Optimistic Exploration for Deep Reinforcement Learning

By

Sam Lobel

B.A., University of Pennsylvania, 2015

Thesis

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Computer Science at Brown University

PROVIDENCE, RHODE ISLAND

May 2026

This dissertation by Sam Lobel is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

George Konidaris, Advisor

Recommended to the Graduate Council

Date _____ _____

Michael Littman, Reader

Date _____ _____

Ronald Parr, Reader

Approved by the Graduate Council

Date _____ _____

David P. Lindstrom, Dean of the Graduate School

Curriculum Vitae

Sam Lobel grew up in Highland Park, New Jersey. He graduated from the University of Pennsylvania in 2015 with a degree in Physics, and rudimentary programming skills. For the next four years, he worked as a software engineer at Forward Philadelphia and MetaMetrics while gaining exposure to machine learning research outside of work, most productively in Lawrence Carin's large and inviting research group at Duke University. In 2019, Sam began his Ph.D. at Brown University, advised by George Konidaris, where his research largely focused on exploration methods for deep reinforcement learning. During his Ph.D., he did a three-month internship at Sony AI, developing exploration techniques for application in PlayStation video games. He is currently employed as a research scientist at Nuro, where he researches and applies reinforcement learning in the context of autonomous vehicles.

Sam's research was supported by the National Science Foundation (Graduate Research Fellowship #2040433, grants #1717569 and #1955361, and CAREER award #1844960), the Defense Advanced Research Projects Agency (grant W911NF1820268), the Office of Naval Research (contracts N00014-17-1-2699 and N00014-21-1-2200, and grant N00014-22-1-2592), and the Army Research Office (grant W911NF2210251).

For my wife, my parents, and my brother

Acknowledgments

A Ph.D. is paradoxically both a solitary pursuit, and takes a village. Thank you to the many people who made these last six years ones of (mostly) joy and growth.

Looking back to when I joined this group, I can't believe how little I knew about reinforcement learning. George, thank you for taking this risk on me. A special thank you for your high-level vision of this field, which often helped keep me out of the weeds and remember that RL is more than just an optimization problem. And in the early days, for doing "gradient ascent on my interests" with me, to find a research direction that gave me what I was looking for—mathematical rigor and real-world impact. Thank you for your mentorship, kindness, enthusiasm, and the fantastic research group you built.

Thank you also to my other faculty mentors. Ron, diving into the details with you on the whiteboard each week were some of the times I got to think most carefully at Brown. I enjoyed all of our projects a lot, even the ones that didn't make it on the page. Michael, thank you especially for the perspective you've shared with me on what makes research with lasting impact.

Thank you also to my closest friends and colleagues (both!) throughout these six years, Akhil, Rafael, and Saket. Akhil, I've never had more fun working (or been more productive) than pair programming CFN with you. Rafa, you were a great temporary roommate and permanent friend. And Saket, among many things I appreciate our many midday walk-and-talks, that were as good for our minds as they were our bodies.

As long as a Ph.D. is, it can't all be academic. The love and support of my family sustained me through this. Megan, you made East Providence a home for me. In these six years we've had three moves, one new dog, a marriage certificate, and too many academic degrees. Thank you for supporting my clickity clacks through it all. Mom and dad, thank you for a wonderful childhood that let me grow up with my curiosity preserved and encouraged. Joe, thank you for smoothing the roadblocks by doing it all first, even the Ph.D.

To the many others at Brown and outside who helped me learn, think, and enjoy myself, thank you as well.

Abstract of *Optimistic Exploration for Deep Reinforcement Learning*, by Sam Lobel, Brown University, May 2026.

Reinforcement learning agents face a challenge unique within machine learning: they must gather their own training data. However, the most informative parts of many environments are quite hard to reach. This makes strategic exploration critical to solve challenging RL problems, especially those with only a sparse reward signal. In tabular settings, "optimism in the face of uncertainty" provides a principled framework for exploration, driving agents to investigate unknown parts of the world by assuming they may be highly valuable. But applying these ideas in large domains requires adapting them to the deep reinforcement learning setting, where it is challenging both to measure uncertainty and to enforce optimism in its face. In this thesis, I develop new methods for quantifying novelty in high-dimensional observation spaces, refine existing theoretical bounds on how uncertainty affects value estimation, and apply the insights from this bound to construct more accurate optimistic value functions in deep reinforcement learning. Taken together, these contributions substantially advance our understanding of how to optimistically explore with modern deep reinforcement learning systems.

Contents

List of Figures

CHAPTER 1

Introduction

Reinforcement learning is the study of learning to interact, through interaction. As such, the two key differences between reinforcement learning and other types of machine learning are as follows: the agent must collect its own training data, and the choices an agent makes now influence which parts of the environment it visits in the future. With the wrong choice of interaction strategy, the agent may never gather the type of useful data it requires to actually learn to solve the task at hand. This is the key challenge in so-called hard exploration problems, where parts of the environment are challenging to reach by chance, or the task only has a sparse reward signal to guide learning progress. In these settings, an agent must engage in directed exploration to make meaningful learning progress.

In simple tabular problems, efficient exploration is well understood and primarily done through optimism in the face of uncertainty, or optimistic exploration. In short, by assuming that unknown parts of the environment are highly valuable, the agent is incentivized to explore them. By visiting these areas, the agent reduces the uncertainty around them, and may discover highly rewarding parts of the environment. Key to this strategy is ensuring that the optimistic estimate of value is, with high likelihood, greater than the true value under the optimal policy.

This thesis ports ideas from tabular reinforcement learning to do better exploration in the deep reinforcement learning setting. First, I introduce an effective way to quantify novelty in the high-dimensional setting, generalizing the idea of count-based bonuses to the deep reinforcement learning setting. Next, I interrogate whether bonus-based exploration is the correct choice for optimistic exploration. I derive an improved bound for the simulation lemma, a foundational result in reinforcement learning, which better quantifies how uncertainty can influence value functions. This improvement comes from more carefully reasoning about how a misspecified transition function interacts with the process of value iteration. Motivated by this result, I demonstrate an improvement to bonus-based exploration that uses the same novelty estimates to create more accurate optimistic value functions. I present a general optimistic framework based on this update rule along with results demonstrating improvement in a suite of challenging continuous control exploration problems. Taken together, this work advances our understanding of how to instantiate optimistic exploration in the deep reinforcement learning context.

CHAPTER 2

Background and Related Work

The general aim of this thesis is to investigate how we can produce interactive agents that self-motivate discovery of the world around them. We do this using the formalism of reinforcement learning, a general framework for describing sequential decision-making tasks. I begin by describing the basics of reinforcement learning, and then move on to introduce optimistic exploration as a method for intrinsic motivation.

2.1 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 2018) is a conceptual framework that can be used to describe a variety of decision-making problems. The three central components of reinforcement learning are the *agent*, the *environment*, and the *task*. The agent is the decision-making component of reinforcement learning. Given its history of interaction with its environment, an agent performs actions that in turn influence the environment in which it resides. We refer to the method by which an agent selects actions as its *policy*, π. The environment is the part of the decision process that evolves according to its internal state as well as the agent's chosen actions.

The task in reinforcement learning is represented by a reward signal: the environment

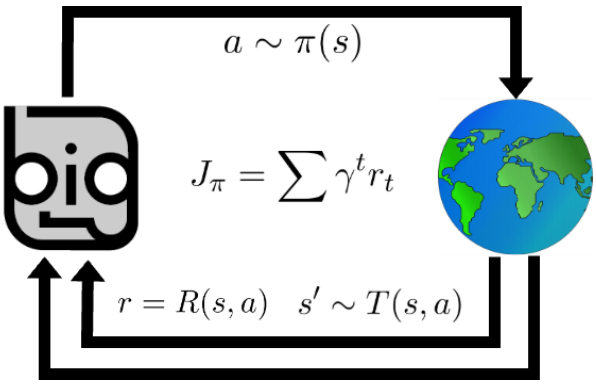

Figure 2.1: The agent-environment interaction loop. The agent observes the world, and chooses an action according to its internal policy with which to interact with the world. The agent's performance is measured by J, the discounted cumulative return.

and the agent's actions together determine the magnitude of this reward. The interactive process of environment, task, and agent is detailed in Figure 2.1. The general goal of reinforcement learning is to learn a policy that performs a given task effectively, meaning it collects a large amount of reward throughout its environmental interactions. There are multiple ways to measure the quality of a policy given the expected trajectory of rewards; a common approach is to geometrically discount rewards by how far in the future they appear. Thus, the quality of a policy can be measured by its expected return as follows:

$$G^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

where t is the timestep at which each reward r_t is received, and $\gamma \in [0, 1]$ controls the agent's preference between short and long-term rewards. In reinforcement learning, the aim of learning is generally to produce an agent with a policy that achieves the highest possible return. As specified so far, there are many forms that an environment, an agent, and a task can take; we next describe the *Markov Decision Process*, which is a common setting for creating and analyzing reinforcement learning algorithms.

2.1.1 Markov Decision Process

The *Markov Decision Process* (MDP) is a simple and widespread formulation of decision processes used in reinforcement learning. A MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$. $\mathcal{S}$ is the state space, which in the MDP framework is the set of all possible observations an agent can make. $\mathcal{A}$ is the action space, the set of all possible actions an agent can take. $T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is the transition function: given a state and action, the transition function determines the distribution of states the environment will transition to next. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function: given a state and action, the reward function produces a scalar quantity. As stated above, γ is a discount factor that determines an agent's time-preference over future rewards. In this thesis, we consider the *discrete-time* MDP formulation, where at each timestep t the agent observes the environment state s_t and chooses an action a_t, resulting in a reward r_t and a new state s_{t+1}.

As implied by its name, the transition and reward functions in an MDP satisfy the *Markov* property: the current state and action are sufficient statistics of the transition and reward functions. Thus, an agent's history of interactions provides no extra useful information for predicting future transitions and returns. Formally:

$$T(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_0, a_0, ...s_t, a_t),$$

and similarly

$$R(s_t, a_t) = R(s_0, a_0, ...s_t, a_t).$$

Likewise, in the MDP framework it is generally sufficient to consider *reactive* policies, where the actions an agent chooses are purely a function of its most recent state. Since transitions and rewards are functions of only the most recent state and actions, knowledge of past states and actions has no bearing on future returns.

2.1.2 Value Functions and Q Functions

When considering a Markov Decision Process and reactive policies, the current state is likewise a sufficient statistic of future return. Thus, we can faithfully refer to the *expected value* of a given state under a policy:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_t \sim \pi(s_t)\right].$$

This is known as the *value function*. Similarly, the *Q-function* is the expected value of taking some action a from state s, and afterwards following the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a, a_t \sim \pi(s_t)\right].$$

From the concept of value functions, we can construct an important definition, that of an *optimal policy*. An optimal policy, π^*, is a policy that, for all states, achieves at least as high value as all other policies:

$$\forall \pi, s: \quad V^{\pi^*}(s) \geq V^{\pi}(s); \quad \forall \pi, s, a: \quad Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a).$$

Under the Markov assumption, both V and Q admit helpful recursive definitions:

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}\left[R(s,a) + \gamma \sum_{s'} T(s'|s,a) V(s')\right]$$

and

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \mathbb{E}_{a' \sim \pi(s')}[Q(s',a')].$$

In words, the current value is equal to the immediate reward, plus the discounted future value. The above two relations of Q-functions and value functions are known as the *Bellman equation* (Bellman, 1966), and suggest forms of recursive improvement, known as

bootstrapping, where an inaccurate Q or value function can be made more precise with knowledge of the environment. While other methods for training reinforcement learning algorithms exist, such as policy gradient and policy search methods, this thesis largely focuses on methods based on the Bellman equation, such as *value iteration* and *temporal difference* (TD) methods (Sutton, 1988).

2.1.3 Value Iteration and Model-Based Reinforcement Learning

The Bellman equation can be used to learn an optimal policy, either through interactive data or a model of the environment. A simple strategy for deriving a policy from a Q-function is choosing the action with the highest Q-value:

$$\pi^Q(s) = \arg\max_a Q(s,a),$$

or the equivalent when using value functions:

$$\pi^V(s) = \arg\max_a \Big[R(s,a) + \gamma \sum_{s'} T(s'|s,a)V(s')\Big].$$

With direct access to T and R, we can perform *value iteration*:

$$V_{k+1}(s) \;=\; \max_a \Big[R(s,a) + \gamma \sum_{s'} T(s'|s,a)V_k(s')\Big] \;=\; V^{\pi^{V_{k+1}}}(s).$$

In the simplest reinforcement learning setting, *tabular MDPs* (to be expanded upon later), value iteration is guaranteed to produce the optimal policy:

$$\lim_{k\to\infty} \pi^{V_k} = \pi^*.$$

As stated above, value iteration requires access to the true generative models of T and R. When the above procedure is instead performed using learned estimates of T and R, it is called *model-based reinforcement learning*.

2.1.4 Q-Learning and Model-Free Reinforcement Learning

Reinforcement learning methods that do not explicitly construct estimates of T and R are called *model-free reinforcement learning*. For example, in *Q-learning* (Watkins and Dayan, 1992), a Q function is updated using data sampled from interacting with the environment, without constructing a proxy model of T and R. Let Q_k be the Q-function after k interactive steps, s_k be the current state, a_k be the current action (not necessarily selected from a specific policy), and let s_{k+1} be the result of taking action a_k from state s_k. α is the step size, which possibly varies with k. Then, we can update the Q function as follows:

$$Q_{k+1}(s_k, a_k) = (1 - \alpha)Q_k(s_k, a_k) + \alpha\big(R(s_k, a_k) + \gamma \max_{a'} Q(s_{k+1}, a')\big). \tag{2.1}$$

Under certain assumptions on α, and provided that each of the finitely-many state-actions is visited/chosen infinitely many times, this converges to the optimal Q-function:

$$\lim_{k\to\infty} \pi^{Q_k} = \pi^*.$$

Since Q-learning generates value estimates without at any point constructing an explicit model of T or R, it is known as a *model-free* reinforcement learning method. Furthermore, since it can be trained with data from any policy (provided the above assumptions are satisfied), it is also an *off-policy* reinforcement learning algorithm.

Now that we have introduced the building blocks of reinforcement learning, we proceed to discuss efficient instantiations of these ideas in tabular reinforcement learning.

2.2 Tabular Reinforcement Learning

The simplest settings for reinforcement learning are so-called *tabular* environments. In a tabular MDP, the state and action spaces are finite and enumerable. Thus, all relevant quantities such as reward functions, transition functions, Q-functions, value functions, and policies can be stored in a simple table that maps states and actions to the relevant quantity. Furthermore, in the most general case the transition and reward functions are *unstructured*: learning about one state's transition function does not tell you anything about another state's transition function. This is in contrast to the intuitive definition of state, perhaps defined by an agent's position and velocity, in which there is some concept of locality and generalization between neighboring states. Because of their simplicity, tabular environments are amenable to theoretical analysis. Algorithms developed within this framework are interesting in their own right, and provide intuition on the interactive learning process in more complicated domains.

2.2.1 Probably-Approximately Correct

What does it mean to learn efficiently? The *Probably-Approximately Correct* (PAC) framework provides a precise answer to this question (Valiant, 1984). In the PAC framework, sample complexity is measured by the number of training samples required for a learning algorithm to return a near-optimal solution with high probability. Following Strehl and Littman (2008), we generalize PAC-complexity to the reinforcement learning setting as follows.

Let $\mathcal{M} = \langle \mathcal{M}, \mathcal{A}, R, T, \gamma \rangle$, let $\epsilon \geq 0$ be an error threshold, and let δ be a probability. Let $c = (s_1, a_1, \ldots s_t, a_t)$ be a sequence of states and actions executed by the algorithm. The **PAC sample complexity** of some algorithm $\mathcal{H}$ with respect to c is the number of timesteps t such that the policy π_t at time t is not ϵ-optimal from state s_t, i.e. when $V^{\pi_t}(s_t) < V^*(s_t) - \epsilon$. Likewise, an algorithm $\mathcal{H}$ is **PAC-efficient** if, for any ϵ, δ, the sample and computational complexity of $\mathcal{H}$ are less than some polynomial in the quantities

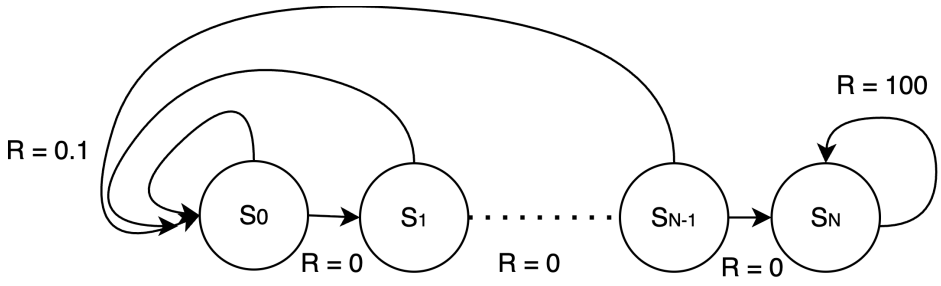

Figure 2.2: A combination-lock MDP illustrating the importance of directed exploration. Taking the "up" action results in a small positive reward at any timestep. Taking the "right" action results in no reward, until the final state under which it results in a very large return.

$|\mathcal{S}|, |\mathcal{A}|, 1/\epsilon, 1/\delta, 1/(1-\gamma)$ with probability at least $1-\delta$.

Intuitively, the idea of PAC sample complexity quantifies how much interaction an agent requires to learn an optimal policy. As described earlier, given a perfect model of a tabular environment, an agent needs exactly zero interactions to learn a near-optimal policy: methods such as value iteration suffice. However, in the learning setting, an agent must use the data it has collected to estimate value. A common approach, as described above, is to estimate the parameters of the transition and reward function directly, and then use value iteration to transform these estimates into a value function and policy.

Why is directed exploration important? In some MDPs, random exploration requires an exponential number of interactions to even visit every state, let alone learn an optimal policy. For example, in Figure 2.2 an "up" action at any time brings the agent back to the start state. Only by executing the "right" action N times in a row can the agent reach the large reward at the end (which is exponentially unlikely to happen by chance). The agent must reach the final state at least once to have any chance of learning the optimal policy, meaning that random exploration is not PAC-efficient. On the other hand, the simple procedure of methodically working through each action in each state explores the MDP much more quickly.

This problem highlights another central challenge in PAC learning: the *exploration-*

exploitation tradeoff. In Figure 2.2, the agent can easily collect a small amount of reward by choosing the *up* action prematurely, but doing so prevents it from ever reaching the highly rewarding state s_n. The agent must temporarily forgo exploiting known rewards in favor of exploring the environment if it is to eventually learn the optimal policy. However, in order to converge to a near-optimal policy, at some point an agent must stop taking exploratory actions.

2.2.2 Optimism in the Face of Uncertainty

Optimism in the face of uncertainty (OFU), also known as optimistic exploration, is a powerful technique for balancing exploration and exploitation (Brafman and Tennenholtz, 2002). This concept is the unifying principle behind a wide variety of theoretically sound exploration algorithms (Auer et al., 2002; Brafman and Tennenholtz, 2002; Jin et al., 2018), and serves as the cornerstone of this thesis. Under OFU, when an agent is uncertain about the quality of some action, it assumes the decision has the highest value consistent with its current experiences. Thus by taking high-valued actions, either the agent will discover a new highly rewarding action sequence, or reduce its uncertainty through its experiences and thus not be attracted to that pathway in the future. OFU naturally balances exploration and exploitation: as uncertainty is whittled away, the agent more closely follows its empirical estimate of action-values.

2.2.3 Count-Based Methods for Optimism

How does one construct a relationship between uncertainty and optimism? To measure uncertainty, in tabular environments we appeal to concentration inequalities such as Hoeffding's Inequality (Hoeffding, 1963), which provide bounds on the likelihood that an empirical estimate of some quantity deviates from its true value by more than some amount. In the tabular setting, using Hoeffding's inequality we can construct a direct relationship between the number of (s, a) interactions and how wrong our estimates of

transition and reward ($\hat{T}$ and $\hat{R}$) may be. For example, after interacting with each (s, a) n times, and expressing $T(\cdot \mid s, a)$ as a vector where $T_{s'}(s, a) = Pr(s' \mid s, a)$, we can produce the following bounds (Jiang, 2018): with probability $(1 - \delta/2)$,

$$\epsilon_T = \|T(s,a) - \hat{T}(s,a)\|_1 \leq 2\sqrt{\frac{1}{2n} \ln \frac{2|S||A|2^{|S|}}{\delta}} \approx \mathcal{O}(\sqrt{\frac{|S|}{n}}) \tag{2.2}$$

and

$$\epsilon_R = |R(s,a) - \hat{R}(s,a)| \leq R_{MAX}\sqrt{\frac{1}{2n} \ln \frac{4|S||A|}{\delta}} \approx \mathcal{O}(\sqrt{\frac{1}{n}}). \tag{2.3}$$

In words, the L_1 error in transition and reward estimates scales inversely with the square root of visitation count. The first algorithm to produce a PAC bound on learning in MDPs (E^3) connects this result to value estimation error in the form of the *simulation lemma*, which bounds the estimated value of executing a policy on misspecified MDP compared with the original (Kearns and Singh, 2002). If for all s, a, T and R are known to some accuracy specified by ϵ_T and ϵ_R, then

$$|Q^\pi(s,a) - \hat{Q}^\pi(s,a)| \leq \frac{\epsilon_r}{1-\gamma} + \frac{\epsilon_T}{2(1-\gamma)^2}. \tag{2.4}$$

These two results taken together suggest a concrete form of optimism in the face of uncertainty. The first quantifies how uncertainty scales with visitation count (in proportion to $\sqrt{\frac{1}{n}}$), and the second produces an optimistic upper bound on the true value of executing a policy in the original MDP compared to the value under an MDP where T and R are empirically estimated. Kearns and Singh (2002) cleverly combined these two results to produce a PAC policy by using an empirical estimate of value when ϵ_T and ϵ_R are small enough for the state and subsequent states, and assuming $V(s) = V_{MAX} = 1/(1-\gamma)$ otherwise. Later in this thesis, I derive a tighter form of the simulation lemma, both as an interesting result in its own right as well as for motivation for a new methodology of exploration in deep reinforcement learning.

2.2.4 Bonus-Based Exploration

Strehl and Littman (2008) derives a much simpler PAC exploration algorithm by simply augmenting the task-specific reward function with a *bonus* proportional to $\sqrt{\frac{1}{n}}$, without modifying the transition function. This formulation, called *model-based interval estimation with exploration bonus* (MBIE-EB) is the theoretical basis for many contributions to exploration in deep reinforcement learning. Since MBIE-EB uses an empirical estimate of the transition function, it can be approximated through simply sampling experience from those the agent has seen, instead of explicitly constructing a model of the environment. As we will see later, this is particularly amenable to the replay-buffer based training methodology on which much of deep reinforcement learning is based.

2.3 Exploration in Deep Reinforcement Learning

I now describe methods by which we can scale the ideas described in the prior section to solve much larger and more challenging learning problems.

2.3.1 Generalizing Reinforcement Learning with Function Approximation

The learning paradigms presented in tabular reinforcement learning are based on powerful and general rules, but are constrained by how we represent the quantities of interest. Representing the reward and transition function (and thus, the Q function and policy) in tabular form involves storing quantities about each state and action as individual entries. Problems we would like to solve frequently have large or possibly infinite state spaces, on which these representations do not scale gracefully. However, the central ideas of bootstrapping and temporal difference learning can be more generally applied by replacing these tabulations with function approximators.

For example, we can represent Q values using a function approximator parameterized

by θ that consumes states and actions and outputs estimates of Q values: $Q_\theta(s, a) \to \mathbb{R}$. Since Q values are not stored separately for each state, we can no longer simply replace Q values with ones closer to their bootstrapped estimates (Eq. 2.1) (Watkins and Dayan, 1992). Instead, we can search for parameters that better satisfy the Bellman equation, for example by using stochastic gradient descent (SGD) (Rumelhart et al., 1986). Let (s_t, a_t, r_t, s_{t+1}) refer to a transition observed in the environment. Then, we can compute bootstrapped target values:

$$y_t = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a'), \tag{2.5}$$

and update our parameterization so as to output values closer to this target:

$$\theta \leftarrow \theta + (y_t - Q_\theta(s_t, a_t))\nabla\theta Q_\theta(s, a). \tag{2.6}$$

With proper choice of function approximator, and careful handling of optimization difficulties, this update rule can produce Q values that are accurate across large input spaces, and generalize well to states that do not have an exact match in the agent's history.

2.3.2 Deep Learning

Deep learning has seen remarkable success over the last two decades at solving complex problems over high-dimensional input spaces, such as language, images, and video (LeCun et al., 2015; Krizhevsky et al., 2012; Brown et al., 2020; Karpathy et al., 2014). Crucially, prior systems of function approximation generally required the input space to in some sense succinctly represent the individual features that are important to solving a problem. Deep learning, by contrast, has the ability to extract meaningful representations from high-dimensional inputs simply by modifying the neural network's weights using SGD. One of the strongest benefits of deep learning is its ability to consume and learn from enormous amounts of data (LeCun et al., 2015; Brown et al., 2020). In this thesis, we

investigate the application of deep learning in the reinforcement learning context.

2.3.3 Deep Reinforcement Learning

In *deep reinforcement learning*, or DRL, an agent uses deep neural networks to represent crucial components of the training algorithm, such as the value function, the policy, or the environment model. This approach has been particularly successful at handling high-dimensional state and action spaces, where powerful function approximation is necessary to make sense of these complicated inputs and outputs. An early impressive result in DRL (Mnih et al., 2015) was *Deep Q-Learning*, which produced strongly-performing policies on the *Atari 2600 Benchmark Suite* (Bellemare et al., 2013) directly from image-based observations of gameplay, and using only the score of the game as a reward signal. This work achieved human-level control on the majority of games in the suite, using significantly more powerful function approximators and fewer hand-crafted features and policies than prior methods. The primary contributions of this work were the introduction of methods to stabilize the learning of Q functions using neural networks, through the introduction of tools such as target networks and replay buffers (Mnih et al., 2015). However, the original work notes that the method struggles on certain games, in particular ones such as *Montezuma's Revenge* where the reward signal is very sparse. To make progress on these tasks, some form of directed exploration is required, since the reward signal on its own is not enough to guide the agent to strong performance.

2.3.4 Optimistic Exploration in Deep Reinforcement Learning

As in tabular reinforcement learning, optimistic exploration is a powerful technique for incentivizing exploration in the absence of an informative reward signal. The underlying principle is that by constructing a measurement of uncertainty, the agent can modify its Q values such that it is attracted to these uncertain regions of its environment. In tabular reinforcement learning, it is generally straightforward to quantify uncertainty, often simply

by observing the number of times the agent has encountered a particular state-action (Brafman and Tennenholtz, 2002; Kearns and Singh, 2002; Strehl and Littman, 2008). However, in deep reinforcement learning, estimating uncertainty is less straightforward. In general, the state spaces in many problems that require deep learning to solve are far too large to simply count over, making it non-trivial to quantify how "familiar" a particular state-action is to the agent.

Two things are needed to instantiate optimistic exploration in the deep reinforcement learning context: a way to quantify uncertainty, and a method for incorporating these estimates into value functions. Bonus-based exploration has emerged as the dominant method for incorporating uncertainty into value; below, we describe this framework in more detail and introduce some of the most popular approaches.

