An Empirical Study of Model-Free Exploration for Deep Reinforcement Learning

by

Xutong Zhao

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Reinforcement learning (RL) is a learning paradigm focusing on how agents interact with an environment to maximize cumulative reward signals emitted from the environment. Exploration versus exploitation challenge is critical in RL research: the agent ought to trade off between taking the known rewarding sequence of actions and exploring unknown actions that might be more rewarding. Exploration research in RL often uses algorithms and environments with many degrees of freedom, which can interfere with the interpretability of results. This thesis presents a systematic, yet simple, study of exploration methods for value-based control algorithms. We present a novel suite of small environments that each pose a distinct exploration challenge. Our environment designs allow us to observe the strengths and weaknesses of individual exploration methods, as well as trends across implementation details and conceptual approaches to exploration. We conduct a literature survey and categorize model-free exploration approaches by their underlying heuristics. We also empirically evaluate the performance of representative exploration methods on our exploration domains. Despite the simplicity of our environments, none of the tested exploration methods achieves good performance in all environments. However, some methods consistently improved upon the Q-learning baseline. Beyond our survey results, our suite of interpretable environments can be used as a sanity check to ensure that an exploration method behaves appropriately in simple situations.

Preface

Parts of this thesis are to be submitted as a journal paper. This is a joint work with Niko Yasui, Martha White, Raksha Kumaraswamy, James Wright, and Adam White.

To my family and friends.

Acknowledgements

I would like to thank my supervisor Adam White for his invaluable support throughout my entire Master's program. He cultivated my research capability and provided innumerous feedback and encouragement.

I would also like to thank Professor Martha White for her substantial support on all projects I worked on. Her guidance helped me make continuous progress on my work.

I am thankful to other collaborators on this project. Niko Yasui laid a solid foundation on this long-term project and he was always able to help. Raksha Kumaraswamy provided insightful ideas and discussions with me that helped shape this project.

I am also grateful to the members of RLAI lab for creating an excellent research atmosphere and providing constant support.

I would finally like to thank the Alberta Machine Intelligence Institute, NSERC, and the Canada CIFAR AI Chairs Program for the funding for this research, as well as Compute Canada for the computing resources used for this work.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a subarea of machine learning concerned with how an intelligent agent learns from interactions with an environment to make optimal decisions (R. S. Sutton et al. 2018). At each discrete time point, the agent takes an action and sends it to the environment, the environment correspondingly updates its underlying state and sends an observation and a numerical reward signal back to the agent. The agent's learning objective is to maximize cumulative rewards emitted from the environment. Consider a maze game as an exemplar RL task. The environment is the game, including all possible situations, the set of available actions, binary rewards indicating whether the goal is reached, and underlying situation-switching mechanisms. The agent is the player of this game. At each time point, the agent interacts with the game by taking a step towards one direction and observing its new location as well as a scalar indicating whether the game is solved.

The exploration-exploitation dilemma exists extensively in real-world decision-making scenarios. The decision-maker needs to balance between taking the best actions based on its current knowledge of the environment (i.e., exploitation), and exploring unfamiliar actions to improve its understanding that could potentially lead to more rewarding long-term strategies (i.e., exploration). Efficient exploration is crucial to solving certain RL problems. In hard exploration tasks, inefficient methods may find the near-optimal policy with learning times exponentially large with respect to the size of the state space (Osband, Van Roy, et al. 2019). In these tasks, the agent must

efficiently explore to avoid local optima.

RL has achieved successes in diverse domains, thanks to the applicability of its learning paradigm and function approximations. Utilizing deep neural networks, modern Deep RL methods have demonstrated their capability in a wide variety of tasks, ranging from game playing such as Atari games (Mnih, Kavukcuoglu, et al. 2015), Go (Silver et al. 2017), and Starcraft (Vinyals et al. 2019), to real-world applications such as robotics (Kober et al. 2013), and recommendation systems (Afsar et al. 2021). Many state-of-the-art approaches such as DQN (Mnih, Kavukcuoglu, et al. 2015), A3C (Mnih, Badia, et al. 2016), TRPO (Schulman et al. 2017), and IMPALA (Espeholt et al. 2018) have achieved super-human performance in many Atari games and continuous control benchmarks. The scale of RL grows every day. It has become commonplace to evaluate deep RL agents with millions of parameters, on video games for hundreds of millions of frames, in complex, high dimensional simulations, and in robotics.

However, there is still a gap between those successful benchmarking results and research on exploration. Given imperfect function approximation, the state-spaces and exploration challenges posed by these problems are vast. Interestingly, many state-of-the-art agents still rely on fairly primitive exploration methods such as ϵ-greedy and entropy regularization, despite significant efforts to specifically design more efficient mechanisms for neural network learners (Bellemare et al. 2016; Fortunato et al. 2019; Osband, Aslanides, et al. 2018; Osband, Van Roy, et al. 2019; Pathak, Agrawal, et al. 2017). Indeed, large-scale learning systems rely on asynchronous training, replay, planning, and vast amounts of training data to brute-force the exploration problem. Given this circumstance, research questions naturally arise: why do state-of-the-art systems still rely on undirected exploration approaches like ϵ-greedy? Are gains from directed exploration methods dominated by other design choices? Do recently proposed deep exploration approaches only make minor improvements over more basic strategies?

To gain traction on these complex, multidimensional issues we turn to em-

pirical study on exploration for value-based methods. Much has been written about the efficiency of exploration in bandits, model-free RL with tabular representation, and model-based RL. Unfortunately, we are still at the beginning of our theoretical understanding of deep neural networks, let alone characterizing exploration with non-linear function approximation. The majority of prior work takes the form of demonstrating new state-of-the-art performance on large-scale benchmarks, or empirical comparisons of existing systems with and without new exploration mechanisms. To gain additional insight, extensive experiments are needed.

Yasui (2020) conducts a comprehensive empirical study on exploration methods for linear agents in the pure exploration setting. In this work, they provide a categorization of environmental properties that reflect diverse exploration challenges, as well as a suite of small diagnostic tasks that implement these challenges. They also present a categorization of common value-based exploration strategies. It is worthwhile to complement this previous work and learn insights on neural network learners in a more complicated setting. In this thesis, we conduct a study on neural network agents in a more practical exploration-exploitation setting. We adopt most of these categorizations, and re-calibrated the environments to obtain more distinct performance across different approaches.

Taiga et al. (2019) systematically benchmark bonus-based exploration methods on hard exploration games with sparse rewards (Bellemare et al. 2016) as well as the whole Atari 2600 suite. They empirically show that bonus-based methods that perform best on the most difficult MONTEZUMA'S REVENGE game often underperform ϵ-greedy on easy exploration games. Their results also suggest that improvement made by bonus-based exploration methods may instead be attributed to other confounding factors such as the model architecture. These findings are interesting, as the lack of efficiency gain could possibly also occur to other common exploration methods. Furthermore, this work investigates bonus-based methods on hard Atari games mainly embodying two exploration difficulties: complex state-action space and sparse

rewards. It is interesting to examine other categories of exploration approaches with more exploration challenges. This thesis thus provides an empirical study complementary to this prior work on bonus-based methods.

It would be tempting to execute such a study in a popular benchmark, such as Atari. However, fair and reliable conclusions require (1) testing and characterizing many hyperparameter combinations, (2) for each algorithm, (3) across a variety of domains, (4) with different function approximation architecture, (5) measuring online and offline performance, and repeating the whole process many times. In this thesis we consider 13 exploration methods, 6 problems, 2 function approximation approaches, and 11648 hyperparameter settings, resulting in 69888 unique experiments. [1]

Our empirical study is designed to tease apart the key characteristics of exploration methods, while also attempting to isolate and test the characteristics of exploration problems that make them difficult. We categorize recent exploration methods by the heuristic that each method employs, and test representative methods of each category. This categorization not only simplifies experiments, but identifies the key approaches promoted in the literature. We describe six properties of environments that can make them easier or harder for agents to explore, and provide a suite of minimal hard exploration tasks—several of which are new to this work. The main idea is that each task should be difficult due to one property, as opposed to common large-scale benchmarks that exhibit numerous interacting challenges or exploration tasks. For instance, DeepSea (Osband, Van Roy, et al. 2019) combines misleading rewards and antagonistic dynamics. Our suite of tasks is designed to be as small as possible to permit extensive experimentation and ensure the domains can be easily analyzed and understood. In addition, our suite emits a score-card-like analysis indicating the degree to which each method can handle different aspects of hard exploration.

Our study results in several novel insights. The main conclusion is, per-

[1] The number of hyperparameter settings and experiments does not consider extended hyperparameter sweeps or deep-dive case studies. More details are covered in Chapter 5.

haps unsurprisingly, no single exploration method performs well in every environment—our tasks are hard and diverse. The form of the representation has a large impact on performance. Almost all methods perform significantly worse with dense network architecture, compared with sparse, high-dimensional tile coding features; however, methods with neural networks perform more robustly under hyperparameter shifts. Moreover, recent deep RL exploration methods, such as noisy networks (Fortunato et al. 2019), and randomized value functions (Osband, Aslanides, et al. 2018), and random network distillation (Burda et al. 2018) perform particularly poorly with tile coding.

The main contributions of this thesis are as follows:

1. We provide high-quality implementations of exploration methods for deep RL. We avoid complex design choices and instead focus on the core ideas behind the exploration strategies. Our software framework will assist researchers to benchmark different methods on diverse exploration challenges.

2. We conduct a systematic empirical study of state-of-the-art model-free exploration methods on the aforementioned suite of diagnostic tasks. Moreover, we dive deeply into some particular aspects of those methods, such as the impact of function approximation, and unexpected poor performance of deep exploration methods in some domains, etc.

3. We obtain empirical insights and learnings through the large comparison of exploration methods. We describe environmental properties that pose the most difficult exploration challenge. We also present several design choices that have an important impact on the agents' performance.

This thesis consists of seven chapters. In Chapter 2 we introduce background concepts and notations. In Chapter 3 we identify the key properties that control the difficulty of exploration challenges, and present our prototypical suite of exploration tasks. In Chapter 4 we propose the categorization

of model-free exploration methods and discuss their underlying heuristics. Chapter 5 and Chapter 6 cover the experimental setup and empirical evaluation of exploration approaches in our domains, respectively. We conclude this thesis with Chapter 7. Furthermore, Appendix A and Appendix B provide implementation details of our environments and selected algorithms, respectively. Appendix C discusses the hyperparameter settings for each representative method.

Chapter 2

Background

This chapter describes the RL problem setting and a model-free value-based baseline approach. We present mathematical definitions of the environment and other important concepts such as the value function. We cover the value-based algorithm with two distinct function approximations, namely tile coding and neural networks, as well as their key characteristics.

2.1 RL Background

In the standard RL framework, the environment is modelled as a Markov Decision Process (MDP), defined as $M = (\mathcal{S}, \mathcal{A}, r, P, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions available to the agent, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition dynamics, and $\gamma \in [0, 1]$ is the discount factor that reduces value of future rewards (R. S. Sutton et al. 2018). If the state space and the action space are finite, then the environment is called a finite MDP. State transitions and rewards are determined by the transition dynamics and the reward function, respectively. The agent takes actions according to its policy, a mapping from states to the distribution of actions $\pi : \mathcal{S} \to \mathcal{A}$.

In this thesis, we consider sequential decision making tasks where an agent interacts with the environment on discrete time steps, $t = 0, 1, 2, \ldots$. In addition, we restrict our focus to episodic tasks, where the agent-environment interaction are split into episodes. At the beginning of each episode, the envi-

ronment samples the initial state S_0 according to an initial state distribution $\rho : \mathcal{S} \to [0, 1]$. On timestep t, the agent observes the state $S_t \in \mathcal{S}$ and decides on an action based on its policy $A_t \sim \pi(\cdot|S_t)$. The environment responds by transitioning to a next state $S_{t+1} \sim P(\cdot|S_t, A_t)$ and emitting a scalar reward $R_{t+1} = r(S_t, A_t)$. The episode ends once the agent reaches a terminal state.

The *return* is defined as the discounted cumulative reward

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$$

where R_T is the reward at the terminal state. In tasks with infinite horizon T is infinitely large. One reason discounting is used in the definition is to upper bound the return in case of infinite horizon. Suppose $R^{\max} = \max r(\cdot, \cdot)$ is the maximum achievable reward in the MDP. Then $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k \leq \sum_{k=t+1}^{\infty} \gamma^{k-t-1} |R^{\max}| \to \frac{|R^{\max}|}{1-\gamma}$.

The *value* function of a state $s \in \mathcal{S}$ under a policy π, denoted as $v_\pi(s)$, is the expected return starting at s and following π thereafter

$$v_\pi(s) := \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s], \text{ for all } s \in \mathcal{S}.$$

Similarly, the value function of a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ under a policy π, denoted as $q_\pi(s, a)$, is the expected return starting at s, taking action a, and following π thereafter,

$$\begin{aligned} q_\pi(s, a) :=& \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ =& \mathbb{E}_\pi[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s, A_t = a], \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned}$$

We call v_π and q_π the state-value function and the action-value function, respectively. For any $s \in \mathcal{S}, a \in \mathcal{A}$, the optimal action-value function is defined as the maximum action-value at (s, a) across all policies $q_*(s, a) := \max_\pi q_\pi(s, a)$. An optimal policy is a policy that achieves the optimal action value, i.e., $\pi_* := \arg\max_\pi q_\pi(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}$.

2.2 Exploration Background

In this section, we introduce two settings for exploration method evaluation. Suppose the learning budget is T_{learn} timesteps. In the pure exploration setting, the learning phase (or training phase) and the evaluation phase (or testing phase) are explicitly separated. The agent interacts with the environment and learns online for T_{learn} timesteps, then it outputs a fixed policy $\hat{\pi}$, with its exploration and learning components disabled. The agent's performance is evaluated by the expected total rewards $\hat{\pi}$ accumulates over T_{eval} offline evaluation timesteps. An equivalent interpretation of this evaluation metric is how large T_{learn} should be such that the cumulative reward over T_{eval} is above some threshold. Here T_{learn} is the sample complexity. The pure exploration setting is also called the offline setting due to the offline phase evaluation. The agent's objective is to obtain a policy $\hat{\pi}$ that maximizes the expected cumulative offline rewards.

Since we are only concerned with how good the output policy is after training, the pure exploration setting does not embody the trade-off between exploiting the agent's current knowledge and exploring the environment. In addition, this learning/evaluation separation may not be applicable to some scenarios. Hence we are also interested in the other setting, the exploration-exploitation setting, where the exploration-exploitation dilemma does exist. The agent needs to learn about the environment to improve its policy while in the meantime accumulating as much rewards as possible along the way.

In the exploration-exploitation setting, the agent interacts with the environment and learns from transitions using the exactly same learning phase in the pure exploration setting. The agent's performance is usually evaluated by the expected total rewards accumulated over the T_{learn} online timesteps. Without offline evaluation involved, the exploration-exploitation setting is also called the online setting. The agent's objective is to maximize the expected cumulative online rewards.

In this thesis, we restrict our attention to the exploration-exploitation set-

ting. Moreover, with more focus paid on the final performance, we evaluate the agent's performance by the expected total rewards accumulated over the final 10% of the learning budget. This evaluation metric serves as a compromise between the cumulative reward and the quality of the final policy learned. Hence, in case two agents achieve similar cumulative rewards over the entire learning budget, the one with a higher final cumulative reward is considered outperforming the other.

In both exploration settings for exploration method evaluation, the performance is based on the cumulative reward, i.e., the result of learning, regardless of online or offline. Data coverage — that is, how well the agent explores the whole state-action space — is not directly reflected in those metrics. Our evaluation metrics are no different from those that are used for tasks without hard exploration challenges. This is because learning and exploration depend on each other. In fact the purpose of efficient exploration is to more accurately learn the action-values, so that the value-based policy becomes better and collects more rewards.

2.3 Baseline Algorithm

In many tasks there are too many states to store and estimate their values individually. In these tasks we use function approximation to approximate the values. In the non-linear setting, a neural network function approximator directly maps the state-action pairs to the value estimates $\hat{q}^{\text{NN}}_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathbf{w}$ is the parameterization. In our study we use neural networks with fully connected layers, also named multilayer perceptron (MLP), where the parameters $\mathbf{w}$ are the weights and biases of each layer.

In contrast to neural networks, under linear setting the state-action pairs are mapped to feature vectors $\phi_t := \phi(S_t, A_t)$, where $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$. This ϕ function produces vectors as a representation of the state-action pair and can be computed in numerous ways, including tile-coding (R. S. Sutton et al. 2018), where ϕ_t would be binary and sparse. The agent's value estimate is

linear in ϕ_t: $\hat{q}^{\text{TC}}_{\mathbf{w}_t}(S_t, A_t) = \phi_t^\intercal \mathbf{w}_t$ and ϕ is a fixed mapping. Under both settings the agent adjusts the parameters $\mathbf{w}_t$, to better estimate the values: $\hat{q}_\pi \approx Q_\pi$. Note that in the linear setting the number of modifiable parameters is typically much smaller than the size of the state-action space: $d \ll |\mathcal{S} \times \mathcal{A}|$.

The most popular methods for estimating $\hat{q}$ and finding near-optimal policies are based on Temporal Difference (TD) learning, an incremental learning mechanism. The essential idea behind TD methods is that they alternate on each step between refining their estimate of $\hat{q}_\pi$ based on the recent reward, and updating the agent's policy to favour actions that maximize $\hat{q}_\pi(S_t, \cdot)$. Here the exploration-exploitation trade-off can be re-interpreted in terms of $\hat{q}_\pi$ of TD methods. If the agent is purely greedy with respect to $\hat{q}_\pi$ it might choose the wrong action; particularly if $\hat{q}_\pi$ is inaccurate in some states. On the other hand, the agent must sometimes intentionally select actions with lower value to check if they are better than the action currently favored by π, thus ensuring convergence to accurate estimates.

We restrict our focus to value-based methods, methods that learn the action-value estimate $\hat{q}$ to directly control the agent's behaviour policy. In particular, we focus on Q-learning (Watkins 1989) with greedy behaviour policy. We also discuss another family of algorithms—policy gradient algorithms, whose policy is directly parameterized by a vector θ.

Incremental value-based methods update their action value estimates from the most recent transition tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. The update rule for Q-learning is

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

where $\alpha \in [0, 1]$ is the learning rate. Note that Q-learning does not learn an action value of the policy that generates the transition, namely the behaviour policy; instead it learns the action-value estimate q_* for the optimal policy π_*, which we call the target policy. Algorithms that learn a value function for a target policy π with data generated from a different behaviour policy b are called off-policy learning algorithms.

