable learning without the need for second-order optimization or explicit KL-divergence constraints.

2.3.4 Advantage Actor-Critic

Advantage Actor-Critic (A2C)[22] is slightly orthogonal to the policy gradient methods so far as it seeks the strengths of both policy-based and value-based reinforcement learning approaches. A2C addresses the high variance issue inherent in pure policy gradient methods while maintaining the ability to learn stochastic policies, crucial for solving complex environments with partial observability or adversarial components. The core innovation of A2C lies in its dual network architecture: an actor network that learns the policy, and a critic network that estimates the value function. This separation allows for more stable and efficient learning, as the critic provides a learned baseline that reduces variance in policy gradient estimates. Formally, the A2C algorithm optimizes the policy parameters θ and value function parameters w according to the following

objectives:

$$\nabla_\theta J(\theta) = \mathbb{E}\pi\theta \left[\nabla_\theta \log \pi_\theta(a|s) A(s,a)\right]$$

$$\nabla_w L(w) = \nabla_w \mathbb{E}\pi\theta \left[(R_t - V_w(s_t))^2\right]$$

where $A(s,a)$ is the advantage function, estimated as:

$$A(s,a) \approx R_t - V_w(s_t)$$

Here, R_t is the discounted cumulative reward from time step t, and $V_w(s_t)$ is the critic's estimate of the state value. The actor's policy update incorporates this advantage estimate:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Simultaneously, the critic is updated to minimize the value estimation error:

$$w \leftarrow w - \beta \nabla_w (R_t - V_w(s_t))^2$$

A unique feature of A2C is its synchronous nature, where multiple actor-learners are used to gather experiences in parallel, but updates are applied synchronously. This approach offers improved data efficiency through parallelized experience collection, enhanced stability due to more diverse and decorrelated experiences, and reduced variance in gradient estimates, leading to more reliable updates.

Despite its advantages, A2C faces challenges including sample inefficiency compared to off-policy methods,[60] sensitivity to hyperparameters,[61] and potential difficulties with long-term credit assignment.[62] Additionally, A2C can suffer from premature convergence to suboptimal policies in complex environments[63] and may struggle with exploration in sparse reward settings.[64]

2.4 Optimistic Planning

In this section, we introduce the framework for optimistic planning algorithms, drawing inspiration from the notation used by Pacchiano et al. (2021).[65] We examine the iterative interaction between an agent and a finite-horizon Markov Decision Process (MDP).

Consider an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, v)$, where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ denotes the action space
- P encapsulates the transition dynamics
- $r \in \mathbb{R}^{S \times A}$ is the reward function
- v describes the initial state distribution

For each state-action pair (s, a), $r(s, a)$ represents the true reward, while P governs the transition probabilities, such that $s' \sim P(\cdot, a)$ with probability $P(s, a, s')$. At each round t, the agent selects a policy π_t, which it employs to accumulate rewards and transition data in $\mathcal{M}$ over H steps. We use k to index episodes and h for timesteps within an episode.

Given that the true transition and reward dynamics are unknown, we must approximate them. Let $\hat{r}(s, a) \in \mathbb{R}$ denote the estimated reward and $\hat{P}(s, a) \in \Delta_{|S|}$ the estimated transition probabilities for each state-action pair (s, a), where k indicates the episode number[1]. During training, the agent gathers transition data from $\mathcal{M}$, using this information in each round t to generate a policy π_t and construct an approximate MDP $\mathcal{M}_t = (S, A, \hat{P}, \hat{r}, v)$.

For any policy π, we define:

- $V^{\pi}(\pi)$: The true (scalar) value of π
- $V^{k}(\pi)$: The value of π in the approximate MDP $\mathcal{M}_k$
- $\mathbb{E}_{\pi}$: Expectation under the true MDP $\mathcal{M}$ dynamics using policy π
- $\mathbb{E}_{\pi}^{k}$: Expectation under the approximate MDP $\mathcal{M}_k$ using policy π

The true and approximate value functions for a policy π are defined as:

$$V(\pi) = \mathbb{E}_{\pi}\left[\sum_{h=0}^{H-1} r(s_h, a_h)\right], \quad V_k(\pi) = \mathbb{E}_{\pi}^{k}\left[\sum_{h=0}^{H-1} \hat{r}_k(s_h, a_h)\right].$$

To evaluate optimistic planning algorithms, the concept of regret is used extensively, which quantifies the performance gap between the optimal policy and the executed policies. In the episodic

[1] Δ_k represents the d-dimensional simplex.

RL context, for an agent using policies $\{\pi_k\}_{k=1}^K$ over K episodes (where $T = KH$), the regret is defined as:

$$R(T) = \sum_{k=1}^{K} V(\pi^*) - V(\pi_k),$$

where π^* is the optimal policy for $\mathcal{M}$ and $V(\pi_k)$ represents its true value function.

The principle of optimism in the face of uncertainty (OFU) is a key strategy in addressing the exploration-exploitation trade-off in sequential decision-making. In the realm of RL, model-based OFU algorithms (e.g., Jaksch et al., 2010; Fruit et al., 2018; Tossou et al., 2019) typically operate as follows: At the start of each episode k, the agent selects an approximate MDP $\mathcal{M}_k$ from a model ensemble $\mathcal{M}_k$ and a policy π_k whose approximate value function $V_k^{\pi_k}$ upper bounds the optimal policy's true value function $V(\pi^*)$.

2.4.1 Optimism Decomposition and OFU Algorithms

Pacchiano et al. (2021)[65] introduced an elegant approach to evaluating OFU-inspired algorithms by decomposing the regret $R(T)$ as follows:

$$R(T) = \underbrace{\sum_{k=1}^{K} (V(\pi^*) - V_k(\pi_k))}_{\text{Optimism}} + \underbrace{\sum_{k=1}^{K} (V_k(\pi_k) - V(\pi_k))}_{\text{Estimation Error}}. \tag{1}$$

This decomposition, referred to as the *Optimism Decomposition*, breaks down the regret into two crucial components. The first term, Optimism, ensures that approximate value functions are sufficiently optimistic. The second term, Estimation Error, measures how far the estimated value function is from the true value function.

To achieve an optimal balance between these components, state-of-the-art OFU algorithms employ sophisticated bonus structure sub-routines, designed to maintain regret within specific bounds. Two prominent model-based OFU algorithms exemplify different approaches to achieving optimism. UCRL2[36] generates optimism by analytically optimizing over the entire dynamics uncertainty set, while UCBVI[66] produces optimism by directly adding a bonus to the value function.

However, these methods have limitations. Their structures do not generalize well to function approximation settings, confining their effectiveness strictly to simple tabular environments. This constraint highlights the need for more versatile approaches in complex, real-world scenarios.
Our work addresses this challenge by developing a framework that implicitly incorporates the OFU principle into its algorithm design. Our approach aims to generalize effectively across a wide spectrum of environments, bridging the gap between theoretical OFU principles and practical applications in diverse reinforcement learning settings. By doing so, we seek to overcome the limitations of current methods and provide a more adaptable solution for optimistic exploration in reinforcement learning.

Chapter 3

Methodology

As introduced in the Optimistic Planning section, when designing a MBRL approach, we are effectively pursuing two interlinked objectives. Firstly, we aim to learn an effective representation of the environment such that our estimated model $\hat{\mathcal{M}}_t$ converges to the true model $\mathcal{M}$ as $t \to \infty$. Formally, we strive for $\hat{\mathcal{M}}_t \to \mathcal{M}$. Second is policy optimization, where conditioned on this learned representation, we seek to discover the optimal policy π^*, ensuring that our policy π_t converges to π^* as $t \to \infty$. Formally, we aim for $\pi_t \to \pi^*$. Crucially, we strive to achieve both these objectives in a sample-efficient manner, minimizing the number of interactions required with the environment to reach near-optimal performance.

In this chapter, we develop a tractable formulation for jointly optimizing the model $\mathcal{M}$ and policy π in a unified framework. Our approach yields two distinct algorithmic variants. The first variant employs a counting-based methodology, akin to traditional OFU algorithms. The second variant extends our approach by leveraging a linear assumption on the underlying Markov Decision Process (MDP), resulting in a more generalized expression capable of handling function approximation. Through extensive experimentation, we demonstrate that our method effectively learns complex policies in three sets of scenarios: environments with large state-action spaces, environments specifically designed to discourage exploration, and environments with continuous state spaces.

3.1 Counting Algorithm

Let us begin by considering the infinite horizon discounted Markov Decision Process (MDP). This setting is equivalent to working with finite horizon MDPs, ensuring that any algorithm we develop will be applicable to both cases. For an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, v)$, we can express the objective function (the cumulative discounted reward) as:

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\nu, \pi_\theta, P}[r(s_t, a_t)]$$

where v is the initial state distribution, π_θ is the parameterized stochastic policy that our agent follows, and P represents the transition dynamics function.
Policy gradient methods, at their core, aim to approximate the gradient of this quantity with

respect to the policy parameters θ. This approach is necessitated by the complexity of directly optimizing $J(\theta)$, which involves both an infinite sum and an infinite recursion, based on a combination of marginal distributions (initial state and reward) and conditional distributions (transition and policy). The challenge lies in the interplay between these distributions and the temporal structure of the problem. To address this, our first step is to unroll this recursion and carefully examine its temporal structure. By doing so, we aim to identify patterns that can lead to a more tractable expression of the gradient.

By considering a arbitrarily chosen trajectory $\tau = (s_0, a_0, s_1, a_1, \cdots, s_t, a_t)$ we can re-write this expression more explicitly:

$$\begin{aligned}\sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{v_0,\pi_\theta,P}[r(x_t, a_t)] &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0,a_0}[\mathbb{E}_{x_1,a_1}[\cdots \mathbb{E}_{x_t,a_t}[r(x_t, a_t)]]] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}[\mathbb{E}_{a_0}[\mathbb{E}_{x_1}[\mathbb{E}_{a_1}[\cdots \mathbb{E}_{x_t}[\mathbb{E}_{a_t}[r(x_t, a_t)] \cdots] \qquad (*)\end{aligned}$$

where:

- $x_0 \sim v$
- $a_0 \sim \pi_\theta(\cdot \mid x_0)$
- $x_i \sim P(\cdot \mid x_{i-1}, a_{i-1}),\ i \in \mathbb{N}^*$
- $a_i \sim \pi_\theta(\cdot \mid x_i),\ i \in \mathbb{N}^*$

Since we are dealing with a strictly tabular setting, we can evaluate these expectation explicitly as summations over states and actions.

Let us evaluate $J(\theta)$ as follows:

$$\begin{aligned}J(\theta) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}\Big[r(x_t, a_t)\Big] \cdots \Big] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}\Big[\sum_{a_t \in \mathcal{A}} \pi_\theta(a_t \mid x_t) r(x_t, a_t)\Big] \cdots \Big] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_{t-1}}\Big[\mathbb{E}_{a_{t-1}}\Big[\mathbb{E}_{x_t}\Big[\mathcal{R}_\pi(x_t)\Big] \cdots \Big] \quad \text{[By Definition of } \mathcal{R}_\pi(s)\text{]} \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_{t-1}}\Big[\mathbb{E}_{a_{t-1}}\Big[\mathcal{R}_\pi(x_t)\Big] \cdots \Big] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{a_{t-1}}\Big[\mathbb{E}_{x_{t-1}}\Big[\sum_{x_t \in \mathcal{S}} \sum_{a_{t-1} \in \mathcal{A}} \pi_\theta(a_{t-1} \mid x_{t-1}) P(x_t \mid x_{t-1}, a_{t-1}) \mathcal{R}_\pi(x_t)\Big] \cdots \Big] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{a_{t-1}}\Big[\mathbb{E}_{x_{t-1}}\Big[\sum_{x_t \in \mathcal{S}} \mathcal{P}_\pi(x_{t-1}, x_t) \mathcal{R}_\pi(x_t)\Big] \cdots \Big] \quad \text{[By Definition of } \mathcal{P}_\pi(s, s')\text{]} \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{a_{t-2}}\Big[\mathbb{E}_{x_{t-2}}\Big[\sum_{x_{t-1} \in \mathcal{S}} \mathcal{P}_\pi(x_{t-2}, x_{t-1}) \sum_{x_t \in \mathcal{S}} \mathcal{P}_\pi(x_{t-1}, x_t) \mathcal{R}_\pi(x_t)\Big] \cdots \Big]\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\sum_{x_1 \in \mathcal{S}} \mathcal{P}_\pi(x_0, x_1) \cdots \sum_{x_{t-1} \in \mathcal{S}} \mathcal{P}_\pi(x_{t-2}, x_{t-1}) \sum_{x_t \in \mathcal{S}} \mathcal{P}_\pi(x_{t-1}, x_t) \mathcal{R}_\pi(x_t) \Big] \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{x_0 \in \mathcal{S}} v(x_0) \sum_{x_1 \in \mathcal{S}} \mathcal{P}_\pi(x_0, x_1) \cdots \sum_{x_{t-1} \in \mathcal{S}} \mathcal{P}_\pi(x_{t-2}, x_{t-1}) \sum_{x_t \in \mathcal{S}} \mathcal{P}_\pi(x_{t-1}, x_t) \mathcal{R}_\pi(x_t) \\
&= \sum_{t=0}^{\infty} \gamma^t v (\mathcal{P}_\pi)^t \mathcal{R}_\pi \\
&= v^T \Big(\sum_{t=0}^{\infty} (\gamma \mathcal{P}_\pi)^t \Big) \mathcal{R}_\pi \\
&= v^T (\mathbb{I} - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi
\end{aligned}
$$

Before delving into how we could turn such an expression into an algorithm, it is worth spending some time to understand what this quantity $v^\top (\mathbb{I} - \gamma \mathcal{P}_\pi)^{-1}$ represents.