2.3.5 Bonus-Based Exploration in Deep RL

Above we see how function approximation allows us to generalize the important quantities in RL to large problems. The same idea can be used to instantiate bonus-based exploration without having to enumerate the uncertainty or novelty of each possible state. To do so, we can modify the reward in Equation 2.5 to construct optimistic targets:

$$y_t = r_t + \mathcal{B}(s_t, a_t) + \gamma \max_{a'} Q_\theta(s_{t+1}, a'), \tag{2.7}$$

where $\mathcal{B}$ is a function that produces exploration bonuses. As such, uncertainty is incorporated into Q-values through the bootstrapping process, and by following its Q function the agent naturally balances exploration and exploitation. Bonus-based exploration is perhaps the most common approach to exploration in deep RL (Bellemare et al., 2016; Burda et al., 2019; Pathak et al., 2017; Tang et al., 2017), due to the simplicity and effectiveness of simply modifying the reward attached to each training sample. Below we present a selection of popular or historically important exploration bonuses:

Pseudocounts Among the most influential bonus-based approaches are those that generalize the classical notion of state visitation counts to complex, high-dimensional domains through the use of *pseudocounts* (Bellemare et al., 2016; Ostrovski et al., 2017). A pseudocount is a quantity that behaves like a count in that it predictably increases with visitation, but is computed using function approximation and thus generalizes over the entire state space. Pseudocounts are most commonly produced via density modeling. The agent trains a density model over its observations in an online fashion using the stream of its interaction data: the amount the density changes after each update can be transformed into a $\sqrt{1/n}$ pseudocount bonus. Density-based pseudocounts were the first exploration method to make meaningful progress on sparse-reward tasks in the Atari Benchmark suite, such as Montezuma's Revenge (Bellemare et al., 2016). However, the technique is limited by the difficulty of training these models. Accurately modeling density over high-dimensional inputs is a challenging task, and the restriction to online training poses an additional constraint. Nonetheless, pseudocounts are a scalable, principled exploration strategy that has produced impressive results when used for bonus-based exploration. Later in this thesis, I introduce a novel method for estimating pseudocounts that does not rely on density models.

Prediction Error Another common approach to creating bonuses is through the error of some function approximator. Similar to pseudocounts, the assumption here is that the more a function approximator is trained on some state, the lower the error on that state will be. Techniques such as Random Network Distillation (Burda et al., 2019) train a function approximator to predict the features produced by a randomly-initialized neural network, and uses the prediction error as an exploration bonus. Intrinsic Curiosity Module (Pathak et al., 2017), or ICM, learns a latent representation over states, and trains a forward model to predict the next-state's representation from the current state and action. The error in this prediction task similarly can be used as an exploration bonus.

Uncertainty-Based Approaches Another approach to bonus-based exploration is

through estimating the uncertainty of forward models of the environment. Pathak et al. (2019), and Stadie et al. (2015) encourage exploration by training an ensemble of forward models, and producing an intrinsic reward based on the variance of their outputs. Plou et al. (2024) trains a Bayesian neural network to approximate transition dynamics, and from this derives an uncertainty measurement from transitions which is used as an intrinsic reward.

2.3.6 Looking Forward

This chapter introduces reinforcement learning as a general framework for sequential decision making, and directed exploration as a necessary ingredient for learning in sparse-reward environments. The unifying principle of the exploration methods described above is *optimism in the face of uncertainty*, whereby an agent seeks out experiences that most inform its learning. Taking inspiration from the decades of work in tabular reinforcement learning theory, the remainder of this dissertation examines how this principle can be instantiated in the deep learning setting. I develop a toolkit for efficient exploration across a range of challenging domains, and in doing so advance our understanding of how uncertainty influences the learning process in both tabular and deep reinforcement learning.

CHAPTER 3

Flipping Coins to Estimate Pseudocounts for Exploration

As described in section 2.3.5, pseudocount bonuses are a particularly promising approach to exploration in deep reinforcement learning, as they generalize the principled approach of count-based bonuses to the deep reinforcement learning context. Previous methods have equated the problem of estimating pseudocounts to the canonical machine learning problem of *density estimation*: the more informative a given state is to the model while learning, the higher the reward for reaching it (Bellemare et al., 2016; Ostrovski et al., 2017).

While providing the first way to estimate pseudocounts, the relationship between counts and probability densities in Bellemare et al. (2016) exists only when the density model meets the following restrictions (Ostrovski et al., 2017):

- It must output normalized probability densities, which precludes many powerful density models.
- It must be *learning-positive*, which means that the probability density of a state must increase when it is encountered by the density model again.

- It must be updated exactly once per state visitation, precluding common techniques such as batching.

These requirements make density-based pseudocounts challenging to implement and sensitive to network architecture and hyperparameters such as learning rate. In light of these restrictions, it is tempting to forego count-based exploration in favor of other novelty estimates based on dynamics (Pathak et al., 2017) or observation (Burda et al., 2019) prediction errors. But prediction-error-based methods do not tell us how an exploration bonus should decay with repeated visits. Furthermore, they do not enjoy the same theoretical foundations afforded by count-based methods (Strehl and Littman, 2008; Azar et al., 2017; Jin et al., 2018). For instance, count-based bonuses lead to near-optimal policies even when environments are highly stochastic; no such guarantees exist for prediction-error-based methods.

In this chapter, we put forth a method for computing pseudocounts that does not rely on online density modeling. Our core insight is that a state's visitation count can be derived from the sampling distribution of Rademacher trials made every time a state is encountered. We train a neural network, the *Coin Flip Network* (CFN), to predict the average of this sampling distribution; by solving this supervised learning problem, we output the inverse of the state's visitation count. Unlike other pseudocount methods (Ostrovski et al., 2017), we do not place any restrictions on the type of function approximator or the procedure used to train it, thereby allowing a practitioner to select the model architecture best suited to their input modality.

We show that in visual versions of Gridworld (Allen et al., 2021) and Taxi (Dietterich, 1998), our method can recover the ground-truth counts while other pseudocount methods cannot. We then evaluate our algorithm on a variety of challenging sparse-reward continuous control problems; in these environments, we outperform baseline actor-critic (Haarnoja et al., 2018) and random network distillation (Burda et al., 2019), with the largest gains on the most challenging exploration domains. On the image-based exploration

benchmark problem Montezuma's Revenge (Bellemare et al., 2013), we outperform baseline Rainbow (Hessel et al., 2018) and are competitive with state-of-the-art exploration algorithms (Ostrovski et al., 2017; Burda et al., 2019), which arguably have been overfit to this domain (Taiga et al., 2020). Finally, we show that increasing transition noise in Gridworld and Montezuma's Revenge causes RND's performance to degrade more rapidly than CFN's, as predicted by the theoretical properties of count-based exploration.

3.1 Coin Flip Networks

A pseudocount $\mathcal{N} : \mathcal{S} \to \mathbb{R}^+$ generalizes the notion of counts to large state spaces and can be used to quantify the novelty of a state (Bellemare et al., 2016). Specifically, visiting a state s affords the agent a novelty bonus of $\mathcal{B}(s) = \frac{1}{\sqrt{\mathcal{N}(s)}}$; we will use a neural network, the *Coin Flip Network* (CFN) f_ϕ, to directly predict this count-based exploration bonus.

To learn f_ϕ, we set up a simple regression problem:

$$f_\phi = \arg\min_\phi \mathbb{E}_{(s_i,y_i)\sim\mathcal{D}}\Big[\mathcal{L}(s_i, y_i)\Big], \tag{3.1}$$

where $\mathcal{L}$ is the mean-squared error loss function and $\mathcal{D}$ is a dataset of state-label pairs. Our main insights relate to the design of the labels y_i in such a way that the resulting function f_ϕ will map each state to its count-based exploration bonus $\frac{1}{\sqrt{\mathcal{N}(s)}}$.

3.1.1 Counts from the Rademacher Distribution

To construct the labels y_i from Eq 3.1, we first notice that averaging multiple samples from a random distribution approaches the distribution's mean at a predictable rate.

Consider the fair coin-flip distribution $\mathcal{C}$ over outcomes $\{-1, 1\}$. Imagine flipping this coin n times, and averaging the results into z_n. In expectation, the average is 0, but any given trial likely results in a non-zero value for z_n. Generally, for all n, the second moment

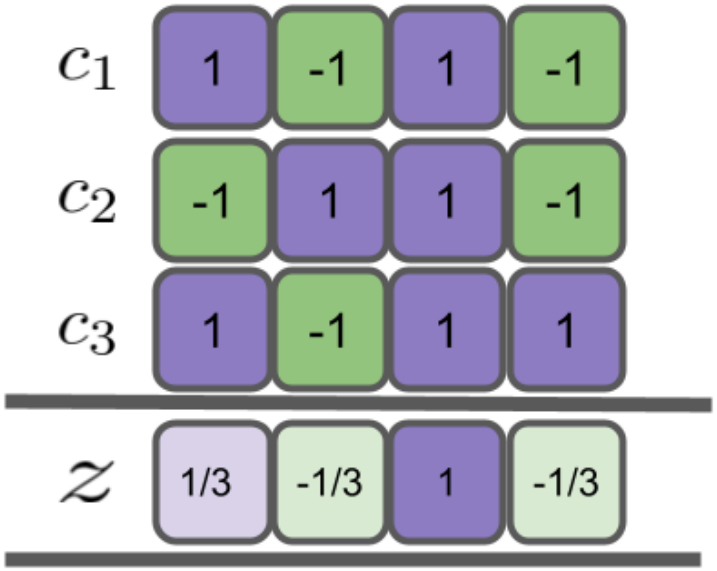

Figure 3.1: Illustration of our counting method for a state s with true count 3. Each occurrence of s creates new coin-flip vectors c_1, c_2, c_3. We average these vectors into z and compute the squared magnitude. Dividing this by the number of coin flips $d = 4$ yields the inverse count $1/\mathcal{N}(s)$.

of z_n is related to the inverse count:

$$\mathcal{M}_2(z_n) = \mathbb{E}[z_n^2] = \sum_i Pr(z_n = i) * i^2 = 1/n. \tag{3.2}$$

This property is a simple restatement of the fact that $\mathbb{E}[z_n^2]$ is the *variance of the sample mean* of the coin-flip distribution, which is well-known to scale inversely with sample size. In fact, this scaling is shared between all zero-mean unit-variance distributions, not just the coin-flip distribution. However, using this distribution leads to the *lowest variance estimates of inverse-counts* out of the entire class of matching distributions. We prove these two facts in Appendix A.1.1 and A.1.2 respectively.

3.1.2 Estimating Counts for a State via Multiple Coin Flips

An additional way to lower the variance of this estimator is to average together multiple estimates of z_n^2: by flipping d coins each time, we get d independent estimates of $\frac{1}{n}$, which reduces variance by a factor of $\frac{1}{d}$ (see Appendix A.1.4).

Consider the contrived case of an MDP with a single state and imagine that we draw a random sample from $\mathcal{C}^d$ each time that state is visited. Equation 3.2 implies that the squared magnitude of the averaged vectors is an unbiased estimator of d/n; this is also illustrated in Figure 3.1. Of course, we do not need a novel method to count the number of elements in a list; but this informs our eventual method for producing bonuses in general MDPs.

3.1.3 Predicting Counts for Multiple States

Having solved the uninteresting problem of extracting counts for a single state MDP, we will now generalize to datasets with multiple occurrences of multiple states. As label y_i for state s_i in Eq 3.1, we generate a d-dimensional random vector $\mathbf{c_i} \sim \{-1, 1\}^d$; this leads to the following simplification of Equation 3.1:

$$\begin{aligned} f_\phi^*(s) &= \arg\min_\phi \sum_{i=1}^{|\mathcal{D}|} \|\mathbf{c_i} - f_\phi(s_i)\|^2 \\ &= \arg\min_\phi \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{d} (c_{ij} - f_\phi(s_i)_j)^2. \end{aligned} \tag{3.3}$$

When there are multiple instances of the same state s in $\mathcal{D}$, each occurrence will be paired with a different random vector. In that case, f_ϕ^* cannot learn a perfect mapping from states to labels, and instead minimizes $\mathcal{L}$ by outputting the *mean* random vector for all instances of a given state:

$$f_\phi^*(s) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{c_i}.$$

Combining Equation 3.2 and 3.3 relates the solution f_ϕ^* to the inverse count:

$$
\begin{aligned}
f_\phi^*(s) &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{c_i} \\
\implies \mathbb{E}\Big[\frac{1}{d}\|f_\phi^*(s)\|^2\Big] &= \frac{1}{d}\sum_{j=1}^{d}\mathbb{E}\Big[\Big(\sum_{i=1}^{n}\frac{c_{ij}}{n}\Big)^2\Big] \\
&= \frac{1}{d}\sum_{j=1}^{d}\mathbb{E}\Big[z_n^2\Big] \\
&= \frac{1}{d}\sum_{j=1}^{d}\frac{1}{n} = \frac{1}{n}.
\end{aligned}
$$

Thus, by training f_ϕ on the objective described in Equation 3.3 we can map states to approximate count-based bonuses:

$$
\boxed{\mathcal{B}(s) := \sqrt{\frac{1}{d}\|f_\phi(s)\|^2} \approx \frac{1}{\sqrt{\mathcal{N}(s)}}}. \tag{3.4}
$$

3.1.4 Generalizing Outside the Training Data

The optimization procedure described so far will eventually derive the correct visitation counts for states in the training data (given a powerful function approximator and sufficient training iterations). But as the agent interacts with the environment, how will the CFN bonus generalize to states absent from the training data? Although in practice we represent f_ϕ as a neural network, to gain intuition on generalization, we mathematically examine the case when f_ϕ is linear. In this case, the bonus for a state s is a linear combination of the bonuses assigned to the right singular vectors of the training data; more discussion and proof is in Appendix A.2. Intuitively, the singular vectors of the training data take on the role of unique states: instead of tracking how many of each state visitation there are, a linear f_ϕ records how much of each singular vector is present in total in the dataset. Interestingly, when states are represented using one-hot vectors, the resulting solution to Equation 3.3 recovers tabular counts.

3.1.5 Improving Predictions for Novel States

As stated above, a learning architecture with infinite capacity would learn the exact inverse-count for each unique state in the training data. But finite capacity and training time imply that the network will not learn this mapping exactly. Next, we will propose two ways to guide our network to favorably trade off prediction errors among states in the dataset: first, we will use *prioritized sampling* to preferentially learn the novelty of rare states; second, we will use *optimistic initialization* to assign a pseudocount of 1 to novel states newly added to the replay buffer.

Prioritizing Novel States

Since training to convergence at every time step is not feasible, we update f_ϕ once every time step on a minibatch of states drawn from a replay buffer. Revisiting the optimization problem from Equation 3.3, we note that a state s with count n will appear in uniform sampling n times more often than a state visited only once. This would make f_ϕ focus too much on learning the bonus of high-count states, which are uninteresting from an exploration perspective. To remedy this problem, we would like to assign more weight to low count states by sampling them with greater probability (Schaul et al., 2015).

Of course, we do not have access to the true count during training; so, we approximate this procedure by prioritizing by our current *estimate* of inverse-count:

$$\text{priority}(s) \leftarrow \frac{1}{d}\|f_\phi(s)\|^2 \approx \frac{1}{\mathcal{N}(s)}.$$

Though this prioritization changes the importance of different states relative to each other, all instances of the same state will be sampled in equal proportion. Therefore, solving the prioritized version of Equation 3.3 still outputs the unbiased average of a state's coin-flip vectors (and therefore the correct pseudocounts).

Prioritizing in this way introduces another difficulty: if a state has been recently added

to the replay buffer, it has not appeared in many gradient updates and thus we cannot trust our estimate of its count. To combat this, we also prioritize sampling by the number of times, $n_{\text{updates}}(s)$, we have sampled s in the past. We combine both these prioritization schemes using an α-weighted sum (we use $\alpha = 0.5$):

$$\text{priority}(s) = \alpha \left(\frac{1}{n_{\text{updates}}(s)} \right) + (1 - \alpha) \frac{1}{d} \|f_\phi(s)\|^2. \tag{3.5}$$

The n_{updates} term weighs different instances of the same state differently, but its effect on prioritization disappears quickly during training, so it does not influence the fixed point either.

Optimistic Initialization of Bonus

Consider a state s that CFN has not been trained on yet. The exploration bonus $\mathcal{B}(s)$ will be determined by CFN's generalization properties. If s is very different from the other states that CFN has already been trained on, then $\mathcal{B}(s)$ tends to be close to 0, when in fact we would like the pseudocount of novel states to be initialized to 1.

We achieve this optimistic initialization using a *random prior* (Osband et al., 2018):

$$f_\phi(s) = \hat{f}_\phi(s) + f_{\text{prior}}(s),$$

where f_{prior} is the output of a frozen and randomly initialized neural network, and $\hat{f}_\phi$ is the trainable component of CFN. We use a running mean and variance to normalize the prior so that $\mathbb{E}_{s \sim \mathcal{D}}[f^{(i)}_{\text{prior}}(s)^2] = 1$ over all output dimensions $i \in \{1, .., d\}$. This ensures that if $\hat{f}_\phi(s) = 0$ on a novel state s, then $\|f_\phi(s)\|^2 = 1$, i.e., states are added to the buffer with an approximate initial pseudocount of 1. As training progresses, the effect of the initialization will wash out and $\mathcal{B}(s)$ will eventually settle to its correct value. An analysis of the optimistic prior's contribution is in Appendix A.5.2.

3.1.6 Integrated CFN Agent

Algorithm 1 outlines how CFN is combined with Rainbow (Hessel et al., 2018) to form a complete bonus-based exploration agent. Naturally, we can combine CFN with most off-the-shelf RL algorithms with minor changes (Haarnoja et al., 2018; Mnih et al., 2015; Lillicrap et al., 2016).

3.2 Experiments

Our empirical results establish CFN as a competitive count-based exploration algorithm. First, we show that CFN can extract accurate counts in domains with visual observations, in contrast to other bonus methods. We then solve eight sparse-reward continuous

Algorithm 1 Rainbow-CFN Agent

Hyperparameters: Reward scale λ, Number of coin flips d

Initialize Q-network Q_θ and target Q-network $Q_{\theta'}$.
Initialize CFN prior f_{prior} and trainable network $\hat{f}_\phi$.
Initialize replay buffer for Rainbow B_r and CFN B_c.
Initialize optimizer for Rainbow and for CFN.
Initialize the running mean μ_t and variance σ_t^2 for the optimistic prior f_{prior}.
while training **do**
  $s_0 =$ `env.reset()`
  while not `done` **do**
    $a_t = \arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a)$
    $R_t, s_{t+1},$ `done` $=$ `env.step`(s_t, a_t)
    Compute intrinsic reward $\mathcal{B}(s_t)$ using Equation 3.4.
    Update μ_t and σ_t^2 using $f_{\text{prior}}(s_t)$.
    Add transition $(s_t, a_t, R_t, \mathcal{B}(s_t), s_{t+1})$ to B_r.
    Sample a random coin-flip vector $\mathbf{c} \sim \{-1, 1\}^d$.
    Add state coin-flip tuple $(s_t, \mathbf{c})$ to B_c.
    Sample minibatch $(s, a, r, \mathcal{B}_s, s') \sim B_r$ and update Q_θ using Rainbow's optimizer and Eq 2.7.
    Update priority for minibatch using Rainbow.
    Sample minibatch $(s, \mathbf{c}) \sim B_c$ and use CFN's optimizer to update f_ϕ via one gradient step on the loss function corresponding to Equation 3.3.
    Update priority for minibatch using Equation 3.5.
  end while
end while

control problems using CFN and show that we significantly outperform RND and baseline SAC. Finally, we show that our method scales gracefully to the challenging Atari game Montezuma's Revenge.

Implementation details. All exploration methods are built on top of Rainbow (Hessel et al., 2018) (when the action-space is discrete) or Soft Actor Critic (SAC) (Haarnoja et al., 2018) (when the action-space is continuous) using the Dopamine library (Castro et al., 2018). CFN has the same neural network architecture as RND's prediction network. We follow the experimental design of Taiga et al. (2020). We use a different set of hyperparameters for each *suite* of tasks, i.e., one for Visual Gridworld, one for *Fetch*, one for Ant, one for Adroit and one for Montezuma's Revenge. Details about CFN, including hyperparameters, can be found in the Appendix A.4; details about environments can be found in Appendix A.3. All code for reproducing results can be found at the linked repository.[1]

3.2.1 Bonus Prediction Accuracy

We compare the exploration bonus from CFN to that of PixelCNN (Ostrovski et al., 2017) and RND (Burda et al., 2019) in Visual Gridworld (Allen et al., 2021). Observations are 84x84 images of a 42x42 grid; these images serve as inputs during training. The agent is initialized in the bottom-left, and achieves a sparse terminal reward of 1 for reaching the top-right within 150 timesteps. For evaluation, we keep track of the tabular state and the ground-truth visitation counts.

Figure 3.2 shows that while CFN is able to predict the count-based exploration bonus with high accuracy, PixelCNN and RND are not. PixelCNN and CFN, being pseudocount methods, should ideally both output bonuses on the dashed line. Not only does PixelCNN mispredict the scale of the exploration bonus, it also assigns the same bonus to states visited once ($x = 1$) as to those visited 25 times ($x = 0.2$). RND's trendline is better

[1] https://github.com/samlobel/CFN

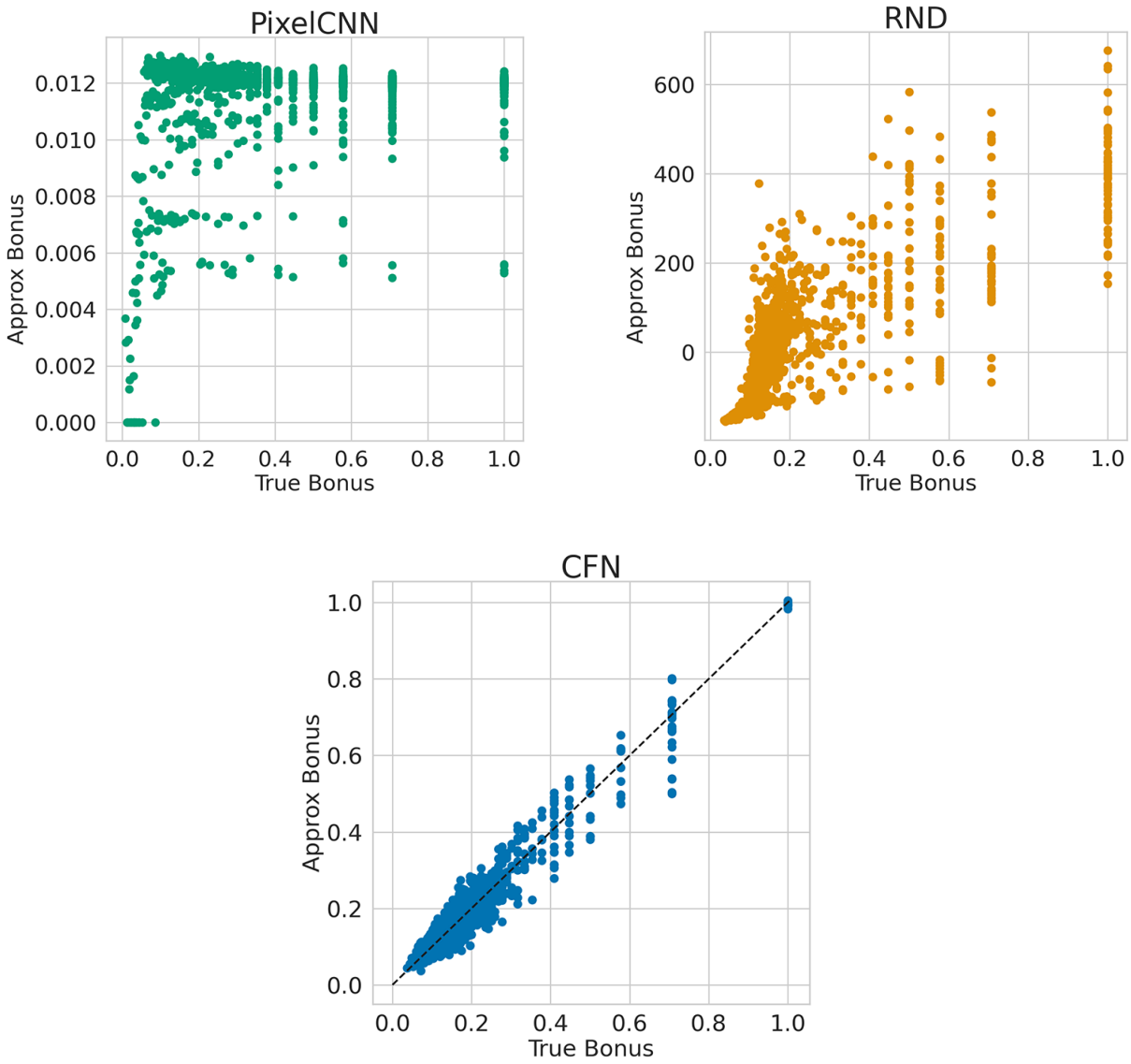

Figure 3.2: Predicted count-based bonuses for all three methods in Visual Gridworld after $100,000$ interactions. The horizontal axis is the ground truth $1/\sqrt{\mathcal{N}(s)}$ bonus, the vertical axis is the exploration bonus predicted by the different methods.

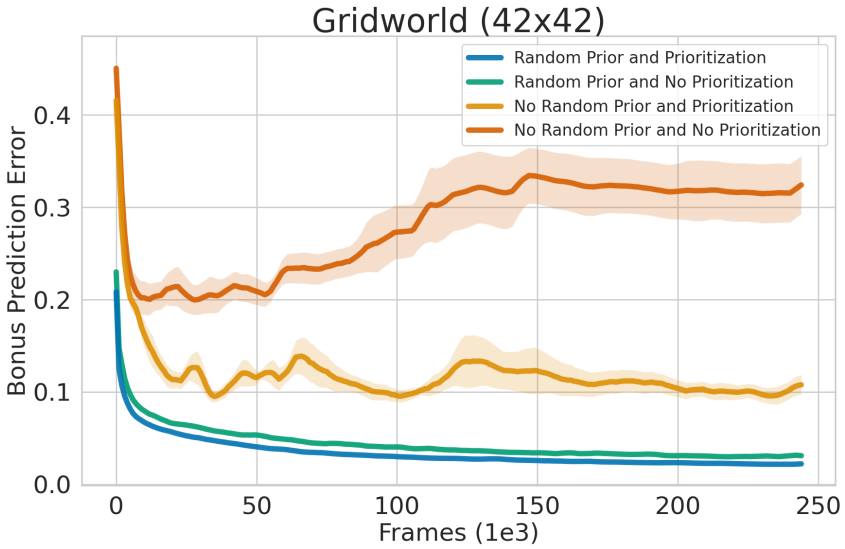

Figure 3.3: Ablating prioritized sampling and optimistic bonus initialization: vertical axis is the mean-squared error between the predicted and ground-truth count-based bonus (averaged over all visited states). Solid lines represent mean and bands represent standard error over 10 random seeds; lower is better.

than PixelCNN, although it has much higher variance than CFN. It is notable that for states with high count, its bonus falls off more sharply than $1/\sqrt{\mathcal{N}(s)}$. A similar experiment is repeated for the more challenging Taxi domain (Dietterich, 1998) (with image observations); results are in Appendix A.5.1.

3.2.2 Ablation: Prioritization and Random Prior

We now ablate the contribution of prioritized sampling (Section 3.1.5) and random prior (Section 3.1.5). In Figure 3.3 we show how mean bonus prediction error evolves over time; the plot indicates that both additions lead to more accurate predictions. The prediction error is the mean-squared difference between the predicted and ground-truth exploration bonuses, averaged over unique states the agent has observed.

In Appendix A.5.3 we provide more insight into each of these curves, and show how both these additions to CFN improve its bonus accuracy on states in the low-count regime.