In addition to incremental updates, Q-learning also supports updates with mini-batches of transition tuples that are not temporally correlated. DQN (Mnih, Kavukcuoglu, et al. 2015) demonstrates mini-batch updates with an experience replay buffer: at each timestep the transition data is fed to the replay, and a mini-batch is randomly sampled from the replay for the batch update. Let $\{(s_i, a_i, r_i, s'_i, a'_i)\}$ denote the batch sampled from replay, the loss function of Q-learning on this batch at timestep t is

$$L_t(\mathbf{w}) = \sum_i \|y_i - \hat{q}_{\mathbf{w}}(s_i, a_i)\|^2, \tag{2.1}$$

where $y_i = r_i + \gamma \max_a \hat{q}_{\mathbf{w}'}(s'_i, a)$ is the target value for transition i. Here in the target value $\hat{q}$ is not parameterized by $\mathbf{w}$; instead it is parameterized by the fixed target parameter $\mathbf{w}'$ due to the usage of the target networks, which will be explained in the following paragraph. The gradient of the loss is obtained by differentiation with respect to the weight parameter $\mathbf{w}$:

$$\nabla_{\mathbf{w}} L_t(\mathbf{w}) = \sum_i \left[(\hat{q}_{\mathbf{w}}(s_i, a_i) - y_i) \nabla_{\mathbf{w}} \hat{q}_{\mathbf{w}}(s_i, a_i) \right]. \tag{2.2}$$

Note that the gradient does not flow through the target value y_i

Another key component of DQN is the target networks. If the target value y_i is also parameterized by the same weights $\mathbf{w}$, when we perform one gradient descent step according to Equation 2.2, the target value is also updated correspondingly. To mitigate the issue of moving targets and stabilize updates, the target network is introduced to replace the policy network in the target value. The target network parameters are synced with the policy network every N timesteps. In our experiments, incremental updates and mini-batch updates are conducted separately, where one hyperparameter setting determines the update type.

Our Q-learning baseline takes actions greedily with respect to the action value estimate at each state. In Chapter 4, all exploration methods are applied to the same Q-learning baseline. Furthermore, we also include a policy gradient algorithm baseline — actor critic (R. S. Sutton et al. 2018) — in our empirical evaluation. Policy gradient methods parameterize their policy

directly by a vector θ, and they improve their policy by performing gradient descent steps on the objective $L(\theta) = \mathbb{E}_{S_0 \sim \rho}[v_{\pi_\theta}(S_0)]$.

Chapter 3

Environments

Exploration can be challenging due to a number of factors. The rewards may be misleading or even adversarial, ticking the agent into a sub-optimal path. The dynamics of the world could make it difficult for the agent to highly rewarding regions of the state-space. Perhaps most realistically, the state-space might be vast with many reasonable policies possible. Many real-world applications might exhibit one or more exploration challenges in addition to other non-exploration challenges such as high-dimensional inputs, continuous actions, partial observability, non-stationarity, or delayed action effects.

In order to focus our investigation, we consider a suite of small, issue-oriented diagnostic exploration test problems, each designed to focus on a particular exploration challenge. We avoid the complexities of complicated function approximation architectures and optimizations designed to improve sample efficiency (such as n-step methods or prioritized replay) and instead focus on simple implementations of core ideas for model-free exploration. In each of the six problems we describe next, good performance requires solving the underlying exploration challenge.

All environments have continuous state space, which makes it difficult for the agent to visit the exact same state multiple times, and simple finite action sets. In our experiments we use the discount factor $\gamma = 0.99$. Each environment is structured so that the value of the optimal policy from the start state is approximately 1. In some of the environments, agents who do not explore enough will tend to learn a specific suboptimal policy, which has

a value around 0.01 from the start state. Details regarding parameter choices and implementation are discussed in Appendix A.

3.1 Exploration Properties Related to Reward

First we present three properties of the reward function that make exploration difficult for value-based RL agents.

3.1.1 High-variance Reward: VarianceWorld

Our first property is simply high variance in the reward function. As this variance increases, functions of reward from a state-action pair require more data to estimate accurately. As a result, action values are more difficult to estimate when rewards are high-variance, and the agent will expect suboptimal actions to be optimal a larger proportion of the time. This property is exemplified by VarianceWorld, a continuous navigation environment adapted from White et al. (2010). This environment is a short, one-dimensional corridor with a small fixed reward on one side and a large in expectation, yet highly varying reward on the other. The agent begins in the center of the corridor, and has two actions that move to the left or right; episodes terminate when the agent moves past either end of the corridor. The agent needs to sample the high-variance reward sufficiently many times to obtain an accurate value estimate of the highly rewarding state.

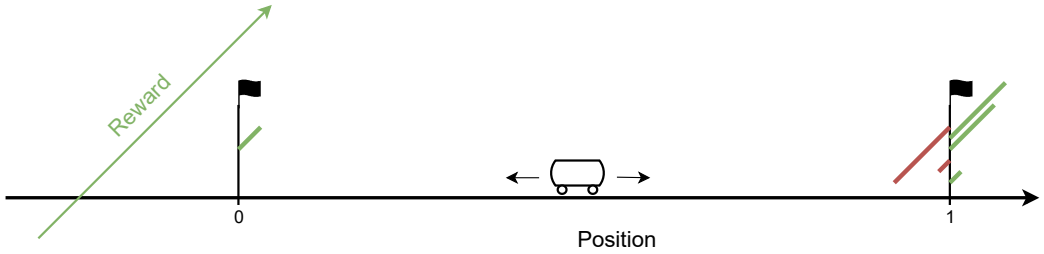

Figure 3.1: Diagram of VarianceWorld. The environment is a one-dimensional corridor with two ends as terminal states. The agent starts in the middle of the corridor. The terminal reward at the left end is a small fixed value, indicated by the short green line. The terminal reward at the right end is large in expectation yet sampled from a discrete uniform distribution, indicated by the three green lines and two red lines. In this diagram, green lines represent positive values and red ones represent negative values. Their lengths illustrate the relative magnitude of values.

3.1.2 Misleading Reward: Antishaping

Reinforcement learning agents explore in state-action space by examining local reward signals. If these local signals are misleading, the agent may fail to make globally optimal decisions. The Antishaping environment, another one-dimensional corridor environment adapted from Langford (2018), gives a simple example of this property. The agent begins at the left end of the corridor and can move left or right. Episodes terminate when the agent reaches the right end of the corridor and receives a large reward. During the episode, a small misleading reward is provided to the agent. The reward decreases as the agent approaches the terminal state on the right. To solve this domain, the agent needs to ignore misleading reward signals in the vicinity and move persistently to the right end.

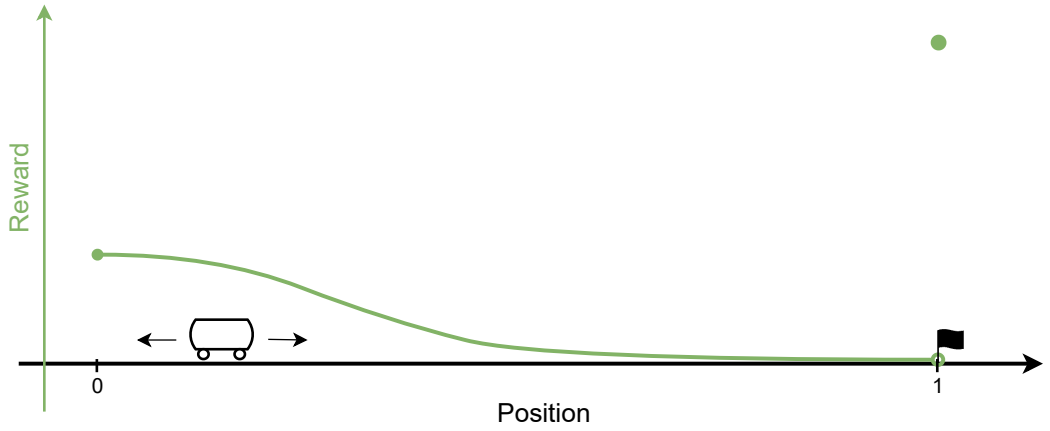

Figure 3.2: Diagram of Antishaping. The environment is a one-dimensional corridor whose rightmost edge is the terminal state. The agent starts at the leftmost edge, and it receives small yet decreasing rewards as it moves right. The terminal reward at the right end is large. In this diagram, the height of the green reward curve at one position illustrates the relative magnitude of the reward in that state.

3.1.3 Sparse Reward: Sparse MountainCar

When the rewards observed by the agent are all equal to zero, there is no signal in the reward for the agent to optimize. In this case, the agent can only expect all the state-action pairs it has visited to have a value of zero. This zero-everywhere value function implies that all policies are equally rewarding. The Sparse MountainCar (SparseMC) environment is a modified version of the classic toy problem MountainCar (Moore 1990). The agent is an underpowered car that uses momentum to drive up a hill to a goal, where the episode terminates. Driving uphill requires a sequence of mildly coordinated actions, but even an agent that chooses actions randomly can reach the goal within around 50k time-steps. The reward is typically -1 per time-step, incentivizing the agent to terminate the episode as quickly as possible. In our "Sparse MountainCar" implementation, the reward is 0 everywhere, with a 3.34 reward upon reaching the goal.

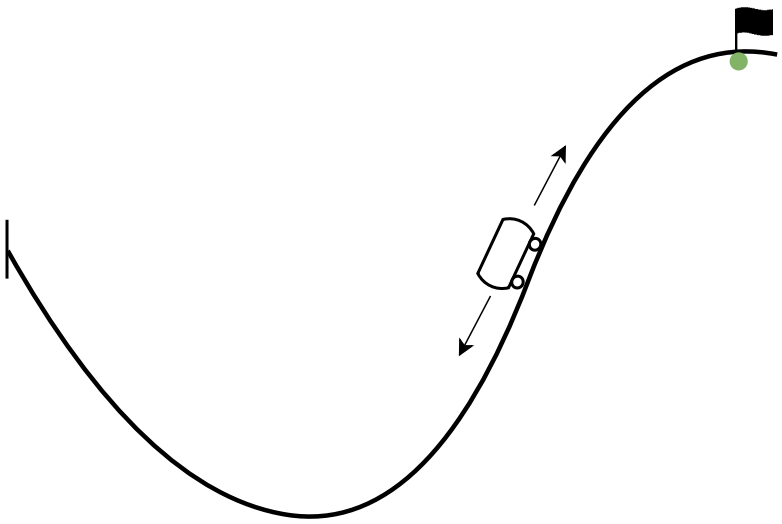

Figure 3.3: Diagram of Sparse MountainCar. The environment depicts a scenario where an underpowered car needs to use momentum to drive up a hill. Specifically, the agent must drive up the left hillside and then drive up the right hill with the help of momentum. The green dot at the terminal state indicates the only nonzero reward in this environment.

3.2 Exploration Properties Related to Transition Dynamics

Next we present three properties of the transition dynamics that make exploration difficult for value-based RL agents. Environments in which it is difficult to frequently visit all states can be challenging to explore, especially when the states that are more difficult to visit have high rewards. An environment can also be challenging to explore if the states that are likely to be visited cause the agent to believe that suboptimal actions are optimal. Lastly, environments can be difficult to explore if the state-action space is quite large, even if the optimal policy only visits a small fraction of state-action pairs.

3.2.1 High-variance Transitions: WindyJump

When transitions are high-variance, the sequence of states visited by an agent is also high-variance. As a result, it can be difficult to estimate the value of a state-action pair without a large number of samples. The WindyJump

environment is similar in spirit to the VarianceWorld environment, while it creates stochasticity in samples of the return by randomizing the state transition dynamics rather than the rewards. The agent's goal is to jump over a pit on the right side of the environment to reach a large reward. Episodes terminate whether the agent falls into the pit or jumps over it, or when the agent reaches the left end of the corridor and receives a small reward. In the wind-free region of the environment, the agent can move left or right according to a low-variance Gaussian distribution. On the windy right half of the environment, the agent's movement is more stochastic, moving left or right according to a nearly symmetric uniform distribution. Due to these stochastic transitions, the returns under the optimal policy are high-variance.

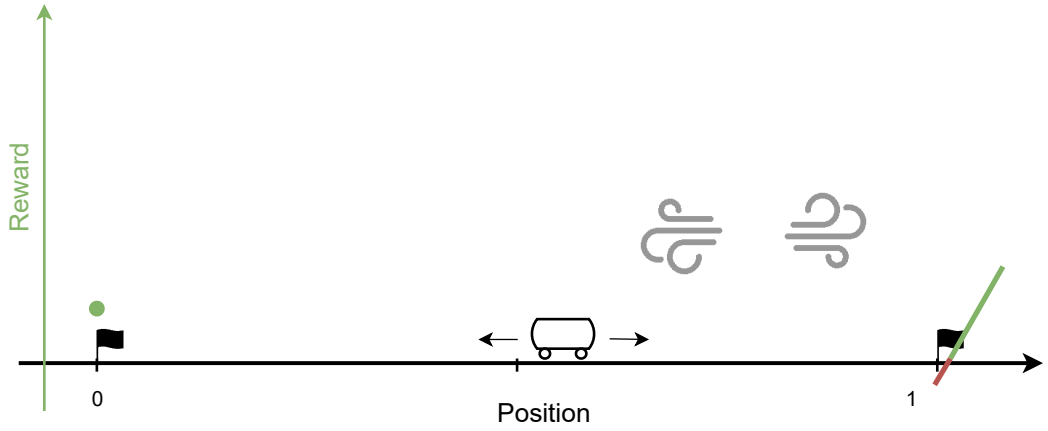

Figure 3.4: Diagram of WindyJump. The environment is a one-dimensional corridor with two ends as terminal states. The agent starts in the middle of the corridor. The terminal reward at the left end is a small value, indicated by the green dot. The terminal reward at the right end depends on how far the agent jumps over the right edge. If the agent falls into the pit, it receives a negative reward, indicated by the short red line segment; if the agent jumps over the pit, it receives a large positive reward, indicated by the long green line segment. In this diagram, the height of the reward line segment at one position illustrates the relative magnitude of the reward in that state.

3.2.2 Antagonistic Transitions: AlpineSki

When the transition dynamics are antagonistic, some states are much more difficult to reach than others. We give an example of an environment with antagonistic transitions inspired by the Combination Lock environment (Lang-

ford 2018) and a highway commute environment (Russo 2019), called AlpineSki. The agent is an alpine skier at the top of a mountain. At each timestep, the skier can choose to ski down, terminating the episode, or traverse across the top of the mountain. Skiing down the slope at the right end of the mountain — which requires a sequence of persistent "traverse" actions first — results in a high reward, whereas skiing down earlier gives a small reward. Undirected exploration methods (e.g., methods that learn a stochastic policy or a policy with noise applied) are less likely to perform well in this environment.

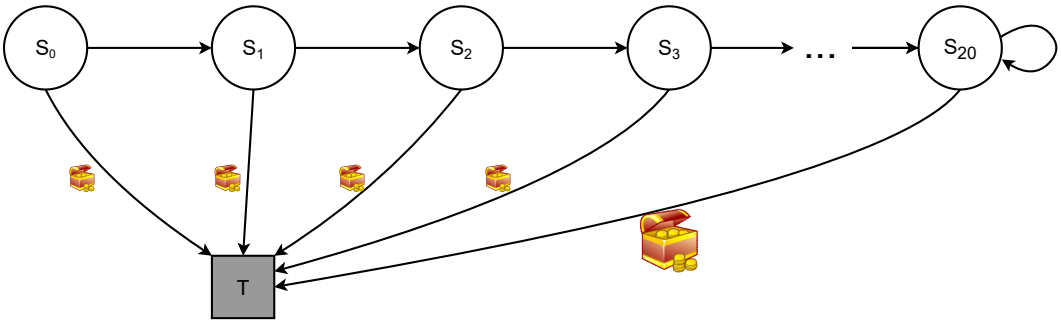

Figure 3.5: Deterministic illustration of AlpineSki. The environment is a one-dimensional continuous corridor. For simplicity this diagram shows the deterministic version with stochasticity in the action distribution ignored. The agent starts at the leftmost edge, and at each state it can choose to terminate the episode by skiing down, or traverse to the right state. Skiing down at the rightmost state results in a large reward, while skiing down at any other state results in a small reward.

3.2.3 Large State-action Space: Hypercube

Any environment can be made trivially more difficult to explore by adding states and actions that are not visited by the optimal policy. Our example environment for large state-action spaces is symmetric along each dimension of the state-action space, so all states are relatively close to the path of an optimal policy. The Hypercube environment has a relatively large 3-dimensional cubic state-space. The agent starts each episode at the origin, and can move up to around 10 steps away from the origin in either direction along any of the axes. The objective of the agent is to travel to any vertex of Hypercube,

where the episode will terminate. Since the state-action space is so large, we add shaping rewards to help the agent find the vertices. The agent receives increased reward as it touches more surfaces of Hypercube. The increase in reward when the agent touches another surface is such that the reward from one time-step of touching $n+1$ surfaces is larger than the discounted return from touching only n surfaces indefinitely. One optimal policy is to move persistently along each axis until reaching one more surface.

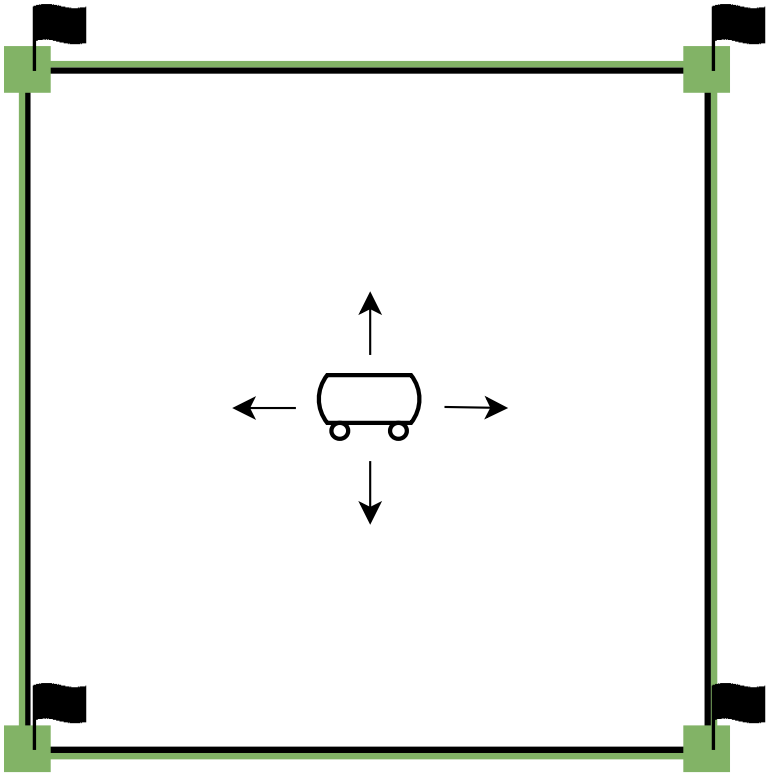

Figure 3.6: Lower dimensional illustration of Hypercube. The environment is a three-dimensional cube whose vertices are terminal states. For simplicity this diagram shows the two-dimensional squared version. The agent starts in the center, and it can move along any axis. There is no reward signal in the interior of the environment. The agent receives a large reward upon reaching a terminal state. To help the agent explore, shaping rewards are added to each surface such that staying in n surfaces forever results in cumulative reward less than touching $n+1$ surfaces for one timestep. In this diagram the width of the green line indicates the relative magnitude of the reward at that state.