DEFINITION 14 *Let $v \in \mathbb{R}^{|\mathcal{S}|}, v^\top \mathbf{1} = 1$ be a distribution over initial states, the discounted weighting of states $d_{v,\gamma,\pi}$ is:*

$$
d_{v,\gamma,\pi}^\top = v^\top (\mathbb{I} - \gamma \mathcal{P}_\pi)^{-1}.
$$

While it is tempting to think of $d_{v,\gamma,\pi}$ as a distribution, a simple calculation for the case $v = e_s$ shows that the rows of $(I - \gamma \mathcal{P}_\pi)^{-1}$ do not sum up to 1:

$$
\sum_{t=0}^{\infty} \sum_{x'} \gamma^t P_\pi(S_t = x' | x) = \sum_{t=0}^{\infty} \gamma^t \sum_{x'} P_\pi(S_t = x' | x) = \frac{1}{1-\gamma}.
$$

Hence, $d_{v,\gamma,\pi}$ neither is a distribution, or for that matter a stationary distribution, but simply is a discounted weighting of states – an expression used by Sutton et al.[56] As we did for the Bellman expectation equations, we can unroll the Neumann series expansion of the discounted weighting of states to obtain a recursive expression:

$$
\begin{aligned}
d_{v,\gamma,\pi}^\top &= v^\top \sum_{t=0}^{\infty} (\gamma P_\pi)^t \\
&= v^\top + v^\top \sum_{t=1}^{\infty} (\gamma P_\pi)^t \\
&= v^\top + \gamma \left(v^\top \sum_{t=0}^{\infty} (\gamma P_\pi)^t \right) P_\pi \\
&= v^\top + \gamma d_{v,\gamma,\pi}^\top P_\pi \ . \qquad (2.7)
\end{aligned}
$$

We observe a very interesting parallel here between these equations and the Bellman expectation equations but where $d_{v,\gamma,\pi}$ plays the role of the "value function" and v is its associated reward function. Hence, for any given initial distribution v, there exists a corresponding linear system

of equations whose solution is the discounted weighting of states. In Sutton and Barto (2018), a similar expression for $d_{v,\gamma,\pi}$ is provided for the so-called on-policy distribution in the undiscounted episodic case. In the same way that the Bellman expectation equations describe the expected sum of discounted rewards, the recursive expression for $d_{v,\gamma,\pi}$ corresponds to an expected sum of discounted indicator variables:

$$\begin{aligned} d_{v,\gamma,\pi}(x) &= \sum_{x_0} v(x_0) \sum_{t=0}^{\infty} \gamma^t P_\pi(S_t = x | S_0 = x_0) \\ &= \sum_{x_0} v(s_0) \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[\mathbb{I}_{S_t=x} | S_0 = x_0] \\ &= \mathbb{E}_v \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}_{S_t=x} \right] . \end{aligned}$$

This interpretation of the discounted weighting of states as sum of indicator variables can be leveraged in the design of learning algorithms. Instead of having a set of evaluation equations for each $x \in \mathcal{S}$, we can also use a more compact recursive expression involving a matrix $\mathbf{A} \in \mathbb{R}^{|S| \times |S|}$ whose rows are initial state distributions:

$$\mathbf{D}_{\mathbf{A},\gamma,\pi} = \mathbf{A} \sum_{t=0}^{\infty} (\gamma \mathcal{P}_\pi)^t = \mathbf{A} + \gamma \mathbf{D}_{\mathbf{A},\gamma,\pi} \mathcal{P}_\pi = \mathbf{A}(\mathbb{I} - \gamma \mathcal{P}_\pi)^{-1} .$$

The matrix $\mathbf{A} \in \mathbb{R}^{|S| \times |S|}$ introduces a generalization of the initial state distribution concept, extending beyond the single vector v to a framework accommodating multiple theoretical distributions. Each row of $\mathbf{A}$ represents a potential initial state distribution, providing a construct for analyzing system behavior under various hypothetical starting conditions. This formulation offers a theoretical tool for exploring policy sensitivity to initial conditions and for deriving general results applicable across a spectrum of scenarios. While our proposed implementation focuses on settings learning a single empirical initial distribution through counting, this object is still useful because iterative evaluation of $\mathbf{D}_{\mathbf{A},\gamma,\pi}$ only involves $\mathcal{O}(|S|^2)$ operations to compute a matrix-vector product rather than the $\mathcal{O}(|S|^3)$ which would otherwise be required for matrix inversion.

Proposed Algorithm

Algorithm 5 Tabular Policy Optimization with Approximate Environment Dynamics

Input: Initialise policy parameters θ_0, batch size N, number of epochs K, total iterations T, discount factor γ, learning rate α
while $t < T$ **do**
    1. Data Collection:
        Collect set of trajectories $\mathcal{D} = \{\tau_t\}_{t=1}^{N}$ by running policy $\pi_t = \pi(\theta_t)$
        Store transition probabilities $P(s'|s,a) = \frac{N(s,a,s')}{\sum_{s'} N(s,a,s')}$
        Store reward values $R(s,a)$
        Store initial state distribution $v_0 = \frac{N(s_0)}{\sum_{s_0} N(s_0)}$
    2. Policy Evaluation:
        Compute $\mathcal{P}_\pi$ and $\mathcal{R}_\pi$
        Compute objective function $\mathbf{J}(\theta_t) = v^T(\mathbb{I} - \gamma\mathcal{P}_\pi)^{-1}\mathcal{R}_\pi$
    3. Policy Improvement:
        Perform K epochs of gradient ascent:
        $\theta_{t+1} \leftarrow \arg\max_{\theta_t} \mathbf{J}(\theta_t)$
    Increment $t \leftarrow t+1$
end while

Algorithm 5 presents our proposed method for tabular policy optimization leveraging approximate environment dynamics. The algorithm initializes policy parameters θ_0 and hyperparameters including batch size N, number of epochs K, total iterations T, and discount factor γ. In each iteration, trajectories are collected by executing the current policy $\pi_t = \pi(\theta_t)$. From these, we estimate transition probabilities $P(s'|s,a)$, reward values $R(s,a)$, and the initial state distribution v_0. Policy evaluation computes $\mathcal{P}_\pi$ and $\mathcal{R}_\pi$, followed by the objective function $\mathbf{J}(\theta_t) = v^T(\mathbb{I}-\gamma\mathcal{P}_\pi)^{-1}\mathcal{R}_\pi$. Policy improvement is achieved through K epochs of gradient ascent: $\theta_{t+1} \leftarrow \arg\max_{\theta_t} \mathbf{J}(\theta_t)$.

This approach combines model-based reinforcement learning with policy gradient methods, leveraging a singular tractable expression to directly optimize the policy using count-based estimates of the approximate MDP $\hat{\mathcal{M}}$. It implicitly applies OFU principles through the inherent stochasticity of policy gradient methods, obviating the need for explicit exploration subroutines often required in value-based methods. Within this model-based reinforcement learning framework, actions are sampled from a stochastic policy that naturally encourages exploration in less-visited states, with exploration possible to control via a temperature parameter. Despite its advantages, the algorithm faces potential limitations as the state-action space grows. Sparse representations in larger spaces may challenge effective optimization through autograd software. The next section introduces a generalization of this method designed to maintain rich feature representations in expanding state-action spaces, addressing this scalability concern.

3.2 A More General Setting

In this section, we examine learning with linearly parameterized Markov Decision Processes (MDPs), commonly referred to as linear MDPs. The core assumption in this framework is the existence of a feature map that enables the expression of transition and reward functions as linear functions. While this may seem like a simplifying assumption, it offers substantial flexibility. Linear MDPs allow us to move beyond purely tabular settings and explore scenarios where feature maps can be represented using kernels or neural networks. This approach facilitates learning in environments with continuous state-action spaces, which was previously infeasible with one-hot encoding. Additionally, it addresses challenges in extremely large discrete state-action spaces, where computing $(\mathbb{I}-\gamma\mathcal{P}_\pi)^{-1}$ - an $\mathcal{O}(|S|^3)$ operation - would lead to computational and numerical stability issues. Let us start by describing our problem setting more precisely:

DEFINITION 15 (Linear MDP (Bradtke & Barto, 1996)) *An MDP $(\mathcal{S}, \mathcal{A}, P, r, v)$ is a linear MDP with feature maps $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and $\Psi : \mathcal{S} \to \mathbb{R}^K$ to describe the transition dynamics and rewards as follows:*

$$
\begin{aligned}
&r(x,a) = \langle w, \varphi(x,a)\rangle && v(x) = \langle z, \Psi(x)\rangle \\
&P(x' \mid x,a) = \langle \Psi(x'), T\varphi(x,a)\rangle && \pi(a \mid x) = SOFTMAX\left(\theta^T\varphi(x,a)\right)
\end{aligned}
$$

with

$$
\begin{aligned}
&T \in \mathbb{R}^{K\times K} && \theta, w, z \in \mathbb{R}^K \\
&\theta^\top\varphi(x,a) \in \mathbb{R}^{|A|}
\end{aligned}
$$

The policy in linear MDPs is defined using a softmax distribution: $\pi(a|x) = \frac{\exp(\theta^\top\phi(x,a)/\tau)}{\sum_b \exp(\theta^\top\phi(x,b)/\tau)}$, where τ is the temperature parameter. This choice offers smoothness for gradient-based optimization and built-in exploration-exploitation trade-off control. As $\tau \to 0$, the policy becomes more deterministic, while higher τ values promote exploration. This acts as a form of regularization, potentially avoiding convergence to suboptimal policies, albeit at the cost of sample efficiency. The careful tuning of τ is crucial to balance thorough state-action space exploration with rapid convergence to high-performing policies.

Before diving into algorithm design for linear MDPs, let's examine its key components through a simple example. This analysis will shape our approach to developing an effective algorithm:

Example: A Simple MDP

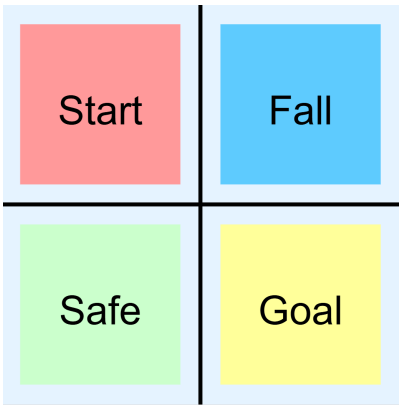


Figure 3.1: A visualisation of a 2x2 Frozen Lake environment

This MDP describes a Frozen Lake environment depicted as a 2x2 grid-based scenario. The agent begins in the start state located at position (0, 0) and must navigate to the goal state at (1, 1) to receive a reward of 1. However, there is also a hazardous "fall into lake" state at position (1, 0), which terminates the episode if reached. The agent can take four actions: up, down, left, and right, to traverse the grid. Transitions are made deterministically. If the agent reaches the goal state, it receives a reward of 1, and the episode ends.

In this example, we can use the one-hot encoding to represent states and state-action pairs. Let $|\mathcal{S}| = S$ and $|\mathcal{A}| = A$ in a linear MDP. Then, the transition matrix $T \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}||\mathcal{A}|}$, To solve for T, we must consider $S \times S \times A$ equations, one for each (x, a, x') triple: $P(x'|x, a) = \langle \Psi(x'), T\phi(x, a)\rangle$. Consequently, the total number of parameters to learn is $|\mathcal{S}|^2|\mathcal{A}|$.

To illustrate this complexity, we consider our simple 2x2 Frozen Lake environment with 4 states and 4 actions. In this case: $T \in \mathbb{R}^{4\times16}$, where $\Psi(x) \in \mathbb{R}^4$ and $\phi(x, a) \in \mathbb{R}^{16}$

Solving for T, we find:

$$T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Even in this small example, we need to solve 64 linear equations (4 states $\times$ 4 states $\times$ 4 actions) to determine T. As S and A increase, explicitly learning T becomes computationally intractable. This suggests to us that any learning algorithm that is developed based on the linear MDP should ideally implicitly learn T in order to be scalable to more realistic environments.