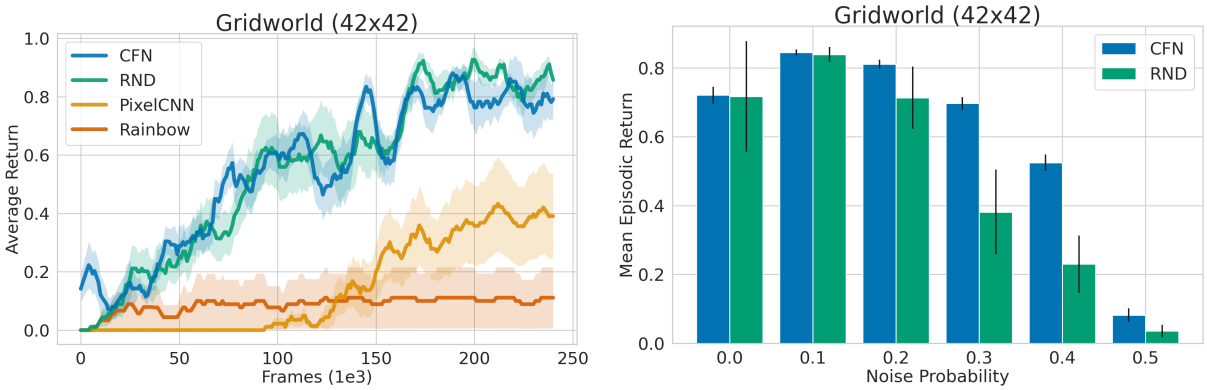

Figure 3.4: **Left**: Learning curves in deterministic Visual Gridworld comparing our method (CFN) with RND, PixelCNN and baseline Rainbow (with noisy networks). Solid lines denote mean episodic return, bands represent standard error. **Right**: Comparison between CFN and RND on increasingly stochastic versions of Visual Gridworld. Bars represent mean episodic return averaged over training run, error bars denote standard error. All results are averaged over 10 random seeds.

3.2.3 RL Performance on Visual Gridworld

Figure 3.4 shows that CFN outperforms baseline Rainbow and PixelCNN, and performs similarly to RND on this task. An important feature to note about this environment is that it is deterministic. As such, the $1/\sqrt{\mathcal{N}(s)}$ bonus may not be appropriate because it is explicitly constructed to deal with stochastic environments (Auer, 2002; Jin et al., 2018). So, we compare CFN to RND on a series of increasingly stochastic versions of Visual Gridworld.[2] Our results show CFN's count-based bonus yields a more significant performance boost over RND in more stochastic versions of the problem. The slight performance bump at noise 0.1 can be attributed to the exploration benefit of random action-selection.

3.2.4 Continuous Control Experiments

We now consider a series of challenging continuous control tasks. These are taken from two different suites: Fetch (Plappert et al., 2018) and D4RL (Fu et al., 2020). For all tasks, we sparsify the reward function to make exploration challenging. We compare CFN

[2]Stochasticity is introduced by replacing the chosen action with a randomly selected action with some predefined probability.

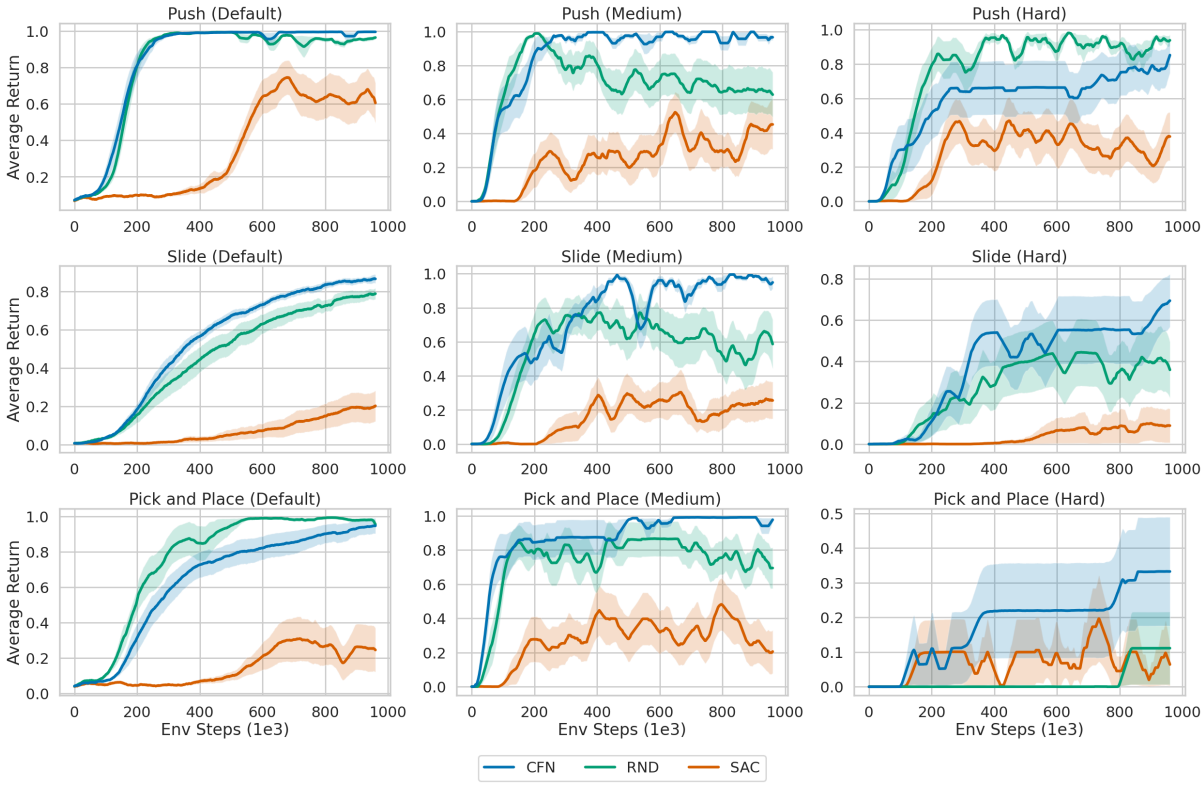

Figure 3.5: Results on the simulated FETCH manipulation tasks with 3 levels of difficulty (default/easy, medium and hard). All curves are averaged over 9 independent runs.

to RND and baseline SAC; we do not compare against PixelCNN because the inputs are not images.

Fetch manipulation tasks. These tasks involve controlling a simulated Fetch robot to perform a series of manipulation tasks: pushing, sliding, or lifting an object to a goal location (Plappert et al., 2018). We consider 3 modes for each task: default, medium and hard; these modes differ in start-goal configurations. The default task randomizes the start-goal states (which occasionally exposes the agent to very simple episodes), medium and hard versions fix them to different levels of difficulty. Figure 3.5 shows that both exploration methods outperform baseline SAC in all tasks; CFN outperforms RND on 6 out of 9 tasks and ties in 1.

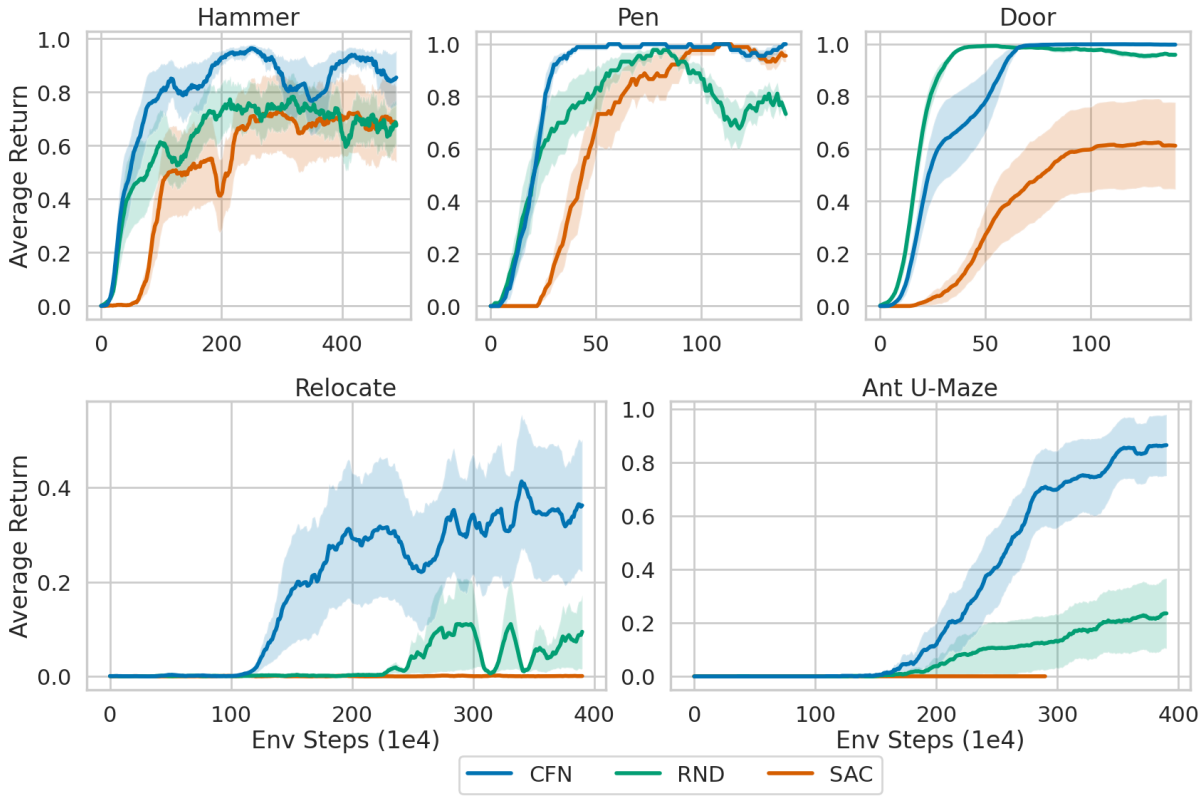

Figure 3.6: Learning curves in the D4RL tasks. Bottom row shows the 2 most challenging tasks in this task suite. All curves are averaged over 9 independent runs.

Ant-navigation and Adriot manipulation tasks. Next, we consider tasks from D4RL(Fu et al., 2020)[3]. The first involves controlling a quadrupedal "ant" robot in a U-shaped maze. The remaining 4 tasks involve controlling a high-dimensional "Adriot" hand to perform various tasks: pick-and-place, reorienting a pen, opening a door and learning how to use a hammer (Rajeswaran et al., 2018). Similar to the Fetch tasks, we remove random restarts because they obviate the need for exploration (Lobel et al., 2022). Figure 3.6 shows that CFN outperforms SAC on all tasks and RND on 4 out of 5 tasks. More interestingly, the performance gains over RND are largest on the hardest exploration tasks (Ant U-Maze and Relocate; as evidenced by SAC's inability to experience any positive rewards). This supports the hypothesis that CFN provides a more thorough exploration bonus than RND.

[3]We use the domains from this suite, not their offline datasets.

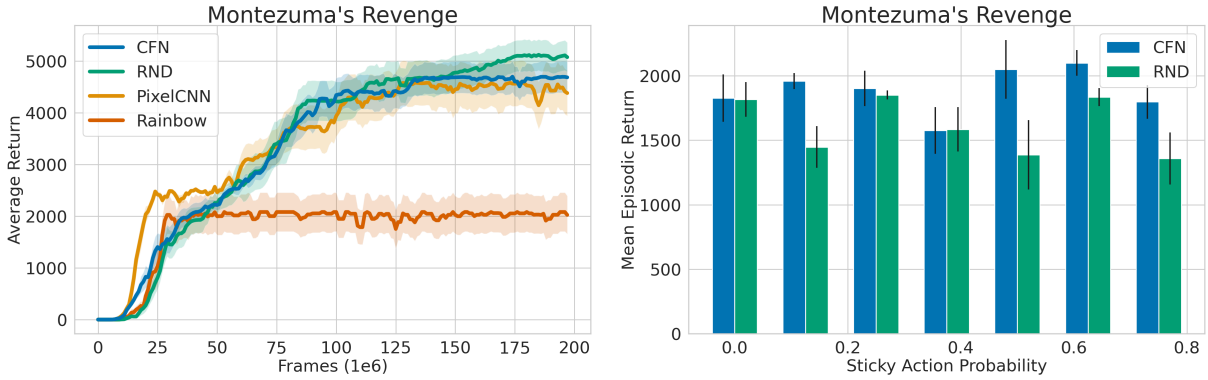

Figure 3.7: **Left**: Learning curve in Montezuma's Revenge comparing our method (CFN) with RND, PixelCNN and baseline Rainbow (with noisy networks). Solid lines denote mean episodic return, bands represent standard error averaged over 12 random seeds. **Right**: Comparison between CFN and RND in terms of mean cumulative reward over 100 million frames on versions of Montezuma's Revenge with varying "sticky action" probabilities (stochastic transitions; 0.25 is the default sticky action probability (Machado et al., 2018)). Error bars represent standard error over 5 seeds.

3.2.5 Performance in Montezuma's Revenge

Finally, we test our method on the challenging exploration benchmark Montezuma's Revenge. We follow the experimental design suggested by Machado et al. (2015) and compare CFN to baseline Rainbow, PixelCNN and RND. Figure 3.7 shows that we comfortably outperform Rainbow with no exploration in this task. All exploration algorithms perform similarly, a result also corroborated by Taiga et al. (2020).

Since all exploration methods perform similarly on the default task, we created a more challenging versions of Montezuma's Revenge by varying the amount of transition noise (via the "sticky action" probability (Machado et al., 2018)). Figure 3.7 *(right)* shows that CFN outperforms RND at higher levels of stochasticity; this supports our hypothesis that count-based bonuses are better suited for stochastic environments than prediction-error-based methods.

Notably, we find that having a large replay buffer for CFN slightly improves performance, which increases memory requirements for this experiment. More discussion about the impact of buffer size can be found in Appendix A.4.3; details about hyperparameters can

be found in Appendix A.4.

3.3 Relationship to Broader Work

This chapter introduces a particularly effective way to compute pseudocounts, and generally estimate novelty, for exploration in deep reinforcement learning. The contribution is made within the realm of bonus-based exploration, where an intrinsic reward is added to the task-specified reward in order to create an optimistic value function. In the next chapter, we will analyze a simplified setting which suggests that additive bonuses can produce overly-optimistic value functions, and describe a simple way to produce a tighter relationship.

CHAPTER 4

An Optimal Tightness Bound for the Simulation Lemma

In reinforcement learning, an agent is frequently tasked with making decisions in an environment that it cannot model perfectly. This may occur because the environment is learned about through sampled data, or because the agent's environment model is simplified through some abstraction. In such cases it is natural to ask how the quality of this approximation might impact an agent's decision-making. This is the subject of the "simulation lemma," a foundational result in reinforcement learning that bounds the error in value estimation when the transition and reward function are known only with some specified degree of precision.

The simulation lemma was introduced in the context of exploration and finds use in a variety of domains that utilize imperfect models, such as hierarchical abstraction (Abel et al., 2016) and offline policy evaluation (Yin et al., 2021). Frequently, results of this kind rely on developing a recursive relationship between value-estimation error at subsequent timesteps. We show how previous approaches implicitly overestimates how probability errors compound over time. By more carefully approximating this quantity, we produce a bound on value-estimation error that is demonstrably tight. We then show that existing

bounds can be derived as a linearization of our result, and finally apply our result to a hierarchical setting to demonstrate broader applicability.

The majority of work in this chapter was first published in Lobel and Parr (2025); in section 4.2 I present a generalization of our prior work that produces a bound in terms on V_{MAX}, not R_{MAX}.

4.1 An Improved Simulation Lemma

We begin by stating the conditions of the original simulation lemma. We consider two MDPs: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \gamma)$, and $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{R}, \hat{T}, \gamma)$, which share a state-action space, but have (boundedly) different transition and reward functions. We are interested in the effect of running the same policy π on these two related MDPs. Let P^π be a matrix that contains the policy-conditioned state-state transition probabilities, and R^π be a vector that contains the per-state expected reward:

$$\begin{aligned} P^\pi_{s,s'} &= \mathbb{E}_{a\sim\pi(s)}[T(s'|s,a)] &&= \sum_{a\in\mathcal{A}} T(s'|s,a)\pi(a|s) \\ R^\pi_s &= \mathbb{E}_{a\sim\pi(s)}[R(s,a)] &&= \sum_{a\in\mathcal{A}} R(s,a)\pi(a|s). \end{aligned} \tag{4.1}$$

We define $\hat{P}^\pi$ and $\hat{R}^\pi$ analogously for MDP $\hat{\mathcal{M}}$. Throughout this work, a single index on a matrix (or vector) extracts the specified row vector (or scalar). Furthermore, P^a and R^a refer to the transition probabilities, and expected reward, of executing action a from each state. Using this notation, we can quantify the difference between two transition or reward functions with the following:

$$\forall s, \pi : \ \|P^\pi_s - \hat{P}^\pi_s\|_1 \leq \epsilon_T \tag{4.2}$$

$$\forall \pi : \ \|R^\pi - \hat{R}^\pi\|_\infty \leq \epsilon_R. \tag{4.3}$$

We additionally assume that $0 \leq R, \hat{R} \leq 1$ for all states and actions. For an improved bound based on a generalization of this assumption (where bounds on V are known instead of bounds on R), see section 4.2. We are interested in the value difference between running π on each MDP. The value of a state for a given policy and MDP is defined as the expected discounted sum of rewards:

$$v^{\pi}(s) = \mathbb{E}_{a_i \sim \pi(s_i)}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, \mathcal{M}\big],$$

where s_t is a random variable representing the state at timestep t.

Noting that $\Pr(s_t = s' | s_0 = s, \pi) = (P^{\pi})^t_{s,s'}$, we can concisely represent value in vectorized notation as follows:

$$V^{\pi} = \sum_{t=0}^{\infty} \gamma^t (P^{\pi})^t R^{\pi} \quad , \quad V^{\pi}_s = \sum_{t=0}^{\infty} \gamma^t \langle (P^{\pi})^t_s, R^{\pi} \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product between two vectors. We define $\hat{V}^{\pi}$ analogously for $\hat{\mathcal{M}}$.

4.1.1 Original Simulation Lemma

We are interested in quantifying the maximum value difference between running the same policy on two different MDPs. The original simulation lemma bounds this quantity:

$$\forall s, \pi: \quad |V^{\pi}_s - \hat{V}^{\pi}_s| \leq \frac{\epsilon_R}{1-\gamma} + \frac{\gamma \epsilon_T}{2(1-\gamma)^2}. \tag{4.4}$$

Existing proofs of the simulation lemma frequently take advantage of the Bellman equation, which provides a recursive representation for value (Howard, 1960):

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} \sum_{t=0}^{\infty} \gamma^t (P^{\pi})^t R^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi}. \tag{4.5}$$

For a complete proof, please refer to Jiang (2018) or see Appendix B.1. The key mathematical idea is to establish the following recursive relationship:

$$\forall s, \pi: \quad |V_s^\pi - \hat{V}_s^\pi| \leq \epsilon_R + \frac{\gamma \epsilon_T}{2(1-\gamma)} + \gamma \|V^\pi - \hat{V}^\pi\|_\infty, \tag{4.6}$$

which can then be easily transformed into the simulation lemma's bound. Analyzing the recursive relationship above, the first term (ϵ_R) represents a one-step reward-prediction error. The second term ($\frac{\gamma \epsilon_T}{2(1-\gamma)}$) represents the maximum value error that results from misspecifying ϵ_T of the next-state distribution's probability mass. However, by defining the recursive relationship as such, the bound implicitly assumes that the process can continually misspecify ϵ_T of its probability at each timestep. Summed over time this quickly amounts to misspecifying more than the entire probability mass, leading to a vast overestimate of the value error, in particular when $\epsilon_T > 1 - \gamma$. By contrast, we carefully track the probability drift at each timestep to avoid this issue.

4.1.2 Bounding Probability Distance

We seek to bound the probability distance tightly at any timestep t. To do so effectively, it is useful to frame distances between probability vectors in terms of their overlap, instead of their L_1 distance. We note that Jiang et al. (2016) use similar machinery to bound compounding probability error (Lemma 1), though applies this insight in a different context. For two probability vectors p, $\hat{p}$, we define their overlap as $\bar{p}$, such that for each index i:

$$\bar{p}_i = \min(p_i, \hat{p}_i).$$

Usefully, because each element of $p - \bar{p}$ (and likewise $\hat{p} - \bar{p}$) is non-negative, the L_1 norm of the difference between these two vectors is equal to the difference between the L_1 norms:

$$\|p - \bar{p}\|_1 = \sum_i |p_i - \bar{p}_i| = \sum_i p_i - \sum_i \bar{p}_i = \|p\|_1 - \|\bar{p}\|_1 \tag{4.7}$$

We use this to derive an equivalence between overlap and L_1 distance, related to the concept of *total variation distance* (Levin and Peres, 2017). Below, we use the notation $[p]^+$ to indicate a thresholded version of p that retains only the non-negative parts, $[p]^+_i = \max(p_i, 0)$:

$$\begin{aligned} \|p - \hat{p}\|_1 &= \|[p - \bar{p}]^+\|_1 + \|[\hat{p} - \bar{p}]^+\|_1 \\ &= \|p - \bar{p}\|_1 + \|\hat{p} - \bar{p}\|_1 \\ &= \|p\|_1 - \|\bar{p}\|_1 + \|\hat{p}\|_1 - \|\bar{p}\|_1 \\ &= 1 + 1 - 2\|\bar{p}\|_1 \\ \implies \|\bar{p}\|_1 &= 1 - \frac{\|p - \hat{p}\|_1}{2}. \end{aligned} \tag{4.8}$$

See Figure 4.1 for a demonstration and explanation of this equivalence. This relationship allows for a simple rewriting of the transition-error condition of the simulation lemma (Equation 4.2):

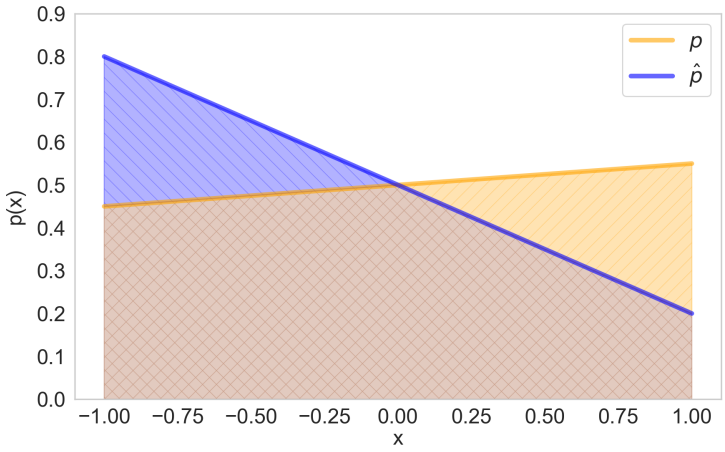

Figure 4.1: Visualization of relation between L_1 distance and overlap of two probability distributions (Equation 4.8). The blue and orange shaded regions together comprise the L_1 distance. The brown region represents overlap. Overlap plus *either* the blue or orange sections constitutes a probability distribution, and therefore has total area 1. Thus the blue and orange regions both individually have area $\|p - \hat{p}\|_1/2$, and so $\|\bar{p}\|_1 = 1 - \|p - \hat{p}\|_1/2$.

$$\forall s, \pi : \ \|\bar{P}_s^\pi\|_1 \geq 1 - \frac{\epsilon_T}{2}. \tag{4.9}$$

Using this framing, we can now lower-bound the overlap of state-distributions at timestep t when starting from s_0, by demonstrating that at every timestep, at least $1 - \epsilon_T/2$ fraction of the prior timestep's distributional overlap is retained. For notational convenience, $P^t_{s_0,s} = (P^\pi)^t_{s_0,s}$, and $\bar{M}^t_{s_0,s} = \min(P^t_{s_0,s}, \hat{P}^t_{s_0,s})$. Thus,

$$\begin{aligned}
\|\bar{M}^{t+1}_{s_0}\|_1 &= \sum_{s'} \min(P^{t+1}_{s_0,s'}, \hat{P}^{t+1}_{s_0,s'}) \\
&= \sum_{s'} \min(\sum_{s} P^t_{s_0,s} \cdot P^\pi_{s,s'} \ , \ \sum_{s} \hat{P}^t_{s_0,s} \cdot \hat{P}^\pi_{s,s'}) \\
&\geq \sum_{s'} \sum_{s} \min(P^t_{s_0,s} \cdot P^\pi_{s,s'} \ , \ \hat{P}^t_{s_0,s} \cdot \hat{P}^\pi_{s,s'}) \\
&\geq \sum_{s'} \sum_{s} \min\Big(\min(P^t_{s_0,s} \ , \ \hat{P}^t_{s_0,s}) \cdot P^\pi_{s,s'}) \ , \ \min(P^t_{s_0,s} \ , \ \hat{P}^t_{s_0,s}) \cdot \hat{P}^\pi_{s,s'}\Big) \\
&= \sum_{s} \sum_{s'} \min(P^t_{s_0,s} \ , \ \hat{P}^t_{s_0,s}) \min(P^\pi_{s,s'} \ , \ \hat{P}^\pi_{s,s'}) \\
&= \sum_{s} \min(P^t_{s_0,s} \ , \ \hat{P}^t_{s_0,s}) \sum_{s'} \min(P^\pi_{s,s'} \ , \ \hat{P}^\pi_{s,s'}) \\
&\geq \|\bar{M}^t_{s_0}\|_1 \cdot \max_s \|\bar{P}^\pi_s\|_1 \\
\implies \|\bar{M}^{t+1}_{s_0}\|_1 &\geq \|\bar{M}^t_{s_0}\|_1 \cdot (1 - \epsilon_T/2).
\end{aligned}$$

The third line can be understood as providing the minimum operator more options to choose from, in that after bringing the minimum inside of the sum, the two elements in the second line are both still possible choices and so the inequality holds. The fourth line can be understood similarly for multiplication.

With $\bar{M}^0 = I$ as the base case, applying recursion yields

$$\|\bar{M}^t_{s_0}\|_1 \geq (1 - \epsilon_T/2)^t. \tag{4.10}$$

We contrast this with the equivalent recursive proof of distributional drift using the L_1 formulation of transition misspecification, akin to the recursion employed by the original simulation lemma (Equation 4.6):

$$\begin{aligned}
\|P_{s_0}^{t+1} - \hat{P}_{s_0}^{t+1}\|_1 &= \|P_{s_0}^{t} P^{\pi} - \hat{P}_{s_0}^{t} \hat{P}^{\pi}\|_1 \\
&= \frac{1}{2}\|(P_{s_0}^{t} - \hat{P}_{s_0}^{t})(P^{\pi} + \hat{P}^{\pi}) + (P_{s_0}^{t} + \hat{P}_{s_0}^{t})(P^{\pi} - \hat{P}^{\pi})\|_1 \\
&\leq \frac{1}{2}\|P_{s_0}^{t} - \hat{P}_{s_0}^{t}\|_1 \|(P^{\pi} + \hat{P}^{\pi})^T\|_1 + \frac{1}{2}\|P_{s_0}^{t} + \hat{P}_{s_0}^{t}\|_1 \|(P^{\pi} - \hat{P}^{\pi})^T\|_1 \\
&= \|P_{s_0}^{t} - \hat{P}_{s_0}^{t}\|_1 + \|P^{\pi} - \hat{P}^{\pi}\|_1 \\
&\leq \|P_{s_0}^{t} - \hat{P}_{s_0}^{t}\|_1 + \epsilon_T \\
\implies \|P_{s_0}^{t+1} - \hat{P}_{s_0}^{t+1}\|_1 &\leq (t+1)\,\epsilon_T,
\end{aligned}$$

where $\|\cdot\|_1$ above refers to both the matrix and vector 1-norm, and on the third line we use the identity $\|Ax\|_1 \leq \|A\|_1 \|x\|_1$. Naïvely using the L_1 formulation leads to unbounded accumulation of drift as horizon approaches infinity, while the overlap formulation smoothly decays from 1 to 0. This difference is crucial to generating a tighter bound.