Chapter 4

Model-Free Exploration Methods

Exploration methods can be either *undirected* or *directed*. Undirected methods inject stochasticity in action selection, typically through the use of soft-greedy approaches like ϵ-greedy or softmax policies. Directed methods, on the other hand, attempt to focus exploration to obtain information that could improve the policy. Most exploration methods are directed (even though most systems use undirected methods), and most directed exploration methods can be seen as encoding the principle of optimism in the face of uncertainty.[1]

Uncertainty estimation can take several forms. Several methods estimate *epistemic uncertainty* in the action values based on data from previous interactions with the world. Others use proxy measures for uncertainty in the estimate for a state, such as counts or model errors, and incorporate that uncertainty directly into the action-value estimate in the form of *reward bonuses*. These reward bonuses affect the return—they propagate through the value function using bootstrapping—and ensure that the uncertainty in the value estimate of a state and action reflects the uncertainty in downstream states.

A critical separating factor amongst these methods is whether or not they explicitly incorporate uncertainty derived from unvisited parts of the state-

[1]There is a large literature on Bayes adaptive MDPs (Duff 2002; J. J. Martin 1967), which formulates the exploration problem itself as a control problem. These approaches do not use optimism, and are typically model-based. There have been a few attempts to make this idea more practical to use, with model-free methods, both for the tabular setting (Şimşek et al. 2006) and under function approximation using meta-learning (Zintgraf et al. 2020). These methods, however, remain quite computationally intensive, and so we do not consider them further here.

action space, which we call *out-of-sample epistemic uncertainty*. *In-sample epistemic uncertainty* methods, like the statistical bootstrap, only reflect uncertainty on the parameters within the data visited. Even if the agent never visits a part of the space, the epistemic uncertainty can go to zero in the visited part. *Anticipatory methods* explicitly model uncertainty from unvisited parts of the state-action space. For example, an anticipatory method might assign a high value to an action that has never been taken in a state, reflecting that it could have a high value (e.g., optimistic initialization). A reward bonus method, on the other hand, without any optimistic initialization, must take an action in a state before it increases its value; it is not anticipatory, but rather retroactive.

Finally, the (optimistic) value estimates, and uncertainties, can be used in different ways for action selection: greedy selection on optimistic values or Thompson sampling. The classic example of the first is UCB (Lai et al. 1985), where actions with the highest estimated upper confidence bound can be selected. This upper confidence bound is the optimistic value estimate, with actions selected greedily according to that optimistic estimate. Other approaches that do not directly estimate uncertainties use a similar idea. For example, the agent can select greedy actions with respect to the optimistic values learned under reward bonuses. Alternatively, the agent can use Thompson sampling (Thompson 1933), where the parameters are sampled from the posterior distribution of parameters. This strategy is limited to approaches that maintain a posterior. For example, it cannot be used with reward bonus methods. However, if a posterior can be computed, then either an upper confidence bound can be used or Thompson sampling. In fact, they can both be seen as implementing optimism, with many regret bounds established for upper-confidence-based approaches extending to Thompson sampling (Russo and Van Roy 2014).

In this chapter, we categorize and discuss model-free exploration methods, based on these described differences. Later, in our experiments we select a representative instance of each category.

4.1 Estimating Value Uncertainty with Count-based Reward Bonuses

Early work in exploration in RL primarily focused on proxy measures of uncertainty. In fact, many of the ideas used today were proposed in early work: preferring less frequently visited state-action pairs based on counts (Barto et al. 1991); preferring state-actions where there was previously high prediction error (Moore 1990; Schmidhuber 1991); using errors in model estimates as a proxy for knowledge gain (Thrun 1992; Thrun and Moller 1991); and using interval estimation (IE) on rewards to obtain local exploration bonuses for action selection (Kaelbling 1993). A key insight from this early work is that local uncertainties, about rewards or state information, need to be propagated to obtain global uncertainties (Meuleau et al. 1999). Many methods estimate or compute local uncertainties, such as reward uncertainties in IE directly to select actions (Kaelbling 1993). Instead, this uncertainty needs to be propagated to account for the fact that an action can lead to a state with high uncertainty.

The Interval Estimation Q-learning (IEQL+) algorithm (Meuleau et al. 1999) was introduced to extend IE (Kaelbling 1993) to compute global uncertainty on value estimates, by propagating local uncertainties. Instead of using local uncertainties to directly select actions, the local uncertainty is incorporated into the value update, in the reward. This approach underlies modern reward bonus strategies, and was the inspiration for a line of model-based exploration approaches, starting with Model-Based Interval Estimation with Exploration Bonuses (MBIE-EB) (Strehl et al. 2008). The idea of propagating exploration bonuses actually arose slightly early, in a model-based algorithm called Dyna-Q+ (R. Sutton 1990), where the Q-learning updates in the planning loop include a recency bonus. These planning updates propagating these local exploration bonuses, to globally increase action-values for actions that have not been recently selected or that lead to parts of the state that have not been recently visited.

The idea of incorporating local bonuses into the reward, to obtain globally optimistic value estimates, has been particularly compelling because it easily extends to function approximation. Consequently, there have been many works relying on this idea, focused on different choices for the reward bonus. The first algorithms relied on estimating counts for continuous state spaces, called pseudo-counts (Bellemare et al. 2016). Exploration is encouraged by adding a multiple of $\frac{1}{\sqrt{n(s)}}$ to the agent's reward, where $n(s)$ is the pseudo-count for state s, estimated using density estimation. This work inspired a series of papers, improving on the estimation of these counts (Machado, Bellemare, et al. 2019; J. Martin et al. 2017; Ostrovski et al. 2017; Tang et al. 2017). One heuristic uses feature activation estimates, and aggregates those estimates for the active features in a state (J. Martin et al. 2017). State visitation can also be approximated using the ℓ_1 norm of a vector called the successor representation (Machado, Bellemare, et al. 2019).

Counts provide a distribution-agnostic measure of uncertainty. Under iid sampling, basic concentration inequalities like Hoeffding's bound tell us that uncertainty decays with $1/\sqrt{n}$. The UCB bandit algorithm employs such confidence estimates, to avoid making strong distributional assumptions (for an extensive overview, see Lattimore et al. (2020)). Such a choice is relatively conservative. For example, another option is to use Bayesian methods for the reward to obtain an estimate of reward uncertainty (namely to maintain a posterior over the reward), as was used in a model-based method with exploration bonuses (Hester et al. 2017). The most common option has been to use proxies for information gain, which we discuss in the next section.

Note that reward bonus approaches, both those discussed in this section and the next, are retroactive, rather than anticipatory; we discuss this more in Section4.3.

4.2 Estimating Value Uncertainty with Learning Progress Reward Bonuses

Many different reward bonuses have been developed to reflect *learning progress*. Here the agent learns an auxiliary function, such as the model, and incorporates some measure of how much it learned for that auxiliary function when taking an action in a state. For example, the agent can learn the transition dynamics for the environment as an auxiliary function, updating with the experience generated by the agent. For state-action pairs where the prediction error is high, the agent receives a high reward bonus to revisit that part of the space (Pathak, Agrawal, et al. 2017; Stadie et al. 2015). Such prediction error reward bonuses are typically inadequate measures of learning progress, as they may simply reflect noise in the targets (see Linke (2021) for a thorough overview).

An alternative strategy is to estimate information gained. Bayesian methods which maintain a posterior over the model facilitate computing the change in the posterior after an update. This idea—called Bayesian surprise (Itti et al. 2005)—was actually first explored in a pure exploration setup (Storck et al. 1995), rather than as a bonus added to the reward, and since has been more generally used either for pure exploration (Antos et al. 2008; Linke 2021; Orseau et al. 2013) or as a bonus (Achiam et al. 2017; Houthooft et al. 2016; Little et al. 2013; Still et al. 2012). The difficulty in using Bayesian surprise is that it requires maintaining posteriors and computing KL divergences between distributions—namely information gain—which can be expensive.

A simple heuristic to approximate information gain is to assess if the agent has identified the correct function from a realizable function class. Prediction errors are problematic because zero prediction error is not always possible, because (a) outcomes may be stochastic and (b) the true model may not be representable by our function class. In Random Network Distillation (RND) (Burda et al. 2018), the secondary function $\hat{f}_w(S_t)$ with weights w predicts a fixed, randomly initialized target $f(S_t)$. Since the target function is deter-

ministic, the agent will not receive large reward bonuses for visiting stochastic areas of the environment. Further, the prediction error will decrease as the agent observes more samples in each region of the state space. Another approach has been to use variance across an ensemble of model estimators (Pathak, Gandhi, et al. 2019), which should also reduce to zero over time as it effectively measures epistemic uncertainty.

4.3 Directly Estimating Value Uncertainty

Another class of methods attempts to directly obtain interval estimates around action-values, rather than propagating local uncertainty estimates. Let us first consider why this is difficult, and why it differs from propagating local uncertainties. Consider the simpler policy evaluation setting, where the goal is to obtain uncertainty estimates on the action-values for a fixed policy π. If we use Monte Carlo rollouts, then we can leverage existing methods for computing uncertainties on our estimates. For example, given a dataset of state-action pairs and corresponding sampled returns, we can use Bayesian linear regression—or more advanced approaches like Gaussian processes—to obtain confidence estimates directly on value estimates. The rate at which the confidence interval shrinks in Bayesian linear regression depends on the number of samples, the variance in the returns, and the initial prior variance over parameters.

This differs from using Bayesian linear regression around rewards, with reward bonuses, for two key reasons. First, the intervals are likely to be wider, since local uncertainties accumulate. Namely, summing variance per state in the trajectory will be larger than the return variance. Second, incorporating reward bonuses is not anticipatory: it does not promote out-of-sample optimism. Direct estimates of uncertainty on action values should both reflect in-sample epistemic uncertainty, as well as out-of-sample epistemic uncertainty due to lack of visitation. In Bayesian linear regression for values, for example, the prior provide out-of-sample optimism. Therefore, though re-

ward bonuses appear similar to uncertainty estimates around action-values, they are notably different for these two reasons.

Practically, there is another important difference: directly estimating uncertainty on action-values is more difficult. For Monte Carlo estimates in policy evaluation, it is straightforward. Once we include *bootstrapping* as in TD learning, however, we can no longer easily apply existing Bayesian methods because the targets are constantly changing and involve estimates. Further, once we move to control with Q-learning, the data distribution changes as the policy changes. These difficulties might explain the focus on model-based exploration methods, which facilitate computing these uncertainties.

More recently, however, several efficient model-free exploration approaches have been developed for the function approximation setting. Some methods have relied on least-squares approaches to obtain intervals based on variance estimates (Jin et al. 2019; Kumaraswamy et al. 2018; O'Donoghue et al. 2018; Osband, Roy, et al. 2016); others have used statistical bootstrap approaches (Chen et al. 2017; Osband, Aslanides, et al. 2018; Osband, Blundell, et al. 2016; White et al. 2010); and others have used more explicit Bayesian strategies (Azizzadenesheli et al. 2019; Dearden et al. 1998; Engel et al. 2005; Gal et al. 2016; Grande et al. 2014).

Least squares approaches facilitate sample-efficient closed-form solutions for both value estimates and variances. Kumaraswamy et al. (2018) propose a method called Upper Confidence Bound Least Squares (UCLS), which estimates the variance of the value function using a second set of weights. The variance estimate is then used to construct an upper confidence bound on the values. UCLS's variance estimates can also be combined with Thompson sampling. This approach resembles RLSVI (Osband, Roy, et al. 2016), which iteratively applies Bayesian linear regression by statistical bootstrap off of parameters sampled from a Gaussian posterior. RLSVI is designed for the finite horizon setting, though practically it could be used in an episodic setting with cutoffs, and has an upper bound on the expected regret in the finite horizon setting. Either approach could use upper confidence bounds or Thompson

sampling, though UCLS originally used upper confidence bounds and RLSVI used Thompson sampling. Both these methods, however, are restricted to linear function approximation.

Statistical bootstrapping approaches are an elegant way to extend exploration methods to nonlinear function approximation. The idea is simple: the experience is resampled N times, with N functions estimated on these different subsets. The variability across these functions reflects the variability due to insufficient samples (epistemic uncertainty); with more and more data, this variability disappears. Bootstrapped DQN (BootstrapQ) (Osband, Blundell, et al. 2016) uses this idea, learning N neural networks on a subset of data screened incrementally. The most straightforward implementation involves learning N separate networks; storing separate replay buffers for each neural network that is a screened subset of all the data; and using the action-value estimates per step to select actions. Practically, however, this can be expensive and the authors provide a different final algorithm for BootstrapQ that uses a multi-headed network and screens updates from a shared mini-batch. Additionally, Thompson sampling is applied only at the start of the episode, where one of the N networks is used to select actions for the episode. Out-of-sample optimism was later incorporated, by adding an initial source of randomness, called Bootstrapped DQN with Randomized Prior Functions (BootstrapQ+prior) (Osband, Aslanides, et al. 2018); the original BootstrapQ without prior algorithm relies solely on differences in value estimates due to random initialization.

Many methods based on Bayesian updates have issues with the fact that values change during learning. Bayesian Q-learning approaches (Azizzadenesheli et al. 2019; Dearden et al. 1998) rely on Bayesian linear regression—designed for stationary targets—and others rely on Gaussian process regression (Chowdhary et al. 2014; Engel et al. 2005). These strategies can result in exponentially slow convergence, because the GP variance reduces too quickly (Grande et al. 2014); the variance, however, needs to stay higher to account for the changing action-values in the target. This was ameliorated in Delayed

Gaussian Process Q-learning (DGPQ) (Grande et al. 2014), where two values are learned and the one using the Gaussian process has the variance periodically reinitialized. Though theoretically appealing, this approach is expensive and difficult to use (Kumaraswamy et al. 2018).

Finally, a simple heuristic, called optimistic initialization (OptimisticInit), does not neatly fit into the above categorization. The idea is to initialize value estimates to be high, with each subsequent update slowly decreasing this value. OptimisticInit could be seen as a heuristic for obtaining an upper confidence bound, though it is not explicitly reasoning about uncertainty. In fact, this approximate upper bound likely decays too quickly, because it decays at the rate that gradient descent changes the values rather than based on any notion of information of variance. It may be better thought of as a simple approach to incorporate a prior over the action-values, that encourages out-of-sample optimism that decays with updates, rather than as a standalone approach. For example, it provides a simple way to incorporate a prior for reward bonus approaches.

Note that OptimisticInit is not always straightforward to use. For linear function, with non-negative features, weights can be simply initialized to be higher than the maximal possible value. For neural networks it is not as straightforward. Machado, Srinivasan, et al. (2014) propose a strategy that is to instead normalize and decrease the rewards so that zero initialization is optimistic. In the nonlinear setting, optimism decays quickly due to the generalization of neural networks. Also, the rewards are normalized by the first non-zero reward seen, which is rarely attained in sparse reward tasks like SparseMC. Rashid et al. (2020) avoid the issue due to the generalization of neural networks by augmenting a count-based upper confidence term to the value function estimate during action selection and bootstrapping. Due to the explicitly state-dependent uncertainty estimation, this approach does not fully align with the classic OptimisticInit.

If we ignore the issue of generalization for a moment, the most straightforward way to obtain an optimistic initialization with neural networks is to

initialize the bias of the output layer to a large positive value. As the agent explores the environment and updates its weights, the action value estimate will be corrected. This approach is equivalent to adding a large constant to the value estimates, where the constant serves as an optimistic prior. Therefore, this approach is also essentially equivalent to a BootstrapQ+prior variant with ensemble size 1, except the randomized prior function does not output the same constant across all state-action pairs. We examine this special BootstrapQ variant in Chapter 6. As for our OptimisticInit implementation, we try a different form without explicitly touching values within the neural networks.

In our experiments on OptimisticInit, we propose to train the initial network towards a large constant, with a set of state-action pairs that cover the space. The initial optimism at unseen state-action pair would still fade away quickly due to the generalization of neural networks. Nonetheless, we evaluate this implementation hoping to gain more empirical insights.

4.4 Undirected Approaches

Undirected methods encourage exploration via random action selection. One of the oldest soft-greedy approaches is ϵ-greedy, which usually selects the action with the highest estimated value but with probability ϵ selects a random action. Since exploration is independent across time-steps, it is difficult for the agent to be persistent and reach hard-to-reach parts of the space. Though ϵ-greedy is sufficient for Q-learning to converge to the optimal policy in tabular MDPs (Melo 2001; Tsitsiklis 1994), it can take exponentially many time-steps to explore environments with antagonistic transitions (Osband, Van Roy, et al. 2019). Related approaches include using a Boltzmann distribution on action-values (Duncan 1959), which samples actions proportionally to action-value magnitudes, and incorporating entropy regularization which can be shown to be equivalent to using a Boltzmann policy (Ziebart et al. 2010).

A simple extension of these simple soft-greedy operators is to learn the noise parameter, such as the temperature or ϵ. The most commonly used approach in this category is Noisy Networks (Fortunato et al. 2019; Plappert et al. 2018), which samples the noise ξ^w after each optimization step. It is implemented by learning a vector of noise parameters σ^w in addition to the neural network parameters μ^w. To query a value from the noisy network, the noise parameters are element-wise multiplied by a sample ξ^w from a fixed, zero-mean noise distribution and added to the neural network parameters, to give $w = \mu^w + \sigma^w \odot \xi^w$. Those noised parameters w are used for that query, namely in the forward pass. Both the neural network parameters and the noise parameters are updated using gradient descent.

These undirected approaches provide some amount of exploration, but do not attempt to assess what actions would provide the most useful information. They nonetheless continue to be popular, due to their simplicity. A positive aspect of these soft-greedy approaches is that they encourage some amount of exploration, which means that they could actually be complementary to directed exploration methods—perhaps most useful in continual non-stationary tasks. In addition, the uncertainty estimates on action values may be inaccurate and soft-greedy action-selection on these inaccurate upper confidence bounds might provide an additional level of robustness. Moreover, for simple environments where minimal exploration is required, they may in fact be sufficient. For this empirical study, we test them on their own—rather than paired with other methods—as baseline methods.