3.2.1 Our Algorithm: ADAPT

We can once again state the cumulative discounted reward, $J(\theta)$, as:

$$\sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{v_0,\pi,P}[r(x_t, a_t)] = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}[\mathbb{E}_{a_0}[\mathbb{E}_{x_1}[\mathbb{E}_{a_1}[\cdots \mathbb{E}_{x_t}[\mathbb{E}_{a_t}[r(x_t, a_t)] \cdots] \tag{*}$$

where:

- $x_0 \sim \nu$
- $x_i \sim P(\cdot \mid x_{i-1}, a_{i-1})$, $i \in \mathbb{N}^*$
- $a_i \sim \pi(\cdot \mid x_i)$, $i \in \mathbb{N}$

We can now evaluate $J(\theta)$ as follows:

$$\begin{aligned}
J(\theta) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}\Big[r(x_t, a_t)\Big] \cdots \Big] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}\Big[\langle w, \varphi(x_t, a_t)\rangle\Big] \cdots \Big] \\
&= \sum_{t=0}^{\infty} \gamma^t \Big\langle w, \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots \mathbb{E}_{a_{t-1}}\Big[\mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}[\varphi(x_t, a_t)]\Big]\Big] \cdots \Big]\Big\rangle
\end{aligned}$$

Consider the quantity $\mathbb{E}x_t[\mathbb{E}a_t[\varphi(x_t, a_t)]]$, where we seek to learn an approximation. This nested expectation encapsulates two key elements of a Markov Decision Process: the policy and the transition dynamics. The inner expectation, $\mathbb{E}_{a_t}[\varphi(x_t, a_t)]$, represents the policy mean embedding—a marginal mean embedding of the policy with respect to the kernel parameterized by φ on a fixed state. The outer expectation, in turn, represents the conditional mean embedding of the transition dynamics function, with respect to the kernel parameterized by the inner expectation. To formalize this concept, we introduce an operator $M : \mathcal{F} \to \mathcal{F}$, where $\mathcal{F}$ denotes a reproducing kernel Hilbert space (RKHS) that φ maps into. We define M as:

$$(Mf)(x, a) = \mathbb{E}_{x' \sim P(\cdot|x,a), a' \sim \pi(\cdot|x')}[f(x', a')]$$

Here, $P(\cdot|x, a)$ denotes the transition probability function, and $\pi(\cdot|x')$ represents the policy. The operator M thus encapsulates the combined effect of the policy and transition dynamics, mapping a function of the current state-action to the expected value of that function in the subsequent state-action. In the context of our original expression, if we consider $f(x, a) = \varphi(x, a)$, then Mf yields precisely the quantity we aim to approximate:

$$\begin{aligned}
(Mf)(x_{t-1}, a_{t-1}) &= M\varphi(x_{t-1}, a_{t-1}) \\
&= \mathbb{E}_{x_t}[\mathbb{E}_{a_t}[\varphi(x_t, a_t)]]
\end{aligned}$$

where $M : \mathbb{R}^K \to \mathbb{R}^K$. Next, we need to explicitly define M in terms of the linear MDP. For simplicity, we develop this definition for M in the discrete state-action space.

$$
\begin{aligned}
M\varphi(x,a) &= \mathop{\mathbb{E}}_{x'a'}[\varphi(x',a')] \\
&= \mathop{\mathbb{E}}_{x'}\left[\sum_{a'\in A}\pi(a'|x')\varphi(x',a')\right] \\
&= \sum_{x'} P(x'|x,a)\sum_{a'\in A}\pi(a'|x')\varphi(x',a') \\
&= \sum_{x'}\langle\psi(x'), T\varphi(x,a)\rangle\sum_{a'\in A}\pi(a'|x')\varphi(x',a') \\
&= \sum_{x'}\psi(x')^T T\varphi(x,a)\sum_{a'\in A}\pi(a'|x')\varphi(x',a') \\
&= \sum_{x'}\left(\sum_{a'\in A}\pi(a'|x')\varphi(x',a')\right)\psi(x')^T T\varphi(x,a) \\
&= \left(\sum_{x'}\left(\sum_{a'\in A}\pi(a'|x')\varphi(x',a')\right)\psi(x')^T T\right)\varphi(x,a)
\end{aligned}
$$

Therefore, we can define M as:

$$
M = \mathop{\mathbb{E}}_{x'a'}\left[\left[\varphi(x',a')\psi(x')^T T\right]\right]
$$

Now, we can continue to evaluate $J(\theta)$ as follows:

$$
\begin{aligned}
J(\theta) &= \sum_{t=0}^{\infty}\gamma^t\Big\langle w, \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots\mathbb{E}_{a_{t-1}}\Big[\mathbb{E}_{x_t}\Big[\mathbb{E}_{a_t}[\varphi(x_t,a_t)]\Big]\Big]\cdots\Big]\Big\rangle \\
&= \sum_{t=0}^{\infty}\gamma^t\Big\langle w, \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots\mathbb{E}_{x_{t-1}}\Big[\mathbb{E}_{a_{t-1}}\Big[M\varphi(x_{t-1},a_{t-1})\Big]\Big]\cdots\Big]\Big\rangle \\
&= \sum_{t=0}^{\infty}\gamma^t\Big\langle w, \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots M\mathbb{E}_{x_{t-1}}\Big[\mathbb{E}_{a_{t-1}}\Big[\varphi(x_{t-1},a_{t-1})\Big]\Big]\cdots\Big]\Big\rangle \\
&= \sum_{t=0}^{\infty}\gamma^t\Big\langle w, \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\mathbb{E}_{x_1}\Big[\mathbb{E}_{a_1}\Big[\cdots M^2\mathbb{E}_{x_{t-2}}\Big[\mathbb{E}_{a_{t-2}}\Big[\varphi(x_{t-2},a_{t-2})\Big]\Big]\cdots\Big]\Big\rangle \\
&= \sum_{t=0}^{\infty}\gamma^t\Big\langle w, M^t\mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\varphi(x_0,a_0)\Big]\Big]\Big\rangle \\
&= \Big\langle w, \sum_{t=0}^{\infty}(\gamma M)^t\mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\varphi(x_0,a_0)\Big]\Big]\Big\rangle \\
&= w^T\Big(\sum_{t=0}^{\infty}(\gamma M)^t\Big)\mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\varphi(x_0,a_0)\Big]\Big] \\
&= w^T\Big(\mathbb{I}-\gamma M\Big)^{-1}W
\end{aligned}
$$

where $W = \mathbb{E}_{x_0}\Big[\mathbb{E}_{a_0}\Big[\varphi(x_0, a_0)\Big]\Big]$ and $M = \mathop{\mathbb{E}\mathbb{E}}_{x'a'}\left[\left[\varphi(x', a')\psi(x')^T T\right]\right]$

Therefore, we have three quantities that we need to empirically learn: w, M, W.

Starting with M, it is useful to restate its initial definition.

$$M\varphi(x, a) = \mathop{\mathbb{E}\mathbb{E}}_{x'a'}[\varphi(x', a')] \tag{3.1}$$

Noting that $M : \mathbb{R}^K \to \mathbb{R}^K$, we can make the linear approximation that $\hat{M} \in \mathbb{R}^{K\times K}$. With this being the case, we can note that

$$\hat{M}\varphi(x, a) = \mathop{\mathbb{E}\mathbb{E}}_{x'a'}[\varphi(x', a')] \tag{3.2}$$

resembles a linear system of equations of the form "Ax = b," where M represents a linear mapping of the previous state-action pair to the *expected* current state-action pair. This is a mapping that we can learn empirically using least squares ridge regression. That is:

$$\begin{aligned}
\hat{M}_* &= \frac{1}{m}\sum_{i=1}^{m}\left\|\hat{M}\varphi(x, a) - \mathbb{E}_{b\sim\pi}\left[\varphi(x', b)\right]\right\|^2 + \lambda\|\hat{M}\|^2 \\
&= \left(X^T X + \lambda\mathbb{I}\right)^{-1} X^T Y \\
&= C_\lambda^{-1} D
\end{aligned}$$

where:

- Training data is $\{(x_i, a_i, x_i') : i = 1, \ldots, m\}$.
- $\lambda > 0$ is the regularization parameter.
- $X = \begin{bmatrix} \varphi(x_1, a_1)^\top \\ \vdots \\ \varphi(x_n, a_n)^\top \end{bmatrix}, \quad Y = \begin{bmatrix} \mu_\pi(x_1')^\top \\ \vdots \\ \mu_\pi(x_m')^\top \end{bmatrix} = \begin{bmatrix} \sum_{b\in\mathcal{A}} \pi(b \mid x_1')\, \varphi(x_1', b) \\ \vdots \\ \sum_{b\in\mathcal{A}} \pi(b \mid x_m')\, \varphi(x_m', b) \end{bmatrix}$
- $C_\lambda = X^T X + \lambda\mathbb{I}$, $D = X^\top Y$

Our algorithm assumes discrete action spaces, prevalent in RL benchmarks [67] and applicable to key domains like game-playing.[68] While this may appear to constrain our algorithm's applicability to continuous control tasks, literature on discretizing environments is well developed. For instance, bang-bang control,[69] a technique where the optimal control switches rapidly between extreme states, provides a theoretical basis for discretizing continuous action spaces. Such approaches suggest that our algorithm loses its generality only slightly through our discrete action assumption.

Similarly for w, we can learn this directly from state-action-reward trajectories

$$\begin{aligned}\hat{w}_* &= \frac{1}{m}\sum_{i=1}^{m}\left\|\hat{w}^\top\varphi(x,a) - r_i\right\|^2 + \lambda\|\hat{w}\|^2 \\ &= \left(X^T X + \lambda\mathbb{I}\right)^{-1} X^T R \\ &= C_\lambda^{-1} E\end{aligned}$$

where:

- Training data is $\{(x_i, a_i, x_i') : i = 1, \ldots, m\}$.
- $\lambda > 0$ is the regularization parameter.
- $X = \begin{bmatrix}\varphi(x_1,a_1)^\top \\ \vdots \\ \varphi(x_n,a_n)^\top\end{bmatrix}, \quad R = \begin{bmatrix} r_1 \\ \vdots \\ r_m\end{bmatrix}$
- $C_\lambda = \mathrm{X}^T X + \lambda\mathbb{I}$, $E = X^\top R$

Now, we can continue to evaluate J(θ):

$$\begin{aligned}J(\theta) &= w^T(\mathbb{I} - \gamma M_\pi)^{-1} W \\ &= w^T(\mathbb{I} - \gamma C_\lambda^{-1} D)^{-1} W \\ &= w^T(C_\lambda^{-1} C_\lambda - \gamma C_\lambda^{-1} D)^{-1} W \\ &= w^T(C_\lambda^{-1}(C_\lambda - \gamma D))^{-1} W \\ &= w^T(C_\lambda - \gamma D)^{-1} C_\lambda W \\ &= (C_\lambda^{-1} E)^T (C_\lambda - \gamma D)^{-1} C_\lambda W\end{aligned}$$

Finally, W is an empirical vector of the form:

$$W = \frac{1}{n}\sum_{i=1}^{n}\sum_{b\in\mathcal{A}}\pi(b|x_i)\varphi(x_i, b)$$

where:

- Training data is $\{(x_0, a_i, x_i') : i = 1, \ldots, n\}$ represents the data collected at the start of each episode.

Algorithm 6 ADAPT (Alternating Dynamics Approximation and Policy opTimization)

Input: Initialize policy parameters θ_0, initialize feature map φ, batch size N, number of epochs K, learning rate α, regularization parameter λ, total iterations T, discount factor γ, temperature τ
while $t < T$ **do**
    1. Data Collection:
        Collect set of trajectories $\mathcal{D} = \{\tau_t\}_{t=1}^{N}$ by running policy $\pi_t = \pi(\theta_t)$
        Compute C_λ, D, E, W
    2. Policy Evaluation:
        Compute $\mathbf{J}(\theta_t) = (C_\lambda^{-1}E)^T(C_\lambda - \gamma D)^{-1}C_\lambda W$
    3. Policy Improvement:
        Perform K epochs of gradient ascent:
        $\theta_{t+1} \leftarrow \arg\max_{\theta_t} \mathbf{J}(\theta_t)$
    Increment $t \leftarrow t + 1$
end while

ADAPT presents a generalized framework for learning the approximate MDP $\hat{\mathcal{M}}$ in both tabular and function approximation settings. This approach extends beyond count-based methods, necessitating the learning of additional parameters. We employ least squares ridge regression to learn $\hat{\mathcal{M}}$, a choice motivated by the desire to avoid multiple matrix inversions, thus potentially improving computational efficiency. The algorithm's performance critically depends on the selection of feature representation φ, which must balance expressiveness with computational tractability. This feature map serves as the bridge between tabular state settings and continuous state state settings. The critical assumption that we have made is that we are dealing with environments that strictly have a discrete number of actions.

Chapter 4

Experimental Results

In the experiments below, we refer to Algorithm 5 as the "Counting" algorithm and the use of one-hot encoding feature representation with Algorithm 6 as the ADAPT algorithm.

We conducted a comprehensive empirical evaluation of both of our proposed algorithms across various discrete action RL environments. Our experiments were designed to assess both the policy optimization procedure in isolation and the full algorithm incorporating exploration. This two-phased approach allows us to disentangle the effects of model learning from policy optimization. Finally, we consider a continuous state space environment to explore our algorithm's ability to handle non-tabular feature representations.

Our evaluation begins with classic tabular reinforcement learning environments implemented in the OpenAI Gym platform.[67] We then extend our focus to environments specifically designed to challenge state-of-the-art algorithms in terms of exploration. This progression allows us to validate our algorithm's performance on well-understood benchmarks before testing its capabilities in more demanding scenarios.

OpenAI Gym is a widely-used toolkit for developing and comparing reinforcement learning algorithms. It provides a diverse suite of environments, ranging from classic control tasks to more complex scenarios, all adhering to a standard interface. This standardization facilitates the benchmarking of different algorithms across various tasks, promoting reproducibility in reinforcement learning research.The toolkit's flexibility and extensive documentation have contributed to its widespread adoption in both academic research and practical applications of reinforcement learning.

We evaluated our algorithm on three classic reinforcement learning environments: Frozen Lake, Cliff Walking, and Taxi. These environments were chosen for their diverse characteristics and well-understood dynamics, facilitating interpretation of results. All experiments were implemented using Pytorch and locally run on Google Collab. Hyperparameter choices are for all figures are referenced in the appendix. In the figure below, we summarise the key properties of these environments.