4.1.3 A Tight Bound on Value Error

We are now ready to prove our main result, a tight bound on the value error.

Theorem 1 *For two MDPs $\mathcal{M}$ and $\hat{\mathcal{M}}$ related as described in Equations 4.2 and 4.3, and reward functions that satisfy $0 \leq R_{sa}, \hat{R}_{sa} \leq 1$ for all s and a, the following inequality holds:*

$$\forall s, \pi: \quad |V_s^{\pi} - \hat{V}_s^{\pi}| \;\leq\; \frac{1}{1-\gamma} - \frac{1-\epsilon_R}{1-\gamma(1-\epsilon_T/2)}. \tag{4.11}$$

Furthermore, this bound is tight.

Proof: Since the conditions of the simulation lemma (Equations 4.2,4.3) are symmetric with respect to $\mathcal{M}$ and $\hat{\mathcal{M}}$, without loss of generality we assume $V^{\pi}_{s_0} \geq \hat{V}^{\pi}_{s_0}$, and thus $|V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0}| = V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0}$. We now add and subtract the same quantity in a way that allows for discarding a strictly non-positive term:

$$
\begin{aligned}
|V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0}| &= V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0} \\
&= \sum_{t=0}^{\infty} \gamma^t \langle P^t_{s_0}, R^{\pi} \rangle - \gamma^t \langle \hat{P}^t_{s_0}, \hat{R}^{\pi} \rangle \\
&= \sum_{t=0}^{\infty} \gamma^t \Big(\langle P^t_{s_0}, R^{\pi} \rangle - \langle \bar{M}^t_{s_0}, R^{\pi} \rangle + \langle \bar{M}^t_{s_0}, R^{\pi} \rangle \\
&\qquad - \langle \bar{M}^t_{s_0}, \hat{R}^{\pi} \rangle + \langle \bar{M}^t_{s_0}, \hat{R}^{\pi} \rangle - \langle \hat{P}^t_{s_0}, \hat{R}^{\pi} \rangle \Big) \\
&= \sum_{t=0}^{\infty} \gamma^t \langle P^t_{s_0} - \bar{M}^t_{s_0}, R^{\pi} \rangle + \gamma^t \langle \bar{M}^t_{s_0}, R^{\pi} - \hat{R}^{\pi} \rangle + \gamma^t \langle \bar{M}^t_{s_0} - \hat{P}^t_{s_0}, \hat{R}^{\pi} \rangle.
\end{aligned}
$$

By construction, $\bar{M}^t_{s_0}$ is the overlap between $P^t_{s_0}$ and $\hat{P}^t_{s_0}$, and thus and entries of $\bar{M}^t_{s_0} - \hat{P}^t_{s_0}$ are strictly non-positive. This allows us to split norms of the form $\|P - M\|$ into $\|P\| - \|M\|$. Since rewards are likewise non-negative, the third inner product in the above sum is always non-positive. Thus, we can drop this term to significantly tighten our bound.

$$
\begin{aligned}
V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0} &\leq \sum_{t=0}^{\infty} \gamma^t \langle P^t_{s_0} - \bar{M}^t_{s_0}, R^{\pi} \rangle + \gamma^t \langle \bar{M}^t_{s_0}, R^{\pi} - \hat{R}^{\pi} \rangle \\
&\leq \sum_{t=0}^{\infty} \gamma^t \| P^t_{s_0} - \bar{M}^t_{s_0} \|_1 \cdot \| R^{\pi} \|_{\infty} + \gamma^t \| \bar{M}^t_{s_0} \|_1 \cdot \| R^{\pi} - \hat{R}^{\pi} \|_{\infty} \\
&\leq \sum_{t=0}^{\infty} \gamma^t \| P^t_{s_0} - \bar{M}^t_{s_0} \|_1 + \gamma^t \| \bar{M}^t_{s_0} \|_1 \epsilon_R \\
&= \sum_{t=0}^{\infty} \gamma^t \| P^t_{s_0} \|_1 - \gamma^t \| \bar{M}^t_{s_0} \|_1 + \gamma^t \| \bar{M}^t_{s_0} \|_1 \epsilon_R \\
&= \sum_{t=0}^{\infty} \gamma^t + \gamma^t (\epsilon_R - 1) \| \bar{M}^t_{s_0} \|_1 \\
&\leq \sum_{t=0}^{\infty} \gamma^t + \gamma^t (\epsilon_R - 1)(1 - \epsilon_T/2)^t \\
&= \frac{1}{1-\gamma} + (\epsilon_R - 1) \sum_{t=0}^{\infty} (\gamma - \frac{\gamma \epsilon_T}{2})^t \\
\implies |V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0}| &\leq \frac{1}{1-\gamma} - \frac{1 - \epsilon_R}{1 - \gamma(1 - \epsilon_T/2)}.
\end{aligned}
$$

■

This proof uses Hölder's inequality (Rudin, 1987) to bound inner products with L_1 and L_∞ norms, as well as the identity in Equation 4.7 to split $\| P^t_{s_0} - \bar{M}^t_{s_0} \|_1$ into $\| P^t_{s_0} \|_1 - \| \bar{M}^t_{s_0} \|_1$. We provide a parallel proof for the finite-horizon undiscounted setting in Appendix B.2. We briefly remark that this bound matches intuition:

- When $\gamma = 0$, then $|V^{\pi}_s - \hat{V}^{\pi}_s| \leq \epsilon_R$ since only the first step contributes to value.
- When $\epsilon_R = 1$, the MDPs can have completely different reward functions and thus $|V^{\pi}_s - \hat{V}^{\pi}_s| \leq \frac{1}{1-\gamma} = V_{MAX}$.
- When $\epsilon_R = \epsilon_T = 0$, the MDPs are identical and thus $|V^{\pi}_s - \hat{V}^{\pi}_s| = 0$.

Additionally we note that the original simulation lemma can be reproduced as a Taylor expansion of our bound around $\epsilon_R = 0$ and $\epsilon_T = 0$, proving that the original bound is

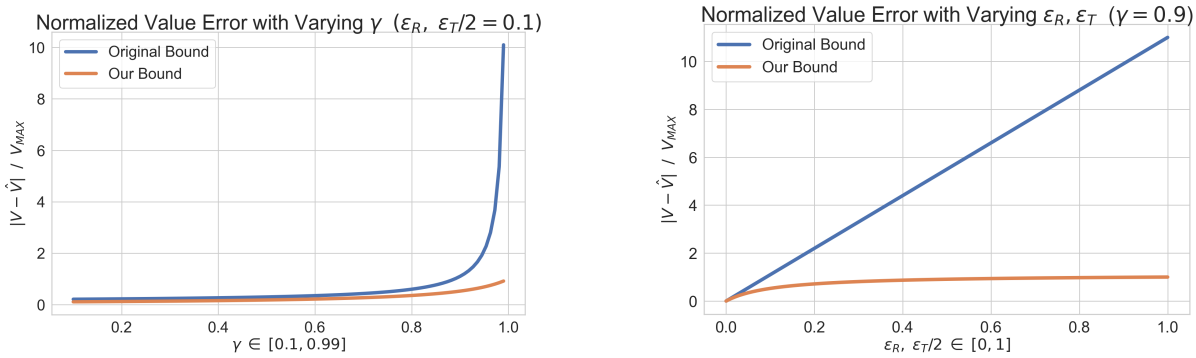

Figure 4.2: Bounds on value error given by original simulation lemma as well as our tighter bounds, normalized by V_{MAX}. (Left) Bound on value error with increasing gamma shows the original lemma's suboptimality with respect to discount. (Right) Bound on value error with increasing misspecification shows looseness of linear approximation compared to the tight bound.

the tightest possible linear approximation to the maximal error as model misspecification approaches 0. Figure 4.2 presents a comparison of our bound with the original simulation lemma, demonstrating superiority in the large-misspecification and large-discount limits.

4.1.4 Proof of Tightness

We now demonstrate that this is the tightest possible bound, including constant factors, by constructing a pair of MDPs with exactly this value error. $\mathcal{M}$ consists of two states, both of which transition to themselves, with $R(s_1) = 1$ and $R(s_2) = 0$. We construct $\hat{\mathcal{M}}$ so that $\hat{V}(s_1)$ is as small as possible given ϵ_R, ϵ_T, by setting $\hat{R}(s_1) = 1 - \epsilon_R$, and transitioning from s_1 to s_2 with probability $\epsilon_T/2$ (and thus self-transitions with $\epsilon_T/2$ less probability, so $\|P^{\pi}_{s_1} - \hat{P}^{\pi}_{s_1}\|_1 = \epsilon_T$). Hence, $V(s_0) = \frac{1}{1-\gamma}$ and $\hat{V}(s_0) = \frac{1-\epsilon_R}{1-\gamma(1-\epsilon_T/2)}$.

Intuitively, this result makes clear the role of ϵ_T as modifying the discount factor of $\hat{\mathcal{M}}$. A discount can be interpreted as entering an absorbing state with probability $1-\gamma$ at each timestep (Sutton and Barto, 2018). In $\hat{\mathcal{M}}$, this instead occurs more frequently, with probability $1 - \gamma(1 - \epsilon_T/2)$.

4.1.5 Value Loss of Optimal Policy

The simulation lemma directly applies to bounding the value difference of executing the same policy on two related MDPs. However, in reinforcement learning the task is frequently to learn an *optimal* policy π^*, that has the following property:

$$\forall \pi, s : V_s^{\pi^*} \geq V_s^{\pi}.$$

It is natural to ask, if one learns the optimal policy $\hat{\pi}^*$ by training on an approximate MDP $\hat{\mathcal{M}}$, how much worse can this policy do than π^* when executed on the actual MDP $\mathcal{M}$? In contrast to the simulation lemma, we are comparing the value loss of *different* policies on the *same* MDP. Noting that $\hat{V}_s^{\hat{\pi}^*} \geq \hat{V}_s^{\pi^*}$:

$$\begin{aligned} V_s^{\pi^*} - V_s^{\hat{\pi}^*} &= V_s^{\pi^*} + (\hat{V}_s^{\pi^*} - \hat{V}_s^{\pi^*}) + (\hat{V}_s^{\hat{\pi}^*} - \hat{V}_s^{\hat{\pi}^*}) - V_s^{\hat{\pi}^*} \\ &= (V_s^{\pi^*} - \hat{V}_s^{\pi^*}) + (\hat{V}_s^{\pi^*} - \hat{V}_s^{\hat{\pi}^*}) + (\hat{V}_s^{\hat{\pi}^*} - V_s^{\hat{\pi}^*}) \\ &\leq (V_s^{\pi^*} - \hat{V}_s^{\pi^*}) + 0 + (\hat{V}_s^{\hat{\pi}^*} - V_s^{\hat{\pi}^*}) \\ &\leq |V_s^{\pi^*} - \hat{V}_s^{\pi^*}| + |\hat{V}_s^{\hat{\pi}^*} - V_s^{\hat{\pi}^*}|. \end{aligned}$$

This is simply twice the value error of executing the *same* policy on *different* MDPs. Thus, by improving the simulation lemma bound, we similarly tighten the estimated value loss when training on an approximate MDP. Similar results are common in inverse RL, e.g., Burchfiel et al. (2016), and have been noted in the context of the simulation lemma as well (Jiang, 2018).

4.1.6 Application to Hierarchy

Analogs to the simulation lemma exist throughout the reinforcement learning literature; here, we present an extension of our proof to one such instance in the field of hierarchical

reinforcement learning. We use the formalism of ϕ-relative options (Abel et al., 2020), a form of approximately value preserving state and action abstractions.

Let $\mathcal{O}^*_\phi$ be a set of options o^* over abstract states $s_\phi \in \mathcal{S}_\phi$, that can be composed to form a policy that is optimal in the base MDP. Let $\hat{\mathcal{O}}_\phi$ be a set of options that approximates $\mathcal{O}^*_\phi$ in that

$$\begin{aligned} &\forall o^* \in \mathcal{O}^*_\phi \;\; \exists \hat{o} \in \hat{\mathcal{O}}_\phi : \\ &\forall s, s' \quad |P^{o^*}_{s,s'} - P^{\hat{o}}_{s,s'}| \leq \epsilon_T \;\text{ and }\; |R^{o^*}_s - R^{\hat{o}}_s| \leq \epsilon_R, \end{aligned}$$

where R^o_s and $P^o_{s,s'}$ represent the reward and multi-time models of Sutton et al. (1999). We define $V^{\pi_{o^*}}$ as the value of executing the best policy over $\mathcal{O}^*_\phi$, and $V^{\pi_{\hat{o}}}$ as the value of executing an approximately equivalent policy using options from $\hat{\mathcal{O}}_\phi$. By bounding probability distances we arrive at the following relation:

$$|V^{\pi_{o^*}}_s - V^{\pi_{\hat{o}}}_s| \leq \frac{R_{MAX}}{1-\gamma} - \frac{R_{MAX} - \epsilon_R}{1 - \gamma + (|S| - 1)\epsilon_T}.$$

This improves on the existing bound (Abel et al., 2020):

$$|V^{\pi_{o^*}}_s - V^{\pi_{\hat{o}}}_s| \leq \frac{\epsilon_R + |S|\epsilon_T R_{MAX}}{(1-\gamma)^2},$$

in much the same way as our original result improves upon the simulation lemma. A proof, more complete definitions, and an example demonstrating tightness are deferred to Appendix B.3. The main difference in applying our technique to this domain is careful treatment of the multi-time transition function, where $\sum_{s'} P^o_{s,s'} \neq 1$.

4.2 A V_{MAX}-Dependent Bound

The result above is optimal for the given assumptions on T, R, ϵ_R, and ϵ_T. Here, I generalize this result to the setting where instead of a bound on R_{MAX} we are given V_{MAX}, the maximum value of any state-action in either $\mathcal{M}$ or $\hat{\mathcal{M}}$. An upper-bound on V_{MAX} can be easily created from knowledge or R_{MAX}: $V_{MAX} \leq \frac{R_{MAX}}{1-\gamma}$. However, for many MDPs of interest the true V_{MAX} is much less than this, and may be easily knowable in advance. For example, in a goal-reaching task where the agent receives a reward of 1 for achieving the goal, $V_{MAX} = R_{MAX} = 1$.

For this bound we use the same principle of separating next-state probability distribution into an overlapping and non-overlapping component. This proof follows more closely to the methodology of Jiang (2018), however treats compounding probability carefully as in our improved bound. We assume that for all states and actions, $R, \hat{R} \geq 0$ and for all states, $V_s^\pi \leq V_{MAX}$. Without loss of generality, we assume $V_s \geq \hat{V}_s$. We begin by expanding value into its recursive form:

$$\begin{aligned} V_{s_0}^\pi - \hat{V}_{s_0}^\pi &= R_{s_0}^\pi - \hat{R}_{s_0}^\pi + \gamma \langle P_{s_0}^\pi, V^\pi \rangle - \gamma \langle \hat{P}_{s_0}^\pi, \hat{V}^\pi \rangle \\ &\leq \left| R_{s_0}^\pi - \hat{R}_{s_0}^\pi \right| + \gamma \left(\langle P_{s_0}^\pi, V^\pi \rangle - \langle \hat{P}_{s_0}^\pi, \hat{V}^\pi \rangle \right) \\ &\leq \epsilon_R + \gamma \left(\langle P_{s_0}^\pi, V^\pi \rangle - \langle \hat{P}_{s_0}^\pi, \hat{V}^\pi \rangle \right). \end{aligned}$$

As in section 4.1.3, we now add and subtract the overlap so that we can discard a strictly non-positive term.

$$
\begin{aligned}
V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0} &\leq \epsilon_R + \gamma \left(\langle P^{\pi}_{s_0}, V^{\pi} \rangle - \langle \hat{P}^{\pi}_{s_0}, \hat{V}^{\pi} \rangle \right) \\
&= \epsilon_R + \gamma \left(\langle P^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0} + \bar{P}^{\pi}_{s_0}, V^{\pi} \rangle - \langle \hat{P}^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0} + \bar{P}^{\pi}_{s_0}, \hat{V}^{\pi} \rangle \right) \\
&= \epsilon_R + \gamma \left(\langle P^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0}, V \rangle - \langle \hat{P}^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0}, \hat{V}^{\pi} \rangle + \langle \bar{P}^{\pi}_{s_0}, V^{\pi} - \hat{V}^{\pi} \rangle \right) \\
&\leq \epsilon_R + \gamma \left(\langle P^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0}, V^{\pi} \rangle + \langle \bar{P}^{\pi}_{s_0}, V^{\pi} - \hat{V}^{\pi} \rangle \right) \\
&\leq \epsilon_R + \gamma \left(\| P^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0} \|_1 \| V^{\pi} \|_{\infty} + \| \bar{P}^{\pi}_{s_0} \|_1 \| V^{\pi} - \hat{V}^{\pi} \|_{\infty} \right).
\end{aligned}
$$

In the final line we again use Hölder's inequality (Rudin, 1987) to bound inner products with L_1 and L_∞ norms. Finally we use the relation that $\| \bar{P}_{s_0} \|_1 \leq 1 - \frac{\epsilon_T}{2}$ from Equation 4.9 to produce the following recursive relationship:

$$
\begin{aligned}
V^{\pi}_{s_0} - \hat{V}^{\pi}_{s_0} &\leq \epsilon_R + \gamma \left(\| P^{\pi}_{s_0} - \bar{P}^{\pi}_{s_0} \|_1 \| V^{\pi} \|_{\infty} + \| \bar{P}^{\pi}_{s_0} \|_1 \| V^{\pi} - \hat{V}^{\pi} \|_{\infty} \right) \\
&\leq \epsilon_R + \gamma \frac{\epsilon_T}{2} V_{MAX} + \gamma (1 - \frac{\epsilon_T}{2}) \| V^{\pi} - \hat{V}^{\pi} \|_{\infty}. \qquad (4.12)
\end{aligned}
$$

Comparing the above with the recursive relationship presented in Jiang (2018) and in Equation 4.6, the relation in Equation 4.12 more tightly bounds the recursive component, by discarding half of the mis-specified probability mass. Solving the above results in a tighter bound for the V_{MAX} assumption.

Theorem 2 *For two MDPs $\mathcal{M}$ and $\hat{\mathcal{M}}$ related as described in Equations 4.2 and 4.3, whose reward functions satisfy $0 \leq R_{sa}$ for all s and a, and for which the values are bounded by $V^{\pi}_s, \hat{V}^{\pi}_s \leq V_{MAX}$ for all policies π and states s, the following inequality holds:*

$$
\forall s, \pi : \quad |V^{\pi}_s - \hat{V}^{\pi}_s| \ \leq \ \frac{\epsilon_R + \gamma \frac{\epsilon_T}{2} V_{MAX}}{1 - \gamma(1 - \frac{\epsilon_T}{2})}. \qquad (4.13)
$$

Furthermore, if $\epsilon_R \leq (1 - \gamma) V_{MAX}$ this bound is tight.

We note that when $V_{MAX} = \frac{1}{1-\gamma}$ this bound exactly matches Theorem 2.

4.2.1 Proof of Tightness

A similar MDP to the one constructed in section 4.1.4 proves the tightness of Theorem 2 in a restricted setting. $\mathcal{M}$ consists of two states, both of which transition to themselves, with $R(s_1) = (1-\gamma)V_{MAX}$ and $R(s_2) = 0$. Thus, $V_{s_1} = V_{MAX}$. We again construct $\hat{\mathcal{M}}$ so that $\hat{V}(s_1)$ is as small as possible given ϵ_R, ϵ_T, by setting $\hat{R}(s_1) = (1-\gamma)V_{MAX} - \epsilon_R$, and transitioning from s_1 to s_2 with probability $\epsilon_T/2$ (and thus self-transitions with probability $1 - \epsilon_T/2$). Hence, $V(s_0) = V_{MAX}$ and $\hat{V}(s_0) = \frac{(1-\gamma)V_{MAX} - \epsilon_R}{1-\gamma(1-\epsilon_T/2)}$. Therefore, $V(s_0) - \hat{V}(s_0) = \frac{\epsilon_R + \gamma\frac{\epsilon_T}{2}V_{MAX}}{1-\gamma(1-\frac{\epsilon_T}{2})}$, matching the above bound presented in Theorem 2 exactly. We note that since $\hat{R}_s \geq 0$ for all s, this example only satisfies our assumptions when $\epsilon_R \leq (1-\gamma)V_{MAX}$, and so while the bound in Theorem 2 is true in all cases, it is only proven to be tight when $\epsilon_R \leq (1-\gamma)V_{MAX}$.

4.3 Relationship to Broader Work

This chapter introduces the *simulation lemma*, and demonstrates that the original proof dramatically overestimates value-error in all but the asymptotically-accurate case. We introduce a proof that avoids this overestimation by treating estimation-error in a multiplicative (decaying) way, rather than an additive way. The simulation lemma, originally developed as part of an exploration algorithm, has a direct application for constructing optimistic value functions. Indeed, bonus-based exploration as a methodology for inducing optimism is derived from a quite similar line of reasoning as the original simulation lemma proof.

This raises the natural question of whether an alternative to bonus-based exploration can be created using a similar multiplicative procedure. In the next chapter, we introduce a method for one such case, where we maintain optimism in deterministic continuous control exploration problems.

CHAPTER 5

Optimistic Initialization for Exploration in Continuous Control

In Chapter 3 we develop machinery to generalize *bonus-based exploration* to the function approximation setting. In this chapter, we do the same with another classical exploration method, *optimistic initialization*. In optimistic initialization, unobserved transitions are assumed to be maximally desirable until proven otherwise, thus encouraging the agent to explore (Brafman and Tennenholtz, 2002; Jaksch et al., 2010). The difference between adding a term to account for uncertainty (BBE), and pinning uncertain values to a high number (optimistic initialization) is analogous to the one between the original simulation lemma and our improved bound; in Chapter 6 we make this connection explicit.

In the discrete state-action settings, optimistic initialization is well understood and leads to strong theoretical regret bounds (Strehl et al., 2006). When using function approximation, however, naïve attempts at optimistic initialization are quickly learned away due to generalization (Rashid et al., 2020; Machado et al., 2015). We introduce Deep Optimistic Initialization for Exploration (DOIE), a novel, practical method for optimistically initializing value functions over continuous states and actions that enables precise control over the effect of generalization on optimism. Our method uses a nearest-

neighbors-based optimism module that identifies the degree to which state-action pairs are similar to those already observed by the agent. We then compute an *optimistic* value function by using this similarity measure to interpolate between a learned value estimate and an optimistic *envelope* that describes an upper bound of the optimal value function. DOIE drastically improves exploration in sparse-reward domains, and is amenable to principled approximation schemes which allow for its use over long time scales. In the limit of complete exploration this method reduces to standard Q-learning and therefore does not modify the fixed point optimal policy. At the other extreme, with no interaction, the effective Q-function is the optimistic envelope, thus satisfying optimistic initialization.

We empirically investigate our method's behavior on a variety of challenging sparse reward continuous control problems, demonstrating state-of-the-art performance on a maze navigation domain and improved sample-efficiency compared with exploratory baselines on sparse-reward tasks in the DeepMind Control Suite (Tassa et al., 2018).

5.1 Optimistic Initialization in Continuous MDPs

The intuition behind DOIE is to construct a modified Q-function that takes on an optimistic value for transitions far outside the agent's experience, and smoothly relaxes to empirical estimates for transitions near those which have been observed. This can be achieved through the use of *knownness*, a quantity which equals 1 for observed transitions, and smoothly decays to 0 for state-actions far away from any observations. Given a knownness function $\kappa(s, a)$ satisfying these properties, we can construct an optimistic Q-function, Q^+, as follows:

$$Q^+(s, a) = \kappa(s, a)Q(s, a) + (1 - \kappa(s, a))Q_{MAX}(s, a). \tag{5.1}$$

Choosing an optimistic upper-bound where $Q_{MAX} \geq Q^*$ everywhere ensures that $Q^+(s, a) \geq Q(s, a)$, and thus incentivizes the agent to explore regions of the state space

with low knownness. Q_{MAX} frequently can be specified using commonly-known quantities of an environment. For example, when the maximum per-timestep reward $r_{\max}$ is known, $Q_{MAX} = \frac{r_{\max}}{1-\gamma}$. When an episode terminates after achieving the reward, such as in goal-directed tasks, $Q_{MAX} = r_{\text{goal}}$. We can then use this optimistic Q-function for both bootstrapping and action selection:

$$Q(s,a) \leftarrow r(s,a) + \max_{a'} \gamma Q^{+}(s,a') \tag{5.2}$$

$$\pi_{Q^{+}}(s) = \arg\max_{a} Q^{+}(s,a). \tag{5.3}$$

We now describe the measure of knownness for a state-action pair. As stated earlier, we would like knownness to quantify similarity of the state-action to previously observed state-actions. We assume $\mathcal{X}$ is endowed with a metric, $d(x_1, x_2)$, that quantifies such similarity, as is common in many works in continuous RL (Ni et al., 2019; Asadi et al., 2018). We define knownness as a function of the distance to the closest state-action which the agent has observed:

$$\kappa(x) = \beta(d_{\min}(x)) \tag{5.4}$$

$$d_{\min}(x) = \min_{x' \in X} d(x, x'). \tag{5.5}$$

where $\beta(d)$ is a kernel function such that $\beta(0) = 1$ and decays monotonically to zero as d goes to infinity, and X is the set of all previously observed state-actions. In this work we use

$$\beta(d) = \frac{1}{1 + (d/d_0)^2} \tag{5.6}$$

where d_0 is a lengthscale parameter that quantifies how quickly the value of Q can change. This function mimics the Gaussian function near the origin, but does not fall away exponentially as d increases.

A natural metric for such a task is L_2. We can make this metric more flexible by allowing for different weighting of different state-action dimensions in the following way:

$$d(x_1, x_2) = |Ax_1 - Ax_2|_2 \tag{5.7}$$

We note that this definition of knownness is similar to one presented in prior work (Nouri and Littman, 2009), except that ours modulates only the next-state's value, rather than the entire update target. This is more suited to continuous control, where domains are often only mildly stochastic and thus every (s, a) in an agent's experience will be largely *known*, while the same cannot be said about every (s', a'). Additionally, prior work has investigated using the distance to the nearest observed state-actions for exploration in continuous spaces. Most notably, Pazis and Parr (2013) constructs a bonus based on the distance to the k^{th}-nearest neighbor that, under assumptions of Lipschitz continuity over T and R, results in a PAC-optimal exploration algorithm.

5.1.1 Covering Sets for Efficient Knownness

The computational complexity of a naïve implementation of our knownness calculation scales linearly with the number of state-actions visited. For tasks which require substantial interaction to solve, this quickly becomes infeasible. We introduce a simple approximation algorithm which maintains a filtered set of state-actions that is an ϵ-covering of all seen state-actions so far (Algorithm 2). A subset $\hat{X}$ of X is an ϵ-covering if:

$$\forall x \in X \;\; \exists \hat{x} \in \hat{X} \;\text{ s.t. }\; d(x, \hat{x}) \leq \epsilon.$$

We define $\hat{d}_{\min}(x)$ as the distance of x to the ϵ-ball around any member of the covering-set:

$$\hat{d}_{\min}(x) = \max \left\{ \min_{x' \in \hat{X}} d(x, x') - \epsilon, \;\; 0 \right\}, \tag{5.8}$$

Algorithm 2 Iterative Covering Set Creation

Input: radius ϵ
Initialize $\hat{X} = \{\}$.
for each episode **do**
  $s \leftarrow$ env.reset()
  for each step **do**
    $a = \pi(s)$
    $x = (s, a)$
    $d_{\min} = \min_{(x') \in \hat{X}} d(x, x')$
    if $d_{\min} > \epsilon$ **then**
      $\hat{X} \leftarrow \hat{X} \cup \{x\}$
    end if
    $s \sim T(s, a)$
  end for
end for

and analogously the approximate knownness as:

$$\hat{\kappa}(x) = \beta(\hat{d}_{\min}(x)). \tag{5.9}$$

We choose $d_{\min}$ as such so that $\hat{\kappa}(x)$ is an efficiently-computable upper bound of the true knownness (proof in Appendix A):

$$\hat{\kappa}(x) \geq \kappa(x). \tag{5.10}$$

Furthermore, if $\mathcal{X}$ is a compact metric space, and $\hat{X}$ is an ϵ-covering of $\mathcal{X}$, it follows that $\hat{\kappa}(x) = 0$ for all $x \in \mathcal{X}$. Thus, this approximation preserves the desirable property that knownness goes to 0 everywhere in the limit of thorough exploration. Figure 5.1 diagrams the filtering process and the relationship between $\hat{\kappa}(x)$ and $\kappa(x)$, and provides intuition on the lower bound of equation 5.10. We examine the time, space, and performance tradeoffs of this approximation in our empirical results. Details for adaptive filtering in domains where the scale of the state space is not known a priori can be found in Appendix B.