4.5 Representative Methods for the Empirical Study

There are a variety of methods within each category. Our goal is to evaluate the exploration ideas behind the methods, and so we select representative methods that are best reflect those ideas and are in common use. We summarize these in the following list. As baselines, we include **Q-learning** with-

out any exploration and **Softmax Actor-Critic**, which only has some initial exploration due to the initialization of the policy variance. We also include **Optimistic Initialization** as a simple anticipatory exploration heuristic.

1. **Undirected Methods**: ϵ-greedy and noisy networks.

2. **Reward Bonus with Counts**: Pseudo-counts on states. We use counts on states, because earlier experiments did not present significant distinctions between counts on states and counts on state-action pairs on their performance.

3. **Reward Bonus with Learning Progress**: Random Distillation Networks (RND). We choose RND, because it is a general-purpose approach, in that it does not rely on using Bayesian methods or computing information gain directly.

4. **Value Uncertainty**: Bootstrapped DQN with Randomized Priors. We include Randomized Priors, to ensure the agent has some mechanism for out-of-sample uncertainty. When sweeping hyperparameters, one hyperparameter setting allows for the prior to be omitted, reducing to the original Bootstrapped DQN algorithm. For simplicity, we use the abbreviation BootstrapQ to refer to both BootstrapQ+prior and BootstrapQ without prior agents in the later text, unless there is a need to explicitly distinguish between the two.

Chapter 5

Experimental Setup

In this chapter we present the experimental setup. Results of our empirical experiments are presented in Chapter 6.

5.1 General Setup

We adapt each representative exploration method introduced in Chapter 4 to work with a Q-learning baseline agent with function approximation and evaluate their performance in each of the exemplary environments described in Chapter 3. The function approximation methods include neural networks for nonlinear approximation and tilecoded representations for linear approximation. Under nonlinear settings, our scope covers both incremental updates that merely use the most recent transition data at each timestep, and mini-batch updates that utilize experience replay and target networks, as mentioned in Chapter 2.

We consider the classic exploration-exploitation trade-off setting: given a certain amount of timesteps, the agent accumulates rewards and at the same time learns about the environment. The performance measure is not straightforward since both the cumulative reward and the quality of the final policy learned are important evaluation metrics. In this thesis, we consider a compromise between the two: rewards accumulated during the final 10% of the learning phase. This performance metric is not only a good approximation to the quality of the final policy, but it also reflects the algorithm's ability to

maximize reward.

Specifically, for each hyperparameter setting, the agent is trained for $500,000$ timesteps, which is sufficient to learn the optimal policy in each environment. We repeat the training phase 30 times, where each repetition uses a unique seed that is shared across exploration methods. Episodes are not cut off by any time-out mechanism unless the budget number of timesteps is exceeded. The performance of the agent is evaluated by the cumulative reward obtained with the final 10% of the given budget.

Each hyperparameter is swept over a group of candidates that are chosen based on reported values from the papers (or accompanying open-sourced implementations) that originally proposed these exploration methods. Most methods perform best with different settings on each environment, and rarely perform best with the most extreme parameter settings, which indicates that the candidates are appropriately diverse. We choose hyperparameters, including tilecoding settings, separately for each environment. Since performance tend to be distributed bimodally, we identify the best hyperparameters by their median cumulative reward. If the best hyperparameter setting is on the edge of the sweeping range, we expand the hyperparameter sweep and re-ran experiments until the best setting is within those ranges. We treat the performance from hyperparameter sweeps as a good representation of the agent's capability. We do not fine-tuning the hyperparameters since the computational costs of conducting it often surpass the benefits from exploration.

5.2 Hyperparameter Settings

In order to study the impact of feature representations on agents' exploration behaviour, we experiment with both linear and nonlinear function approximations. For nonlinear approximation, our function approximator is a multilayer perceptron (MLP) neural network model with rectified linear unit (ReLU) activation function applied on the hidden layers. Each layer applies a linear transformation to the incoming data x: $y = xA^\intercal + b$, where $A^\intercal$ and b are the

learnable weights and bias, respectively. The ReLU activation is applied to its input element-wise: $\text{ReLU}(x) = \max(0, x)$. Due to the small scale and relatively lower dimensionality of our domains, we use a network model with two layers and 50 neurons on each layer. Unless otherwise specified we initialize network weights according to Kaiming uniform initialization (He et al. 2015) at the beginning of the learning phase. All neural network modules and optimization algorithms applied to nonlinear agents are implemented with the PyTorch framework (Paszke et al. 2019).

For the linear agent, we use three variants of tile coding for each environment, depending on the number of dimensions in the environment's state space. All three variants produce nearly the same number of features, with different levels of generalization (the ability to generalize over the state space) and discrimination (the ability to discriminating between nearby states). The first variant generalizes significantly over the state space but does not discriminate finely between nearby states. The second variant has a better local function approximation, with decreased generalization across regions of the state space, whereas the third variant is intermediate in both discrimination and generalization. All variants include a "bias" feature with a constant value of 1.

To update value function estimates, we use the Adam optimizer (Kingma et al. 2017) for nonlinear agents, and sweep both Adam and stochastic gradient descent (SGD) optimizers for linear agents. For nonlinear algorithms, we sweep over the learning rate $\alpha \in \{1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7\}$. Unless otherwise specified we initialize network weights according to Kaiming uniform initialization (He et al. 2015) at the beginning of the learning phase. As for linear algorithms, we set the learning rate $\alpha = \frac{\alpha_0}{\#\text{tilings}}$, and sweep over learning rates of $\alpha_0 \in \{1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125\}$. Unless otherwise specified we initialize all weights to zero at the beginning of the learning phase. More detailed hyperparameter settings are discussed in Appendix C.

5.3 Algorithm Details

In this section we describe implementation details of our nonlinear optimistic initialization (OptimisticInit). More detailed descriptions and pseudocode for each method are introduced in Chapter B

We obtain an OptimisticInit agent by firstly training a network towards a large constant, and then letting the Q-learning agent load this pre-trained network at the beginning of the learning phase. The network is randomly initialized before pretraining. The offline training has $N_{\text{pretrain}} = 100\text{k}$ iterations. At each iteration, a batch of 256 state-action pairs are randomly sampled from the environment. Using this minibatch, one gradient descent step is performed to minimize the mean squared error between the network's current predictions and the optimistic value. Given the small scale and low dimensionality of our domains, this set of state-action pairs should be able to achieve a good state coverage.

Each optimistically initialized network is trained with a random seed uniquely determined by the environment, the optimistic value, and a network index. Due to the generalization of neural networks, updating predictions at some data points would also affect outputs of unseen data points. Therefore, even after training with such a huge amount of data, the resulting network does not predict the optimistic value everywhere in the state-action space. In fact, even for the same environment and optimistic value, networks with different indices would produce an ensemble of diverse function approximators. This property serves as the basis of our BootstrapQ variant, which is discussed and empirically investigated in Chapter 6.

Chapter 6

Results

In this chapter we present the results from experiments described in Section 5.1. We firstly present overview results of representative nonlinear agents in Section 6.1. Then we show the performance of the same group of agents with tilecoding representation in Section 6.2. In Section 6.3 we provide a meta-analysis on all methods and discuss the robustness of each algorithm to their hyperparameter settings. In Section 6.4 we present a more in-depth discussion of the method BootstrapQ. Finally we discuss other lessons the RL community could learn from our empirical study.

All plots we present in subsequent sections are bar plots made with R (Team 2013). Unless otherwise specified, the height or length of each rectangular bar indicates the median total rewards accumulated by each agent in one environment during the final 10% of the learning phase (i.e., the final 50k timesteps), normalized by the rewards accumulated by the domain-specific near-optimal policy. All near-optimal policies are deterministic and they are described in detail in Appendix A. We note that the near-optimal performance for each environment is not obtained by simple calculations based on the environment's definition with stochasticity ignored. Instead, we evaluate each near-optimal policy by averaging cumulative rewards it collects in 50k timesteps over 100 random seeds. [1] Since a representative explo-

[1] For any optimal policy whose expected episode length is not a divisor of 50k, we adjust the evaluation length and rescale the result accordingly. For instance, the optimal policy in SparseMC attains an episode length of approximately 105 timesteps, hence in each run we evaluate its performance with 52.5k timesteps, and divide the cumulative reward by 1.05.

ration method's median cumulative reward could potentially exceed the near-optimal performance, especially in a domain with high variance, the cumulative reward that we use to normalize performance is the highest cumulative reward achieved by all learning agents and the near-optimal agent.

Bars are coloured according to the agents' exploration method categorization identified in Chapter 4. Q-learning and Actor-Critic baselines have the same colour despite distinct exploration heuristics, because Actor-Critic — the only method that is not based on the Q-learning implementation — is also considered as a baseline. Intrinsic reward methods are also presented with the same colour.

6.1 Neural Network Results Overview

We present the overall results of representative agents with neural network approximations. For each agent in any environment, the best hyperparameter setting is determined by the cumulative reward over the final 10% of the given learning budget. To emphasize various patterns, we show three plots with different groupings: the performance across all domains is shown in Figure 6.1, bar plots grouped by agent are shown in Figure 6.2, and bar plots grouped by environment are shown in Figure 6.3. Only the best hyperparameter settings are shown in these three figures. In Figure 6.1, the performance of each agent is the average performance across all environments. To highlight the comparison between whole performance and final performance, we also present Figure 6.4 to show the performance of the same agents over the *entire* learning phase rather than the final 10%.

Figure 6.1 shows that all methods except BoostrapQ are able to improve upon the greedy Q-learning baseline. Within the same category except for anticipatory methods, agents also have similar performance. The Actor-Critic baseline achieves the best performance overall. The count-based method marginally outperforms the greedy Q-learning baseline. In general many approaches introduced in papers demonstrating their feasibility perform poorly

in our study. Four exploration methods including the Actor-Critic baseline have performance above one-half of optimal cumulative reward, and no agent is able to reach 75% of optimal cumulative reward. Only two methods recently proposed for deep RL match or slightly outperform ϵ-greedy.

Our results also suggest that BoostrapQ variants, which have been successful in hard exploration games, may potentially have drawbacks that are not present in other methods. In Section 6.4 we take a deep dive into BootstrapQ with an effort to mitigate its ineffectiveness.

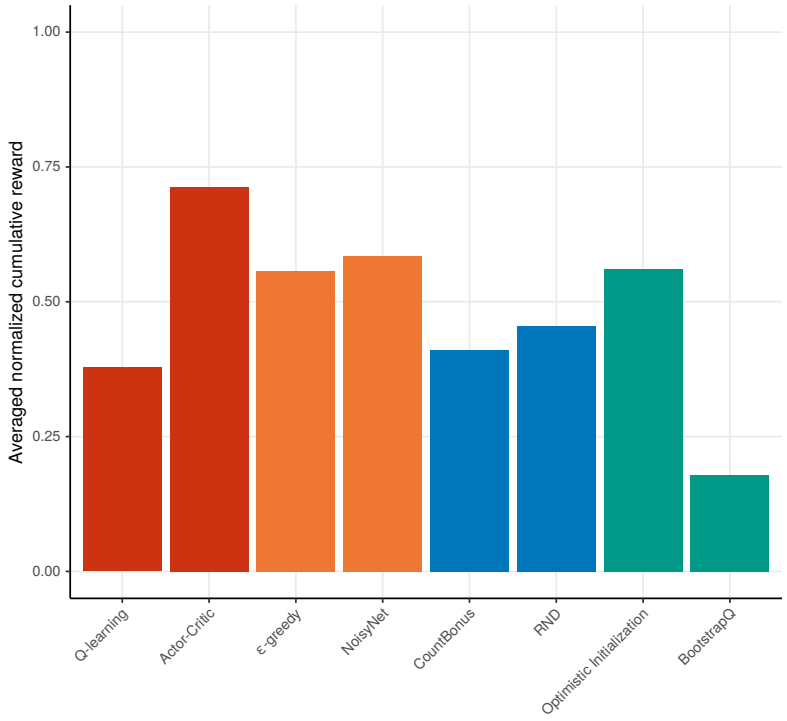

Figure 6.1: Median normalized cumulative reward *averaged across all environments* over final 10% learning budget for Q-learning agents with neural networks. Performance of the same agents with tilecoding is shown in Figure 6.5.

Figure 6.2 shows that ϵ-greedy exploration is competitive or outperforms retroactive methods — CountBonus and RND — in some environments, which aligns with recent conclusions on bonus-based methods benchmarking (Taiga et al. 2019). Most methods achieve high median cumulative reward in less than half of the environments. The Actor-Critic baseline is the only agent to achieve near-optimal performance in many environments. CountBonus is the

only method to consistently improve upon the vanilla Q-learning baseline in every environment, but the improvement in most environments is minimal. This suggests that the previous success of CountBonus methods in other benchmarking suites such as Atari may critically depend on the strength of the base agent. One aspect that influences the agent's strength is the neural network architecture, including the convolutional net that is usually applied to visual inputs. Other methods with Q-learning as their base agent fail to match Q-learning in at least one environment (e.g., ϵ-greedy, NoisyNet, and RND in Antishaping, OptimisticInit in Hypercube, and BootstrapQ in most environments except SparseMC and VarianceWorld).

Comparing Figure 6.2 and Figure 6.4, it is evident that most agents have whole performance similar to their final performance, which suggests that their policies improve quickly and converge soon in the early learning stage. The only exceptions are the undirected methods, namely ϵ-greedy and NoisyNet, across all domains, and RND in Antishaping. Their whole normalized performance in much worse than their corresponding final performance. This gap indicates that these methods improve slowly throughout the entire learning phase.

Figure 6.3 indicates that Antishaping is the easiest domain, where the majority of exploration methods reach the near-optimal policy. However, their performance has high variance: the best runs are near-optimal while the worst runs have zero cumulative rewards. SparseMC is the most difficult environment. Only the noisy methods — ϵ-greedy and NoisyNet — are able to obtain a relatively strong suboptimal performance. Most agents have poor or virtually zero performance in this domain. Moreover, methods under the same categorization tend to find similar suboptimal policies in each environment despite a few exceptions such as RND in SparseMC and BootstrapQ in most environments. This implies that similar methods often fall into the same trap in each environment.

Solving any of our environments requires the agent to execute a persistent sequence of actions. Particularly in AlpineSki and SparseMC, one wrong step

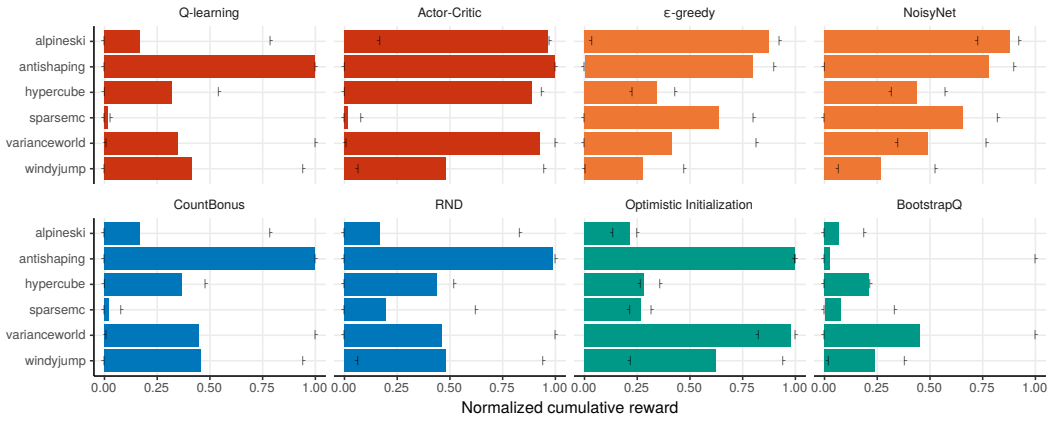

Figure 6.2: Median normalized cumulative reward over final 10% learning budget for Q-learning agents with neural networks. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Performance of the same agents over the entire learning budget is shown in Figure 6.4. Performance of the same agents with tilecoding is shown in Figure 6.6. Performance of the same agents with neural networks in one single hyperparameter setting is shown in Figure 6.9.

followed by a sequence of optimal steps will cause all previous efforts wasted. ϵ-greedy does not conduct temporally extended exploratory actions; however, it performs surprisingly well in these two environments, which contradicts our previous hypothesis that undirected exploration methods are less likely to perform well in those domains. It could be because occasional random actions due to the noise help the agent to jump out of the pitfalls laid in each environment.

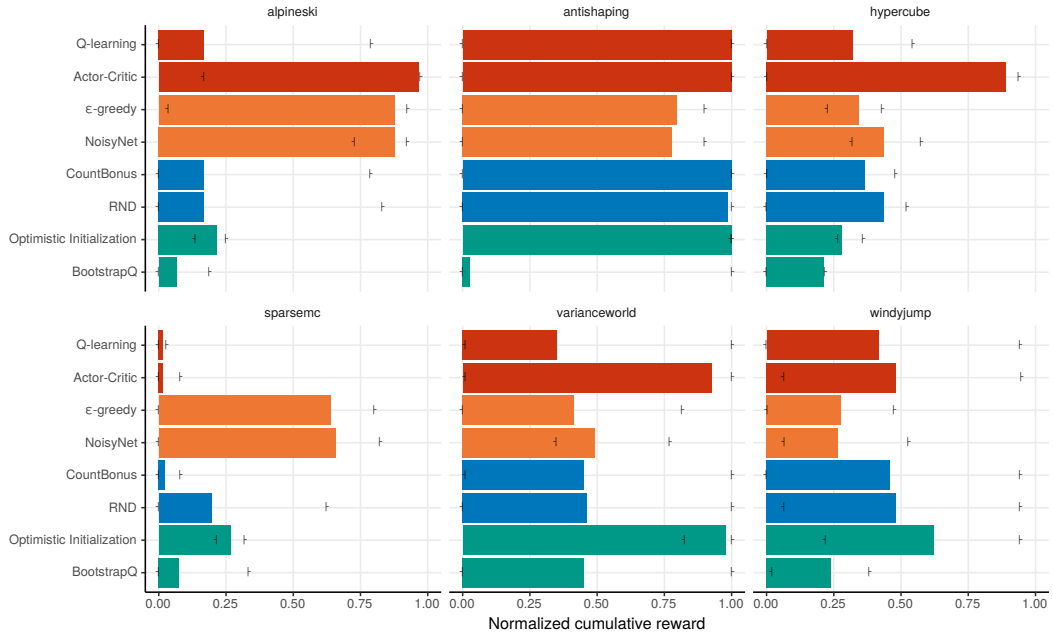

Figure 6.3: Median normalized cumulative reward over final 10% learning budget for Q-learning agents with neural networks. Bars are grouped by *environment*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively.