Environment	Description	State Space	Action Space	Challenges
Frozen Lake	Agent must navigate from start to goal on a grid with holes	Discrete, 16 states (4x4 grid)	4 (cardinal directions)	Sparse rewards
Cliff Walking	Agent must reach a goal while avoiding a cliff, balancing shortest path and risk	Discrete, 12x4 grid size	4 (cardinal directions)	Risk-reward trade-off, negative rewards
Taxi	Agent must pick up and drop off passengers at specific locations in a grid-based city	Discrete, 500 states	6 (movement/ pickup/dropoff)	Sequential decision making, larger state space

Table 4.1: Overview of the classic reinforcement learning environments used in our experiments. These environments, available in OpenAI Gym, offer a range of challenges including stochastic dynamics, sparse rewards, and varying levels of complexity in state and action spaces.

4.1 Policy Optimization without Exploration

To isolate the effectiveness of our policy optimization procedure, we first conducted experiments assuming a perfect model of the environment.

4.1.1 Optimizer Comparison

A key design choice in our experiments is the selection of an optimizer. In Figures 4.1 & 4.2, we plot the average rewards for both ADAPT and counting algorithms over 1,000 epochs of policy iteration for three tasks (averaged over 10 runs). Across all tasks, Adam consistently performs significantly better than stochastic gradient descent (SGD) in each environment. The notable exception to this trend is the Counting Algorithm in the Cliff Walking environment. We note that each environment exhibits a sparse reward structure to varying degrees, leading to sparse gradients. Adam's generally superior performance in these scenarios may be attributed to its adaptive learning rates and momentum-based update rule. By maintaining per-parameter learning rates adjusted via exponential moving averages of past gradients, Adam can effectively scale updates for infrequently occurring gradients. Its first moment estimate retains information from previous updates, facilitating progress along sparse dimensions, while the second moment estimate acts as an adaptive learning rate. This adjustment enables larger updates for sparse parameters while controlling updates for denser ones. Moreover, Adam's scale-invariant updates and bias correction mechanisms enhance its robustness to variations in gradient magnitudes, potentially improving early-stage learning — an advantage particularly relevant for reinforcement learning tasks with sparse rewards and gradients. Thus unless stated otherwise, we will proceed with Adam as our default optimizer of choice.

Figure 4.1: **ADAPT:** SGD versus Adam across three OpenAI Gym envrionments: Frozen Lake, Cliff Walking, and Taxi (Results averaged over 10 runs)

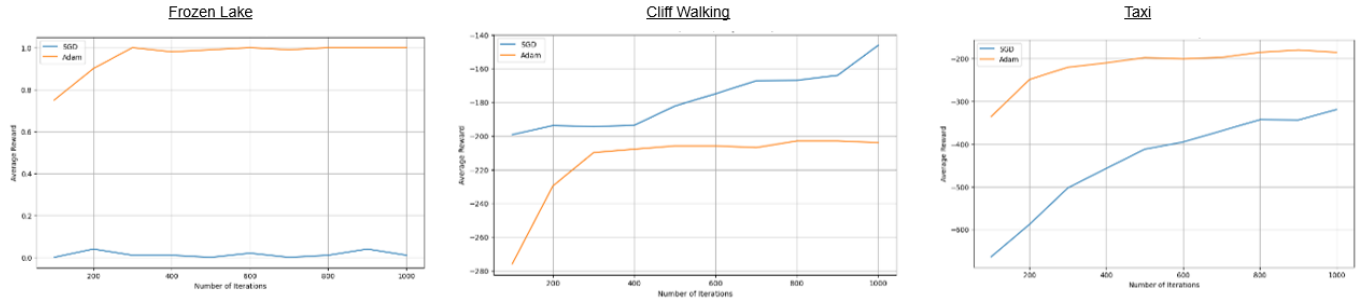


Figure 4.2: **Counting Algorithm:** SGD versus Adam across three OpenAI Gym environments: Frozen Lake, Cliff Walking, and Taxi (Results averaged over 10 runs)

4.1.2 Sample Efficiency Analysis

In Figures 4.3, 4.4, and 4.5, we study the sample efficiency of the ADAPT algorithm versus the Counting algorithm across the three tasks. We observe a significantly faster rate of improvement in the ADAPT algorithm versus the Counting Algorithm. In the Taxi environment, ADAPT achieves significantly higher rewards than Counting across both 80,000 and 100,000 timesteps, demonstrating its effectiveness in complex, state-rich scenarios. The Cliff Walking environment shows ADAPT outperforming Counting across various timesteps (15,000 to 50,000), with the performance gap narrowing as timesteps increase, indicating Counting's gradual improvement with more data. In the Frozen Lake environment, ADAPT exhibits rapid convergence to near-optimal performance within 200 iterations, even in short time horizons (800 to 10,000 timesteps), while Counting's performance improves more slowly and remains highly dependent on the number of timesteps. These results across diverse environments suggest that ADAPT is able to learn effective representations of these environments at a speed that its more naive counterpart cannot.

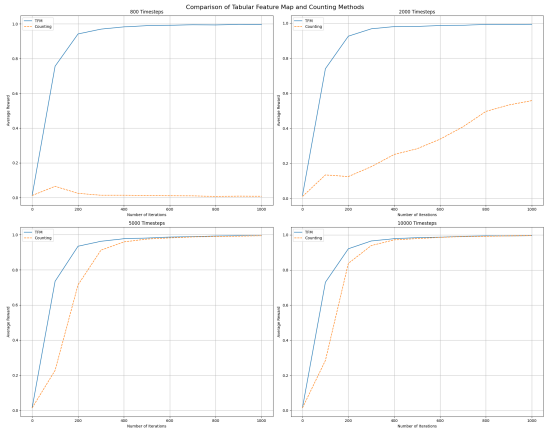


Figure 4.3: Frozen Lake sample efficiency.

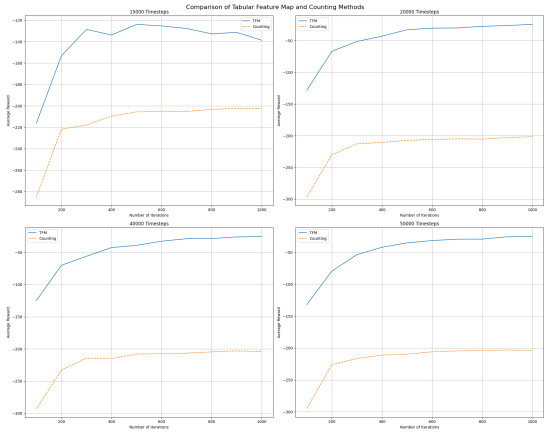


Figure 4.4: Cliff Walking sample efficiency.

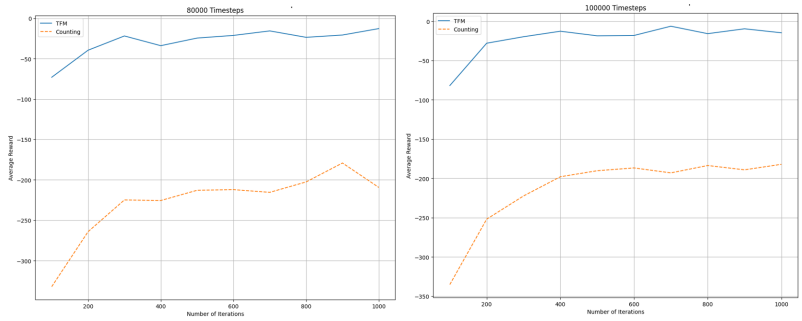


Figure 4.5: Taxi sample efficiency

4.1.3 Learning Rate Sensitivity

If our algorithms are able to learn effective representations quickly, one might expect that the use of large learning rates may be optimal in the simplest settings. Indeed, this is what we observe in the Figures below for the Frozen Lake and Cliff Walking environments. As we transition to environments with higher complexity (such as Taxi) we observe that using learning rates close to 1 has a marginally detrimental impact on performance. Overall, this analysis suggests to us that our algorithms are able to leverage learning rates that are larger than one might use for traditional policy gradient methods[70]

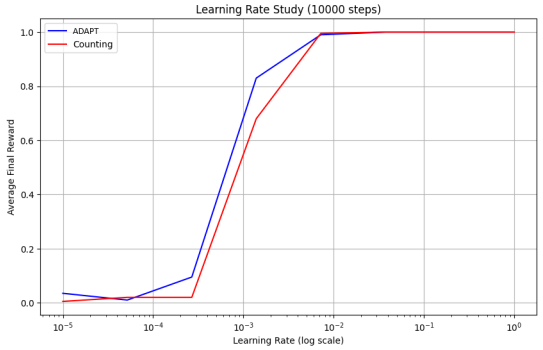


Figure 4.6: Frozen Lake

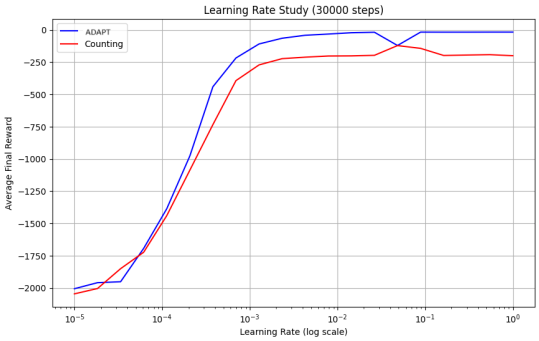


Figure 4.7: Cliff Walking

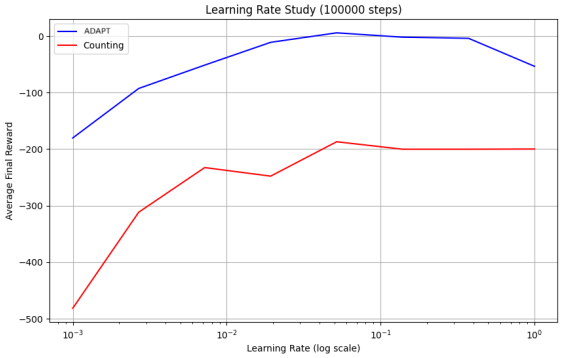


Figure 4.8: Taxi

4.2 Exploration versus Exploitation

We are now in a position to explore Algorithms 5 and 6 where we collect data in an online fashion by following the policy π. The success of our algorithmic class hinges critically on the relationship between batch size and the number of policy iteration epochs per batch. As demonstrated in our exploitation-only analysis, results are highly sensitive to hyperparameter choices, a challenge further compounded by the stochasticity introduced in the data collection process. To address this complexity, we employ Optuna,[71] a sophisticated hyperparameter optimization framework. Optuna leverages advanced techniques such as Tree-structured Parzen Estimators to efficiently navigate multidimensional parameter spaces, defining an objective function that maps hyperparameters to performance metrics and iteratively refining choices based on previous outcomes. This approach significantly outperforms traditional grid or random search methods,[72] adaptively focusing on promising regions to expedite the discovery of optimal configurations. While Optuna undoubtedly adds structure to our hyperparameter search efforts, the inherently complex loss landscapes typical in reinforcement learning can challenge its ability to tune multiple parameters simultaneously. Consequently, we restrict our use of Optuna to optimizing at most two hyperparameters concurrently, interpreting its results as a heuristic guide rather than an absolute truth. This conservative approach allows us to harness Optuna's strengths while maintaining a critical perspective on its limitations within the context of our research.

Specifically for our algorithm, Optuna generates contour plots depicting average reward as a function of batch size and policy iteration epochs. We illustrate this for the Frozen Lake environment, deliberately choosing a challenging scenario for both algorithms: learning a policy with only 1,000 steps and a learning rate of 0.01. The resulting performance differential is striking, as shown in Figure 4.9. ADAPT demonstrates remarkable robustness, achieving optimal policy performance across a broad range of hyperparameter combinations. In contrast, the counting algorithm exhibits highly localized areas of high performance, indicating an extremely narrow window of effective hyperparameter settings.

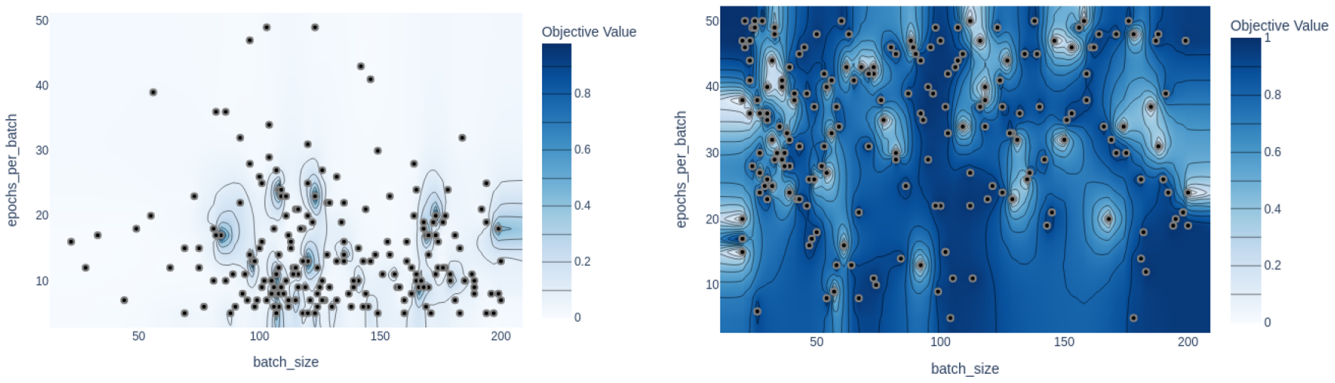


Figure 4.9: Contour plots of the counting (LHS) and ADAPT (RHS) algorithms

To contextualize our results within the broader landscape of reinforcement learning approaches, we benchmark our algorithm against a selection of policy gradient methods. We employ the stable-baselines3[70] implementations of Proximal Policy Optimization (PPO) and Advantage Actor-Critic

(A2C), representing state-of-the-art general-purpose algorithms capable of handling both discrete and continuous state-action spaces. Additionally, we include a custom implementation of a tabular REINFORCE algorithm parameterized with the squaremax policy, specifically tailored for tabular settings. This diverse set of comparisons allows us to evaluate our algorithm's performance against both versatile, widely-used methods and a specialized tabular approach, providing a well-rounded benchmark for performance.