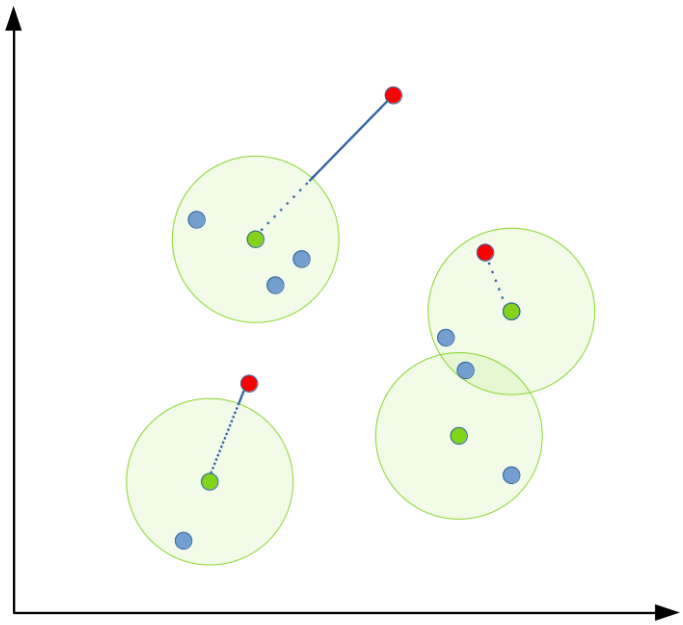

Figure 5.1: A visualization of $\hat{d}_{\min}$ in a two-dimensional state-action space. Green points at the center of each circle represent the filtered covering-set. Blue points represent observed state-actions. Red points represent "query points". The blue solid line is $\hat{d}_{\min}$. Note that $\hat{d}_{\min} = 0$ for the red point within the covering-ball, and that $\hat{d}_{\min} \leq d_{\min}$ always. The approximation $\hat{d}_{\min}$ is equivalent to assuming we have encountered every point in each green ball. If the green balls cover the entire state-action space, then $\hat{d}_{\min} \equiv 0$.

5.1.2 Value Shaping

An attractive property of optimistic initialization is that it allows for easy incorporation of domain knowledge through specifying a shaped upper bound to the Q-function. At a high level, optimistic initialization works by pinning the values of completely unknown state-actions to Q_{MAX}, and "whittling down" the excess optimism through interaction, until $Q(s,a)$ approaches $Q^*(s,a)$. How much whittling is necessary is a function of how over-optimistic Q_{MAX} is. For example, if $Q_{MAX}(s,a) = Q^*(s,a)$ then the initial policy of Q^+ will be optimal, and no further learning is necessary.

Clearly, being given $Q^*(s,a)$ as an initialization is an unrealistic scenario, because learning Q^* is the goal of interaction. However, it is often possible to use domain knowledge to lessen the distance between Q_{MAX} and Q^* while maintaining the crucial property that $Q_{MAX}(s,a) \geq Q^*(s,a)$ everywhere. In our empirical results, we provide an example of

value shaping which accelerates learning for a task where the agent has access to the location of its terminal goal-state.

5.2 Empirical Results

We test our method on three sets of domains. First, we investigate our method's exploration performance in detail on a challenging point navigation task, achieving state-of-the-art sample efficiency. We then compare our method to a variety of exploration baselines on modified versions of the benchmark tasks in the DeepMind Control Suite (Tassa et al., 2018). Finally, we investigate the efficacy of reward and value shaping in MountainCar (Singh and Sutton, 1996). Details on architectures, training procedures, resource usage and shaping functions are included in Appendix D. All code used to generate results is included as supplementary material.

5.2.1 PointMaze

PointMaze (Trott et al., 2019) is a challenging continuous control problem with sparse rewards where an agent attempts to navigate a 2D maze from the lower-left corner to the upper-right corner. As in prior work, the agent has 50 steps before it is reset to the starting region; an optimal agent would reach the goal in roughly 25 steps Trott et al. (2019). Neither uniform random exploration nor ϵ-greedy training Sutton and Barto (2018) solves this problem within 5,000 episodes, so some form of directed exploration is necessary for efficient learning.

We compare against the following bonus-based exploration methods:

- Random Network Distillation (RND), a bonus derived from the error of predicting a randomly-initialized neural network's output Burda et al. (2019).
- Model-Predictive Error (MPE), a bonus derived from the error of predicting the environment's transition model Pathak et al. (2017).

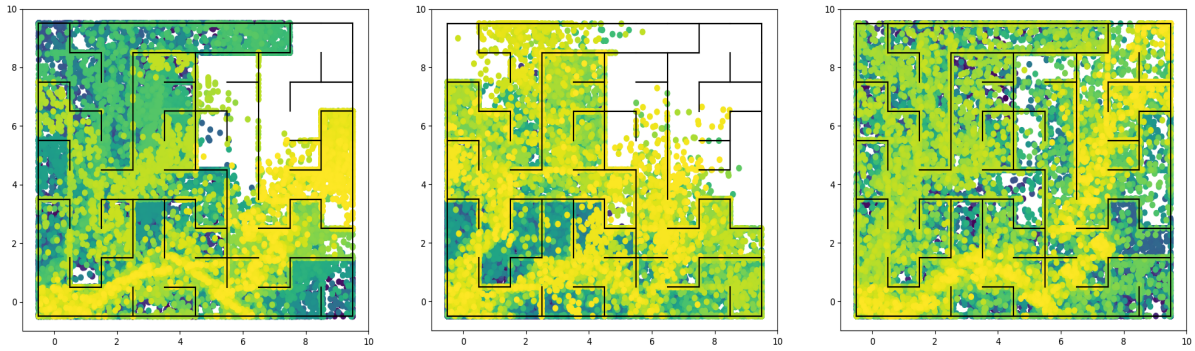

Figure 5.2: State visitation of the three best-performing models throughout the first 1000 episodes on PointMaze. Left: OPIQ, center: OMEGA, right: DOIE. Color denotes episode number, with blue being near the beginning of training and yellow near the end.

- Pseudocounts with action-selection bonus (OPIQ), with counts calculated from a discretization of the normalized state-action space (Rashid et al., 2020).

We also compare against a previous method which enforces optimistic initialization through the value function's bias term (OptBias) Machado et al. (2015), and against OMEGA Pitis et al. (2020), a goal-conditioned exploration algorithm which achieves previous state-of-the-art performance on this domain. For OMEGA we use publicly available code; for all other methods we integrate the novelty-bonus or initialization into an RBFDQN Asadi et al. (2021) base agent. Finally, we include ϵ-greedy RBFDQN and a random agent as non-exploratory baselines. Details of all architectures and bonuses are included in Appendix D.

Ability to Explore

We begin by investigating the ability of our method to efficiently explore the state space. We visualize the agent's trajectories through training in Figure 5.2. By the 1000th episode, optimistic initialization has guided the agent through the majority of the state space, while none of the baseline exploration methods have yet explored in the region of the goal. We quantify this exploration in Figure 5.3 by partitioning the maze into a 40

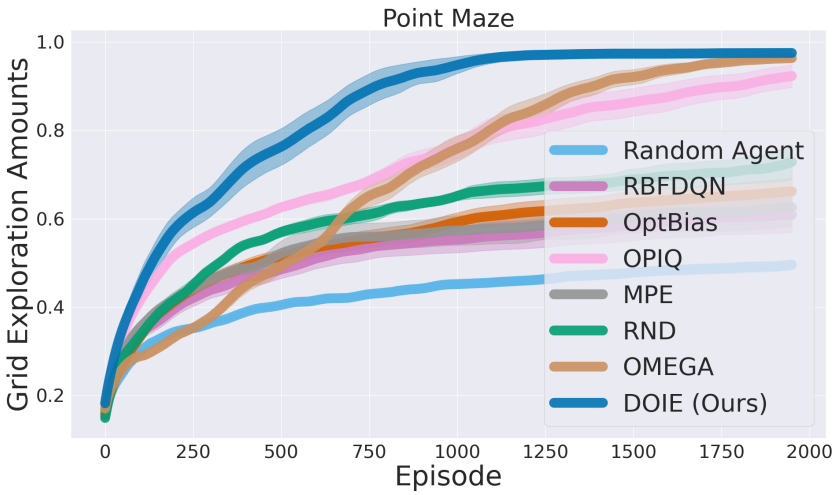

Figure 5.3: Fraction of state space explored by all exploration methods over 2000 episodes. While both DOIE and OMEGA eventually cover the state space, DOIE accomplishes it in roughly half the episodes.

by 40 grid, and tracking the number of these squares that each agent visits throughout training. Optimistic initialization quickly covers the state space after 1000 episodes, while the strongest baseline takes 2000 episodes to cover similar area.

Performance Results

Figure 5.4 demonstrates the learning performance of each method over 2000 episodes. Besides OMEGA, the remaining baseline methods are unable to reach the goal even after extensive hyperparameter tuning. This is consistent with the exploration graphs presented in the prior section. OMEGA is able to reach the goal consistently, but first reaches the goal nearly 300 episodes after our method.

Effect of approximation

Naïve nearest-neighbor calculations involve computing the distance between a query point and every point the agent has encountered, and thus exact nearest-neighbors is impractical for long learning interactions. Earlier, we described a filtering method that makes this process much faster at the expense of the granularity of the knownness calculation. We investigate the relationship between filtering amount and exploration

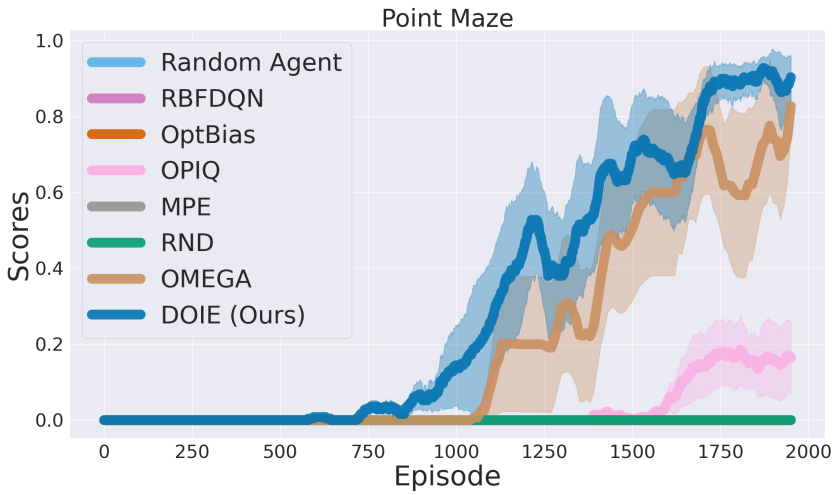

Figure 5.4: Success percentage of Optimistic Initialization and baselines on PointMaze over 2000 episodes. The shaded region represents the standard deviation over 5 runs. Only OMEGA and DOIE reach the goal in the allotted time.

by plotting the speedup of the knownness calculation versus exploration amounts at 500 episodes for a variety of filtering radii (Figure 5.5). As expected, we see that increasing the amount filtered decreases exploration performance. Our experiments are performed with intermediate values of filtering, so as to train acceptably quickly while still providing the benefits of exhaustive exploration.

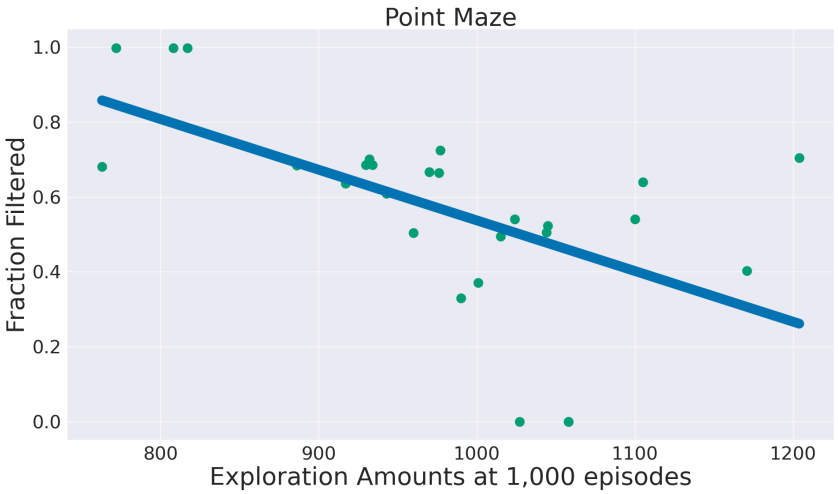

Figure 5.5: Comparison of exploration amounts versus knownness speedup for various approximation filter radii. As filtering becomes more aggressive, exploration is less effective.

5.2.2 DeepMind Control Suite

To gauge the general exploration ability of our method, we test our method on modified versions of four sparse-reward tasks from the DeepMind Control Suite Tassa et al. (2018): Pendulum, Hopper Stand, Acrobot, and Ball in Cup (Figure 5.6). Normally, each of these environments would be initialized, on reset, to a random state, which actually provides good coverage of the state space (including near the goal) and largely obviates the need for directed exploration. To make the problems more challenging, we modify the environments by performing 1000 *no-op* actions at the beginning of each episode in order to settle to a state far from the rewarding region, thus increasing the exploration required to solve the task. For more details on the environments, refer to Appendix C. We compare to all methods used for PointMaze except for OMEGA, as these tasks are not goal-directed. The tasks vary from low-dimensional (Pendulum, $|\mathcal{X}| = 4$) to relatively high-dimensional (Hopper Stand, $|\mathcal{X}| = 19$). Our method performs competitively on all domains, outperforming all baselines at early-stage learning on Acrobot, Ball in Cup, and

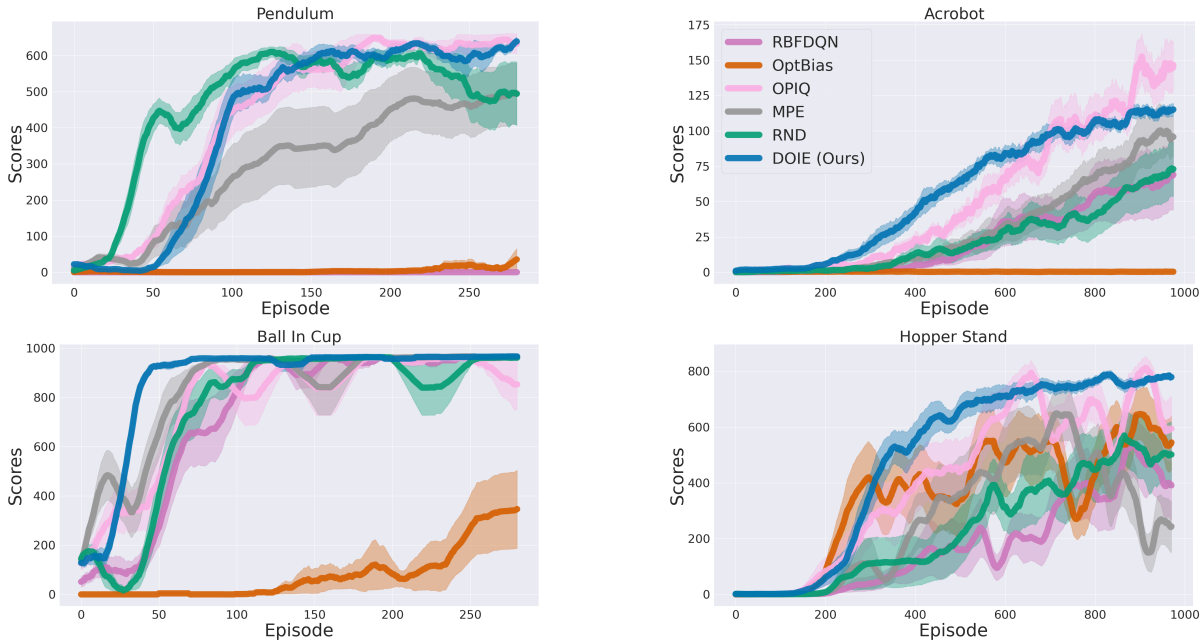

Figure 5.6: Performance on tasks in the DM Control Suite. The shaded region represents the standard deviation over 8 runs. Colors correspond to the same methods across domains.

Hopper Stand, and learning more quickly than all but RND on Pendulum.

5.2.3 Value Shaping

We now demonstrate how our method allows us to incorporate domain knowledge through specifying a state-dependent Q_{MAX}, which we call "value shaping." We test our method on MountainCar, a low-dimensional yet challenging continuous control exploration problem that requires first moving away from the goal in order to build momentum. Using the goal-position s_g, the agent's max speed $v_{\max}$, and the reward for reaching the goal r_g, we can calculate an upper bound of the optimal Q-function Q^*:

$$Q_{MAX}(s,a) \;=\; r_g \gamma^{|s_g - s|/v_{\max}}. \tag{5.11}$$

We limit the episode length to 80 steps, which necessitates directed exploration in order to reach the goal. Our results are presented in Figure 5.7.

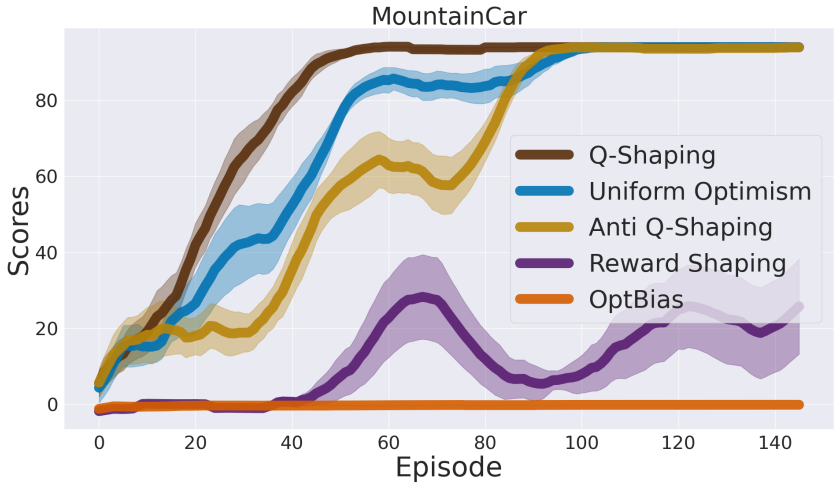

Figure 5.7: Learning curves for uniform optimistic initialization (blue) as well as shaping (brown) and anti-shaping (gold) on MountainCar. Optimistic initialization allows us to use value shaping for more efficient learning, while eventually learning an optimal policy is robust to value-shaping misspecification. In comparison to *reward* shaping, value shaping is much more effective at finding a solution. For this task, OptBias and RBFDQN (not shown) were unable to solve the task in the allotted time. The shaded region represents the standard deviation over all 10 runs.

In comparison to a uniform Q_{MAX}, when we provide a tighter upper bound through value shaping we see the agent explore much more quickly. We also include a reversal of this shaping (anti-shaping), which maintains the upper-bound property $Q_{MAX} \geq Q^*$ but sets the upper-bound value higher *away* from the goal. Though this deliberately provides the agent with bad advice, the agent still learns to reach the goal, albeit more slowly. While value shaping specifies regions of the state-space which should be explored first, since the optimism is learned away the agent still eventually explores until it finds the solution in all runs.

5.3 Relationship to Broader Work

This chapter introduces an instantiation of *optimistic initialization* in the deep reinforcement context. Key to this approach is an optimistic Q-value update rule, which interpolates between an empirical estimate of value and V_{MAX} depending on the "knownness" of a given state and action. This update rule looks remarkably similar to the improved version of the simulation lemma derived in Chapter 4. In the next section, we make this connection explicit, and create an optimistic Bellman equation based on this reasoning. This new form of optimism uses the same novelty estimates as bonus-based exploration, and thus can be used as a drop-in replacement, but produces much tighter estimates of optimistic value.

CHAPTER 6

From Additive Bonuses to V_{MAX} Interpolation

In chapter 4, we derive an optimally tight bound for the simulation lemma, a foundational result in reinforcement learning relating model uncertainty to error in predicting expected return. We improved the existing bound by noticing that transition misspecification must be conserved: probability mass that ends up somewhere needs to come from somewhere else. Therefore, if transition probability is taken from a low-value state and moved to a high-valued state, there is less low-valued probability to be wrong about in future timesteps. This sharply limits how much an incorrect model can degrade value estimation.

Many instantiations of optimism in the face of uncertainty (OFU) are derived analogously to the simulation lemma: enforcing optimism by assuming that any estimation errors are in the direction that increases value the most. This connection is unsurprising, as the simulation lemma was originally developed as part of a PAC-MDP exploration algorithm. It is natural to ask whether the lesson from chapter 4 can be applied to produce tighter optimistic value estimations for the purpose of exploration.

In this chapter, we draw a connection between the two approaches to proving the simulation lemma, and the two PAC exploration algorithms presented in the paper *An Analysis of Model-Based Interval Estimation for Markov Decision Processes* (Strehl and Littman, 2008). The first algorithm, *model-based interval estimation with exploration bonuses* (MBIE-EB), serves as the theoretical basis for bonus-based exploration (BBE) in deep reinforcement learning. The second algorithm, *model-based interval estimation* (MBIE) has attractive theoretical advantages in its optimistic estimates, but presents difficulties when porting to sample-based training.

Our primary contribution in this chapter is presenting two different update rules for approximating this tighter form of optimism in the deep reinforcement learning setting. The first of these reproduces MBIE-EB exactly using a version of distributional Q-networks (Dabney et al., 2018; Bellemare et al., 2017). The second of these provides missing theoretical motivation for the knownness-based objective presented in Chapter 5, and generalizes it to serve as a drop-in replacement for BBE. We then empirically demonstrate the advantages of this optimistic objective, which we call V_{MAX} *Interpolation*, over equivalent BBE learning rules on a variety of bonuses and methods presented earlier in this thesis. Due to its generality and simplicity, V_{MAX} Interpolation shows promise as a simple tool that can improve many existing exploration bonuses.

6.1 Background

As this work builds upon prior PAC algorithms, we describe their core ideas in more detail here.

6.1.1 The Empirical MDP

Both PAC algorithms described below work by modifying the *empirical MDP* $\tilde{\mathcal{M}}$ such that the modified MDP $\hat{\mathcal{M}}$ has optimistic values for all states and actions. The empirical

MDP is simply the MDP whose transitions and rewards exactly match the distribution observed so far. Let $B = \{(s_t, a_t, r_t, s_{t+1}) : 0 \leq t \leq T-1\}$ be the collection of observed transitions after T timesteps. Then,

$$\tilde{R}(s,a) = \mathbb{E}_{\{r_t \sim B | s_t = s, a_t = a\}}[r_t]$$

and

$$\tilde{T}(s,a,s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

Because the parameters of the empirical MDP are derived from the agent's experiences, a randomly sampled batch of experiences definitionally has the same average statistics as the empirical MDP. Therefore, minibatch training from a replay buffer, as is common in deep reinforcement learning (Mnih et al., 2015), is an approximation to solving the empirical MDP.

6.1.2 Model-Based Interval Estimation with Exploration Bonus

The heart of MBIE-EB is constructing a bonus for each state and action such that, if all other $Q(s,a)$ are optimistic (or correct), each execution of bootstrapping or value iteration is optimistic as well. We can construct such a bonus using reasoning similar to Equation 4.6. Using Hoeffding's Inequality from Equations 2.2, 2.3, we define high-confidence intervals $\epsilon_R(s,a) \propto \sqrt{\frac{1}{N(s,a)}}$, $\epsilon_T(s,a) \propto \sqrt{\frac{1}{N(s,a)}}$. From these, we construct a bonus $\mathcal{B}(s,a) = \beta(\epsilon_R(s,a) + \frac{\gamma V_{MAX}}{2}\epsilon_T(s,a))$[1].

MBIE-EB constructs an optimistic MDP $\hat{\mathcal{M}}$ by augmenting the empirical MDP with this bonus:

$$\hat{R}(s,a) = \tilde{R}(s,a) + \mathcal{B}(s,a).$$

[1]MBIE-EB derives its confidence intervals using Hoeffding's Inequality in a tighter way that does not depend on ϵ_R, ϵ_T directly, however the same $\sqrt{\frac{1}{n}}$ scaling, and overestimation arguments hold. See Section Lemma 7 of Strehl and Littman (2008)

Value iteration on $\hat{M}$ produces optimistic Q-values with high confidence. By acting greedily with respect to the optimistic Q values, the agent efficiently explores its environment. As described above, since $\hat{T} = \tilde{T}$, this optimistic learning procedure can easily be ported to the deep reinforcement learning context through minibatch experience replay (Bellemare et al., 2016; Mnih et al., 2015).

Similarly to the looser form of the simulation lemma, this bonus is constructed to upper-bound $Q(s, a)$ at each step of value iteration, but can be quite loose when considering how the optimism compounds. This is because the bonus must be large enough such that $Q(s, a)$ is optimistic even when, for example, all other $Q(s', a')$ are accurate (not optimistic). When other $Q(s', a')$ are also *also* optimistic, this bonus leads to vacuously high optimism in exactly the same way as Equation 4.4. As a contrived example, if each (s, a) has been encountered once, $\mathcal{B}(s, a) \approx V_{MAX}$. This ensures that each step of value iteration is appropriately optimistic, but when converged on will produce $Q(s, a) \approx V_{MAX}/(1 - \gamma)$.

6.1.3 Model-Based Interval Estimation

MBIE modifies both the reward and transition function of the empirical MDP, as opposed to only the reward function. For each state, reward and transition confidence intervals $\epsilon_R(s, a)$ and $\epsilon_T(s, a)$ are created. The empirical reward function is made optimistic by choosing the tail of this confidence interval:

$$\hat{R}(s, a) = \tilde{R}(s, a) + \epsilon_R(s, a).$$

At each step of value iteration, the empirical transition function is likewise modified, to produce the highest values while respecting the $\epsilon_T(s, a)$ confidence interval:

$$\hat{T}(s, a) = \underset{|\hat{T}(s,a) - \tilde{T}(s,a)|_1 \leq \epsilon_T(s,a)}{\arg\max} \sum_{s'} \hat{T}(s, a, s') \hat{V}_i(s'), \tag{6.1}$$

where $\hat{V}(s) = \max_a \hat{Q}(s,a)$. A $\hat{T}$ that satisfies Equation 6.1 moves $\frac{\epsilon_T(s,a)}{2}$ probability mass from the lowest-value successor states of each (s,a) to a V_{MAX} state[2]. At the i^{th} step of value iteration, MBIE then performs the following optimistic backup:

$$\begin{aligned}\hat{Q}_t(s,a) &\leftarrow \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s,a,s')\hat{V}_{i-1}(s') \\ &= \sum_{s'} \hat{T}(s,a,s')\Big(\hat{R}(s,a) + \gamma \hat{V}_{i-1}(s')\Big). \end{aligned} \tag{6.2}$$

At most, using $\hat{T}$ in place of $\tilde{T}$ increases the target of value iteration by $\frac{\gamma V_{MAX}}{2}\epsilon_T(s,a)$, in the case where an observed successor to s has value exactly 0. However, since the probability added to a V_{MAX} state must have come from somewhere, the effective optimism can be much smaller when the empirical successors of (s,a) themselves have high value. Note that early in exploration, all states will have high value before optimism is whittled away. Thus, especially early in training this leads to much tighter optimistic Q-value estimates. Additionally, just as in Chapter 4, this reweighting scheme naturally bounds learning targets below V_{MAX}, an intuitively desirable property for bounded optimism.