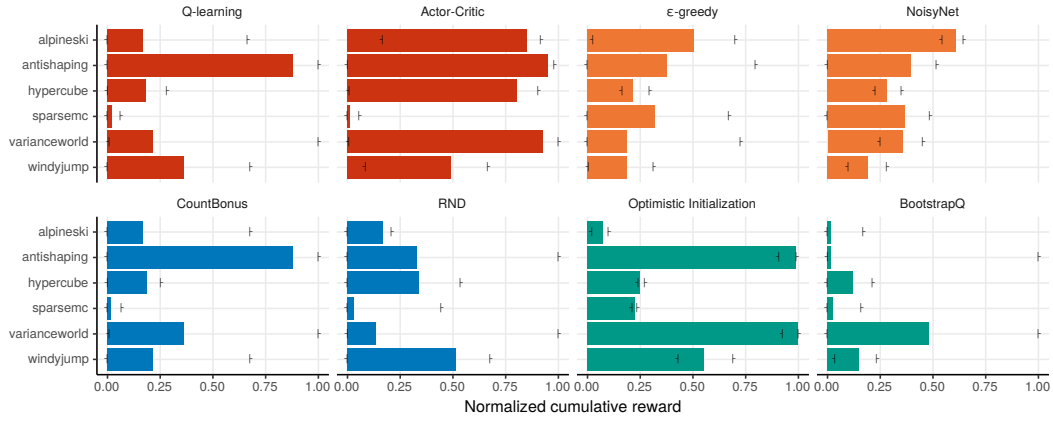

Figure 6.4: Median normalized cumulative reward over the *entire* learning budget for Q-learning agents with neural networks. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Performance of the same agents over the final 10% of learning budget is shown in Figure 6.2.

6.2 Tilecoding Results Overview

Results from Section 6.1 suggest that those representative exploration methods with neural network approximation perform poorly in general. As a comparison, we evaluate the performance of the same agents with tilecoding function approximation. For each agent in any environment, the best hyperparameter setting is determined by the cumulative reward over the final 10% of the given learning budget. Similarly we show figures with different groupings: the performance across all domains is shown in Figure 6.5, and bar plots grouped by agent are shown in Figure 6.6. Only the best hyperparameter settings are shown in these three figures. In Figure 6.1, the performance of each agent is the average performance across all environments.

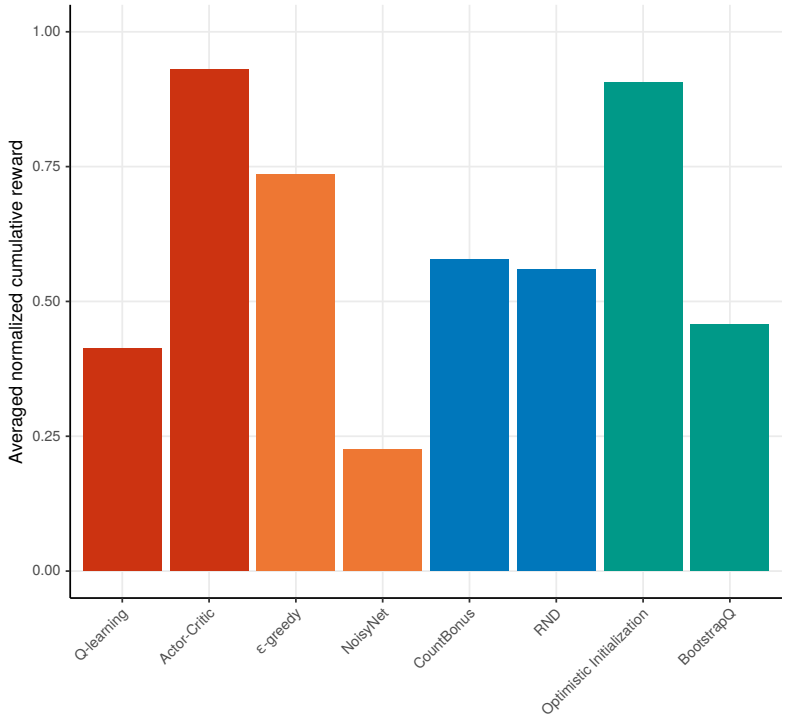

Figure 6.5: Median normalized cumulative reward *averaged across all environments* over final 10% learning budget for Q-learning agents with tilecoded features. Performance of the same agents with neural networks is shown in Figure 6.1.

Comparing to agents equipped with neural networks, Figure 6.5 suggests that with tilecoded features all methods other than NoisyNet outperform their

nonlinear selves by a significant margin. NoisyNet is also the only agent that is not able to improve upon the base linear Q-learning agent. This phenomenon could be attributed to the noisy linear layer, which is originally proposed for neural networks. The idea is basically random perturbations in weights could potentially induce highly complex changes in the Q-values, hence it drives complex exploratory behaviour (Fortunato et al. 2019). In the linear setting, however, perturbations in the weights end up being random noise in the value estimates.

In general the representative methods with tilecoding are able to learn much stronger policies than themselves with neural networks, even though there are still only two methods outperforming the classic ϵ-greedy exploration. The results also suggest that BootstrapQ, which has been failed in most environments with neural networks, is now able to marginally outperform the Q-learning baseline. The performance gap between OptimisticInit and BootstrapQ still persists with linear feature representation, hence only retroactive methods out of all categories have similar performance. Actor-Critic and OptimisticInit achieve the highest cumulative reward overall.

This comparison between linear and nonlinear agents highlights the significance of good feature representations. Tilecoding constructs sparse features by thoroughly discretizing the state space. Good tile & tiling settings can build up representations with strong generalizability and discriminability, and this feature mapping remains fixed since it is constructed. On the other hand, weights in neural networks are commonly initialized randomly. Despite its potential brought by higher model complexity, an agent with neural network approximation must learn a good representation while exploring the environment. This inherent nature might be a big disadvantage for nonlinear agents, especially at the early learning stage.

The comparison of Figure 6.2 and Figure 6.6 illustrates that Actor-Critic is the only agent that outperforms itself with neural networks in all domains. OptimisticInit almost accomplishes this cross-domain improvement except for a minimal drop in VarianceWorld. OptimisticInit learns a stronger (yet still

weak) suboptimal policy in Hypercube, which suggests that the initial optimism cannot be retained for long, especially when the agent is guided to explore a large state-action space. Hypercube's shaping rewards essentially break the domain into smaller checkpointed segments that can be solved individually with ϵ-greedy's local search.

Most agents with tilecoding attain high median cumulative reward in more than half of the environments. Interestingly CountBonus cannot manage to make any improvement with tilecoding in SparseMC while its base Q-learning learns a much stronger policy. Intrinsic rewards now harm the performance rather than incentivizing exploration. The other bonus-based method RND, however, does not damage Q-learning's policy. OptimisticInit and ϵ-greedy are the only two methods to consistently improve upon the linear Q-learning baseline in every environment. RND and BootstrapQ are slightly worse than Q-learning in WindyJump.

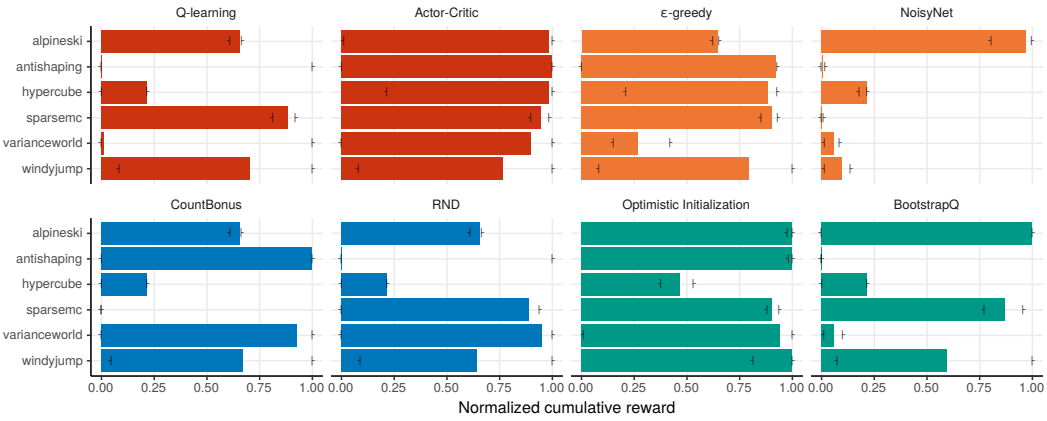

Figure 6.6: Median normalized cumulative reward over final 10% learning budget for Q-learning agents with tilecoded features. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Performance of the same agents with neural networks is shown in Figure 6.2. Performance of the same agents with tilecoding in one single hyperparameter setting is shown in Figure 6.10.

Furthermore, we present the median cumulative reward of two linear methods, IEQL+ (White et al. 2010) and UCLS (Kumaraswamy et al. 2018), in Figure 6.7. IEQL+ and UCLS are methods that include explicit optimistic initial values. These plots provide us with a demonstration that all our toy

environments are solvable: other than UCLS being suboptimal in Hypercube, both algorithms are able to achieve near-optimal or good sub-optimal policies in all domains.

In addition, we conduct experiments on an OptimisticInit variant that is augmented by count-bonus intrinsic rewards. The results are shown in Figure 6.8. This agent serves as a nonlinear analogue to IEQL+. The extra exploration bonus should be able to help create better value estimates after the initialized optimism decays. Interestingly, this variant achieves a very similar performance as the standalone OptimisticInit agent. Only small improvements have been made in Hypercube and SparseMC.

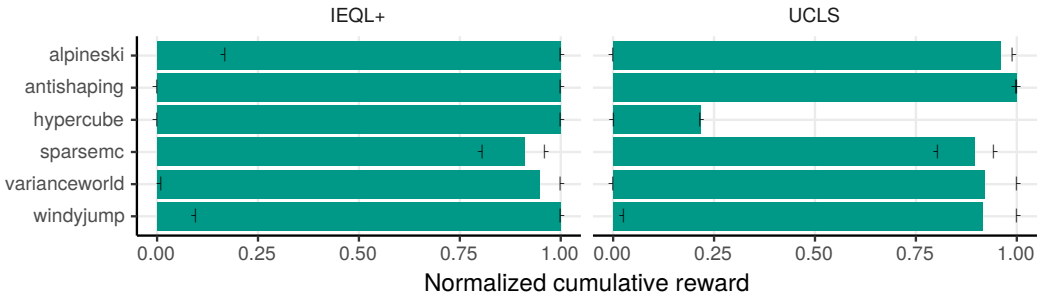

Figure 6.7: Median normalized cumulative reward over final 10% learning budget for IEQL+ and UCLS with tilecoded features. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Performance of the same agents with tilecoding in one single hyperparameter setting is shown in Figure 6.10.

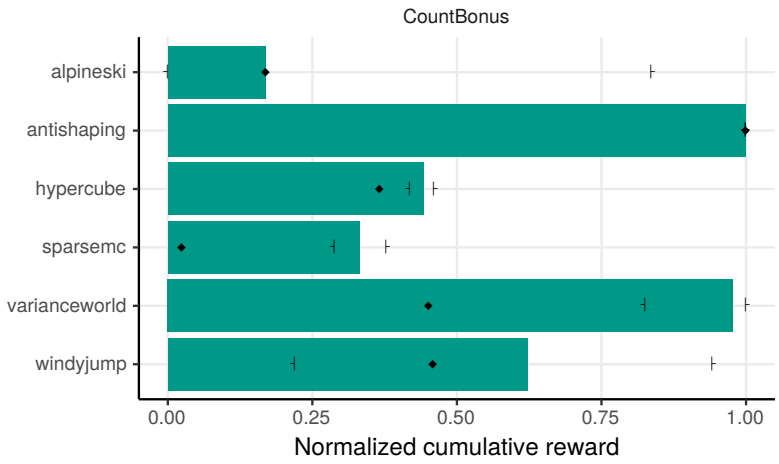

Figure 6.8: Median normalized cumulative reward over final 10% learning budget for OptimisticInit augmented by CountBonus with neural networks. This variant serves as the nonlinear analogue to IEQL+. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Each black solid diamond denotes the CountBonus agent's best performance, which corresponds to the bar plot in Figure 6.2. Performance of IEQL+ with tilecoding is shown in Figure 6.7.

6.3 Meta Hyperparameter Tuning Results

In this section we evaluate the robustness of exploration methods to hyperparameter choices. Specifically, for each agent we pick the hyperparameter setting with the highest averaged performance across all domains, and report its performance in this best single hyperparameter setting. The performance for agents with neural networks and tilecoded features is shown in Figure 6.9 and Figure 6.10, respectively.

Most agents do not suffer from drastic performance drop with one single hyperparameter setting. In fact NoisyNet achieves its best performance in every environment with one hyperparameter, which is consistent with conclusions in the original paper that random perturbations promote robust exploration with less hyperparameter tuning thereby less computational cost. Only undirected exploration methods reach a good policy in SparseMC. Moreover, NoisyNet is also the only agent that outperforms or matches Q-learning in every domain. Actor-Critic remains strong in all environments except SparseMC.

One single hyperparameter choice further exacerbates BootstrapQ's performance.

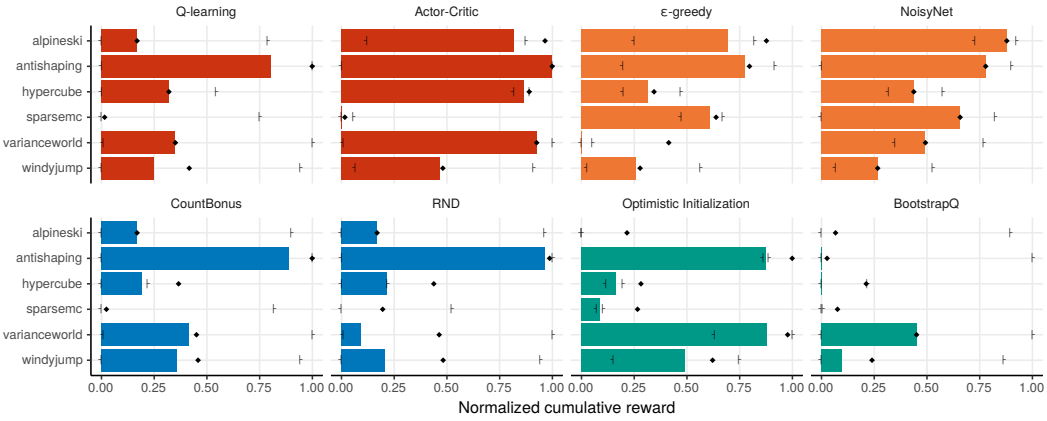

Figure 6.9: Median normalized cumulative reward over final 10% learning budget for Q-learning agents with neural networks *in one single hyperparameter setting*. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Each black solid diamond denotes the agent's performance with its independent best hyperparameter setting in one specific environment, which corresponds to the bar plot in Figure 6.2.

Hyperparameter choices have a notably more significant impact on linear agents. All agents have severely declined performance in almost all environments. The base Q-learning agent could not find a good policy in any environment, especially in SparseMC Q-learning fails to accumulate any rewards. As for those previous strongest linear agents Actor-Critic, OptimisticInit, IEQL+, and UCLS, each of them could merely obtain a high median cumulative reward in at most three environments. ϵ-greedy and OptimisticInit manage to accumulate non-zero rewards across all domains. Furthermore, Actor-Critic, OptimisticInit, and BootstrapQ maintain their performance in SparseMC on a good level.

This section is intended to demonstrate representative exploration methods' capability to robustly performing well across all environments of our suite with one hyperparameter setting. Comparing to previous Section 6.1 and Section 6.2 where the best hyperparameter setting is found independently in each environment, agents in this section have declined performance to

varying degrees. In general agents with neural network approximations are more robust to hyperparameter choices than themselves with tilecoded representation. Undirected exploration approaches benefit from different forms of stochastic perturbations and therefore they are strongly robust to hyperparameter choices.

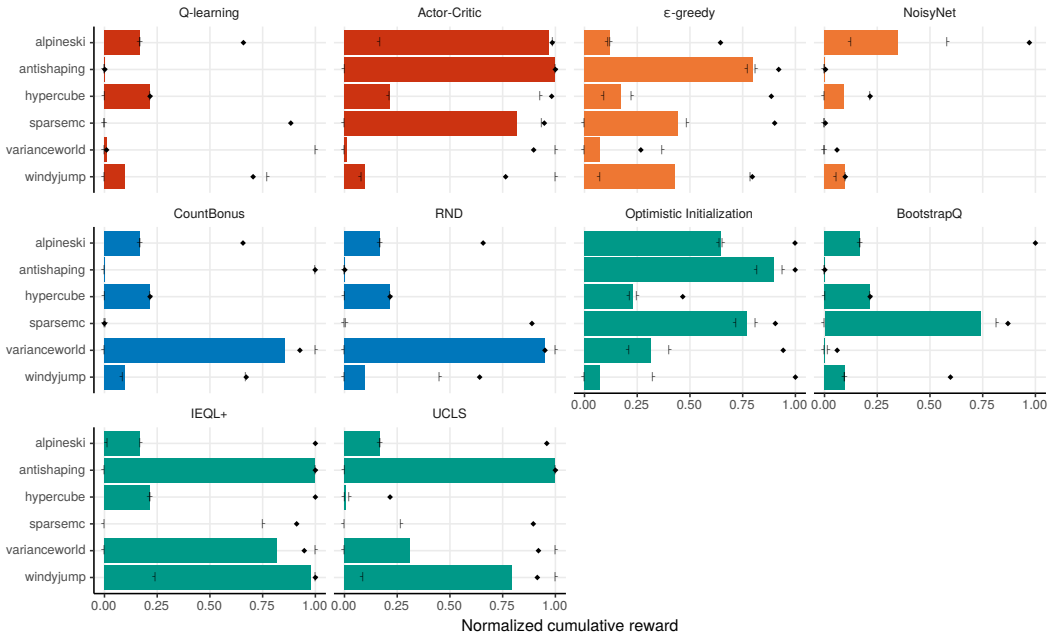

Figure 6.10: Median normalized cumulative reward over final 10% learning budget for Q-learning agents with tilecoded features *in one single hyperparameter setting*. Bars are grouped by *agent*. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Each black solid diamond denotes the agent's performance with its independent best hyperparameter setting in one specific environment, which corresponds to the bar plot in Figure 6.6 and Figure 6.7.