We note that the dynamics of these environments are relatively simple and given enough samples, we would expect stable-baseline3 implementations of PPO and A2C to learn the optimal policy. Thus, to study comparative sample efficiency in the context of these toy environments, we set the total steps for training to be challenging to learn optimally. Our custom algorithms have been tuned with the Optuna library and performance has been averaged over 10 episodes. In environments where extremely large episode sizes are possible, we limit our evaluation function to episodes of size 500.

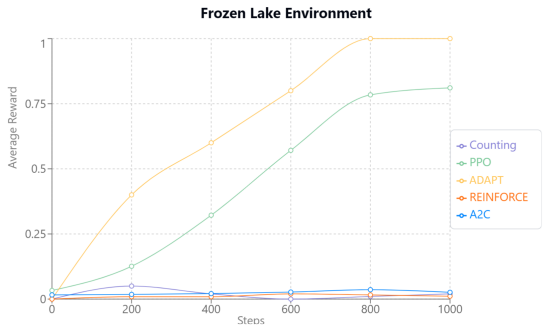


Figure 4.10: Frozen Lake Comparison

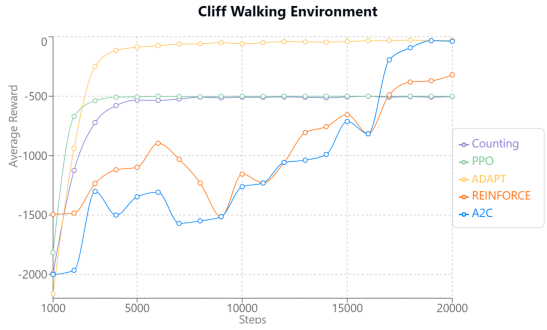


Figure 4.11: Cliff Walking Comparison

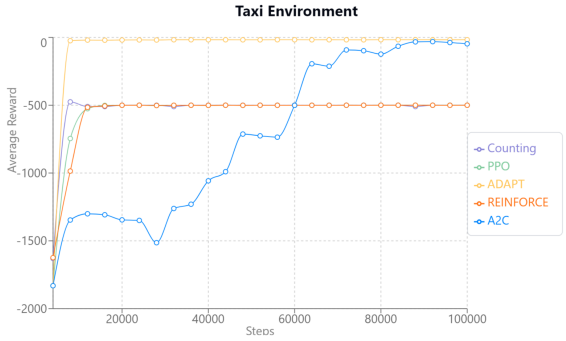


Figure 4.12: Taxi sample efficiency

From the Figures above, ADAPT learns in the most sample efficient manner across each of the three tasks. The Counting algorithm struggles in all three environments whilst stable-baseline algorithms have varying amounts of success in these settings: A2C learns the optimal policy in stable manner whilst PPO is unable to learn the optimal policy in the allocated time-frame. REINFORCE shows promising results in the Cliff Walking environment but is unable to replicate this performance in Frozen Lake and Taxi.

4.2.1 Deep Sea Exploration

The results thus far indicate that ADAPT effectively and efficiently learns representations of the underlying Markov Decision Processes (MDPs) in Frozen Lake, Cliff Walking, and Taxi environments. We now aim to evaluate our algorithm's capacity for deep exploration, a sophisticated form of exploration that considers long-term learning opportunities.[73] Just as an agent seeking to 'exploit' must consider the long-term consequences of its actions on cumulative rewards, an agent aiming to 'explore' must evaluate how its actions can position it to learn more effectively in future timesteps.

The Deep Sea problem[74] is implemented as an $N \times N$ grid, where each state is represented by a one-hot encoding. The agent initiates each episode in the top-left corner of the grid and descends one row per timestep, with episodes terminating after N steps when the agent reaches the bottom row. In each state, there is a random but fixed mapping between actions $\mathcal{A} = 0, 1$ and the transitions 'left' and 'right'. At each timestep, moving right incurs a small cost of $r = -0.01/N$, while moving left results in no cost ($r = 0$). However, if the agent manages to transition right at every timestep of the episode, it receives an additional reward of $+1$ upon reaching the bottom row. The agent is typically given 10,000 episodes of length N to learn the optimal policy of always moving to the right.

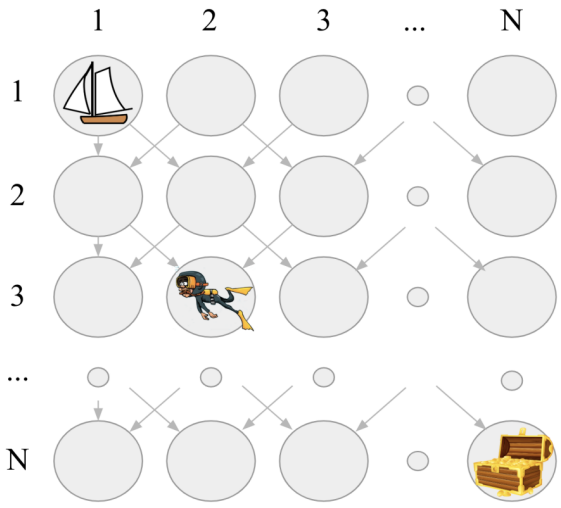


Figure 4.13: Visualisation of the Deep Sea Environment[74]

This environment presents a particularly challenging exploration problem for two primary reasons. First, the gradient of small intermediate rewards misleads the agent away from the optimal policy. Second, a policy that explores with uniformly random actions has only a 2^{-N} probability of reaching the rewarding state in any given episode. Given the algorithms we have developed thus far rely heavily upon gradient ascent methods, these characteristics of the environment make it an excellent testbed for evaluating deep exploration capabilities. Notably in the original paper, it is shown that powerful neural network based approaches such as A2C and DQN can perform consistently poorly even for small N.[74]

For our own experiments, we have replicated this framework as an OpenAI Gym customised environment; in line with the original paper specification,[74] agents will interact with the environment for 10,000 episodes (i.e. 10,000 * N steps) and average reward through the training process will be monitored. As N increases, it is instructive to study how the relationship between batch size and number of epochs of policy iteration in producing a stable learning process. Once again, we can produce contour plots to guide our hyperparameter search and also help us gauge the stability of our solutions. Let us start with N=5 and N=6:

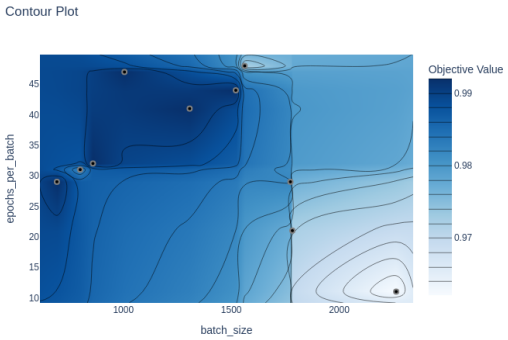


Figure 4.14: Counting (N=5) (lr=0.01)

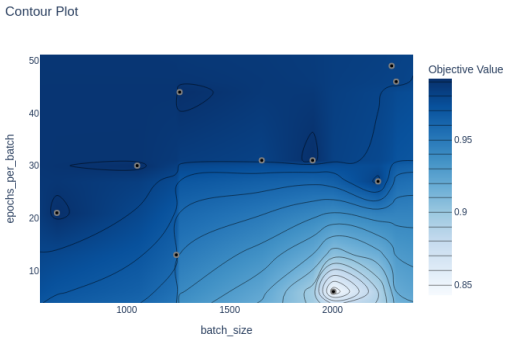


Figure 4.15: ADAPT (N=5)(lr=0.01)

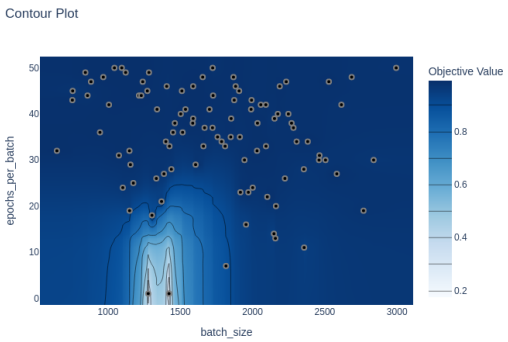


Figure 4.16: Counting (N=6)(lr=0.01)

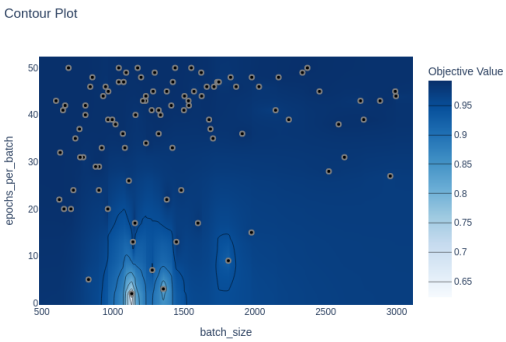


Figure 4.17: ADAPT (N=6)(lr=0.01)

Figures 4.14, 15, 16 & 17 reveal that the learning process remains largely consistent across the specified hyper-parameter search, regardless of batch size, in terms of converging to the optimal policy. As expected, using too few epochs per batch leads to sub-optimal results. Conversely, the model tends to perform better with a higher number of epochs per batch.

Moving onto N= 7,8,9, and 10:

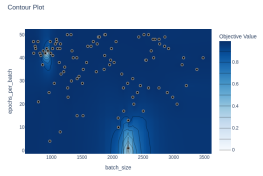


Figure 4.18: Counting (N=7)(lr=0.01)

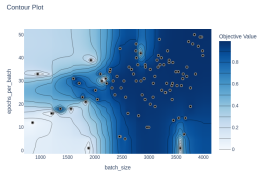


Figure 4.19: Counting (N=8)(lr=0.01)

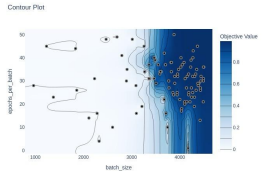


Figure 4.20: Counting (N=9)(lr=0.01)

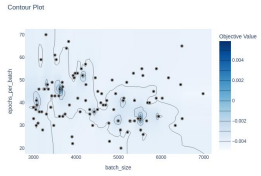


Figure 4.21: Counting (N=10)(lr=0.0015)

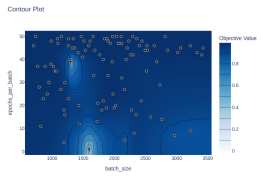


Figure 4.22: ADAPT (N=7)(lr=0.01)

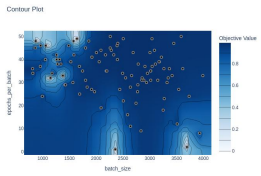


Figure 4.23: ADAPT (N=8)(lr=0.01)

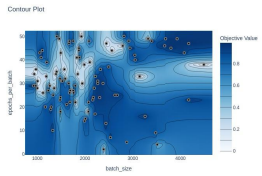


Figure 4.24: ADAPT (N=9)(lr=0.01)

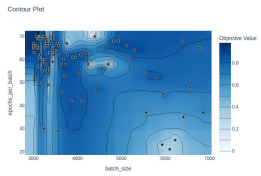


Figure 4.25: ADAPT (N=10)(lr=0.0015)

Figure 4.26: Deep Sea Exploration for N = 7,8,9,10

As N increases, we observe a rapid diminishing of efficacy for smaller batch sizes across both algorithms. This trend is particularly pronounced in the counting algorithm. By N=10, it reaches a critical point where even increasing the batch size fails to yield good hyperparameters, suggesting the algorithm has hit the limits of its deep exploration capabilities. ADAPT, while also affected, shows a more gradual decline. Our contour plots reveal an increasing sparsity of good hyperparameters for this algorithm, albeit at a slower rate than its counting counterpart. Interestingly, both algorithms consistently favor a batch size between 3000 and 4000, even when larger batches are available. This preference persists across different values of N, hinting at a potential sweet spot for batch size in these algorithms.

At this point, it is instructive to compare performance with our selection of benchmark algorithms before moving onto considering larger N. This can help us contextualise performance with respect to the state-of-the art algorithms. In the figures below, average rewards are computed over 10 runs and hyper parameters have been chosen through a mixture of manual tuning and Opuna-assisted tuning.