However, since $\hat{T} \neq \tilde{T}$, there is no clear way to approximate this learning target by sampling from a replay buffer. In the next section, I develop a way to compute this target with only sampled transitions, using a variant of distributional reinforcement learning (Dabney et al., 2018; Bellemare et al., 2017). I then provide an approximation to this optimistic target that generalizes my *DOIE* algorithm from Chapter 5 to a wider variety of novelty estimates.

[2]In MBIE, the probability mass is transferred to the state with highest estimated value. Assuming there is a state with exactly value V_{MAX} simplifies our argument, and is equivalent to augmenting the MDP with a self-transitioning state with value V_{MAX}.

6.2 Deep Learning Analogs to MBIE

I now demonstrate an implementation of MBIE that does not require directly estimating the optimistic transition function. This allows us to implement an approximation to MBIE in the deep learning setting. For simplicity, in the remainder of this chapter I assume that reward is deterministic—a common assumption since confidence intervals surrounding reward are generally dominated by those arising from transition uncertainty (Abel et al., 2021; Wang et al., 2020).

6.2.1 V_{MAX} Interpolation

Recall that at each step of value iteration MBIE performs the backup specified by Equation 6.2, which involved modifying the empirical transition function to put more weight on high-valued successor states. However, as described above, this learning target cannot be easily estimated using experience replay, as it requires modifying the empirical transition function. In Appendix D.1, I present an exact implementation of MBIE amenable to sample-based training, which relies on an optimistic distributional Q function (Dabney et al., 2018; Bellemare et al., 2017) instead of an optimistic transition function. Below, I further simplify that update rule to work with a standard Q function, in a way that maintains the intuitive benefits of MBIE while losing the perfect analogy to this PAC-MDP method.

Computing $\hat{T}$ from Equation 6.1 exactly is challenging: it requires linear programming and does not generalize well to the sample-based regime (Strehl and Littman, 2008). However, if we settle for a transition function that is guaranteed to be optimistic compared to $\tilde{\mathcal{M}}$ but not necessarily to $\mathcal{M}$, this task becomes much easier. Instead of drawing $\frac{\epsilon_T(s,a)}{2}$ probability from the lowest-value successor states, we draw from all successor states in proportion to their empirical likelihood:

$$\hat{T}(s,a,s') = (1 - \frac{\epsilon_T(s,a)}{2})\tilde{T}(s,a,s').$$

We then reassign the remaining $\epsilon_T/2$ probability as entering an absorbing state s_{max}, where $V(s_{max}) = V_{MAX}$:

$$\hat{T}(s,a,s_{max}) = \epsilon_T(s,a)/2.$$

This leads to a simplified update rule that only requires the empirical transition function:

$$\begin{aligned}\hat{Q}(s,a) &\leftarrow \tilde{R}(s,a) + \gamma\Big(\frac{\epsilon_T(s,a)}{2}V_{MAX} + (1 - \frac{\epsilon_T(s,a)}{2})\sum_{s'}\tilde{T}(s,a,s')V(s')\Big) \\ &= \tilde{R}(s,a) + \gamma\sum_{s'}\tilde{T}(s,a,s')\Big(\frac{\epsilon_T(s,a)}{2}V_{MAX} + (1 - \frac{\epsilon_T(s,a)}{2})V(s')\Big). \qquad (6.3)\end{aligned}$$

I call this method V_{MAX} Interpolation (VMI) because the uncertainty measurement interpolates between the empirical next-value distribution and V_{MAX}. Instead of acting as a bonus, the uncertainty measure acts like an additional discount, putting less weight on value estimates of states that the agent has not frequently visited. We note that in Jiang et al. (2015) it is shown that using a lower discount factor leads to better planning when using a mis-specified model—we expect to see similar benefits in this context.

6.2.2 Relationship Between V_{MAX} Interpolation and Bonus-Based Exploration

Consider some estimate of novelty or uncertainty, $\mathcal{B}(s,a)$. This could be a pseudocount (Bellemare et al., 2016; Lobel et al., 2023), a model-prediction error (Pathak et al., 2017), or another novelty-prediction methodology (Burda et al., 2019). In the BBE framework, we add to the task-specified reward this novelty measure, scaled by a hyperparameter κ,

as an exploration bonus. Thus, the BBE optimistic update rule can be written as follows:

$$Q^+(s,a) \leftarrow R(s,a) + \kappa\mathcal{B}(s,a) + \gamma V(s')$$

which adds $\kappa\mathcal{B}(s,a)$ to the empirical bootstrapping. Alternatively, we could use this same measure of uncertainty within the VMI update rule. Scaling $\mathcal{B}(s,a)$ by a new hyperparameter λ produces the analogous VMI update rule:

$$\begin{aligned} Q^+(s,a) &\leftarrow R(s,a) + \gamma\Big(\lambda\mathcal{B}(s,a)V_{MAX} + (1-\lambda\mathcal{B}(s,a))V(s')\Big) \\ &= R(s,a) + \gamma\lambda\mathcal{B}(s,a)\Big(V_{MAX} - V(s')\Big) + \gamma V(s'), \end{aligned} \tag{6.4}$$

which adds $\gamma\lambda\mathcal{B}(s,a)\Big(V_{MAX} - V(s')\Big)$ to the empirical bootstrapping at each step. Just as the original simulation lemma bound can be viewed as a linearized upper bound to our improved result, so too can the BBE bonus be viewed when compared to Equation 6.4. Setting

$$\lambda = \frac{\kappa}{\gamma V_{MAX}} \tag{6.5}$$

produces an identical one-step update when $V(s')$ is small, and so the above relation allows us to set up a one-to-one correspondence between bonus methods and their VMI equivalents. But as optimism compounds, $V(s')$ will grow larger, in which case BBE can converge to be over-optimistic by a factor of up to $1/(1-\gamma)$. In contrast, by construction VMI stays bounded by V_{MAX}.

6.3 Experimental Results

The results from Chapter 5 demonstrate the frequent superiority of *Deep Optimistic Initialization for Exploration* (DOIE), to a variety of optimistic baselines including an instantiation of bonus-based exploration from pseudocounts. However, these results did

not distinguish between the impact of the two major innovations within DOIE. First, DOIE quantifies uncertainty using a nearest-neighbors module especially well suited for continuous control. And second, DOIE uses the V_{MAX} Interpolation update rule from Equation 6.4 to train the optimistic Q function in place of BBE.

Here, we attempt to clarify the impact of these two contributions on the learning process. We investigate all four combinations of two uncertainty-modules from Chapter 5 (DOIE's nearest-neighbors module, and OPIQ's discretized counting module) and the two update rules (BBE and VMI), at a variety of uncertainty-bonus scales. For each method and environment, we adopt the same training configuration as used in the prior chapter, for example characteristic lengths for discretization and knownness calculation. To compare BBE and VMI we set the relative bonus scale according to Equation 6.5, and sweep over κ. Figure 6.1 presents a sweep of bonus scales on the four continuous control domains used to evaluate DOIE in Chapter 5, for both uncertainty modules listed above. As expected, we find that performance of both algorithms track each other closely for small κ scales. However, VMI successfully learns for a much larger range of bonus scales than BBE, successfully learning at 10x the value of κ with which BBE training collapses[3]. This provides evidence that V_{MAX} Interpolation serves as a more robust instantiation of optimistic exploration than its bonus-based counterpart, and that the VMI update rule can be successfully applied to a broader class of uncertainty measurements than were used for training DOIE.

[3]Note that we clip Q value targets at V_{MAX} during bootstrapping as this substantially improves stability and performance for both methods.

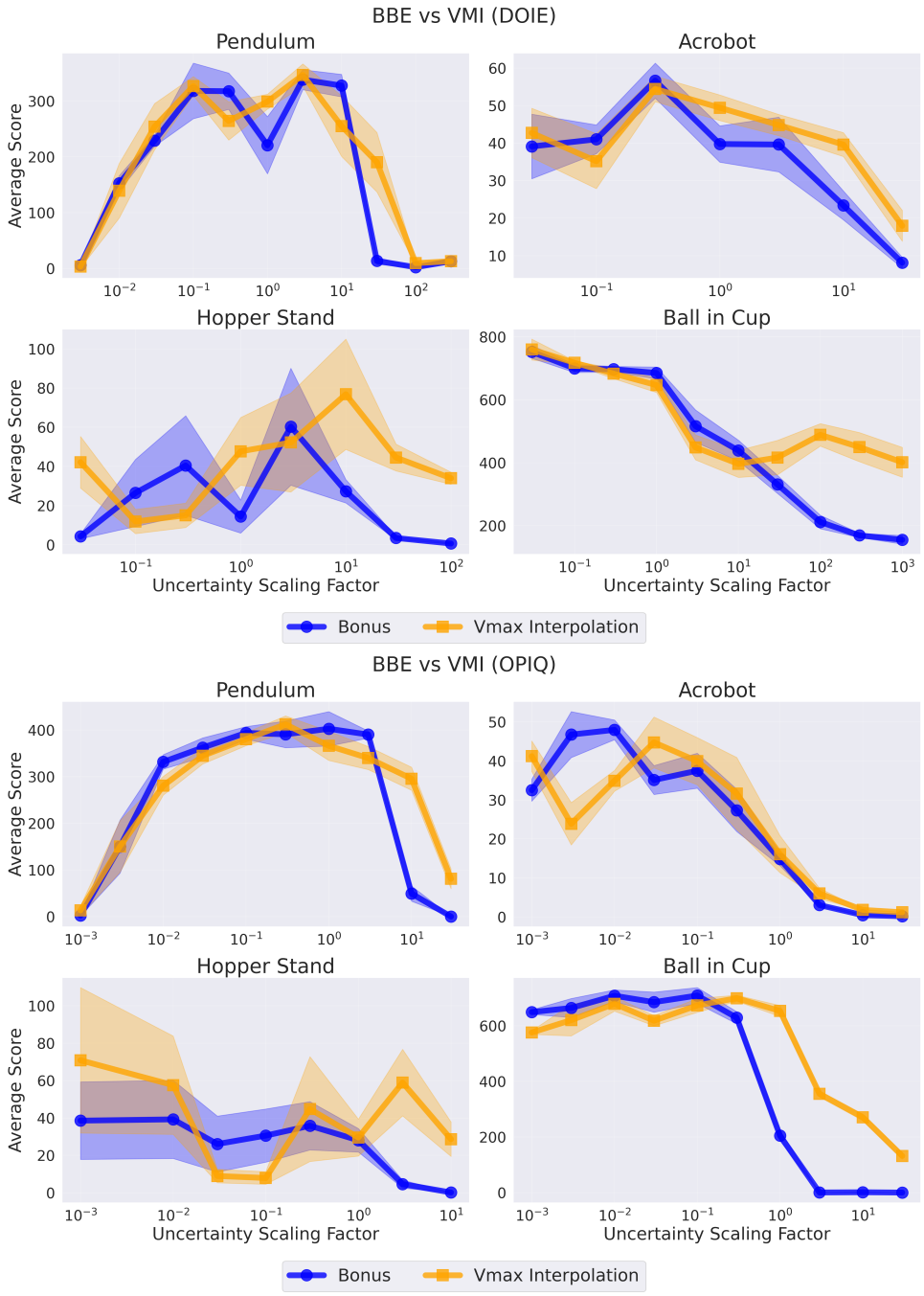

Figure 6.1: Performance on tasks in the DM Control Suite. Each data point represents the average return throughout training (as opposed to a final score), for a given method and optimism scale, averaged over 5 seeds. The shaded region represents the standard error. The x-axis represents the bonus-scale κ: VMI scale λ is computed using Equation 6.5. **(TOP)**: Uncertainty $\mathcal{B}$ is computed using DOIE's nearest-neighbor uncertainty module. In many environments, VMI training remains stable at larger optimism scales than BBE, whose performance sharply degrades. **(BOTTOM)**: Uncertainty $\mathcal{B}$ is computed through discretized counting, as used for OPIQ in Chapter 5. VMI is similarly more robust to uncertainty scale than BBE for this method of computing uncertainty.

6.4 Relationship to Broader Work

This chapter builds on the one before, providing a better-grounded theoretical source for the DOIE update rule, and generalizing it to a broader class of uncertainty estimates. Most importantly, I show how this generalization, which I call V_{MAX} Interpolation, is a modification to bonus-based exploration that comes from reasoning more carefully about compounding transition misspecification.

This chapter serves to tie the prior three works together in a single algorithm. The careful reasoning about compounding transition misspecification was inspired by my modified simulation lemma (Chapter 4), and the improvements come from the same source. The resulting algorithm generalizes DOIE (Chapter 5) to work with many ways of estimating novelty. In particular, I apply it to a (discretization-derived) count-based bonus, where it stabilizes learning over a wider range of bonus scales. In future work I intend to similarly modify my CFN-derived pseudocount bonus (Chapter 3), which will cleanly integrate the most important ideas of this dissertation in one project.

CHAPTER 7

Conclusion

A hallmark of reinforcement learning, the source of its generality as well as its difficulty, is that a reinforcement learning agent must gather its own training data. Though an agent is usually equipped with a reward function that judges the quality of its current policy, it is not given a way to know a priori which parts of its environment are important to visit as part of learning. That is the problem with which this dissertation concerns itself: how can an agent determine what is worth learning more about, and how can it use that knowledge to learn?

This dissertation is a small but concrete step forward for one of the most important problems in deep reinforcement learning, that of efficient exploration in large-scale domains. The challenge of exploration starts at the beginning: in deep RL, even identifying what parts of the world are novel or unknown is the subject of active research. The first contribution in this dissertation is a method for estimating novelty in environments with rich observation spaces. To do this, we derived a rigorous connection between random sampling and novelty estimation that is amenable to function approximation. We turn this insight into an algorithm by way of an exploration bonus, that rewards the agent for visiting new parts of the environment. We then move our focus towards how to use a given novelty estimate for better exploration. By deriving an improved form of the *simulation*

lemma, we formalize a way in which bonuses can lead to over-optimistic values. Finally, we apply this logic to deep RL to create a class of exploration algorithms that offer a general alternative to bonus-based exploration. This dissertation serves as a toolkit for, and advances our understanding of, how to do optimistic exploration in deep reinforcement learning.

The challenge of exploration for interactive agents is evergreen. As reinforcement learning methods are applied to larger models and more general domains, the goal shifts from discovering a narrowly-defined optimal policy to growing the class of tasks that an agent can solve. In particular, modern foundation models begin with an incredible amount of world knowledge, and the next generation of exploration techniques must take advantage of this starting point and expand from there. Novelty estimation directly over states is in some sense a tabula-rasa approach; RL exploration in this new world requires an exciting new toolkit. This dissertation introduces simple methods for using prior knowledge such as value shaping in Chapter 5, and insights such as the uncertainty estimation on which CFN is based are finding exciting application in these more general domains. Much work, however, needs to be done to discover how to best explore and use that exploration in these large interactive models. At the time of this writing, pretrained foundation models remain largely bounded by the knowledge encoded in their initial training data, despite its breadth. Exploration is the mechanism by which these systems can move beyond static competence toward open-ended growth through interaction.

References

David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pages 2915–2923. Proceedings of Machine Learning Research, 2016.

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. Proceedings of Machine Learning Research, 2020.

David Abel, Will Dabney, Anna Harutyunyan, Mark K Ho, Michael Littman, Doina Precup, and Satinder Singh. On the expressivity of Markov reward. *Advances in Neural Information Processing Systems*, 34:7799–7812, 2021.

Cameron Allen, Neev Parikh, Omer Gottesman, and George Konidaris. Learning Markov state abstractions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 264–273. Proceedings of Machine Learning Research, 2018.

Kavosh Asadi, Neev Parikh, Ronald E Parr, George D Konidaris, and Michael L Littman. Deep radial-basis value functions for continuous control. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 6696–6704, 2021.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for

reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 263–272, 06–11 Aug 2017.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. Proceedings of Machine Learning Research, 2017.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Ronen I Brafman and Moshe Tennenholtz. R-max—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, pages 213–231, 2002.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Benjamin Burchfiel, Carlo Tomasi, and Ronald Parr. Distance minimization for reward learning from scored trajectories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL `http://arxiv.org/abs/1812.06110`.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 1, 2018.

Thomas Dietterich. The MAXQ method for hierarchical reinforcement learning. In *International Conference on Machine Learning*, volume 98, pages 118–126, 1998.

Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 240–247, 2008.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. Proceedings of Machine Learning Research, 2018.

Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Linear Algebra*, 2:134–151, 1971.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

In *International Conference on Machine Learning*, pages 1861–1870. Proceedings of Machine Learning Research, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Ronald A Howard. Dynamic programming and Markov processes. 1960.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(4), 2010.

Nan Jiang. Notes on tabular methods. 2018. URL `https://nanjiang.cs.illinois.edu/files/cs542f22/note3.pdf`.

Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189, 2015.

Nan Jiang, Satinder Singh, and Ambuj Tewari. On structural properties of MDPs that bound loss due to shallow planning. In *International Joint Conferences on Artificial Intelligence*, volume 8, page 1, 2016.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is Q-learning provably efficient? *Advances in Neural Information Processing Systems*, 31, 2018.

Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning*, pages 306–312, 2003.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar,

and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

Sam Lobel and Ronald Parr. An optimal tightness bound for the simulation lemma. *Reinforcement Learning Journal*, 2:785–797, 2025.

Sam Lobel, Omer Gottesman, Cameron Allen, Akhil Bagaria, and George Konidaris. Optimistic initialization for exploration in continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7612–7619, 2022.

Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In *International Conference on Machine Learning*, pages 22594–22613. Proceedings of Machine Learning Research, 2023.

Marlos C Machado, Sriram Srinivasan, and Michael Bowling. Domain-independent

optimistic initialization for reinforcement learning. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

Chengzhuo Ni, Lin F Yang, and Mengdi Wang. Learning to control in metric space with optimal regret. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing*, pages 726–733, 2019.

Ali Nouri and Michael Littman. Multi-resolution exploration in continuous spaces. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, pages 1209–1216, 2009.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. Proceedings of Machine Learning Research, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. Proceedings of Machine Learning Research, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. Proceedings of Machine Learning Research, 2019.

Jason Pazis and Ronald Parr. Pac optimal exploration in continuous space Markov decision processes. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 7750–7761. Proceedings of Machine Learning Research, 2020.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. 2018. URL `http://arxiv.org/abs/1802.09464`.

Carlos Plou, Ana C Murillo, and Ruben Martinez-Cantin. Active exploration in bayesian model-based reinforcement learning for robot manipulation. *arXiv preprint arXiv:2404.01867*, 2024.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems*, 2018.

Tabish Rashid, Bei Peng, Wendelin Boehmer, and Shimon Whiteson. Optimistic exploration even with a pessimistic initialisation. In *International Conference on Learning Representations*, 2020.

Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, Inc., 1987.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1):123–158, 1996.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael Littman. PAC model-free reinforcement learning. *International Conference on Machine Learning*, 2006.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

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

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

Adrien Ali Taiga, William Fedus, Marlos C. Machado, Aaron Courville, and Marc G. Bellemare. On bonus based exploration methods in the arcade learning environ-

ment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJewlyStDr.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems*, pages 10376–10386, 2019.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

Ruosong Wang, Simon S Du, Lin Yang, and Russ R Salakhutdinov. On reward-free reinforcement learning with linear function approximation. *Advances in Neural Information Processing Systems*, 33:17816–17826, 2020.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4): 279–292, 1992.

Ming Yin, Yu Bai, and Yu-Xiang Wang. Near-optimal provable uniform convergence in offline policy evaluation for reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1567–1575. Proceedings of Machine Learning Research, 2021.

Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

Appendix

Appendix A

Flipping Coins to Estimate Pseudocounts for Exploration

A.1 Proofs

A.1.1 Any Zero-Mean Unit-Variance Distribution can be Used for Counting

Let $z_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ where $x_i \sim \mathcal{X}$. Assume that the distribution $\mathcal{X}$ is such that $\mathbb{E}[\mathcal{X}] = 0$ and $\mathrm{Var}[\mathcal{X}] = 1$. In many cases we make use of the fact that $\mathbb{E}[x_i x_j] = \delta_{ij}$ (Kronecker delta function), because if $i \neq j$, then $\mathbb{E}[x_i x_j] = \mathbb{E}[x_i]\mathbb{E}[x_j] = 0$.

$$\begin{aligned}
\mathbb{E}\big[z_n^2\big] &= \mathbb{E}\Big[\Big(\frac{1}{n}\sum_{i=1}^{n} x_i\Big)^2\Big] \\
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j\Big] \\
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n} x_i x_i + \sum_{i=1}^{n}\sum_{j\neq i}^{n} x_i x_j\Big]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n} x_i^2\Big] + \frac{1}{n^2}\mathbb{E}\Big[\sum_{i=1}^{n}\sum_{j\neq i}^{n} x_i x_j\Big] \\
&= \frac{1}{n^2}(n+0) \\
&= \frac{1}{n}
\end{aligned}
$$

This proves the well-known fact that the variance of the sample mean scales inversely with the number of samples.

A.1.2 Functional Form of the Variance of z_n^2

We now know that z_n^2 is an unbiased estimator of $\frac{1}{n}$, the inverse count. What is the variance of this estimate?

$$
\begin{aligned}
\mathrm{Var}\big[z_n^2\big] &= \mathbb{E}\big[z_n^4\big] - \mathbb{E}\big[z_n^2\big]^2 \\
&= \mathbb{E}\Big[(\frac{1}{n}\sum_{i=1}^{n} x_i)^4\Big] - \frac{1}{n^2} \\
&= \frac{1}{n^4}\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] - \frac{1}{n^2} \\
\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] &= \mathbb{E}\Big[\sum_i\sum_j\sum_k\sum_l x_i x_j x_k x_l\Big] \\
&= \mathbb{E}\Big[\sum_{i=j=k=l} x_i^4 + \sum_{i=j,k=l\neq i} x_i^2 x_k^2 + \sum_{i=k,l=j\neq i} x_i^2 x_l^2 \\
&\quad + \sum_{i=l,j=k\neq i} x_i^2 x_k^2 + \sum_{\text{remaining}} x_i x_j x_k x_l\Big]
\end{aligned}
$$

where here we have broken the summation across all indices i, j, k, l apart into four sets of terms with even exponents, and one set which all have at least one odd exponent (and thus has expectation 0). There are n elements in the first sum, and $n(n-1)$ elements of

the second, third, and fourth sum. Thus:

$$\mathbb{E}\Big[\sum_{i=j=k=l} x_i^4\Big] = n\mathbb{E}\big[\mathcal{X}^4\big]$$
$$\mathbb{E}\Big[\sum_{i=j,k=l\neq i} x_i^2 x_k^2\Big] = n(n-1)\mathbb{E}\big[\mathcal{X}^2\big]^2$$
$$\mathbb{E}\Big[\sum_{\text{remaining}} x_i x_j x_k x_l\Big] = 0$$

and therefore,

$$\mathbb{E}\Big[(\sum_{i=1}^{n} x_i)^4\Big] = n\mathbb{E}\big[\mathcal{X}^4\big] + 3n(n-1)\mathbb{E}\big[\mathcal{X}^2\big]^2$$
$$= n\mathbb{E}\big[\mathcal{X}^4\big] + 3n^2 - 3n$$

Plugging into the original equation we arrive at a functional form of $\mathrm{Var}[z_n^2]$:

$$\mathrm{Var}\big[z_n^2\big] = \frac{1}{n^3}\mathbb{E}\big[\mathcal{X}^4\big] + \frac{3}{n^2} - \frac{3}{n^3} - \frac{1}{n^2}$$
$$= \frac{1}{n^3}\mathbb{E}\big[\mathcal{X}^4\big] + \frac{2}{n^2} - \frac{3}{n^3} \tag{A.1}$$

A.1.3 Proof that Using the Coin-Flip Distribution Yields the Lowest-Variance Estimator of $\frac{1}{n}$

Above, we show that to reduce $\mathrm{Var}\big[z_n^2\big]$ the only knob we can turn is reducing the 4^{th} moment, because $\mathbb{E}\big[\mathcal{X}\big] = 0$ and $\mathbb{E}\big[\mathcal{X}^2\big] = 1$ by construction. The 4^{th} moment of the coin-flip distribution $x \sim \mathcal{C} \in \{-1, 1\}$ is simply

$$\mathbb{E}[\mathcal{C}^4] = \frac{1}{2}1^4 + \frac{1}{2}(-1)^4 = 1.$$

Plugging this into Equation A.1 yields, for $x \sim \mathcal{C}$:

$$\mathrm{Var}\big[z_n^2\big] = \frac{2}{n^2} - \frac{2}{n^3}.$$

This holds for all $n \geq 1$. Therefore, $\mathrm{Var}\big[z_1^2\big] = 0$, which is easy to confirm. Since variance must always be positive, this means that $\mathbb{E}[x^4] \geq 1$ for all $\mathcal{X}$ satisfying our assumptions, and hence using the coin-flip distribution achieves minimum possible variance in its estimation of $\frac{1}{n}$.

A.1.4 Proof that Variance Scales Inversely with Vector-Length

We have another, simpler method for reducing variance: we can increase the number of trials (equivalently, the number of coin flips), and average the results together.

Let $\{z_{ni} | i \in 1, ..., d\}$ be a set of d independent draws of z_n. Then

$$\mathbb{E}\Big[\frac{1}{d}\sum_{i=1}^{d} z_{ni}^2\Big] = \frac{1}{d}\sum_{i=1}^{d}\mathbb{E}\big[z_{ni}^2\big] = \mathbb{E}\big[z_n^2\big] = \frac{1}{n}$$

as expected (the average of i.i.d samples is simply the expectation). Similarly:

$$\mathrm{Var}\Big[\frac{1}{d}\sum_{i=1}^{d} z_{ni}^2\Big] = \frac{1}{d^2}\sum_{i=1}^{d}\mathrm{Var}\big[z_{ni}^2\big] = \frac{1}{d}\mathrm{Var}\big[z_n^2\big]$$

This is a well-known fact of how variance scales with samples. Therefore, we can decrease the variance of our estimator in two ways: picking a good distribution (coin-flip is best) and also increasing the number of trials.

A.2 Analysis of Linear Coin Flip Network

We now analyze the solution to Equation 3.3 when our function approximator f_ϕ is a linear mapping between states and coin flips. Our goal is to recover an intuitive

understanding of how bonuses are estimated when the model has to generalize across inputs. For simplicity, we consider the case of a single coin flip per encountered state. We represent each state $\mathbf{s}$ as a p-dimensional vector, with $\mathbf{S}$ being the $n \times p$ matrix of all encountered states , and $c \sim \{\text{-}1, 1\}^n$ being the n-dimensional vector of sampled coin flips.