6.4 Deep Dive into BoostrappedDQN

Our results from Section 6.1 suggest that BootstrapQ's performance is not as good as one might expect. It has the worst performance out of all selected algorithms including the Q-learning baseline. In particular, ϵ-greedy outperforms BoostrapQ in almost all domains. Given that BootstrapQ has been successful in much more complicated Atari games, there should be much room

for improvement in our domains. In this section we investigate a few variants applied to the original implementation individually, hoping to achieve a better performance.

We first consider two modifications inspired by the original paper that proposes BootstrapQ (Osband, Blundell, et al. 2016). One is concerned with the value function sampling frequency. In the vanilla implementation, the value function is sampled at the beginning of each episode, and the agent's behaviour is guided by this sampled value until the episode terminates. This setting brings up a potential issue: if this specific sample is not sufficiently exploratory, that is, the agent easily gets stuck in a region of the state spaces and cannot escape until the learning budget is exceeded, then BootstrapQ could only accumulate minimal reward or no reward, like in SparseMC. One idea is to set a time-out mechanism in the agent. If the agent fails to complete an episode after acting in the environment for a certain number of timesteps T_{sample}, a flag is triggered, then the agent resamples a value function and follows it from then on. However, the most sensible value for T_{sample} is domain-specific, which correspondingly requires domain-specific knowledge to set it properly.

In our experiment, we use a setting similar to Thompson sampling by setting T_{sample} to domain-independent constants. Two extreme cases are $T_{\text{sample}} = 1$ and $T_{\text{sample}} = \infty$. Note that $T_{\text{sample}} = 1$ is exactly the sampling frequency of Thompson sampling, in which case BootstrapQ is identical to the Thompson DQN ablation in the original paper. We also experiment with an intermediate case $T_{\text{sample}} = 4$, in accordance with the network updating period. Given the aforementioned scenario of insufficiently exploratory value sample, we expect a higher value function sampling frequency will help the agent to jump out of the trap thereby obtaining higher performance. On the other hand, in environments where there is no obvious traps, sampling the value function too frequently will be detrimental to persistent exploration.

The other modification explores the influence of data sharing. In the setting described in the BootstrapQ original paper, at each update step all net-

works are trained with a shared minibatch, and depending on the value of p for the Bernoulli bootstrap masking distribution $\text{Ber}(p)$, the agent determines which network is trained with which data. Under this circumstance, even with $p = 0.5$, it is still highly likely that some networks are trained with many identical transitions. On the other hand, in the implementation of the follow-up work on BootstrapQ+prior, BootstrapQ implements an ensemble buffer that maintains one independent buffer for each network. Therefore, at each update step each network is trained with a minibatch sampled from its own replay memory. Clearly all networks are not trained with shared minibatches. Even with $p = 1$, it is much less likely that the networks are trained with data in common.

We study the effect of data sharing in terms of shared vs. independent minibatch in our experiment. Our implementation of BootstrapQ only has a centralized replay buffer, but we can achieve independent minibatches by resampling a batch for each network update. Results from Osband, Blundell, et al. (2016) show that BootstrapQ with diverse levels of data sharing in terms of different values for probability p performed similarly. We expect a similar phenomenon in our experiments.

Moreover, we consider a variant of BootstrapQ in which randomized prior functions are replaced by an alternative of 'prior' mechanism (Osband, Aslanides, et al. 2018). Inspired by our earlier results (see Section 6.1), we use OptimisticInit as the prior effect to incentivize exploration. Despite their good performance in this domain, undirected methods are not selected since their exploratory behaviour is due to random perturbations rather than some intrinsic motivation. In our BootstrapQ with OptimisticInit variant, each network is given an initial optimism according to our OptimisticInit implementation. The maximum optimistic initial value is a hyperparameter swept over $H_{\text{maxOptInit}} = \{1\} \cup \{5x \mid x \in \{1, \ldots, 20\}\}$. For each hyperparameter setting $i \in H_{\text{maxOptInit}}$. the optimistic value for each network is randomly sampled from $\{\tilde{i} \in H_{\text{maxOptInit}} \mid \tilde{i} \leq i\}$. Note that each network is initialized with different initial random weights and data samples, hence networks that happen to

sample the same optimistic initial value still produce diverse generalization with smaller variance across them.

To summarize the experimental setup in this section, we consider three variants of BootstrapQ (shared minibatch, independent minibatch, and OptimisticInit prior) as well as three value function sampling frequency (two extreme settings in which value is sampled every timestep and every episode respectively, and one intermediate setting in which value is sampled every 4 timesteps), resulting in 9 combinations. The results are shown in Figure 6.11.

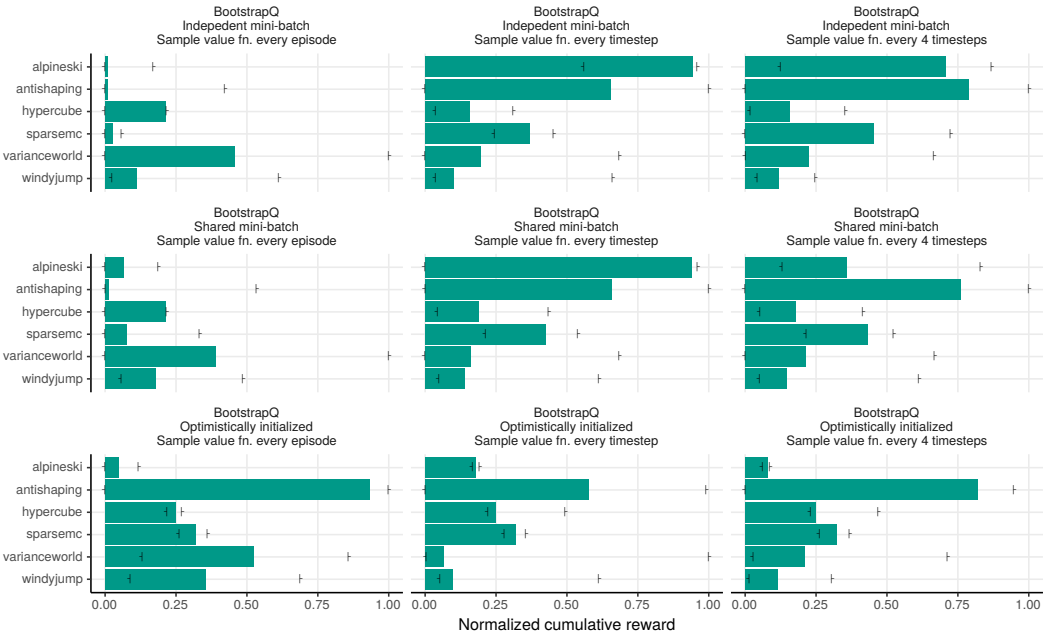

Figure 6.11: Median normalized cumulative reward over final 10% learning budget for BootstrapQ variants with neural networks. One variant sweeps over different value sampling frequencies and independent *vs*. shared minibatches, and the other variant has optimistically initialized value functions. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. The BoostrapQ variant that shares minibatches and samples value function every episode is the vanilla implementation shown in Figure 6.2.

The BootstrapQ with prior variant that shares minibatches across networks and samples the value function every episode is the vanilla implementation we previously present in Section 6.1. Its performance also conforms to earlier results although experiments in this section are carried out with ran-

dom seeds different from seeds used earlier (see Figure 6.2). For each BoostrapQ variant, sampling value functions every timestep results in a substantial difference in performance comparing to sampling every episode, while the performance of resampling every four timesteps and every one timestep is similar.

In particular, the Thompson sampling variant surpasses the vanilla BootstrapQ drastically in AlpineSki, Antishaping, and SparseMC, while it also performs more poorly in VarianceWorld. In Hypercube and WindyJump, Thompson sampling receives a relatively lower median cumulative reward, yet the best runs in both environments are better. The results in general coincide with our previous hypothesis that more frequent value function sampling is beneficial in environments with traps from which the agent must escape.

Note that one conclusion in Osband, Blundell, et al. (2016) suggests that that only BoostrapQ with episode-wise sampling drives efficient exploration whereas Thompson DQN does not. Our findings do not fully contradict it, since the toy environment DeepSea they experimented on is inherently different from our suite. DeepSea is a deterministic chain environment with a fixed episode length and finite discrete state space. The value function is thus sampled periodically, and the agent could not get trapped in a pitfall region forever like it tends to do in some of our environments.

However, the part of our results that does not align with their conclusion lies in this comparison: the variant that samples value function every four timesteps is also capable of jumping out of pitfalls, plus it conducts more temporally extended exploration than Thompson sampling does, but it achieves very similar performance across almost all environments. One possible hypothesis is that the distinction between sampling every timestep and resampling every four timesteps is insignificant, especially in relatively larger environments like Antishaping and SparseMC. In AlpineSki, on the other hand, after the prior guides the agent to find the most rewarding state, jumping out of the pitfall sooner becomes much more important than continuing a temporally extended sequence of actions, hence we observe a better perfor-

mance from the sampling per timestep variant. More research is needed to study how value function sampling frequency, either domain-dependent or not, affects exploration.

It is also evident that data sharing does not play an equivalently crucial role as value function sampling frequency. Performance of shared minibatch is almost identical to that of independent minibatch. Although in AlpineSki BootstrapQ with independent minibatches outperforms the other variant by a big margin, their max-min ranges are comparable. The results in general conform our expectation mentioned previously.

As for the BootstrapQ with OptimisticInit variant, it achieves the best performance with episode-wise value sampling. In contrast to the BootstrapQ with prior agent, this variant has worse performance with more frequent value function sampling. When the agent samples the value function only at the beginning of each episode, due to the large horizon of each episode, the optimistic initialization in each sampled value network dominates the variance of the ensemble distribution so that it essentially determines the exploratory behaviour. For this reason, its performance is similar to the OptimisticInit agent alone (see Figure 6.2).

Results from more frequent value sampling show both similar and different patterns found in the BootstrapQ with prior variant. As the value function is sampled more often, the performance in VarianceWorld and WindyJump decreases, which demonstrates that both environments require more temporally extended exploration. The prior variant establishes the benefit from more frequent resampling in AlpineSki and SparseMC, but this case does not fully apply to the OptimisticInit variant. The OptimisticInit variant obtains minor improvement in AlpineSki, while its performance in SparseMC remains the same. The reason could be that given OptimisticInit itself can only play suboptimally, the small variance across all networks fails to provide the agent with sufficient diversity to escape the valley region, no matter how often the agent switches its value function.

As resampling occurs more frequently, all variants are only able to ob-

tain an improvement in the best runs in Hypercube, while the median performance stays roughly the same. Optimistic initialization usually explores the entire state space to let the inflated action value to fade away. Since the OptimisticInit agent in Section 6.1, as well as the BootstrapQ with OptimisticInit variant, are only able to achieve suboptimal policies in Hypercube and SparseMC, it is valid to hypothesize that our implementation of OptimisticInit loses its initial optimism too quickly.

A key property of OptimisticInit is that its optimistically initialized values decrease only at the state-action pairs used for updates. In the linear setting it is easy to index state-action pairs and update values accordingly. In the nonlinear setting, however, values of unseen state-action pairs will also be changed after training with seen transitions due to the generalization of neural networks. Moreover, the changed values could become unpredictably pessimistic, neutral, or even potentially more optimistic. The value estimate of a highly rewarding state-action pair may drop very quickly after updating net weights with seen transitions, thus our OptimisticInit agent could totally ignore this particular state-action.

A similar argument also applies to our BootstrapQ with OptimisticInit variant: all optimistic nets will correct their estimates at seen state-action pairs, while those estimates in unseen state-action space could drop substantially as well. Due to different initialization and pre-training data as mentioned above, those networks could have diverse value estimates in unseen state-action space, but it is much less likely any of them remains optimistic, which eventually leads to ineffective exploration. Exploration motivated by an additive prior as an analogue to OptimisticInit will be examined in Section 6.5.

6.5 More Lessons Learned

In this section, we present additional empirical insights trying to improve the performance of deep RL methods. We also investigate exploration motivated by prior functions as a form of optimistic initialization for deep RL agents.

We hope these experiments will help researchers to examine the drawbacks and potentials of existing approaches, hence improving the performance of nonlinear agents.

6.5.1 Neural Network with Tilcoded Inputs

Neural network Sarsa(0) with tile-coded inputs has exhibited better performance in the classic MountainCar environment (Ghiassian et al. 2020). Motivated by this work we empirically investigate this representation with the Sparse MountainCar task, the task where the majority of representative methods have poor performance. In previous experiments with nonlinear approximation, we input the continuous states (normalized by the range of state space) directly to the neural network. Tile coding obtains features through discretization of the state space in a smart way. We expect neural networks to benefit from estimating action values for simpler, discrete inputs. The results are shown in Figure 6.12.

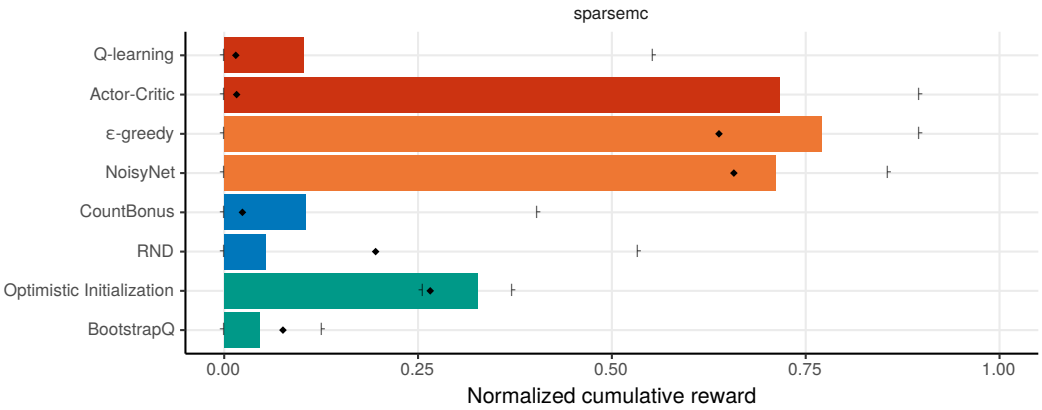

Figure 6.12: Median normalized cumulative reward over final 10% learning budget for Q-learning agents in Sparse MountainCar with neural networks. The neural networks take tile-coded features of states as input. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Each black solid diamond denotes the agent's performance with its independent best hyperparameter setting and continuous state input to the neural network, which corresponds to the bar plot in Figure 6.3.

Most agents obtain an improvement in their final performance, especially Actor-Critic, which has improved from virtually zero median cumulative re-

wards to around three quarters of the near-optimal performance. The results are in general consistent with our expectation.

On the other hand, two agents RND and BootstrapQ suffer from performance drop. This phenomenon could be interpreted from the perspective of representation. While it makes the inputs simpler, this tilecoded net architecture also induces less complicated value estimates. As a result, the ensemble distribution becomes less diverse, hence the less effective uncertainty estimation leads to less effective exploration. A similar argument also applies to RND. Simpler inputs cause the predictor network and the target network to produce small prediction errors even in unseen regions of the state-action space, which creates less effective uncertainty estimation.

Although these results demonstrate the high potential of smaller input space, one thing to be aware of is that tilecoding is not particularly extendable to image inputs. As the capability of constructing good representations of highly complex state space is critical, both in game playing and real-world applications, more research is needed for reliable state abstraction methods.

6.5.2 Randomized Prior as Optimism

Furthermore, in order to evaluate the impact of the randomized prior function on exploration alone, we experimented with the extreme case where the size of the value function ensemble is 1. This additive prior applied to one value function alone plays the role of optimistic initializations. Common neural network initialization schemes initialize layer weights and biases with zero-mean symmetric distributions (common choices are Gaussian distribution and uniform distribution). Due to the non-negative activation ReLU and the prior scaling coefficient, the Q-value estimate as a posterior is initializaed to be optimistic in expectation, i.e., $\mathbb{E}[\max_a \hat{q}_{\mathbf{w}}(s,a)] = \mathbb{E}[\max_a (\hat{f}_{\mathbf{w}} + \beta p)(s,a)] > 0$ at states s, where $\tilde{q}_{\mathbf{w}}$ is the original Q-value estimate without a prior, β is the prior scale, and p is the prior function.

By controlling the prior scale this initial optimism could be arbitrarily high. As the agent is guided to explore the environment and trained with seen

transitions, the Q-value estimate at those state-action pairs will be corrected, that is, $\tilde{q}_{\mathbf{w}}$ will become pessimistic to counter the positive prior at those state-action pairs. In addition, it should also encounter the same issue due to the generalization of neural networks as our OptimisticInit implementation for the same reason.

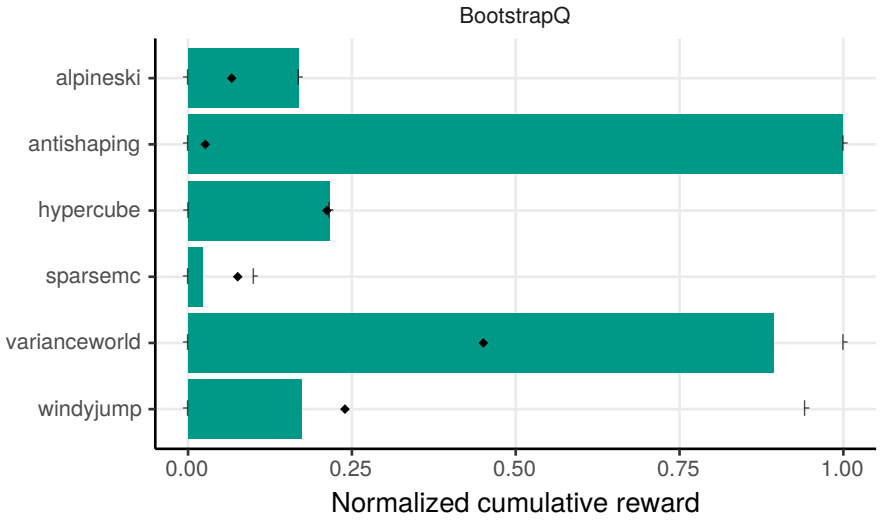

Figure 6.13: Median normalized cumulative reward over final 10% learning budget for BootstrapQ with neural networks. The size of the ensemble is 1. The box drawings "┤" and "├" denote the worst and best runs in each hyperparameter setting, respectively. Each black solid diamond denotes the agent's performance with its independent best hyperparameter setting and ensemble with size $K > 1$, which corresponds to the bar plot in Figure 6.2.