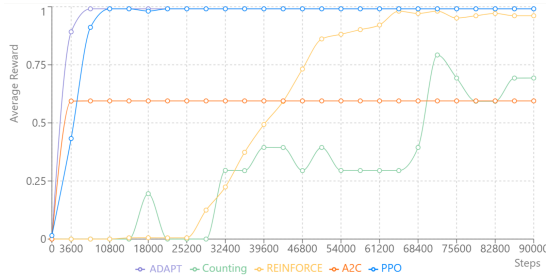


Figure 4.27: Average Reward Comparison (N=9)

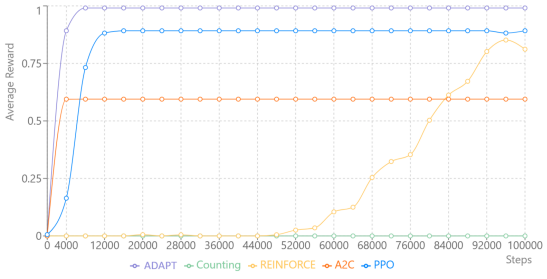


Figure 4.28: Average Reward Comparison (N=10)

For N=9, both the ADAPT and PPO algorithms consistently converge to the optimal policy across all 10 runs, demonstrating robust performance. This consistency holds true for ADAPT even when N increases to 10. However, PPO begins to show some variability in performance at N=10, indicating a potential sensitivity to increased problem complexity. A2C exhibits the most pronounced variability in performance. At N=10, in some runs it converges to the optimal policy even faster than ADAPT, showcasing its potential for rapid learning. However, in other cases, it struggles to converge at all, highlighting its instability in more complex environments.

REINFORCE, while less sample efficient than the aforementioned algorithms, demonstrates impressive stability in its convergence towards optimality. This consistency across runs suggests a trade-off between learning speed and reliability. The counting algorithm presents a more complex picture: at N=9, it shows a high degree of variance in performance, though it generally appears to be learning to act optimally. However, as our contour plots suggested, the algorithm fails to learn at all when N increases to 10, indicating a critical threshold in its capabilities.

Finally, let us consider ADAPT performance for N=11, 12:

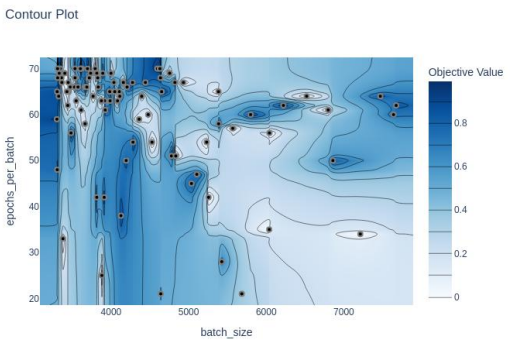


Figure 4.29: ADAPT (N=11)(lr=0.0012)

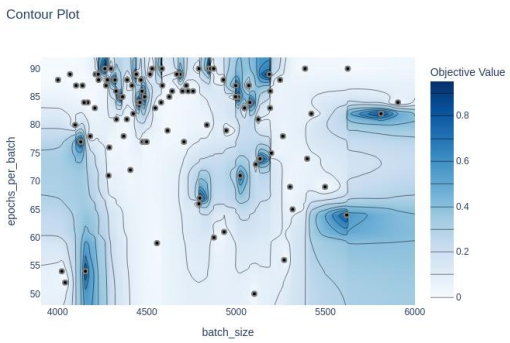


Figure 4.30: ADAPT (N=12)(lr=0.001)

The results for N=11 and N=12 reveal a significant narrowing in the range of effective hyperparameters for our problem. While we can still identify points that lead to learning an optimal policy, the landscape has become increasingly sensitive. Points sampled very close to these optimal regions can yield sub-optimal results, highlighting the precarious nature of hyperparameter selection in these more complex scenarios. This increased sensitivity is likely attributable, in large part,

to the exponentially increasing stochasticity introduced with each additional layer of the "deep sea" environment. As N grows, the problem space expands dramatically, making it progressively more challenging to navigate the hyperparameter landscape effectively. To properly contextualize our algorithm's performance in these more demanding settings, it's instructive to we examine how other methods fare under these conditions:

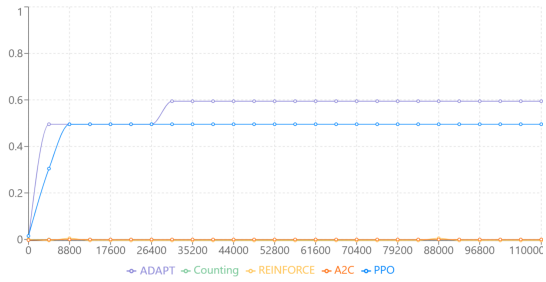


Figure 4.31: Average Performance Comparison (N=11)

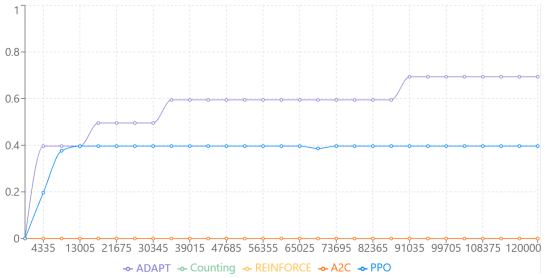


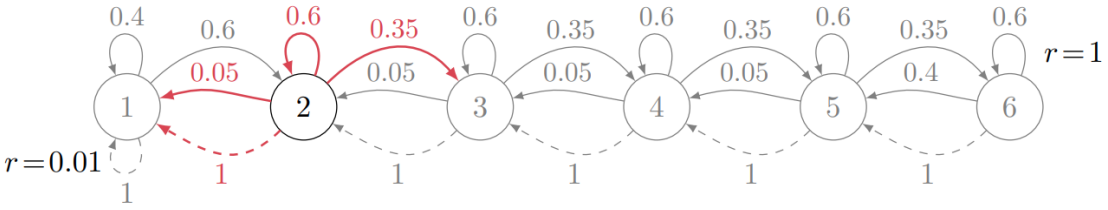
Figure 4.32: Average Performance Comparison (N=12)

ADAPT and PPO algorithms demonstrate the ability to learn the optimal policy, albeit inconsistently. We can see that for N=12 (over 10 runs), the ADAPT algorithm is able to learn the optimal policy 60% of the time by the end of training whereas PPO learns optimal policy 40% of the time. In contrast, A2C, REINFORCE, and Counting algorithms fail to show any meaningful improvement, with their performance remaining at or near zero throughout the experiment, suggesting an inability to learn this task effectively.

4.2.2 River Swim

So far, ADAPT has shown promise in its ability to perform deep exploration and learn representations across a variety of tasks. We introduce the River Swim task,[75] a task that combines the challenge of learn stochastic transition dynamics and deep exploration.

The River Swim environment consists of six states. The agent starts from the left side of the row and, in each state, can either swim left or right ($|\mathcal{A}| = 2$). Swimming to the right (against the current of the river) is successful with probability 0.35; it leaves the agent in the same state with a high probability equal to 0.6, and leads him to the left with probability 0.05 (see Figure 4.36). On the contrary, swimming to the left (with the current) is always successful. The agent receives a small reward when he reaches the leftmost state, and a much larger reward when reaching the rightmost state – the other states offer no reward.



- $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$, $\mathcal{A} = \{L, R\}$
- $\pi_L(s) = L$, $\pi_R(s) = R$

Figure 4.33: Illustration of the River Swim MDP[76]

In Figure 4.34, we compare the performance of various algorithms by measuring how quickly (if at all) they asymptotically converge to the optimal policy, which always selects action R. The expected reward for this policy has been approximated by averaging the rewards accumulated over 100,000 episodes. We set the total number of steps to 200,000, a relatively large value chosen to highlight the asymptotic behavior of each algorithm. Results have been averaged over five runs for consistency.

Both the ADAPT and Counting algorithms converge to the average reward of the optimal policy. PPO demonstrates rapid improvement but fails to reach optimality within the given time frame. A2C results are highly variable, while REINFORCE consistently fails to learn throughout the episodes.

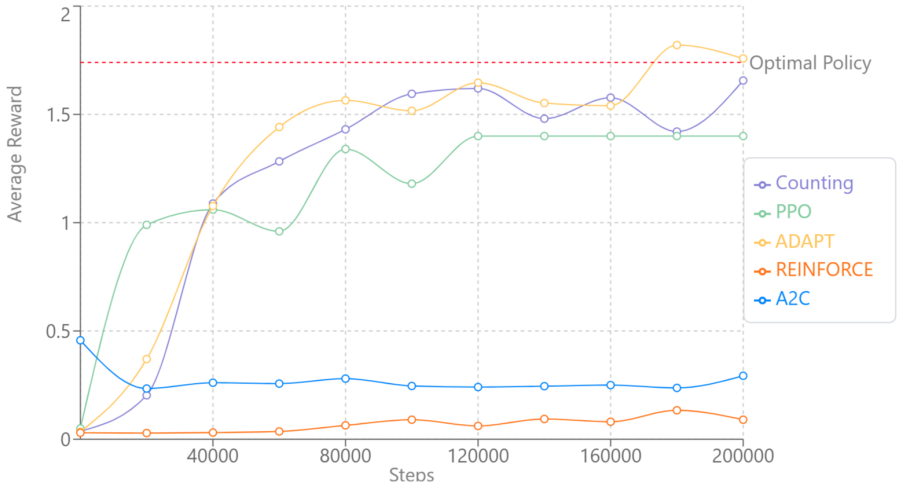


Figure 4.34: Illustration of the River Swim MDP[76]

4.3 Dense State Spaces

To study the dense setting (i.e. continuous state spaces), we have designed a simple continuous environment, **Temperature Control**. The task is simple: temperature is randomly initilisated at a float number between 15 and 30. The action space is discrete, with two possible actions: decreasing (0) or increasing (1) the temperature by 0.25°C. The agent's goal is to bring the temperature within a threshold of the target temperature (22.5°C ± 0.5°C). The environment provides a sparse reward signal, with the agent receiving a reward of -1 for each step taken and 0 at termination. Maximum step limit is set at 100 steps; this design choice prevents episodes from running indefinitely and encourages the agent to learn efficient control strategies.

4.3.1 Random Fourier Features (RFF) for Dense State Spaces

To address the challenge of dense state spaces, we employ Random Fourier Features (RFF),[77] a technique that approximates shift-invariant kernels via Fourier transforms. Bochner's theorem forms the basis of this approach, stating that any continuous, shift-invariant kernel $k(x, y)$ can be represented as the inverse Fourier transform of a non-negative measure $\mu(\omega)$:

$$k(x - y) = \int_{\mathbb{R}^d} e^{i\omega^T(x-y)} d\mu(\omega).$$

For real-valued kernels, we can express this as:

$$k(x - y) = \int_{\mathbb{R}^d} \cos(\omega^T(x - y)) d\mu(\omega).$$

By leveraging this theorem, we approximate the kernel function using a finite set of random features, mapping input data into a higher-dimensional space where linear techniques can be applied.

The kernel function $k(x, y)$ is approximated as:

$$k(x, y) \approx \varphi(x)^T \varphi(y),$$

where $\varphi(x)$ is a feature map consisting of random Fourier features.

To construct this feature map, we first sample D frequencies $\omega_1, \ldots, \omega_D$ independently from the probability distribution $p(\omega)$, which is the Fourier transform of the kernel $k(x-y)$. We also sample D bias terms $b_1, \ldots, b_D$ uniformly from $[0, 2\pi]$.
The feature map $\varphi(x)$ is then defined as:

$$\varphi(x) = \sqrt{\frac{2}{D}} \left[\cos(\omega_1^T x + b_1), \ldots, \cos(\omega_D^T x + b_D)\right].$$

This approximation converges to the true kernel function as D increases:

$$\lim_{D \to \infty} \varphi(x)^T \varphi(y) = k(x, y).$$

In the original paper,[77] it is demonstrated that the error of this approximation is bounded in expectation:

$$\mathbb{E}\left[\left(\varphi(x)^T \varphi(y) - k(x, y)\right)^2\right] \leq \frac{1}{D} \left(\int_{\mathbb{R}^d} p(\omega) d\omega\right)^2.$$

This RFF approach allows us to approximate complex kernel functions with a finite-dimensional feature representation, enabling efficient computation in high-dimensional spaces and facilitating the use of linear methods in traditionally nonlinear domains.

The choice of distribution for ω_i is tied to the kernel being approximated. For instance, when approximating the Gaussian (RBF) kernel, ω_i is sampled from a multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where $\Sigma = \sigma^2 I$ and σ represents the bandwidth of the Gaussian kernel. Similarly, for the Cauchy kernel, ω_i is drawn from a Cauchy distribution Cauchy$(0, \gamma)$, with γ serving as a scale parameter that controls the spread and approximation of the Laplace kernel involves sampling ω_i from a Laplace distribution Laplace$(0, b)$, where b denotes the scale parameter. This flexible approach to sampling allows for the approximation of a diverse range of kernel functions; Gaussian approximations tend to lend themselves to smooth approximations whereas Cauchy and Laplace are heavy-tailed.

RFFs can alternatively be formulated using both sine and cosine functions:[77]

$$\varphi(x) = \sqrt{\frac{2}{D}} \left[\cos(\omega_1^T x + b_1), \sin(\omega_1^T x + b_1), \ldots, \cos\left(\omega_{\frac{D}{2}}^T x + b_{\frac{D}{2}}\right), \sin\left(\omega_{\frac{D}{2}}^T x + b_{\frac{D}{2}}\right)\right].$$

The sine-cosine formulation of RFFs offers advantages over the cosine-only variant. While both converge to the true kernel function,[77] sine-cosine RFFs exhibit superior performance with fewer features,[78] lower variance in kernel approximation,[79] and tighter approximations for certain kernels.[78] However, cosine-only RFFs can be more computationally efficient,[80] presenting a trade-off between approximation quality and computational cost.

4.3.2 Temperature Control Experiments

In the context of our Temperature Control environment, the optimal choice of kernel for capturing the underlying MDP dynamics is not immediately apparent. To investigate this issue, Figures 4.35 & 4.36 focus on an exploitation-only scenario, examining the performance of Gaussian, Cauchy, and Laplace Random Fourier Feature (RFF) maps.