Thus, $f_\phi(\mathbf{s}) = \mathbf{s} \cdot \boldsymbol{\phi}$, where $\boldsymbol{\phi}$ is the weight vector that parameterizes f_ϕ. Under this formulation, Equation 3.3 reduces to solving the following linear least-squares regression problem:

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi'}} \|\mathbf{S}\boldsymbol{\phi'} - \mathbf{c}\|^2.$$

For the following derivation, we assume that $\mathbf{S}$ has rank p (and $n > p$) in order to recover a unique solution, however this result can be generalized by replacing the inverse with the pseudo-inverse. The solution to this linear regression (Golub and Reinsch, 1971) is

$$\boldsymbol{\phi} = (\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{c}.$$

We now rewrite $\mathbf{S}$ using its singular value decomposition (Golub and Reinsch, 1971): $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^T$, where $\mathbf{U}$ is the $n \times n$ orthonormal "left singular vector" basis, $\mathbf{V}$ is the $p \times p$ orthonormal "right singular vector" basis, and $\boldsymbol{\Lambda}$ is a $n \times p$ rectangular diagonal "singular value" matrix. Thus, $\mathbf{S^T S} = \mathbf{V\Lambda^T U^T U\Lambda V^T} = \mathbf{V\Lambda^T\Lambda V^T}$. Therefore:

$$\begin{aligned}
\boldsymbol{\phi} &= (\mathbf{S^T S})^{-1}\mathbf{S^T c} \\
&= (\mathbf{V\Lambda^T\Lambda V^T})^{-1}\mathbf{V\Lambda^T U^T c} \\
&= \mathbf{V}(\mathbf{\Lambda^T\Lambda})^{-1}\mathbf{V^T V\Lambda^T U^T c} \\
&= \mathbf{V}(\mathbf{\Lambda^T\Lambda})^{-1}\mathbf{\Lambda^T U^T c} \\
&= \mathbf{V\Lambda^{-1}U^T c}
\end{aligned}$$

where $\boldsymbol{\Lambda}^{-1}$ is the pseudo-inverse of $\boldsymbol{\Lambda}$, which in the case of a rectangular diagonal matrix

is simply the element-wise reciprocal of the diagonal entries, transposed.

Recall that $f_\phi(s) = \mathbf{s} \cdot \boldsymbol{\phi}$. We can gain more intuition about f_ϕ by representing $\mathbf{s}$ using the orthonormal basis $\mathbf{V}$: $\mathbf{s} = \sum_i (\mathbf{s^T} \cdot \mathbf{V_i^T}) \mathbf{V_i^T} = \mathbf{p V^T}$ where $\mathbf{p}$ says how much of each basis vector there is in $\mathbf{s}$. Now we can derive a simple formula for the expected inverse-count of $\mathbf{s}$:

$$\begin{aligned}
\mathbb{E}[\frac{1}{\mathcal{N}(\mathbf{s})}] &= \mathbb{E}[\|f_\phi(\mathbf{S})\|]^2 \\
&= \mathbb{E}[\mathbf{s}\boldsymbol{\phi}\boldsymbol{\phi}^{\boldsymbol{T}}\mathbf{s^T}] \\
&= \mathbb{E}[\mathbf{p V^T V \Lambda^{-T} U^T c c^T U \Lambda^{-1} V^T V p^T}] \\
&= \mathbf{p V^T V \Lambda^{-T} U^T}\ \mathbb{E}[\mathbf{c c^T}]\ \mathbf{U \Lambda^{-1} V^T V p^T}
\end{aligned}$$

Since each element of $\mathbf{c}$ is independent of all others, $\mathbb{E}[\mathbf{cc^T}] = \mathbb{I}$. Furthermore, $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases and so $\mathbf{U^T U} = \mathbb{I}$ and $\mathbf{V^T V} = \mathbb{I}$. Thus, we get the following simplification:

$$\mathbb{E}[\frac{1}{\mathcal{N}(\mathbf{s})}] = \mathbf{p}(\mathbf{\Lambda^T \Lambda})^{-1}\mathbf{p^T}.$$

In other words, when using a linear model, CFN stores in its weights how much of each singular vector is present in the dataset $\mathbf{S}$. When presented with a new state $\mathbf{s}$, it computes an inverse count for each singular vector, and returns a linear combination of those, weighted by how much of that singular vector is present in $\mathbf{s}$.

This matches intuition in the tabular case. If each $s \in S$ is represented by a one-hot vector, then $\mathbf{S^T S}$ is a diagonal matrix that counts how many times each unique state has been seen, and $\mathbf{p}(\mathbf{\Lambda^T \Lambda})^{-1}\mathbf{p^T}$ returns the the inverse count of a given state. In the more general linear case, counting simply happens over a different basis than the identity matrix.

A.3 Environment Details

Visual Gridworld. We used a 42x42 gridworld, observations were 84x84 images (each grid-cell was visualized using 2x2 pixels). The player always started the episode at the bottom-left and the goal was at the top-right. Episodes lasted a maximum of 150 steps, unless the agent reached the goal, in which case the episode terminates with a sparse reward of 1. For the stochasticity experiments in Figure 3.4(right), the maximum number of steps per episode was determined by $\frac{150}{1-\eta}$ where η is the action-noise probability. We did this to make sure that more stochastic versions of the task still had long-enough episodes to be solvable by a reasonably good agent: with this scaling the agent has the same number of actions under its own control in a given episode, independent of η.

Visual Taxi. We used two variants of the Taxi—one with a 5x5 grid (as in the default version) and another with a 10x10 grid (Diuk et al., 2008); episodes lasted a maximum of 50 steps in the former and 100 in the latter case. Similar to visual gridworld, the agent observed images of the game state; to show that the passenger was inside the taxi, we shaded the taxi differently and included a black border around the image.

Fetch and Adroit Manipulation. As mentioned in Section 3.2.4, we first sparsified the reward function—this means that there are no shaping rewards for reaching the object or for moving it to non-goal locations. To set the goal locations for the non-default versions of the tasks, we first visualized the environment and rendered the effect of random actions. The exact goal locations that we settled on can be found in our linked code (file `wrappers.py`). In the Relocate task, we truncate the episode when the ball leaves the table.

Ant U-Maze. D4RL randomly sets the goal location to a small distribution around $(x = 0, y = 8)$ and the ant's location to a small distribution around $(x = 0, y = 0)$. Predictably, we used the sparse-reward version of the task. Episodes last a maximum of

1000 steps.

Montezuma's Revenge We followed the experimental protocol of Machado et al. (2015) which means that we used sticky actions, a frame stack of 4, action repeat of 4, grayscale images of shape 84x84 and a training budget of 200 million frames (50 million agent-environment interactions).

A.4 Hyperparameters, Architecture and Training Details

The final values of hyperparameters for all experiments have been set as the default values in our configuration files, which can be found in the `configs/*.gin` files in our codebase. The file `intrinsic_motivation/intrinsic_rewards.py` contains architectural details such as number of layers and layer sizes, and unless otherwise noted are the same as presented in (Taiga et al., 2020). The neural network architecture of CFN is chosen to match that of RND's prediction network.

All hyperparameters *not* listed are chosen to match the default Rainbow Hessel et al. (2018) and SAC Haarnoja et al. (2018) implementations. For each task group, the tested hyperparameters are listed; multiple value indicate a grid search, with the chosen value listed in bold. On Montezuma's Revenge we use the best reported RND and PixelCNN hyperparameters from (Taiga et al., 2020).

A.4.1 Shared Hyperparameters

We used Rainbow for Visual Gridworld (Section 3.2.3) and Montezuma's Revenge (Section 3.2.5) and SAC for all the continuous control experiments (Section 3.2.4). The hyperparameters for these base agents are reported below:

Rainbow Hyperparameter	Gridworld	Atari
Discount Factor γ	0.99	0.99
Adam LR	**1.25e-4**, 1e-5	**1.25e-4**, 6.25e-5
Adam ϵ	1.5e-4	1.5e-4
Multi-step return n	3	3
Min history to start learning	1,000	20,000
Distributional Atoms	51	51
Distributional Min/Max values	$\pm$10	$\pm$10
Batch Size	32	32

SAC Hyperparameter	Fetch	Adroit	Ant
Discount Factor γ	0.99	0.99	0.99
Adam LR	3e-4	**1e-4** 3e-4	3e-4
Adam ϵ	1e-8	1e-8	1e-8
Multi-step return n	3	3	3
Min history to start learning	10,000	10,000	10,000
SAC Reward Scale Factor	1.0	1.0, 3.0, 10.0, **30.0**	0.1
Batch Size	256	256	256

A.4.2 CFN Hyperparameters

Hyperparameter	Gridworld	Fetch	Adroit	Ant	Atari
Intrinsic Reward Scale	0.001, 0.003 **0.01**, 0.03	**0.001**, 0.003 0.01, 0.03	**0.001**, 0.003 0.01, 0.03	0.001, 0.003 **0.01**, 0.03	0.001, 0.003 **0.01**, 0.03
Reward Normalization*	**Yes**, No	No	No	No	No
CFN learning rate	1e-4	1e-4	1e-4	1e-3	1e-4 **1e-5**
CFN replay buffer size	1e6	1e6	1e6	1e6	2e7
CFN batch size	1024	1024	1024	1024	512
CFN update period	1	1	1	1	4
Number of coin flips d	20	20	20	20	20

* **Reward normalization.** In Visual Gridworld, we found it helpful to use reward normalization (Burda et al., 2019), which normalizes the exploration bonus by subtracting its running mean and dividing by its running variance.

A.4.3 CFN Replay Buffer Size

Note that we used a much larger CFN replay buffer for Montezuma's Revenge. Usually, a small FIFO queue is used to implement the replay buffer. As shown by Zhang and Sutton (2017), larger replay buffers hurt RL performance because it makes the Q-learning updates more off-policy. Since the CFN objective is a standard regression problem (and does not use bootstrap targets), larger buffers almost always improve performance. Furthermore, shorter CFN buffers often cause high-count states to be removed from the replay buffer and eventually re-appear as novel to the agent; a problem that is mitigated with larger buffer sizes. In future work, we would like to revert to smaller replay buffers by implementing a "forgetting" strategy in which low novelty states are discarded from replay with higher probability.

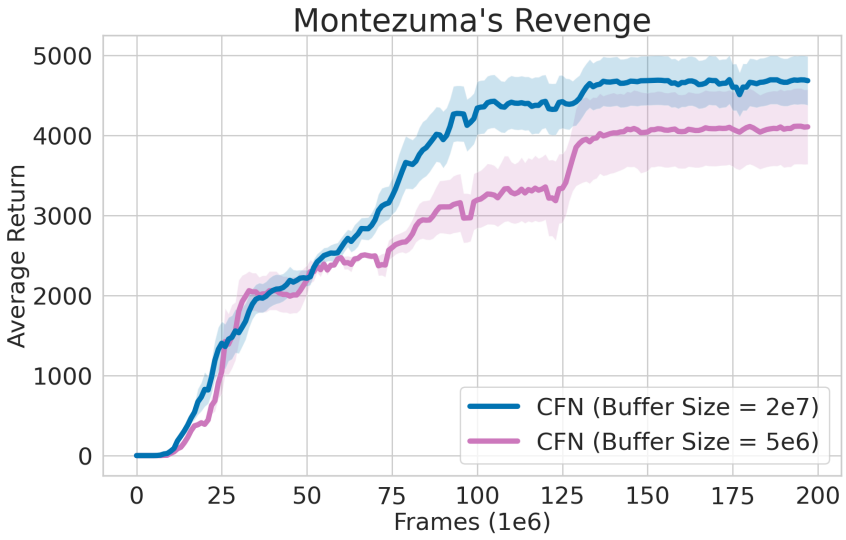

Figure A.1: Analyzing the impact of large replay buffer for CFN: while a very large buffer size of $2e7$ leads to better performance, a smaller buffer size of $5e6$ still performs well. Results are averaged over 6 random seeds.

Figure A.1 shows that while a very large replay buffer does indeed yield better performance in Montezuma's Revenge, a moderately sized replay buffer also performs respectably.

A.4.4 PixelCNN Hyperparameters

Hyperparameter	Gridworld	Atari
Intrinsic Reward Scale	0.1, **0.5**, 1.0	0.1
Prediction Gain Scale	0.1, **0.5**, 1.0, 5.0	1.0
PixelCNN learning rate	1e-4	1e-4

A.4.5 RND Hyperparameters

Hyperparameter	Gridworld	Fetch	Adroit	Ant	Atari
Intrinsic Reward Scale	5e-5,**1e-4**, 5e-4,1e-3	**5e-5**,1e-4, 5e-4,1e-3	**5e-5**,1e-4, 5e-4,1e-3	**5e-5**,1e-4, 5e-4,1e-3	5e-5
RND learning rate	1e-4	1e-4	1e-4	1e-4	1e-4

A.4.6 Compute Resources

The wallclock time of all intrinsic reward methods is roughly similar, however on Atari domains CFN requires significantly more memory—see Section A.4.2 for discussion. All experiments are performed on a SLURM cluster using nodes equipped with 4 CPUs and 1 3090-Ti NVIDIA GPU. We reserve 16GB of memory for all experiments except for CFN's Montezuma's Revenge, where we reserve 160GB.

A.5 Additional Experiments

A.5.1 Counts on Visual Taxi

We report count reconstruction-accuracy on visual versions of both the 5x5 and the 10x10 TAXI environments in Figure A.2. See Appendix A.3 for environment details. These tasks are visually more complex and have many more possible states than Visual Gridworld. Since the policy used to collect data is a confounding variable when comparing counting accuracy, in these experiments all interaction is performed with a random policy, and only the bonus modules are trained. For both domains, we present bonuses computed after $200,000$ interactions, with each method taking one training step per interaction. In this time, the random policy visits approximately 900 unique states in the 5x5 domain, and approximately $6,500$ unique states in the 10x10 domain. We note a similar trend as in Gridworld, where CFN procudes more accurate bonuses than PixelCNN and RND.

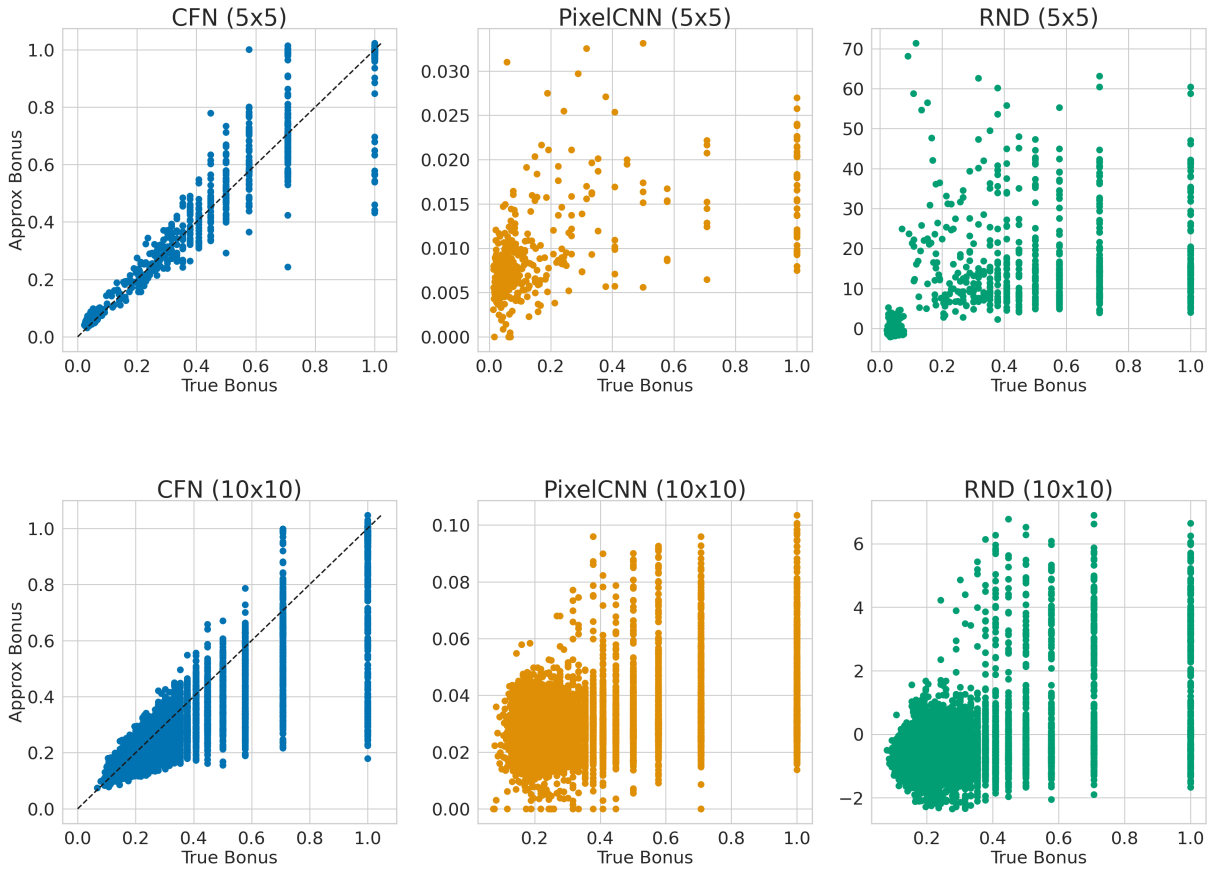

Figure A.2: Bonuses for CFN, PixelCNN, and RND on both Taxi domains.

A.5.2 Effect of Random Prior Without Coin Flips

Can the accuracy of the predicted exploration bonuses in Figure 3.2 solely be attributed to the use of the random prior (discussed in Section 3.1.5)? To answer this question, we devised an experiment in which we removed the randomness introduced by the coin-flip vectors by setting them to $\{0\}^d$ instead of sampling them from $\{-1, 1\}^d$.

Figure A.3 shows the result of this experiment: states seen for the first time get a high exploration bonus, implying that the random prior initializes their pseudocount near 1 (as intended). On the other hand, states visited more than once get almost no exploration bonus; this highlights the importance of the full CFN objective to get an exploration bonus that falls off smoothly as $1/\sqrt{\mathcal{N}(s)}$.

Furthermore, notice that states with true bonus of 1 form two distinct clusters—with high and low predicted bonus respectively. We posit that states in the lower bonus cluster were encountered less recently and thus have been sampled by CFN, which has wiped out the effect of their optimistic initialization.

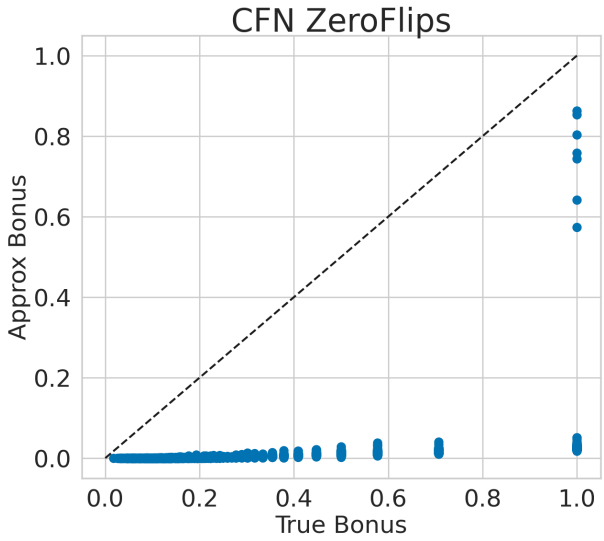

Figure A.3: CFN bonus prediction versus ground-truth bonus on the 42x42 Visual Gridworld domain using $c \sim \{0\}^d$.

A.5.3 Prioritization and Random Prior Ablation Count Plots

Figure A.4 shows the importance of prioritized sampling and optimistic bonus initialization. Using both methods results in the most accurate bonus predictions across the entire range of novelty. When only random prior is used, CFN does not train as frequently on novel states, and thus underestimates their count. When only prioritization is used, a novel state may be first inserted into the dataset with low novelty (and thus low priority); so, the state may not be sampled for training, which retains its underestimate of novelty. When neither prioritization nor random prior are used, bonuses are still accurate for states observed more than 5 times, but are inaccurate for very novel states.

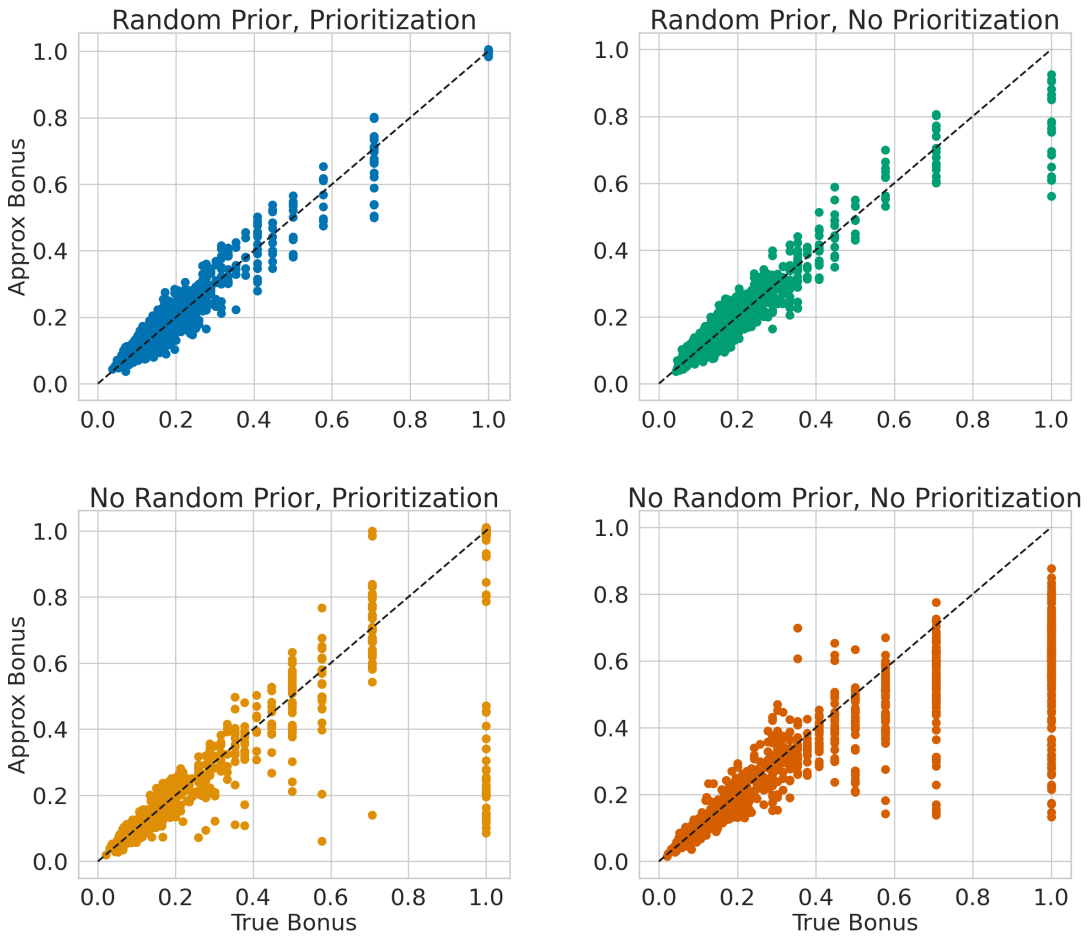

Figure A.4: True versus approximate bonus of including/excluding random prior and prioritization.

Appendix B

An Optimal Tightness Bound for the Simulation Lemma

B.1 Full Proof of the Simulation Lemma

For completeness, we include the proof of the simulation lemma found in Jiang (2018). We adopt notation from Section 4.1.

$$
\begin{aligned}
|V_s^\pi - \hat{V}_s^\pi| &= |R_s^\pi + \gamma\langle P_s^\pi, V^\pi\rangle - \hat{R}_s^\pi - \gamma\langle \hat{P}_s^\pi, \hat{V}^\pi\rangle| \\
&\leq \epsilon_R + \gamma|\langle P_s^\pi, V^\pi\rangle - \langle \hat{P}_s^\pi, V^\pi\rangle + \langle \hat{P}_s^\pi, V^\pi\rangle - \langle \hat{P}_s^\pi, \hat{V}^\pi\rangle| \\
&= \epsilon_R + \gamma|\langle P_s^\pi, V^\pi - \frac{\mathbf{1}}{2(1-\gamma)}\rangle - \langle \hat{P}_s^\pi, V^\pi - \frac{\mathbf{1}}{2(1-\gamma)}\rangle + \langle \hat{P}_s^\pi, V^\pi\rangle - \langle \hat{P}_s^\pi, \hat{V}^\pi\rangle| \\
&\leq \epsilon_R + \gamma\|P_s^\pi - \hat{P}_s^\pi\|_1 \cdot \|V^\pi - \frac{\mathbf{1}}{2(1-\gamma)}\|_\infty + \gamma\|\hat{P}_s^\pi\|_1 \cdot \|V^\pi - \hat{V}^\pi\|_\infty \\
&\leq \epsilon_R + \frac{\gamma\epsilon_T}{2(1-\gamma)} + \gamma\|V^\pi - \hat{V}^\pi\|_\infty \\
\implies |V_s^\pi - \hat{V}_s^\pi| &\leq \frac{\epsilon_R}{1-\gamma} + \frac{\gamma\epsilon_T}{2(1-\gamma)^2}.
\end{aligned}
$$

This proof makes use of Hölder's inequality to bound inner products with L_1 and L_∞ norms, as well as centers the value $0 \leq V_s^\pi \leq \frac{1}{1-\gamma}$ through subtracting the midpoint for improved bounds.