The results are shown in Figure 6.13. In general this variant outperforms the vanilla BootstrapQ with ensemble size $K > 1$. It achieves the most improvement in Antishaping and VarianceWorld. In fact its performance in many environments is aligned with our OptimisticInit (see Figure 6.2). The performance in Hypercube remains at the same level, which can be explained by the quickly diminishing optimism as described in Section 6.4. Surprisingly its performance in SparseMC and WindyJump is worse. It is likely that some initial Q-value estimates are not optimistic, since our discussions above only cover the case that the initialization is optimistic *in expectation*. Figure 6.14 presents the initial maximum Q-value estimates for all runs in WindyJump.

It shows that in some runs the initial value estimates are neutral or even pessimistic, in spite of the optimism shown in most runs.

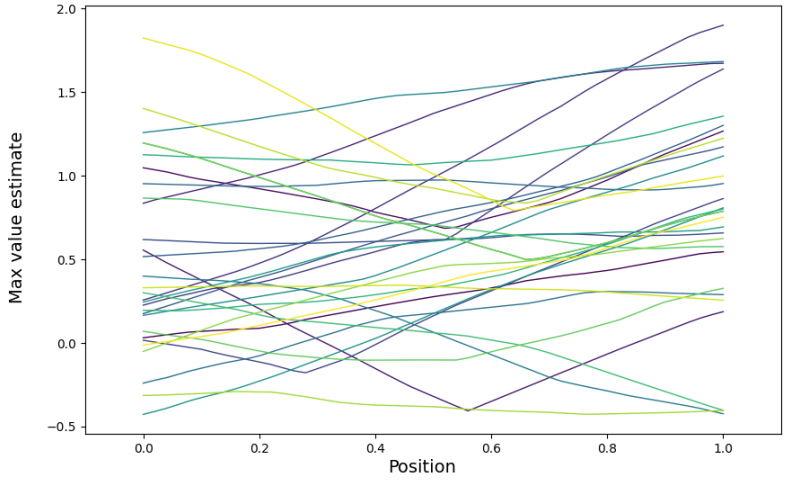

Figure 6.14: Initial maximum action value estimates $\max_a \hat{q}_{\mathbf{w}}(s, a)$ for BootstrapQ with neural networks across all states s in WindyJump. The prior scale is 1. The size of ensemble is 1. Each line segment corresponds to one random seed.

Both the additive prior function and our OptimisticInit implementation are hindered by the issue caused by the generalization of neural networks. Rashid et al. (2020) avoids this issue by augmenting a count-bonus as an upper confidence bound to the Q-value estimates $\hat{q}^+_{\mathbf{w}}(s, a) = \hat{q}_{\mathbf{w}}(s, a) + \frac{C}{(N(s,a)+1)^M}$, here $N(s, a)$ is the state-action visitation count, and C and M are hyperparameters. This $\hat{q}^+_{\mathbf{w}}$ is used at both action selection and bootstrapping. During exploration the optimism is captured through the count-bonus as an uncertainty estimate, hence generalization in the Q-value network does not damage optimism in unseen state-action pairs. Ideally any uncertainty estimation could serve as the optimism at action selection, and one interesting future direction is to experiment on various choices.

Chapter 7

Conclusion

In this thesis, we conduct a large empirical study designed to tease apart the key characteristics of model-free exploration methods in deep reinforcement learning, as well as to isolate and test the characteristics of exploration problems that make them difficult. We identify challenges posed by properties of the reward function (high-variance, sparsity, and misguiding) and the transition dynamics (high-variance, large state-action space, and adversarial) of the environment, and introduce a new set of benchmark exploration problems, each of which embodies one exploration property. Our suite of tasks is designed to be as small as possible to permit extensive experimentation and ensure the domains can be easily analyzed and understood.

We introduce a categorization of model-free exploration methods based on their underlying exploration heuristics. We firstly identify the undirected exploration methods, methods that inject stochastic perturbations to the parameters or directly to the policy so that exploratory actions are induced. We then identify the bonus-based methods, which augment the extrinsic reward from the environment by an intrinsic reward based on some form of novelty or uncertainty estimation so that the agent could have more accurate value estimates. Finally we introduce a group of methods that directly estimate uncertainty on state-action values. These methods estimate value uncertainty and use it to guide its exploration behaviour.

We conduct a systematic empirical study of representative model-free exploration methods on the aforementioned suite of toy tasks. Our study results

in several practical insights. No single exploration method performs well in every environment, highlighting the utility of our task suite. Performance is usually grouped according to methods that use a reward bonus, take actions according to inflated values, or use randomness to explore. There is some inconsistency across related methods. For example, some upper confidence estimation methods achieve high robustness across representations, while others achieve the best performance with one representation, but poor with others.

Our results also demonstrate the form of the representation has a large impact on performance. Almost all methods perform significantly worse with dense neural network representations, compared with sparse, high-dimensional tilecoded features. In particular, recent deep RL exploration methods perform particularly poorly with tile coding. Furthermore, the choice of hyperparameter settings also plays a significant role in affecting performance. Agents with neural network approximations perform much more robustly than themselves with tilecoding. Finally we take a deeper dive into methods that directly estimate value uncertainty. We propose some modifications resulting in multiple variants of the same algorithm, and conduct experiments to gain empirical insights that could potentially lead to more effective exploration methods.

7.1 Future Work

There are interesting future research directions beyond the scope of this thesis. Methods that directly estimate value uncertainty is more investigated in the linear setting. Although there are practical concerns about the generalization of neural networks, those nonlinear agents have shown great potential for high performance across exploration domains. For instance, Optimistic Initialization is critical in linear settings. Our proposed nonlinear analogue has severe drawbacks but it still managed to achieve promising performance. It is an interesting research direction to alleviate the generalization of neural

networks, or bypassing this issue with different forms of uncertainty estimation.

Another future direction could focus on the non-stationary online setting. A realistic scenario in which the learning/evaluation separation is not applicable is when the environment is non-stationary. In such a case the RL agent is expected to continuously improve the performance and adapt to changes in environments. A sensible evaluation metric for a non-stationary setting thus needs to consider both the cumulative reward and how quickly the agent adapts to environmental shifts. Each of our tasks exhibits a unique exploration challenge, hence it is interesting to extend this work by applying a different form of non-stationarity to each individual environment.

One other future research direction is to empirically study exploration methods in environments with combined exploration challenges. Each of our tasks embodies one single exploration challenge. Although some properties are inherently not very compatible, such as sparse rewards and misleading rewards, the six exploration properties could result in interesting combinations that are worth further investigations.

References

Achiam, Joshua and Shankar Sastry (Mar. 2017). "Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning." In: *Deep Reinforcement Learning Workshop*. 26

Afsar, M. Mehdi, Trafford Crump, and Behrouz Far (Jan. 2021). "Reinforcement Learning Based Recommender Systems: A Survey." In: *arXiv:2101.06286 [cs]*. 2

Antos, András, Varun Grover, and Csaba Szepesvári (2008). "Active Learning in Multi-Armed Bandits." In: *Algorithmic Learning Theory*, pp. 287–302. 26

Azizzadenesheli, Kamyar and Animashree Anandkumar (Sept. 2019). "Efficient Exploration through Bayesian Deep Q-Networks." In: *arXiv:1802.04412 [cs, stat]*. 28, 29

Barto, Andrew G and Satinder P Singh (Jan. 1991). "On the Computational Economics of Reinforcement Learning." In: *Connectionist Models*, pp. 35–44. 24

Bellemare, Marc G., Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos (Nov. 2016). "Unifying Count-Based Exploration and Intrinsic Motivation." In: *arXiv:1606.01868 [cs, stat]*. 2, 3, 25

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov (Oct. 2018). "Exploration by Random Network Distillation." In: *arXiv:1810.12894 [cs, stat]*. 5, 26, 80

Chen, Richard Y., Szymon Sidor, Pieter Abbeel, and John Schulman (Nov. 2017). "UCB Exploration via Q-Ensembles." In: *arXiv:1706.01502 [cs, stat]*. 28

Chowdhary, G., Miao Liu, T. Walsh, and J. How (2014). "Off-Policy Reinforcement Learning with Gaussian Processes Citation." In: *Acta Automatica Sinica*. 29

Dearden, Richard, Nir Friedman, and Stuart Russell (1998). "Bayesian Q-Learning." In: *AAAI Conference on Artificial Intelligence*, p. 8. 28, 29

Duff, Michael O'Gordon (Jan. 2002). "Optimal Learning: Computational Procedures for Bayes -Adaptive Markov Decision Processes." PhD thesis. University of Massachusetts Amherst. 22

Duncan, Luce R (1959). *Individual Choice Behavior: A Theoretical Analysis*. 31

Engel, Yaakov, Shie Mannor, and Ron Meir (2005). "Reinforcement Learning with Gaussian Processes." In: *Proceedings of the 22nd International Conference on Machine Learning - ICML '05*, pp. 201–208. 28, 29

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu (June 2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." In: *arXiv:1802.01561 [cs]*. 2

Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg (July 2019). "Noisy Networks for Exploration." In: *arXiv:1706.10295 [cs, stat]*. 2, 5, 32, 45, 87

Gal, Yarin and Zoubin Ghahramani (June 2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." In: *International Conference on Machine Learning*, pp. 1050–1059. 28

Ghiassian, Sina, Banafsheh Rafiee, Yat Long Lo, and Adam White (Mar. 2020). "Improving Performance in Reinforcement Learning by Breaking Generalization in Neural Networks." In: *arXiv:2003.07417 [cs]*. 57

Grande, Robert C, Thomas J Walsh, and Jonathan P How (2014). "Sample Efficient Reinforcement Learning with Gaussian Processes." In: *International Conference on Machine Learning*, p. 9. 28–30

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (Feb. 2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *arXiv:1502.01852 [cs]*. 36

Hester, Todd and Peter Stone (June 2017). "Intrinsically Motivated Model Learning for Developing Curious Robots." In: *Artificial Intelligence* 247, pp. 170–186. 25

Houthooft, Rein, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "VIME: Variational Information Maximizing Exploration." In: *Advances in Neural Information Processing Systems*. 26

Itti, Laurent and Pierre Baldi (Dec. 2005). "Bayesian Surprise Attracts Human Attention." In: *Advances in Neural Information Processing Systems*, pp. 547–554. 26

Jin, Chi, Zhuoran Yang, Zhaoran Wang, and Michael I. Jordan (Aug. 2019). "Provably Efficient Reinforcement Learning with Linear Function Approximation." In: *arXiv:1907.05388 [cs, math, stat]*. 28

Kaelbling, Leslie Pack (1993). *Learning in Embedded Systems*. 24

Kingma, Diederik P. and Jimmy Ba (Jan. 2017). "Adam: A Method for Stochastic Optimization." In: *arXiv:1412.6980 [cs]*. 36, 86

Kober, Jens, J. Andrew Bagnell, and Jan Peters (Sept. 2013). "Reinforcement Learning in Robotics: A Survey." In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274. 2

Kumaraswamy, Raksha, Matthew Schlegel, Adam White, and Martha White (2018). "Context-Dependent Upper-Confidence Bounds for Directed Exploration." In: *Advances in Neural Information Processing Systems* 31. 28, 30, 46, 81

Lai, T. L and Herbert Robbins (Mar. 1985). "Asymptotically Efficient Adaptive Allocation Rules." In: *Advances in Applied Mathematics* 6.1, pp. 4–22. 23

Langford, John (2018). *RL Acid*. 16, 19, 71

Lattimore, Tor and Csaba Szepesvári (2020). *Bandit Algorithms*. 25
Linke, Cameron (2021). *Adapting Behaviour via Intrinsic Reward*. https://era.library.ualberta.ca/items/3a8b07d6-bab6-4916-8cd1-2485e84309a0. 26
Little, Daniel Y. and Friedrich T. Sommer (Mar. 2013). "Learning and Exploration in Action-Perception Loops." In: *Frontiers in Neural Circuits* 7, p. 37. 26
Machado, Marlos C., Marc G. Bellemare, and Michael Bowling (Nov. 2019). "Count-Based Exploration with the Successor Representation." In: *arXiv:1807.11622 [cs, stat]*. 25
Machado, Marlos C., Sriram Srinivasan, and Michael Bowling (Oct. 2014). "Domain-Independent Optimistic Initialization for Reinforcement Learning." In: *arXiv:1410.4604 [cs]*. 30
Martin, J. J. (1967). *Bayesian Decision Problems and Markov Chains*. 22
Martin, Jarryd, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter (June 2017). "Count-Based Exploration in Feature Space for Reinforcement Learning." In: *arXiv:1706.08090 [cs]*. 25, 80
Melo, Francisco S (2001). *Convergence of Q-Learning: A Simple Proof*. 31
Meuleau, Nicolas and Paul Bourgine (May 1999). "Exploration of Multi-State Environments: Local Measures and Back-Propagation of Uncertainty." In: *Machine Learning* 35.2, pp. 117–154. 24
Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (June 2016). "Asynchronous Methods for Deep Reinforcement Learning." In: *arXiv:1602.01783 [cs]*. 2
Mnih, Volodymyr, 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 (Feb. 2015). "Human-Level Control through Deep Reinforcement Learning." In: *Nature* 518.7540, pp. 529–533. 2, 12
Moore, A. (1990). "Efficient Memory-Based Learning for Robot Control." PhD thesis. 17, 24, 72
O'Donoghue, Brendan, Ian Osband, Rémi Munos, and Volodymyr Mnih (2018). "The Uncertainty Bellman Equation and Exploration." In: *International Conference on Machine Learning*. Vol. abs/1709.05380. 28
Orseau, Laurent, Tor Lattimore, and Marcus Hutter (2013). "Universal Knowledge-Seeking Agents for Stochastic Environments." In: *Algorithmic Learning Theory*, pp. 158–172. 26
Osband, Ian, John Aslanides, and Albin Cassirer (Nov. 2018). "Randomized Prior Functions for Deep Reinforcement Learning." In: *arXiv:1806.03335 [cs, stat]*. 2, 5, 28, 29, 52
Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (July 2016). "Deep Exploration via Bootstrapped DQN." In: *arXiv:1602.04621 [cs, stat]*. 28, 29, 51, 52, 54

Osband, Ian, Benjamin Van Roy, and Zheng Wen (2016). "Generalization and Exploration via Randomized Value Functions." In: *International Conference on Machine Learning*, p. 10. 28

Osband, Ian, Benjamin Van Roy, Daniel Russo, and Zheng Wen (Sept. 2019). "Deep Exploration via Randomized Value Functions." In: *Journal of Machine Learning Research*. 1, 2, 4, 31

Ostrovski, Georg, Marc G. Bellemare, Aaron van den Oord, and Remi Munos (June 2017). "Count-Based Exploration with Neural Density Models." In: *arXiv:1703.01310 [cs]*. 25

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (Dec. 2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *arXiv:1912.01703 [cs, stat]*. 36

Pathak, Deepak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell (May 2017). "Curiosity-Driven Exploration by Self-Supervised Prediction." In: *arXiv:1705.05363 [cs, stat]*. 2, 26

Pathak, Deepak, Dhiraj Gandhi, and Abhinav Gupta (June 2019). "Self-Supervised Exploration via Disagreement." In: *International Conference on Machine Learning*. 27

Plappert, Matthias, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz (Jan. 2018). "Parameter Space Noise for Exploration." In: *arXiv:1706.01905 [cs, stat]*. 32

Rashid, Tabish, Bei Peng, Wendelin Böhmer, and Shimon Whiteson (Feb. 2020). "Optimistic Exploration Even with a Pessimistic Initialisation." In: *arXiv:2002.12174 [cs, stat]*. 30, 60

Russo, Daniel (2019). *Algorithmic Foundations of Learning and Control*. 20

Russo, Daniel and Benjamin Van Roy (Feb. 2014). "Learning to Optimize Via Posterior Sampling." In: *arXiv:1301.2609 [cs]*. 23

Saxe, Andrew M., James L. McClelland, and Surya Ganguli (Feb. 2014). "Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks." In: *arXiv:1312.6120 [cond-mat, q-bio, stat]*. 88

Schmidhuber, Jürgen (1991). *Adaptive Confidence And Adaptive Curiosity*. Tech. rep. Institut fur Informatik, Technische Universitat Munchen, Arcisstr. 21, 800 Munchen 2. 24

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel (Apr. 2017). "Trust Region Policy Optimization." In: *arXiv:1502.05477 [cs]*. 2

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis (Dec.

2017). “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.” In: *arXiv:1712.01815 [cs]*. 2

Şimşek, Özgür and Andrew G. Barto (June 2006). “An Intrinsic Reward Mechanism for Efficient Exploration.” In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 833–840. 22

Stadie, Bradly C., Sergey Levine, and Pieter Abbeel (Nov. 2015). “Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models.” In: *arXiv:1507.00814 [cs, stat]*. 26

Still, Susanne and Doina Precup (Sept. 2012). “An Information-Theoretic Approach to Curiosity-Driven Reinforcement Learning.” In: *Theory in Biosciences* 131.3, pp. 139–148. 26

Storck, Jan and S. Hochreiter (1995). “Reinforcement Drive Information Aquisition in Nondeterministic Evnvironments.” In: *International Conference on Artificial Neural Networks*. 26

Strehl, Alexander L. and Michael L. Littman (Dec. 2008). “An Analysis of Model-Based Interval Estimation for Markov Decision Processes.” In: *Journal of Computer and System Sciences* 74.8, pp. 1309–1331. 24

Sutton, R. (1990). “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming.” In: *ML*. 24

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement Learning: An Introduction*. 1, 7, 10, 12, 72

Taiga, Adrien Ali, William Fedus, Marlos C. Machado, Aaron Courville, and Marc G. Bellemare (Sept. 2019). “On Bonus Based Exploration Methods In The Arcade Learning Environment.” In: *International Conference on Learning Representations*. 3, 40

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (Dec. 2017). “#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning.” In: *arXiv:1611.04717 [cs]*. 25, 80

Team, R Core (2013). *R: A Language and Environment for Statistical Computing*. 38

Thompson, William R. (1933). “On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples.” In: *Biometrika* 25.3/4, pp. 285–294. 23

Thrun, Sebastian (1992). “The Role of Exploration in Learning Contro.” In: *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. 24

Thrun, Sebastian and Knut Moller (1991). “Active Exploration in Dynamic Environments.” In: *Advances in Neural Information Processing Systems*. 24

Tsitsiklis, John N (1994). “Asynchronous Stochastic Approximation and Q-Learning.” In: *Machine Learning* 16.3, pp. 185–202. 31

Vinyals, Oriol, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin

Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver (Nov. 2019). "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning." In: *Nature* 575.7782, pp. 350–354. 2

Watkins, Christopher (Jan. 1989). "Learning From Delayed Rewards." PhD thesis. 11

White, Martha and A. White (2010). "Interval Estimation for Reinforcement-Learning Algorithms in Continuous-State Domains." In: *Advances in Neural Information Processing Systems*. 15, 28, 46, 70

Xiao, Zhiqing (2019). *Reinforcement Learning: Theory and Python Implementation*. 72

Yasui, Nikolaus Winget (2020). *An Empirical Study of Exploration Strategies for Model-Free Reinforcement Learning*. `https://era.library.ualberta.ca/items/806a27af-5bf4-466e-a544-57df23f87fce`. 3

Ziebart, Brian D, J Andrew Bagnell, and Anind K Dey (2010). "Modeling Interaction via the Principle of Maximum Causal Entropy." In: *International Conference on Machine Learning*, p. 8. 31

Zintgraf, Luisa, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson (Feb. 2020). "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning." In: *International Conference on Learning Representations*. 22

Appendix A

Environment Hyperparameters

In this appendix, we describe the specific hyperparameters that we chose for each environment. All environments except SparseMC have stochastic transition dynamics to make it difficult for the agent to visit the exact same state multiple times. In SparseMC, a similar effect is achieved by starting in a random location near the center of the valley. While the transitions are stochastic, they are very low variance outside of WindyJump, so we usually do not consider the transition stochasticity to be part of the exploration challenge. Additionally, environments except Antishaping have sparse rewards in some regions of the state space. We only consider SparseMC embodies the challenge of sparse reward, since in other environments the start state is close to the rewarding states.