In our experimental set-up, we collect 10,000 steps of data, followed by 100 epochs of policy iteration with a learning rate of 0.01. Results are averaged over five independent runs. In our analysis, we distinguish between cosine-only implementations of RFFs (denoted as gaussian1, laplace1, and cauchy1) and their cosine-sine counterparts (gaussian2, laplace2, and cauchy2). Each kernel is initialised with 30 features

Figure 4.35 illustrates the progression of the objective function J across training epochs for each RFF variant. The resulting curves demonstrate stable learning trajectories across all algorithms, with consistent improvements over time. Complementing this, Figure 4.36 presents the average rewards achieved by the learned policies, providing evidence that the observed enhancements in the objective function values translate effectively into improved policy performance in the Temperature Control task. Notably, average rewards across kernels and average reward performance appears to stabilise between 40-60 epochs. Performance is grouped closely together but there does appear to be signs of slight performance deterioration as the number of epochs of policy deterioration increases. This may suggest possible instability in our solution, perhaps due to the high degree of expressive power in our feature representation.

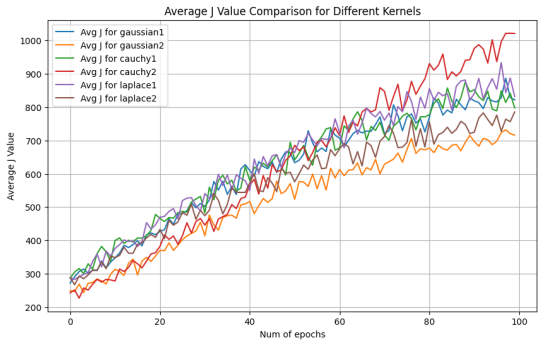


Figure 4.35: Average J versus number of epochs (10 runs)(lr=0.01)

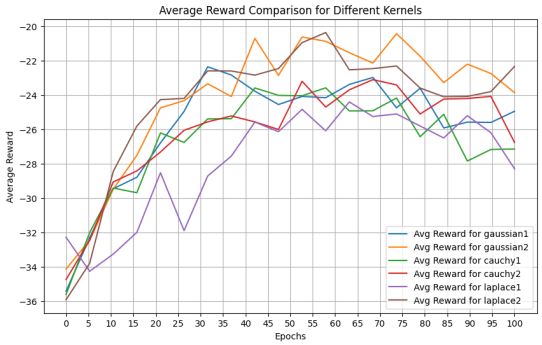


Figure 4.36: Average Reward versus number of epochs (10 runs)(lr=0.01)

The analysis above arguably did not provide a conclusive best choice of kernel for our problem. In light of this, we opt to proceed with the cosine-sine implementation of the Gaussian Kernel (gaussian2) for our subsequent experiments. This decision is motivated by the Gaussian kernel's widespread adoption and proven track record in diverse machine learning applications. Additionally in contemporary literature, cosine-sine appears to be the dominant implementation.

In Figures 4.37 and 4.38, we examine learning rate sensitivity of ADAPT when using Gaussian feature maps, averaging results over five runs. Once again, the experimental set-up collects 10,000 steps of data followed by 100 epochs of policy iteration. Figure 4.38 reveals that learning rates of

0.001 and 0.01 yield stable performance, while larger rates of 0.1 and 1 prove detrimental to learning stability. Interestingly, Figure 4.37 shows that the magnitude of increase in the objective function (J) can be misleading. While learning rates of 0.01 and 0.1 may appear to drive rapid gradient ascent on J, this does not always translate to a policy close to optimality. In contrast, learning rates of 0.001 and 0.01, though slower in improving J, result in policies that are consistently closer to the optimal solution.

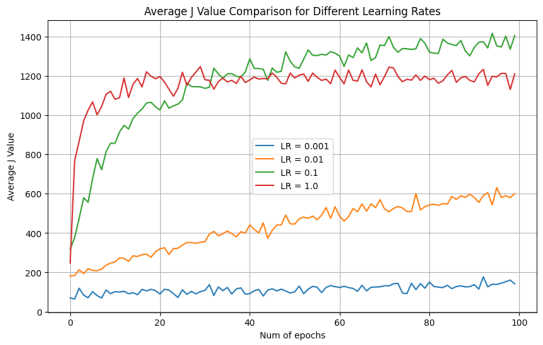


Figure 4.37: Average J versus number of epochs (5 runs)

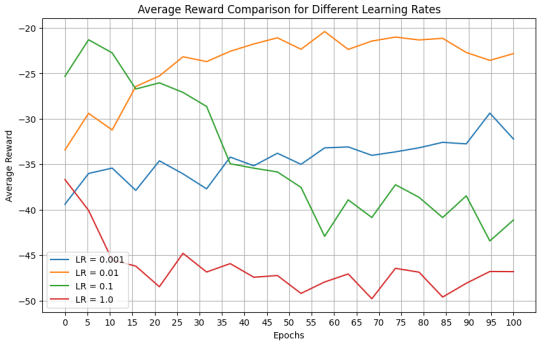


Figure 4.38: Average Reward versus number of epochs (5 runs)

We now focus on the online version of the ADAPT algorithm, which alternates between learning dynamics and policy optimization. Using Optuna, we generate a contour plot (Figure 4.39) showing average rewards of final policies as a function of batch size and epochs per batch. Our analysis reveals that batch size significantly impacts performance, with a notable drop for batches below 3000 samples. Early excessive policy iteration combined with small batches increases the risk of suboptimal end-episode behavior. However, for larger batch sizes, the number of policy updates has minimal impact on the learning process.

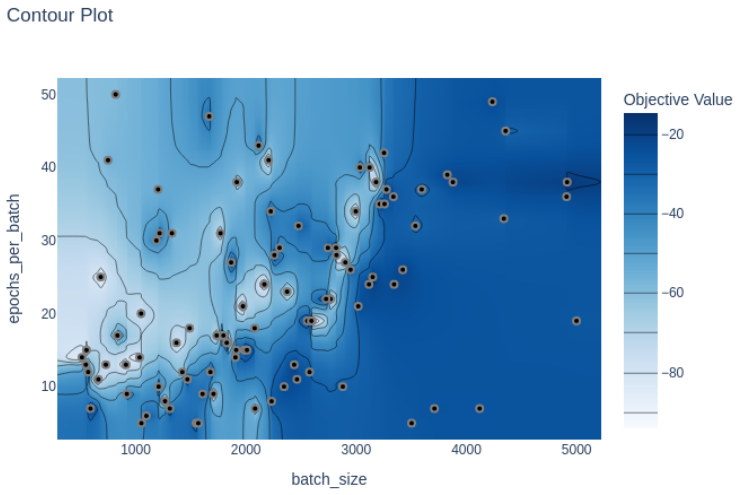


Figure 4.39: Temperature Control Environment: Contour Plot

Before comparing the performance of ADAPT to benchmark algorithms, it is important to first clarify what constitutes optimal behavior in this environment. The optimal policy involves moving to the right when the agent's state is less than 22.0 ($s < 22.0$) and moving to the left when the state exceeds 23.0 ($s > 23.0$). To empirically estimate the expected value of this policy, we averaged the rewards over 100,000 episodes, yielding a value of -12.7.

Figure 4.40 presents average rewards recorded at 2,000-step intervals throughout a 20,000-step training process, with results averaged over 10 runs. We observe that while both PPO and A2C exhibit learning progress, it is relatively sample-inefficient. Although these algorithms appear to be on a path toward convergence, ADAPT is able to achieve near-optimal performance within 10,000 steps.

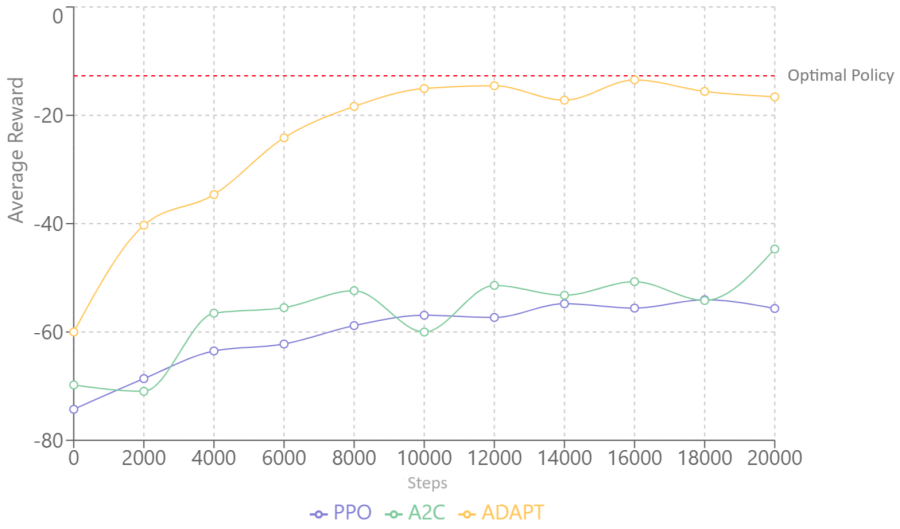


Figure 4.40: Temperature Control Environment: Performance Comparison

Chapter 5

Discussion

5.1 Summary

This thesis draws inspiration from the Optimism in the Face of Uncertainty (OFU) principle, a sub-field of RL that has developed a solid theoretical foundation to apply approximate dynamic programming (ADP) techniques for efficient exploration of discrete state-action MDPs. Our approach, ADAPT, brings fresh perspective to this principle by building on the blueprint of policy gradient methods instead, allowing for stochastic exploration of state-action spaces without relying on the environment-specific subroutines typically required by ADP methods.

In Chapter 2, we developed the theoretical foundations of both Approximate Dynamic Programming (ADP) and Policy Gradient methods, identifying that the objective function in both cases can be written as a Neumann series. We discussed why this approach in traditionally avoided in ADP settings and successfully developing one in the policy gradient space. We achieved this by leveraging the linear MDP assumption—whereby a feature map $\varphi(s, a)$ exists such that both the transition dynamics and reward function are linear with respect to $\varphi(s, a)$. By studying the properties of the linear MDP, we were able to pinpoint potential bottlenecks (such as the explicit calculation of the transition operator, T) and avoid them in our algorithm design. Ultimately, we derived an expression that seamlessly integrates the learning of environment dynamics with the agent's optimistic planning and policy iteration. Crucially, our algorithm made the assumption that the environment it is interacting with has discrete actions.

In our experiments, we incrementally demonstrated the full potential of the ADAPT algorithm. We began with classical tabular reinforcement learning tasks from OpenAI Gym[67], equipping the ADAPT algorithm with a "perfect" feature representation (the one-hot encoding of state-action pairs). In this setting, the algorithm quickly learned optimal policies. To push the boundaries of our algorithm's performance under perfect feature representation, we introduced a new class of deep exploration environments. These environments posed two primary challenges: (1) they introduced significant stochasticity into the tabular setting through a "layer" framework, exponentially increasing the sparsity of rewards, and (2) their reward structures misled naive policy gradient methods away from the optimal policy. Despite these challenges, ADAPT performed effectively across two levels of difficulty: Deep Sea Exploration (deterministic transitions) and RiverSwim

(stochastic transitions).

Finally, we explored settings where ADAPT does not have access to a "perfect" feature map. To test this, we designed a dense state-space environment, Temperature Control, where the agent must use discrete actions (Left or Right) to regulate the temperature and bring it to room temperature. We introduced Random Fourier Features (RFFs) to harness the representative power of kernels such as the Gaussian, Laplace, and Cauchy kernels, which have infinite-dimensional feature maps, by approximating them with finite feature maps.

A key design choice that we have made in all of our experiments is the manual selection / tuning of the feature representation for the problem at hand. In these settings, we were able to find information rich representations but make this algorithm truly generalisable, the learning of feature maps should be integrated into the algorithm structure.

5.2 Future Work

Our future work on ADAPT can be broadly divided into two areas: theoretical and experimental. From a theoretical standpoint, further work is needed to establish formal regret bounds for the ADAPT algorithm in tabular settings. This analysis is crucial, as it would provide a rigorous measure of the algorithm's performance relative to an optimal policy, offering valuable insights into its efficiency and sample complexity. Moreover, our approach currently relies on the assumption of discrete actions. A possible solution to this assumption is to use Bang-bang control's principle of using extreme values to discretizing continuous action spaces. This discretization aligns with the concept of solving minimum energy problems, as bang-bang control often yields time-optimal and, by extension, energy-optimal solutions in many systems. Informally, discretizing a continuous action space to extreme values (e.g., full acceleration or full brake) can simplify the learning process while potentially converging to optimal policies, especially in systems where rapid state transitions are beneficial. This approach effectively transforms a continuous control problem into a discrete decision-making task, reducing the complexity of the action space and potentially speeding up learning. Thus in certain RL environments, particularly those with dynamics akin to linear systems, it may be possible to learn optimal policies learned through this discretization. However, it's crucial to note that while this discretization can be powerful, it may not be suitable for all RL tasks, especially those requiring fine-grained control or involving complex, nonlinear dynamics where intermediate actions are crucial for optimal performance. Further work is required to study whether continuous actions can be handled more directly within the ADAPT algorithm, without relying on an additional discretization sub-routine.