B.2 Application to the Finite-Horizon Setting

We now extend our improved bound to the finite-horizon, undiscounted setting, where an agent interacts with an environment for H steps. One difference in this setting is that policies are conditioned on timestep as well as state; hence we define $\pi = [\pi_0, \ldots, \pi_{H-1}]$. Existing bounds in the finite-horizon setting establish a relationship between values at subsequent timesteps. Noting that $0 \leq V_{h,s}^\pi \leq H - h$ (and defining $V_{H,s}^\pi = 0$), Then,

$$
\begin{aligned}
|V_{h,s}^\pi - \hat{V}_{h,s}^\pi| &= |R_s^{\pi_h} + \langle P_s^{\pi_h}, V_{h+1}^\pi \rangle - \hat{R}_s^{\pi_h} - \langle \hat{P}_s^{\pi_h}, \hat{V}_{h+1}^\pi \rangle| \\
&\leq \epsilon_R + |\langle P_s^{\pi_h}, V_{h+1}^\pi \rangle - \langle \hat{P}^{\pi_h}, V_{h+1}^\pi \rangle + \langle \hat{P}^{\pi_h}, V_{h+1}^\pi \rangle - \langle \hat{P}_s^{\pi_h}, \hat{V}_{h+1}^\pi \rangle| \\
&= \epsilon_R + |\langle P_s^{\pi_h}, V_{h+1}^\pi - \frac{H-h-1}{2} \cdot \mathbf{1} \rangle - \langle \hat{P}^{\pi_h}, V_{h+1}^\pi - \frac{H-h-1}{2} \cdot \mathbf{1} \rangle \\
&\qquad + \langle \hat{P}^{\pi_h}, V_{h+1}^\pi \rangle - \langle \hat{P}_s^{\pi_h}, \hat{V}_{h+1}^\pi \rangle| \\
&\leq \epsilon_R + \|P_s^{\pi_h} - \hat{P}_s^{\pi_h}\|_1 \cdot \|V_{h+1}^\pi - \frac{H-h-1}{2} \cdot \mathbf{1}\|_\infty + \|V_{h+1}^\pi - \hat{V}_{h+1}^\pi\|_\infty \\
&\leq \epsilon_R + \epsilon_T \frac{H-h-1}{2} + \|V_{h+1}^\pi - \hat{V}_{h+1}^\pi\|_\infty \\
\implies |V_{h,s}^\pi - \hat{V}_{h,s}^\pi| &\leq \sum_{i=h}^{H-1} \epsilon_R + \epsilon_T \frac{H-i-1}{2} \\
\implies |V_{0,s}^\pi - \hat{V}_{0,s}^\pi| &\leq \epsilon_R H + \epsilon_T \frac{H(H-1)}{4}
\end{aligned}
$$

For our bound, the only change from the discounted setting is replacing the discounted infinite sums of Section 4.1.3 with finite undiscounted ones. Redefining $P^t = \prod_{0 \leq i < t} P^{\pi_i}$, and WLOG assuming that $V_{0,s_0}^\pi \geq \hat{V}_{0,s_0}^\pi$ we can show:

$$
\begin{aligned}
|V^{\pi}_{0,s_0} - \hat{V}^{\pi}_{0,s_0}| &= V^{\pi}_{0,s_0} - \hat{V}^{\pi}_{0,s_0} \\
&= \sum_{t=0}^{H-1} \langle P^t_{s_0}, R^{\pi_t} \rangle - \langle \hat{P}^t_{s_0}, \hat{R}^{\pi_t} \rangle \\
&= \sum_{t=0}^{H-1} \langle P^t_{s_0} - \bar{M}^t_{s_0}, R^{\pi_t} \rangle + \langle \bar{M}^t_{s_0}, R^{\pi_t} - \hat{R}^{\pi_t} \rangle + \langle \bar{M}^t_{s_0} - \hat{P}^t_{s_0}, \hat{R}^{\pi_t} \rangle \\
&\leq \sum_{t=0}^{H-1} \langle P^t_{s_0} - \bar{M}^t_{s_0}, R^{\pi_t} \rangle + \langle \bar{M}^t_{s_0}, R^{\pi_t} - \hat{R}^{\pi_t} \rangle \\
&\leq \sum_{t=0}^{H-1} \|P^t_{s_0} - \bar{M}^t_{s_0}\|_1 \cdot \|R^{\pi_t}\|_\infty + \|\bar{M}^t_{s_0}\|_1 \cdot \|R^{\pi_t} - \hat{R}^{\pi_t}\|_\infty \\
&\leq \sum_{t=0}^{H-1} \|P^t_{s_0} - \bar{M}^t_{s_0}\|_1 + \|\bar{M}^t_{s_0}\|_1 \epsilon_R \\
&= \sum_{t=0}^{H-1} \|P^t_{s_0}\|_1 - \|\bar{M}^t_{s_0}\|_1 + \|\bar{M}^t_{s_0}\|_1 \epsilon_R \\
&\leq \sum_{t=0}^{H-1} 1 + (\epsilon_R - 1)(1 - \epsilon_T/2)^t \\
\implies |V^{\pi}_{0,s_0} - \hat{V}^{\pi}_{0,s_0}| &\leq H - (1 - \epsilon_R)\frac{2}{\epsilon_T}(1 - (1 - \epsilon_T/2)^H)
\end{aligned}
$$

Again, we note that Taylor expanding this relation at $\epsilon_R = 0$ and $\epsilon_T = 0$ recovers the original bound.

B.3 Proof of Hierarchy Bound

This proof exactly mirrors the one in the main body, with additional care taken to handle multi-time models. We first describe the ϕ-relative options framework (definitions largely taken from Abel et al. (2020)), and then provide a tighter bound on value loss.

An *option* $o \in \mathcal{O}$ is an abstract action defined by the tuple $(\mathcal{I}_o, \beta_o, \pi_o)$, where $\mathcal{I}_o \subseteq \S$ is

the subset of the state space the option can initiate in, $\beta_0 \subseteq \S$ is the subset the option terminates in, and π_o is a policy. For a given state abstraction $\phi : \S \to \S_\phi$, an option o_ϕ is said to be ϕ-relative if and only if $\exists s_\phi \in \S_\phi$ such that

$$s \in s_\phi \implies s \in \mathcal{I}_{o_\phi} \qquad s \notin s_\phi \implies s \in \beta_{o_\phi} \qquad \forall s \in s_\phi,\ \ \pi_{o_\phi}(s) \to \Delta(\mathcal{A}).$$

In words, a ϕ-relative option is one that executes from anywhere in one abstract state, and terminates upon leaving that abstract state. Furthermore, $\mathcal{O}_\phi$ denotes a set of only ϕ-relative options, with at least one option that executes at each abstract state.

Let $\mathcal{O}^*_\phi$ be a set of ϕ-relative options o^* that can be composed to form an optimal policy in the base MDP. Let $\hat{\mathcal{O}}_\phi$ be a set of options that approximates $\mathcal{O}^*_\phi$ in that

$$\begin{aligned} &\forall o^* \in \mathcal{O}^*_\phi \ \ \exists \hat{o} \in \hat{\mathcal{O}}_\phi : \\ &\forall s, s' \ \ |P^{o^*}_{s,s'} - P^{\hat{o}}_{s,s'}| \leq \epsilon_T \ \text{ and } \ |R^{o^*}_s - R^{\hat{o}}_s| \leq \epsilon_R \end{aligned} \tag{B.1}$$

where R^o_s and $P^o_{s,s'}$ represent the multi-time reward and transition functions described in Sutton et al. (1999):

$$R^o_s = \mathbb{E}_{a\sim o}[\sum_{t=0}^{\infty} \gamma^t r_t] \qquad P^o_{s,s'} = \sum_{t=1}^{\infty} \gamma^t \Pr(s_t = s', t_\beta = t).$$

In words, R^o_s is the expected discounted reward accumulated over the course of an option execution, and $P^o_{s,s'}$ is the total discounted probability that an option terminates in s' when starting from s. Crucially, $\sum_{s'\in\S} P^o_{s,s'} \leq \gamma < 1$. We also note that the ϵ_T bound is per-entry, not per-vector. This was the form of the conditions in the original simulation lemma (Kearns and Singh, 2002), which was replaced with a vectorized version in subsequent work (Kakade et al., 2003).

Since $\|P^o_s\|_1$ may take on different values for different options and starting states, we can no longer directly use a relation similar to Equation 4.8. However, we can augment

the MDP by adding an absorbing state s_x, and modify each option such that

$$R^o_{s_x} = 0 \quad , \quad P^o_{s,s_x} = \gamma - \sum_{s' \neq s_x} P^o_{s,s'}.$$

By doing this, $\|P^o_s\|_1 = \gamma$ without modifying the behavior of the given option in the base MDP. This allows our proof to proceed treating options in roughly the same way as we do actions in the main body. Noting that since $P^o_{s,s} \equiv 0$ by construction, for two options $o^*, \hat{o}^*$ satisfying the relations of Equation B.1 we have that:

$$|P^{o^*}_{s,s_x} - P^{\hat{o}}_{s,s_x}| \leq \sum_{s' \neq s_x, s} |P^{o^*}_{s,s'} - P^{\hat{o}}_{s,s'}| \leq (|S| - 1)\epsilon_T.$$

Thus we can recover a condition similar to that of Equation 4.2:

$$\|P^{o^*}_s - P^{\hat{o}}_s\|_1 = \sum_{s' \in \S + s_x} |P^{o^*}_{s,s'} - P^{\hat{o}}_{s,s'}| \leq 2(|S| - 1)\epsilon_T.$$

Due to the addition of s_x, we can now describe the above bound in terms of overlap. Defining $\bar{P}^{o^*,\hat{o}}_{s,s'} = \min(P^{o^*}_{s,s'}, P^{\hat{o}}_{s,s'})$, we can produce a similar relation to Equation 4.9:

$$\|\bar{P}^{o^*,\hat{o}}_s\|_1 \geq \gamma - (|S| - 1)\epsilon_T.$$

Let $\Pi_{\mathcal{O}_\phi}$ be the set of abstract policies representable by $\mathcal{O}_\phi$. Let π_{o^*} be a policy within $\Pi_{\mathcal{O}^*_\phi}$ that is optimal in the base MDP. Let $\pi_{\hat{o}}$ be a policy in $\Pi_{\hat{\mathcal{O}}_\phi}$ produced by replacing each o^* chosen by π_{o^*} with an option $\hat{o}$ satisfying the relations of Equation B.1. Then, we can follow the same algebraic steps as in the main body to produce the following bound:

$$|V^{\pi_{o^*}}_s - V^{\pi_{\hat{o}}}_s| \leq \frac{R_{MAX}}{1-\gamma} - \frac{R_{MAX} - \epsilon_R}{1 - \gamma + (|S| - 1)\epsilon_T}.$$

B.3.1 Proof of Tightness

We can generate an abstract MDP that achieves this bound using a similar recipe as in Section 4.1.4. We construct an abstract MDP where each option o^* transitions uniformly to each other state with discounted probability $\frac{\gamma}{|S|-1}$, receiving a reward of R_{MAX}. We then construct a new set of options that uniformly transition with discounted probability $\frac{\gamma}{|S|-1} - \epsilon_T$, receiving reward $R_{MAX} - \epsilon_R$. This exactly reproduces the provided bound.

Appendix C

Optimistic Initialization for Exploration in Continuous Control

C.1 Proofs

Proof of Equation 11: Since $\beta(d)$ decays monotonically for $d \geq 0$ and $\hat{\kappa}(x) = \beta(\hat{d}_{\min}(x))$, to prove $\hat{\kappa}(x) \geq \kappa(x)$ we can show that $\hat{d}_{\min}(x) \leq d_{\min}(x)$.

Let $x^* \in X$ be the closest point to x, meaning $d_{\min}(x) \equiv |x - x^*|$. Let $\hat{x}^* \in \hat{X}$ be (one of) the points in the covering set for which $|\hat{x}^* - x^*| \leq \epsilon$. Then,

$$
\begin{aligned}
|x - \hat{x}^*| &\leq |x - x^*| + |x^* - \hat{x}^*| \\
|x - \hat{x}^*| &\leq |x - \hat{x}^*| + \epsilon \\
|x - \hat{x}^*| - \epsilon &\leq |x - x^*|.
\end{aligned}
$$

Now, we can relate $\hat{d}_{\min}$ with $d_{\min}$. If

$$
\min_{x' \in \hat{X}} d(x, x') - \epsilon \leq 0,
$$

then $\hat{d}_{\min} = 0 \leq d_{\min}$. Otherwise,

$$\begin{aligned}
\hat{d}_{\min}(x) &= \min_{x' \in \hat{X}} d(x, x') - \epsilon \\
&\leq |x - \hat{x}^*| - \epsilon \\
&\leq |x - x^*| \\
&= d_{\min}(x).
\end{aligned}$$

C.2 Adaptive Filtering

When the approximate size of the state space is known, one can normalize the state space as in equation 8 and choose an appropriate length-scale d_0 through domain knowledge or hyperparameter sweep. However, in many problems the relative sizes of each dimension may vary significantly and are not known in advance. For example, in the Acrobot domain the velocity features take on roughly 50x the range of values compared with the position features. Without prior knowledge, we would like to explore along each dimension equally, however in such situations a single feature dominates the knownness calculation. To overcome this imbalance, we use a simple adaptive rescaling procedure: after each episode we update the distance metric with a new diagonal normalizing matrix A based on the current minimum and maximum values observed for each feature,

$$A_{ii} = \frac{1}{\max_{x \in X} x_i - \min_{x \in X} x_i + \delta},$$

where δ is a small constant (we use 10^{-6}) to prevent divide-by-zero errors when a feature has a constant value. This transforms the interaction data so that each feature has a range close to 1 during parameter updates. Since the denominator is monotonically increasing, this ensures that updates to A only ever shrink the distances between points; thus any summarizing set S which covers interaction history X under A^t will still cover X under

the updated A^{t+1}. Therefore we can use Algorithm 1 with either fixed or adaptive scaling.

C.3 Policy Gradient Methods

We focus on value-function-only methods in this chapter because for this class of methods, optimism in the Q-function is immediately represented in the policy. In contrast, for more standard policy-gradient methods, optimism must first be distilled to the policy network before it affects exploration. Therefore, value-function-only architectures such as RBFDQN (Asadi et al., 2021) are a cleaner testbed for bonus-based exploration methods. Despite this extra machinery, we find that we achieve qualitiatively similar results when each of our exploration methods (ours and baselines) are applide to TD3, a state-of-the-art policy gradient architecture. We report these results in Figure C.1.

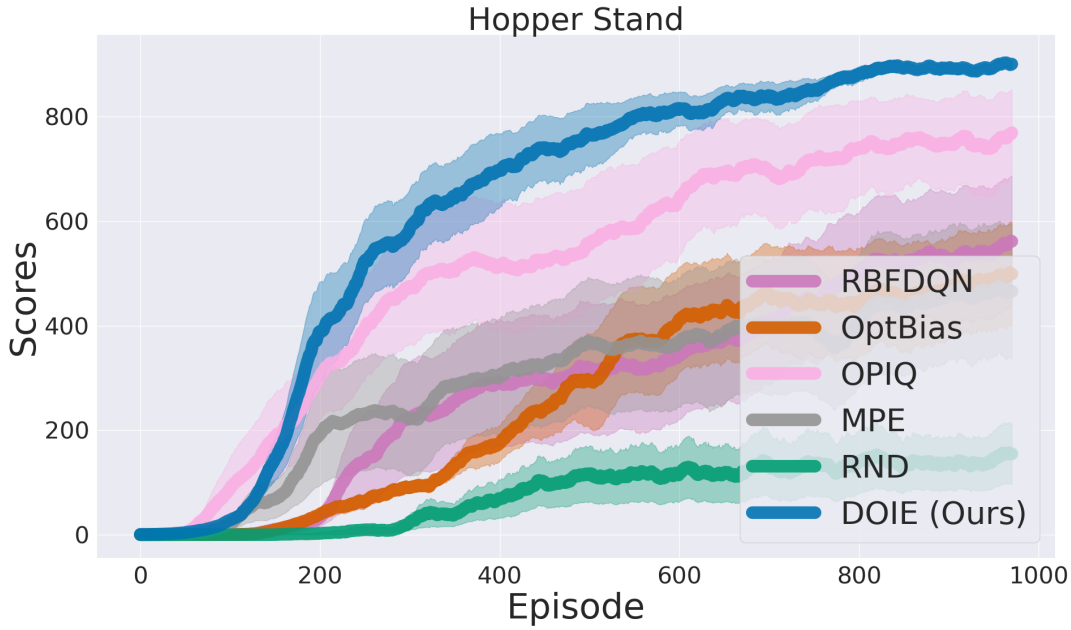

Figure C.1: Learning curves for all exploration methods applied to a TD3 base-architecture, on the Hopper Stand task.

C.4 Architectures and Hyperparameters

Here we describe the architectures we use for all experiments. All experiments were run on a cluster, with each job having access to a 12GB GPU, 32 GB RAM, and 2 CPUs. Complete implementations and instructions for running experiments can be found in our code submission. All hyperparameters not listed are shared between all experiments, and can be found in the hyperparameters directory.

C.4.1 OMEGA

We use publicly available code with reported hyperparameters, to re-run experiments with OMEGA on the PointMaze domain Pitis et al. (2020).

C.4.2 RBFDQN

For the results in Chapter 5, we apply all exploration methods besides OMEGA using an RBFDQN base agent Asadi et al. (2021). We also include comparisons with RBFDQN using a decaying ϵ-greedy exploration strategy for all experiments, as a non-directed exploration baseline.

C.4.3 TD3

To demonstrate our method's applicability to policy-gradient methods we run an additional set of experiments using Twin-Delayed Deep Deterministic Policy Gradient (TD3) as a base agent (Fujimoto et al., 2018). TD3 trains a policy network to maximize a Q-function, and reduces overestimation bias by using the minimum value of two Q-functions when bootstrapping. Our implementation of TD3 is included in the Code Appendix. For bonus-based methods we optimistically modify both Q-function using the same optimism module. As TD3 uses a policy network for action selection, applying a bonus during interaction for action-selection has no effect. Hyperparameters for TD3 on the "Hopper Stand" domain are included in the tables below.

C.4.4 DOIE

Our method, Deep Optimistic Initialization for Exploration (DOIE), modifies the effective Q-function for bootstrapping and action selection, as detailed in equation 3 and equation 4. Unless otherwise specified, we choose $Q_{MAX} = r_{\text{goal}}$ for goal-directed tasks, and $Q_{MAX} = \frac{r_{\max}}{1-\gamma}$ otherwise. We use RBFDQN as our base agent. For tasks with known sizes for the state and action spaces, we normalize the spaces to have range 1. Otherwise, we use the adaptive scaling procedure described above. We further scale the action space by some constant α, which is chosen through hyper-parameter search. The length-scale d_0 is also chosen through hyperparameter search. We choose the filtering-radius ϵ on a per-task basis such that learning is sufficiently fast. All hyperparameters are listed below. When multiple values are in a cell, it represents the values swept, with the reported value in bold.

	PointMaze	Pendulum	Acrobot	Ball In Cup	Hopper	MountainCar	TD3 Hopper
Updates per episode	100	1,000	1,000	1,000	1,000	1,000	1,000
Action scaling	**0.01**, 0.03, 0.1, 0.3, 1.0	0.5, 1.0, 2.0, 3.0, **4.0**	0.5, 1.0, **2.0**, 3.0, 4.0	0.5, 1.0, **2.0**, 3.0, 4.0	0.5, 1.0, **2.0**, 3.0, 4.0	0.5, **1.0**, 2.0	0.5, 1.0, **2.0**
d_0	**0.1**, 0.3, 1.0, 2.0, 3.0	0.1, 0.3, 1.0, **2.0**, 3.0	0.1, **0.3**, 1.0, 2.0 3.0	0.1, 0.3, 1.0, **2.0** 3.0	0.1, 0.3, 1.0, 2.0 **3.0**	0.1, **0.3**, 1.0	0.1, 0.3, **1.0**, 3.0
Adaptive filtering	No	Yes	Yes	Yes	Yes	No	Yes
Filtering radius	0.001	0.1	0.25	0.75	0.25	0.0	0.5
Training time per run	2h 45m	3h 45m	7h 15m	3h 30m	6h 15m	1h 15m	5h 15

Table C.1: Hyperparameters for Deep Optimistic Initialization for Exploration (DOIE)

C.4.5 OPIQ

We implement a count-based bonus using the update rule introduced in OPIQ (Rashid et al., 2020):

$$
\begin{aligned}
\theta \;&\leftarrow\; \theta + \alpha\delta\Delta_\theta \hat{Q}(s,a;\theta) \\
\text{where} \quad \delta &= r + \frac{\beta}{\sqrt{N(s,a)}} + \gamma \max_{a'} \hat{Q^+}(s',a';\theta) - Q(s,a;\theta) \\
\text{and} \quad Q^+(s,a) &= Q(s,a) + \frac{C}{(N(s,a)+1)^M}
\end{aligned}
$$

As in DOIE, Q^+ is used during action-selection as well as bootstrapping. The term modulated by β is the standard count-based bonus, and the modification to Q^+ ensures optimistic bootstrapping even of state-action pairs which appear zero times in the agent's experience. As is common for counting-based bonuses, we calculate our counts by discretizing the state-action space and tracking the number of visited (s, a) within each bin (Rashid et al., 2020; Ecoffet et al., 2021). For fair comparison, we use the same normalization procedure described in Appendix B, and discretize this normalized space into hypercubes of side-length d_0. We treat d_0, β, and C as hyperparameters, and use $M = 2$ as recommended by the original OPIQ paper. We additionally scale the action-space by some constant α as we do for DOIE. All hyperparameters are listed below. When multiple values are in a cell, it represents the values swept, with the reported value in bold.

C.4.6 RND

We implement Random Network Distillation Burda et al. (2019)(RND) using RBFDQN as our base agent. RND modifies the effective reward used for training:

$$r_t(s,a) = r(s,a) + \alpha\,\mathrm{RND}(s)$$

where α is a scaling hyperparameter. RND(s) is the calculated by determining the L_2 distance between two neural network outputs, one randomly initialized and frozen, and

	PointMaze	Pendulum	Acrobot	Ball In Cup	Hopper	TD3 Hopper
Updates per episode	100	1,000	1,000	1,000	1,000	1,000
Action scaling	**0.01**, 0.1, 1.0	**1.0**, 2.0 4.0	**1.0**, 2.0, 4.0	1.0, 2.0, 4.0	**0.5**, 1.0 2.0	**0.5**, 1.0 2.0
d_0	**0.1**, 0.3, 1.0, 2.0, 3.0	0.03, 0.1, 0.3, 1.0	0.03, 0.1, 0.3, 1.0	0.03, 0.1, 0.3, 1.0	0.1, 0.3 **0.5**, 1.0	0.1, 0.3 **0.5**, 1.0
β	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	0.0001 **0.001**, 0.01 0.1 1.0	0.0001 **0.001**, 0.01 0.1 1.0
C	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	0.0001 0.001, 0.01 0.1 1.0	**0.0001** 0.001, 0.01 0.1 1.0	**0.0001** 0.001, 0.01 0.1 1.0
Adaptive filtering	No	Yes	Yes	Yes	Yes	Yes
Training time per run	1h 30m	2h 0m	6h 30m	3h 30m	6h 45m	5h 15m

Table C.2: Hyperparameters for Optimistic Pessimistically Initialized Q-Learning (OPIQ)

the other trained to mimic the first's output. Following the original implementation, we normalize the outputs of this network so that $\mathbb{E}[\text{RND}(s)] = 0$ and $Std[\text{RND}(s)] = 1$ over the current replay buffer. We update the trained network with the same data and frequency as we update the RBFDQN base agent. We learn a single value head which estimates the intrinsic and extrinsic expected reward. All hyperparameters are listed below. When multiple values are in a cell, it represents the values swept, with the reported value in bold.

	PointMaze	Pendulum	Acrobot	Ball In Cup	Hopper	TD3 Hopper
Updates per episode	100	1,000	1,000	1,000	1,000	1000
RND scaling	**0.01**, 0.03, 0.1, 0.3, 1.0	0.01, 0.03, 0.1, 0.3, **1.0**	0.01, **0.03**, 0.1, 0.3, 1.0	0.01, 0.03, **0.1**, 0.3, 1.0	0.001, 0.003, **0.01**, 0.03, 0.1, 0.3, 1.0	0.001, **0.003**, 0.01, 0.03, 0.1, 0.3, 1.0
Training time per run	45m	2h 0m	6h 0m	1h 30m	4h 0m	3h 45m

Table C.3: Hyperparameters for Random Network Distillation (RND)

C.4.7 MPE

A variety of methods augment the training reward using the error of a learned transition model Stadie et al. (2015); Pathak et al. (2017):

$$r_t(s, a, s') = r(s, a) + \alpha \, \text{MPE}(s, a, s')$$

$$\text{MPE}(s, a, s') = |\phi(s') - T_\theta(s, a)|$$

where ϕ maps states to a compact latent space, T_θ is a function approximator trained to mimic the transition function, and α is a scaling hyperparameter. Since, unlike images, our inputs already are a compact feature space, we set ϕ to the identity transformation. We implement T_θ as outputting the residual between s and s', which we find models the environment more accurately since the state only changes slightly per timestep. Unlike RND, MPE does not generally have reward normalization. Thus, we manually inspect r_{MPE} and choose our hyperparameter sweep such that the effective intrinsic reward generally lies between 0.01 and 1.0. The exception is for the Hopper task, for which we could not generate strong results without the normalization procedure. All hyperparameters are listed below. When multiple values are in a cell, it represents the values swept, with the reported value in bold.

	PointMaze	Pendulum	Acrobot	Ball In Cup	Hopper	TD3 Hopper
Updates per episode	100	1,000	1,000	1,000	1,000	1,000
MPE scaling	**0.01**, 0.03, 0.1, 0.3, 1.0	300, 1,000, **3,000**, 10,000, 30,000	1.0, **3.0**, 10.0, 30.0, 100	0.3, 1.0 3.0, **10.0**, 30.0	**0.003**, 0.001 0.03, 0.01 0.3, 0.1	0.003, 0.001 0.03, 0.01 0.3, **0.1**
Training time per run	45m	2h 0m	6h 0m	1h 30m	4h 0m	3h 45m

Table C.4: Hyperparameters for Model-Predictive Error (MPE)

C.4.8 Value Shaping

We investigate the effect of value-shaping on learning speed in the MountainCar domain. Value shaping does not introduce any hyperparameters to the algorithm: besides changing Q_{MAX}, we use the architecture and hyperparameters tuned to perform will with DOIE. Using the goal-position s_g, the agent's max speed $v_{\max}$, and the reward for reaching the goal r_g, we can calculate an upper bound of the optimal Q-function Q^*:

$$Q_{MAX}(s,a) \;=\; r_g \gamma^{|s_g - s|/v_{\max}}.$$

We also include a reversal of this shaping, which we refer to as anti-shaping:

$$Q_{MAX}(s,a) \;=\; r_g \gamma^{-|s_g - s|/v_{\max}}.$$

This maintains the upper-bounding property, but directs exploration away from the true goal.

C.4.9 Reward Shaping

We additionally implement a dense reward for MountainCar, which we add to the sparse reward during training as a shaping signal. The dense reward is derived from a state-based potential function:

$$r_{\text{dense}}(s,a,s') = \gamma V(s') - V(s)$$

with

$$V(s) = |s - s_g|$$

being the difference in the position dimension between one state and the next. We modify the training reward by adding the dense reward with the environment's reward, modulated

by a hyperparameter α:

$$r_t = r + \alpha r_{\text{dense}}$$

We choose $\alpha = 10.$ as the best-performing value from the range $(0.1, 0.3, 1.0, 3.0, 10.0, 30.0, 100.0)$. As r_{dense} is a potential-based shaped reward, it does not modify the fixed point of Bellman updates Ng et al. (1999).

Appendix D

From Additive Bonuses to V_{MAX} Interpolation

D.1 Distributional MBIE

At each step of value iteration, MBIE performs the following backup:

$$\begin{aligned}\hat{Q}_t(s,a) &\leftarrow \tilde{R}(s,a) + \gamma \sum_{s'} \hat{T}(s,a,s') V_{t-1}(s') \\ &= \sum_{s'} \hat{T}(s,a,s') \Big(\tilde{R}(s,a) + \gamma \hat{V}_{t-1}(s') \Big), \end{aligned} \tag{D.1}$$

where $\hat{V}(s) = \max_a \hat{Q}(s,a)$, and $\hat{T}$ of Equation 6.1 moves $\epsilon_T/2$ probability mass from the lowest-value successor states to V_{MAX}. Since this can be written as the expectation with respect to the distribution of $r + \gamma V$ target values, this backup can be simplified through the notion of a distributional Q function, which instead of representing a scalar value represents a probability distribution over values:

$$Q_D(s,a,q) \;=\; \Pr\Big[\,\tilde{R}(s,a) + \gamma V(s') = q \;\Big|\; s' \sim \tilde{T}(s,a)\,\Big]. \tag{D.2}$$

The standard Q value is then the expectation of this distribution:

$$Q(s,a) = \int_0^{V_{MAX}} q \cdot Q_D(s,a,q)dq.$$

The benefit of this representation is that it can be be made optimistic without modifying the underlying transition, by moving $\epsilon_T/2$ of the lowest-value probability mass to $Q_D(s,a,V_{MAX})$. Let V_ϵ be the value for which there is $\epsilon_T(s,a)$ probability mass below: $\int_0^{V_\epsilon} Q_D(s,a,q)dq = \epsilon_T(s,a)$. Then, we can easily construct our *optimistic* Q value:

$$\hat{Q}(s,a) = \tilde{R}(s,a) + \gamma\Big(\frac{\epsilon_T(s,a)}{2}V_{MAX} + \int_{V_\epsilon}^{V_{MAX}} q \cdot Q_D(s,a,q)dq\Big), \tag{D.3}$$

and likewise $\hat{V}(s)$. This cleanly represents the process of moving probability mass to a high-value successor state, and can be trained purely using the empirical transition function. Therefore, with a deep distributional Q function (Bellemare et al., 2017) trained to represent Equation D.2, we can implement MBIE's optimism in the deep learning context. As described in Section 6.1.3, this bound naturally controls the degree to which error compounds, and especially early in training produces much tighter optimistic values than MBIE.