A key design choice that we have made in all of our experiments is the manual selection / tuning of the feature representation for the problem at hand. In these settings, we were able to find information rich representations but make this algorithm truly generalisable, the learning of feature maps should be integrated into the algorithm structure. Using correct feature representation (i.e. one-hot encoding) in tabular settings gives us confidence in the robustness of ADAPT to compete with state-of-art. However, further experimentation is required to understand how compatible this algorithm is with modern deep learning architectures.

More specifically, future experiments for ADAPT could explore its efficacy in high-dimensional visual domains, such as Atari games.[67] This approach could leverage pre-trained deep learning vision architectures, like those trained on ImageNet, to extract rich feature representations from raw game frames. This two-stage process—feature extraction followed by ADAPT's dynamics learning and policy optimization—could potentially enhance sample efficiency and generalization across diverse game environments. By capitalizing on transfer learning principles, this method may bridge the gap between ADAPT's strengths in continuous control and the challenges posed by complex visual inputs. Such an investigation would not only extend ADAPT's applicability but also provide insights into the synergies between model-based reinforcement learning and deep visual representation learning. Key areas of inquiry would include comparative performance against end-to-end learning approaches, sample efficiency improvements, and the method's generalization capabilities across various Atari games.

Appendix A

Hyperparameters

A.1 Figures 4.1 & 4.2

Environment	Discount Factor	Learning Rate	Total Time Steps
Frozen Lake	0.99	0.01	5000
Cliff Walking	0.99	0.01	20000
Taxi	0.99	0.01	75000

A.2 Figures 4.3, 4.4 & 4.5

Environment	Discount Factor	Learning Rate	Optimizer
Frozen Lake	0.99	0.01	Adam
Cliff Walking	0.99	0.01	Adam
Taxi	0.99	0.01	Adam

A.3 Figures 4.6, 4.7 & 4.8

Environment	Discount Factor	Learning Rate	Optimizer	Total Time Steps
Frozen Lake	0.99	0.01	Adam	5,000
Cliff Walking	0.99	0.01	Adam	20,000
Taxi	0.99	0.01	Adam	75,000

A.4 Figure 4.10, 4.11, & 4.12

ADAPT

- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.1]

- Batch Size: Frozen Lake: [200], Cliff Walking: [1000], Taxi [4000]
- Epochs of Policy Optimisation: [40]

Counting Algorithm
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.1]
- Batch Size: Frozen Lake: [200], Cliff Walking: [1000], Taxi [4000]
- Epochs of Policy Optimisation: [40]

PPO
- For our implementation of the PPO algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation[1].

A2C
- For our implementation of the A2C algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation[2].

REINFORCE
- Policy Parameterisation (θ): Squaremax
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.01]

A.5 Figure 4.27, 4.28

ADAPT
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): N=9: [0.01], N=10: [0.0015]
- Batch Size: N=9: [3600], N=10: [4000]
- Epochs of Policy Optimisation: [62]

Counting Algorithm
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.0015]
- Batch Size: N=9: [0.01], N=10: [0.0015]
- Epochs of Policy Optimisation: [62]

PPO
- For our implementation of the PPO algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

A2C
- For our implementation of the A2C algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

[1] `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`
[2] `https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html`

REINFORCE
- Policy Parameterisation (θ): Squaremax
- Discount Factor (γ): [0.99]
- Learning Rate (α): N=9: [0.01], N=10: [0.0015]

A.6 Figure 4.31, 4.32

ADAPT
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): N=11: [0.0012], N=12: [0.001]
- Batch Size: N=11: [4000], N=12: [4335]
- Epochs of Policy Optimisation: [62]

Counting Algorithm
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): N=11: [0.0012], N=12: [0.001]
- Batch Size: N=11: [4000], N=12: [4335]
- Epochs of Policy Optimisation: [62]

PPO
- For our implementation of the PPO algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

A2C
- For our implementation of the A2C algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

REINFORCE
- Policy Parameterisation (θ): Squaremax
- Discount Factor (γ): [0.99]
- Learning Rate (α): N=11: [0.0012], N=12: [0.001]

A.7 Figure 4.34

ADAPT
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.001]
- Batch Size: [1019]
- Epochs of Policy Optimisation: [45]

Counting Algorithm
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.001]

- Batch Size: [1019]
- Epochs of Policy Optimisation: [45]

PPO
- For our implementation of the PPO algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

A2C
- For our implementation of the A2C algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

REINFORCE
- Policy Parameterisation (θ): Squaremax
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.001]

A.8 Figure 4.35 & 4.36

Random Fourier Features
- Total Timesteps: [10000]
- Number of Features: [30]
- Learning Rate (α): [0.01]
- Total Epochs of Policy Optimisation: [100]

A.9 Figure 4.37 & 4.38

Random Fourier Features
- Number of Features: [30]
- Total Timesteps 10000
- Total Epochs of Policy Optimisation: [100]

A.10 Figure 4.34

ADAPT
- Policy Parameterisation (θ): Softmax with temperature, $\tau = 1$
- Discount Factor (γ): [0.99]
- Learning Rate (α): [0.01]
- Batch Size: [4000]
- Epochs of Policy Optimisation: [45]

PPO
- For our implementation of the PPO algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

A2C
- For our implementation of the A2C algorithm, we utilized the Stable Baselines 3 library [70]. We used the default hyperparameters as specified in the library's documentation.

Appendix B

Bibliography

Bibliography

[1] H. A. Simon, *The sciences of the artificial*, English (MIT Press, Cambridge, MA, 1996).

[2] R. Bozulich, *The go player's almanac 2001*, English, Contains discussions on Go's complexity and its challenges for AI (Kiseido Publishing Company, Tokyo, 2001).

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: a survey", The International Journal of Robotics Research **32**, 1238–1274 (2013).

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., "Mastering the game of go with deep neural networks and tree search", nature **529**, 484–489 (2016).

[5] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: a deep reinforcement learning framework for news recommendation", in Proceedings of the 2018 world wide web conference (2018), pp. 167–176.

[6] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: a survey", IEEE Transactions on Intelligent Transportation Systems **23**, 4909–4926 (2021).

[7] H. van Hasselt, *Lecture 2: exploration and exploitation*, Lecture slides, COMP0089: Reinforcement Learning course, University College London, 2021.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction* (MIT press, 2018).

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", arXiv preprint arXiv:1707.06347 (2017).

[10] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning", arXiv preprint arXiv:1703.03864 (2017).

[11] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization", Methodology and computing in applied probability **1**, 127–190 (1999).

[12] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies", The Journal of Machine Learning Research **15**, 949–980 (2014).

[13] N. Hansen, "The cma evolution strategy: a tutorial", arXiv preprint arXiv:1604.00772 (2016).

[14] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks", in Artificial life, Vol. 15, 2 (MIT Press, 2009), pp. 185–212.

[15] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", Machine learning **8**, 229–256 (1992).

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in International conference on machine learning (PMLR, 2015), pp. 1889–1897.

[17] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming* (Athena Scientific, 1996).

[18] M. L. Puterman and M. C. Shin, "Modified policy iteration algorithms for discounted markov decision problems", Management Science **24**, 1127–1137 (1978).

[19] C. J. Watkins and P. Dayan, "Q-learning", Machine learning **8**, 279–292 (1992).

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning", Nature **518**, 529–533 (2015).

[21] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms", Advances in neural information processing systems **13** (2000).

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", International conference on machine learning, 1928–1937 (2016).

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

[24] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting", in Acm sigart bulletin, Vol. 2, 4 (1991), pp. 160–163.

[25] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: a survey", arXiv preprint arXiv:2006.16712 (2020).

[26] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning", arXiv preprint arXiv:1907.02057 (2019).

[27] M. Deisenroth and C. E. Rasmussen, "Pilco: a model-based and data-efficient approach to policy search", Proceedings of the 28th International Conference on Machine Learning (ICML-11), 465–472 (2011).

[28] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models", Advances in Neural Information Processing Systems **31** (2018).

[29] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization", arXiv preprint arXiv:1802.10592 (2018).

[30] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: model-based policy optimization", Advances in Neural Information Processing Systems **32** (2019).

[31] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models", arXiv preprint arXiv:2005.05960 (2020).

[32] P. Ball, J. Parker-Holder, A. Pacchiano, K. Choromanski, and S. Roberts, "Ready policy one: world building through active learning", in International conference on machine learning (PMLR, 2020), pp. 591–601.

[33] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Tuning bandit algorithms in stochastic environments", in International conference on algorithmic learning theory (Springer, 2007), pp. 150–165.

[34] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning", in European conference on machine learning (Springer, 2006), pp. 282–293.

[35] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation", arXiv preprint arXiv:1810.12894 (2019).

[36] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning", Journal of Machine Learning Research **11**, 1563–1600 (2010).

[37] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation", Advances in neural information processing systems **29** (2016).

[38] S. Frederick, G. Loewenstein, and T. O'donoghue, "Time discounting and time preference: a critical review", Journal of economic literature **40**, 351–401 (2002).

[39] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming* (John Wiley & Sons, New York, 1994).

[40] D. Silver, *Lecture 2: markov decision processes*, Lecture Notes in Reinforcement Learning, Retrieved from https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf, 2015.

[41] I. M. Gelfand, "Normierte ringe", Matematicheskii Sbornik **9**, 3–24 (1941).

[42] E. Çinlar, *Probability and stochastics*, Vol. 261, Graduate Texts in Mathematics (Springer, New York, 2011).

[43] R. A. Horn and C. R. Johnson, *Matrix analysis*, 2nd (Cambridge University Press, Cambridge, 2012).

[44] C. D. Meyer, *Matrix analysis and applied linear algebra* (SIAM: Society for Industrial and Applied Mathematics, Philadelphia, 2000).

[45] D. S. Watkins, *Fundamentals of matrix computations*, 2nd (John Wiley Sons, Hoboken, NJ, 2004).

[46] A. S. Householder, "The theory of matrices in numerical analysis", Courier Corporation (1964).

[47] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales", Fundamenta Mathematicae **3**, 133–181 (1922).

[48] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "Numerical linear algebra for high-performance computers", Society for Industrial and Applied Mathematics (1988).

[49] M. Dubois and C.-W. Chin, "The effects of memory hierarchy on vector algorithm performance", in Proceedings of the 1996 acm/ieee conference on supercomputing (1996), 37–es.

[50] V. Volkov and J. W. Demmel, "Benchmarking gpus to tune dense linear algebra", in Sc'08: proceedings of the 2008 acm/ieee conference on supercomputing (IEEE, 2008), pp. 1–11.

[51] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach* (Elsevier, 2011).

[52] R. Bellman, *Dynamic programming* (Princeton University Press, 1957).

[53] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization", arXiv preprint arXiv:1412.6980 (2014).

[54] Y. Wu, M. Ren, R. Liao, and R. Grosse, "Understanding and improving the adam optimizer", arXiv preprint arXiv:1804.00957 (2018).

[55] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch", in Nips 2017 workshop on automatic differentiation (2017).

[56] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", Advances in neural information processing systems **12** (1999).

[57] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control", International conference on machine learning, 1329–1338 (2016).

[58] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients", Neural networks **21**, 682–697 (2008).

[59] R. J. Williams and J. Peng, "Function optimization using connectionist reinforcement learning algorithms", Connection Science **3**, 241–268 (1991).

[60] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor", International Conference on Machine Learning, 1861–1870 (2018).

[61] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters", Proceedings of the AAAI Conference on Artificial Intelligence **32** (2018).

[62] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: return decomposition for delayed rewards", in Advances in neural information processing systems (2019), pp. 13566–13577.

[63] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A closer look at deep policy gradients", in International conference on learning representations (2018).

[64] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration", in International conference on learning representations (2017).

[65] A. Pacchiano, P. J. Ball, J. Parker-Holder, K. Choromanski, and S. Roberts, *Towards tractable optimism in model-based reinforcement learning*, 2021.

[66] M. G. Azar, I. Osband, and R. Munos, *Minimax regret bounds for reinforcement learning*, 2017.

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

[68] Y. Li, "Deep reinforcement learning: an overview", arXiv preprint arXiv:1701.07274 (2017).

[69] D. E. Kirk, *Optimal control theory: an introduction* (Dover Publications, 2004).

[70] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-baselines3: reliable reinforcement learning implementations*, 2021.

[71] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: a next-generation hyper-parameter optimization framework*, 2019.

[72] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", Journal of Machine Learning Research **13**, 281–305 (2012).

[73] I. Osband, B. V. Roy, D. Russo, and Z. Wen, *Deep exploration via randomized value functions*, 2019.

[74] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. V. Roy, R. Sutton, D. Silver, and H. V. Hasselt, *Behaviour suite for reinforcement learning*, 2020.

[75] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for markov decision processes", Journal of Computer and System Sciences **74**, Learning Theory 2005, 1309–1331 (2008).

[76] A. L. Mohammad Ghavamzadeh and M. Pirotta, *Lecture notes from aaai 2020 tutorial*, `https://rlgammazero.github.io/docs/2020_AAAI_tut_part1.pdf`, 2020.

[77] A. Rahimi and B. Recht, *Random features for large-scale kernel machines*, 2007.

[78] D. J. Sutherland and J. Schneider, *On the error of random fourier features*, 2015.

[79] F. X. Yu, A. T. Suresh, K. M. Choromanski, D. N. Holtmann-Rice, and S. Kumar, *Orthogonal random features*, 2016.

[80] Z. Li, J.-F. Ton, D. Oglic, and D. Sejdinovic, *Towards a unified analysis of random fourier features*, 2019.