We also set the hyperparameters in each environment so that the expected value of an episode following the optimal policy is close to 1. When there is an obvious sub-optimal policy that the agent may learn instead, the corresponding value under the sub-optimal policy is close to 0.01 or less.

A.1 Exploration Properties Related to Reward

A.1.1 High-variance Reward: VarianceWorld

VarianceWorld is adopted from the noisy reward navigation environment proposed by White et al. (2010). This environment is a one-dimensional corridor between 0 and 1. The agent starts uniformly randomly in $[0.45, 0.55]$, and

can move left or right according to a $\mathcal{N}(0.05, 0.01^2)$ distribution. Either end of the corridor is 10 steps away from the start point in expectation. Moving past 0 terminates the episode with a reward of $0.01\gamma^{-10} \approx 0.011$, while moving past 1 terminates the episode with a reward of $X\gamma^{-10}$, where X is drawn uniformly randomly from $\{10, -10, 0.5, -0.5, 5\}$. The 'always go left' fixed policy has a value approximately equal to 0.01, while the 'always go right' fixed policy — that is, the near-optimal policy — has a value approximately equal to 1. VarianceWorld is similar to a two-armed bandit problem, with one arm producing fixed rewards while the other arm producing noisy rewards.

A.1.2 Misleading Reward: Antishaping

This environment is adapted from the original Antishaping environment proposed by Langford (2018). The goal of Antishaping is to provide an agent with a local reward structure that does not reflect globally optimal behaviour. An agent that generalizes aggressively may learn that the reward decreases as the agent moves further from the start state, failing to explore sufficiently to confirm its action-value estimates, and failing to identify the optimal policy.

Antishaping is a one-dimensional corridor between 0 and 1. The agent starts at state 0, and can move left or right according to a $\mathcal{N}(0.005, 0.001^2)$ distribution. Reaching the other side of the corridor takes about 200 steps in expectation. The reward is designed to follow a bell curve centered at state 0. As the agent moves to the right, it receives less and less reward until it passes state 1, at which point the episode terminates and it receives a large reward of 7.452. The non-terminal reward is generated by the following function:

$$r(s) = \frac{1}{7264\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{s}{0.15}\right)^2\right),$$

where $s \in [0, 1]$. With a discount factor of 0.99, the optimal policy of always moving right has a value near 1, while the policy that always moves left to stay in the state 0 has a value near 0.0055.

A.1.3 Sparse Reward: Sparse MountainCar

MountainCar (Moore 1990) is a classic continuous environment. The agent is an underpowered car in a valley, and it must use momentum to drive to the top of a hill, where the episode terminates. This environment has a two-dimensional state space, which represents the position and velocity of the car. The agent starts with 0 velocity, with position drawn uniformly randomly from $[-0.6, -0.4]$. The reward is typically -1 per timestep, incentivizing discounted agents to terminate the episode as quickly as possible.

Actions $A_t \in \{-1, 0, 1\}$ correspond to reversing, coasting, and accelerating respectively. Let X denote the position and $\dot{X}$ denote the velocity, the state update rule follows the equations as follows (R. S. Sutton et al. 2018):

$$\dot{X}_{t+1} = \text{bound}\left[\dot{X}_t + 0.001A_t - 0.0025\cos(3X_t)\right]$$
$$X_{t+1} = \text{bound}\left[X_t + \dot{X}_{t+1}\right]$$

The bound operator keeps the position within $[-1.2, 0.5]$ and the velocity within $[-0.07, 0.07]$. If the dynamics update causes the agent's position X_{t+1} to go beyond -1.2, the velocity $\dot{X}_{t+1}$ is set to 0. In our SparseMC implementation, rewards are always 0 unless X_{t+1} is greater than or equal to 0.5, in which case the episode terminates with a reward of 3.34. This reward is chosen so that the near-optimal policy has a value close to 1 in the start state distribution. The optimal policy utilizes the momentum, and it is a deterministic policy (Xiao 2019) described as follows:

$$\pi_*(\cdot \mid X_t) = \begin{cases} 2, & \text{lb}_t \le \dot{X}_t \le \text{ub}_t, \\ 0, & \text{otherwise}, \end{cases}$$

where $\text{lb}_t = \min\left(-0.09(X_t + 0.25)^2 + 0.03, 0.3(X_t + 0.9)^4 - 0.008\right)$ and $\text{ub}_t = -0.07(X_t + 0.38)^2 + 0.07$.

A.2 Exploration Properties Related to Transition Dynamics

A.2.1 High-variance Transitions: WindyJump

This environment follows the familiar $[0, 1]$ corridor model with uniform random start states in $[0.45, 0.55]$. WindyJump has two actions, going left and going right. When the agent is in the left half of the environment with no wind, it moves according to a $\mathcal{N}(0.0116, 0.01^2)$ distribution. In the right half of the environment where there is wind, the agent moves according to a $\mathcal{U}(-0.1, 0.125)$ random variable. Both ends of the corridor are terminal states, which are about 40 steps away from the start state. The terminal reward at state 0 is $0.01\gamma^{-40} \approx 0.015$. The agent's terminal reward on the right end depends linearly on its final position. The agent's goal is to jump over a pit on the right side of the environment to reach a large reward. Writing the final position of the agent as $x > 1$, the agent's terminal reward is given by $r(x) = (281(x-1) - 10)\,\gamma^{-40}$. Non-terminal rewards are 0. With a discount factor of 0.99, the optimal policy of always moving right has a value near 1, while the policy that always moves left has a value near 0.01.

A.2.2 Antagonistic Transitions: AlpineSki

AlpineSki is another $[0, 1]$ corridor, with the start state at 0. There are two actions, *traverse* and *descend*. The *traverse* action moves the agent to the right according to a $\mathcal{N}(0.05, 0.01^2)$ random variable. The *descend* action moves the agent to the left, and it terminates the episode with a reward of 0.01 if the agent's position is less than 0.95, and a reward of γ^{-20} otherwise. Unlike the other environments, AlpineSki only terminates if the agent takes the *descend* action, and does not terminate based on the agent's position. The optimal policy is to move to the right end and descend one step back. Like the other environments, the value of the optimal policy from the start state is roughly 1, and the value of *descend*ing immediately is 0.01.

A.2.3 Large State-action Space: Hypercube

In this thesis the Hypercube environment is a $n = 3$ dimensional cube with radius (distance from its center to surface) of 10, i.e., $s \in [-10, 10]^n$. For a more challenging exploration task the number of dimensions or size of the cube can be increased. The agent starts at the origin and can move in any direction along any of the axes by a $\mathcal{N}(1, 0.15^2)$ random variable. The agent's movement is clipped to stay within the boundaries of Hypercube. One optimal policy is to move persistently along each axis until reaching one more surface.

Rewards depend on the number of surfaces the agent touches at a given timestep. The reward is increased as the agent touches more surfaces. They are designed so that the value of touching i surfaces forever is less than the reward from touching $i + 1$ surfaces for a single timestep, in particular $\sum_{k=0}^{\infty} \gamma^k r_i \lessapprox 0.9 r_{i+1}$, where r_i is the reward for touching i surfaces. The design choice also satisfies the optimal policy has a value near 1 at the origin. The reward vector r used in this thesis is $[0, 0.000081, 0.009, 1.245]$. When the agent transitions to a vertex, that is, it touches n surfaces (with $n = 3$, its position might be $[10, 10, -10]$), then the episode terminates with a reward of r_n.

Appendix B

Algorithm Details

In this appendix we provide the detailed pseudocode for each algorithm. To avoid redundant repetition of pseudocode, we mainly present the baseline algorithms and describe how each exploration scheme is applied. The function approximation methods include neural networks for nonlinear approximation $\hat{q}^{\text{NN}}_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and tilecoded representations for linear approximation $\hat{q}^{\text{TC}}_{\mathbf{w}}(S, A) = \mathbf{w}^\top \boldsymbol{\phi}(S, A)$, where $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ is the tile coding feature mapping.

B.1 Baselines

The section presents the pseudocode for the baseline agents greedy Q-learning and Softmax Actor-Critic, with both tilecoding and neural network function approximations. Algorithms in the following sections are based on the Q-learning agent.

Algorithm 1: Q-learning with tilecoding function approximation

Input: arbitrary value function weights $\mathbf{w} \in \mathbb{R}^d$
discount factor $\gamma \in [0, 1]$
Parameters: learning rate $\alpha > 0$
feature map $\boldsymbol{\phi} : S \times A \to \mathbb{R}^d$

while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  $A \leftarrow \arg\max_a \mathbf{w}^\top \boldsymbol{\phi}(S, a)$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    $A' \leftarrow \arg\max_a \mathbf{w}^\top \boldsymbol{\phi}(S', a)$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma \max_a \mathbf{w}^\top \boldsymbol{\phi}(S', a) - \mathbf{w}^\top \boldsymbol{\phi}(S, A)]\boldsymbol{\phi}(S, A)$
    $S \leftarrow S', A \leftarrow A'$
    Exit if *training budget exhausted*
  end
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \mathbf{w}^\top \boldsymbol{\phi}(S, A)]\boldsymbol{\phi}(S, A)$
end

Algorithm 2: Q-learning with neural network function approximation

Input: value function $\hat{q}_{\mathbf{w}}$ parameterized by weights $\mathbf{w} \in \mathbb{R}^d$ initialized with Kaiming uniform initialization
discount factor $\gamma \in [0, 1]$

Parameters: learning rate $\alpha > 0$
mini-batch size n_{batch}
target network sync period T
replay buffer $\mathcal{B}$ with capacity N
number of environment steps per update n_{env}
number of optimization steps per update n_{opt}

Iteration $t \leftarrow 0$
Target network $\mathbf{w}' \leftarrow \mathbf{w}$
while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  $A \leftarrow \arg\max_a \hat{q}_{\mathbf{w}}(S, a)$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    $A' \leftarrow \arg\max_a \hat{q}_{\mathbf{w}}(S, a)$
    Add transition (S, A, R, S') to $\mathcal{B}$
    Run UpdateWeights-DQN
    $S \leftarrow S', A \leftarrow A'$
    $t \leftarrow t + 1$
    Exit if *training budget exhausted*
  end
  Run UpdateWeights-DQN
end

Algorithm 3: Q-learning update with neural networks

if $len(\mathcal{B}) > n_{batch}$ *and* $t \mod n_{env} == 0$ **then**
  for $i = 1, \ldots, n_{update}$ **do**
    Sample a minibatch $\{(S_i, A_i, R_i, S_i')\}_{i=0}^{n_{\text{batch}}}$ from $\mathcal{B}$
    Perform one gradient descent step according to Equation 2.2
  end
end
if $t \mod T == 0$ **then**
  Sync target network $\mathbf{w}' \leftarrow \mathbf{w}$
end

Algorithm 4: Softmax Actor-Critic with tilecoding function approximation

Input: arbitrary value function weights $\mathbf{w} \in \mathbb{R}^m$
arbitrary policy weights $\mathbf{u} \in \mathbb{R}^n$
discount factor $\gamma \in [0, 1]$

Parameters: value function learning rate $\alpha > 0$
policy learning rate $\beta > 0$
value function feature map $\boldsymbol{\phi}: S \to \mathbb{R}^m$
policy feature map $\boldsymbol{\psi}: S \times A \to \mathbb{R}^n$

while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  $A \sim \text{softmax}(\boldsymbol{u}^\top \boldsymbol{\psi}(S, \cdot))$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    $A' \sim \text{softmax}(\boldsymbol{u}^\top \boldsymbol{\psi}(S', \cdot))$
    $\delta \leftarrow R + \gamma \max_a \mathbf{w}^\top \boldsymbol{\phi}(S', a) - \mathbf{w}^\top \boldsymbol{\phi}(S, A)$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \boldsymbol{\phi}(S, A)$
    $\mathbf{u} \leftarrow \mathbf{u} + \beta \delta \nabla_{\boldsymbol{u}} \ln \text{softmax}(\boldsymbol{u}^\top \boldsymbol{\psi}(S, A))$
    $S \leftarrow S', A \leftarrow A'$
    Exit if *training budget exhausted*
  end
  $\delta \leftarrow R - \mathbf{w}^\top \boldsymbol{\phi}(S, A)$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \boldsymbol{\phi}(S, A)$
  $\mathbf{u} \leftarrow \mathbf{u} + \beta \delta \nabla_{\boldsymbol{u}} \ln \text{softmax}(\boldsymbol{u}^\top \boldsymbol{\psi}(S, A))$
end

Algorithm 5: Softmax Actor-Critic with neural network approximation

Input: value function $\hat{v}_{\mathbf{w}}$ parameterized by weights $\mathbf{w} \in \mathbb{R}^m$ initialized with Kaiming uniform initialization
policy function $\hat{a}_{\mathbf{u}}$ parameterized by weights $\mathbf{u} \in \mathbb{R}^n$ initialized with Kaiming uniform initialization
discount factor $\gamma \in [0, 1]$

Parameters: value function learning rate $\alpha > 0$
policy learning rate $\beta > 0$

while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  $A \sim \text{softmax}(\hat{a}_{\mathbf{u}}(S, \cdot))$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    $A' \sim \text{softmax}(\hat{a}_{\mathbf{u}}(S', \cdot))$
    $\delta \leftarrow R + \gamma \max_a \hat{v}_{\mathbf{w}}(S', a) - \hat{v}_{\mathbf{w}}(S, A)$
    Perform one gradient descent step with gradient $-\delta \nabla_{\mathbf{w}} \hat{v}_{\mathbf{w}}(S, A)$ to update $\mathbf{w}$
    Perform one gradient descent step with gradient $-\delta \nabla_{\mathbf{u}} \ln \text{softmax}(\hat{a}_{\mathbf{u}}(S, A))$
    $S \leftarrow S', A \leftarrow A'$
    Exit if *training budget exhausted*
  end
  $\delta \leftarrow R - \hat{v}_{\mathbf{w}}(S, A)$
  Perform one gradient descent step with gradient $-\delta \nabla_{\mathbf{w}} \hat{v}_{\mathbf{w}}(S, A)$ to update $\mathbf{w}$
  Perform one gradient descent step with gradient $-\delta \nabla_{\mathbf{u}} \ln \text{softmax}(\hat{a}_{\mathbf{u}}(S, A))$
end

B.2 Undirected Methods

The ϵ-greedy agent takes actions according to the following policy

$$A' = \begin{cases} \arg\max_a \hat{q}_{\mathbf{w}}(S, a), & \text{with probability } 1 - \epsilon, \\ \text{random action}, & \text{otherwise}, \end{cases}$$

and the rest of the algorithm is identical to the greedy Q-learning.

NoisyNet utilizes the noisy linear layer rather than the regular fully connected layer. New parameters of the Q-value network are sampled after each step of optimization, i.e., each gradient descent step according to Equation 2.2. The rest of the algorithm is identical to the greedy Q-learning.

B.3 Bonus-Based Methods

Our linear count-bonus implementation estimates pseudocounts in the feature space (J. Martin et al. 2017), whereas our non-linear implementation utilizes static hashing (Tang et al. 2017). The agent updates visitation count estimates $\hat{N}(S)$ at state S before updating weights of the Q-value network. Each extrinsic reward signal is augmented by an intrinsic reward proportional to $\frac{1}{\sqrt{N(S}}$. The rest of the algorithm is identical to the greedy Q-learning.

Random network distillation (Burda et al. 2018) computes the reward bonus as the prediction error between its secondary prediction network and a fixed randomly initialized target network. In the linear setting, the 'network' is realized by the inner product of the feature vector and the weight vector. Before computing the reward bonus, the secondary network is updated towards the target network at seen transitions. The bonus term is added to the extrinsic reward similar to what count-bonus method does. The rest of the algorithm is identical to the greedy Q-learning.

B.4 Directly Estimating Value Uncertainty

Optimistic initialization essentially initializes the value estimates with an optimistic value before its interaction with the environment. The rest of the

algorithm is identical to the greedy Q-learning.

The IEQL+ approach combines optimistic initialization and count-bonus methods. Given parameters standard deviation of return $\sigma_{\max}$, confidence interval width $1-\theta$, and the discount factor γ, the value estimates are initialized to $Q(\cdot,\cdot) = \frac{\sigma_{\max} Z_{\theta/2}}{1-\gamma}$. The rest of the algorithm is identical to the count-bonus agent.

UCLS (Kumaraswamy et al. 2018) is a linear exploration method. Its pseudocode is shown in Algorithm 6.

BootstrapQ maintains an ensemble of $K > 1$ Q-value functions. This ensemble plays the role of uncertainty estimation. The detailed pseudocode for nonlinear function approximation setting is shown in Algorithm 8. In the linear setting, the 'network' is realized by the inner product of the feature vector and the weight vector. Note that in our implementation we update the network weights every n_{env} timesteps, as opposed to the setting proposed by the original paper that updates are only performed at the end of each episode. Since none of our diagnostic environments has a fixed episode length, more frequent optimization steps would help guide more efficient behaviour. We discuss and evaluate more modifications to the original setting in Chapter 6.

Algorithm 6: UCLS with tilecoding function approximation

Input: arbitrary value function weights $\mathbf{w} \in \mathbb{R}^d$
discount factor $\gamma \in [0, 1]$
Parameters: value function learning rate $\alpha > 0$
variance function learning rate $\alpha_{\text{var}} > 0$
confidence interval width $1 - \theta$
feature map $\boldsymbol{\phi} : S \times A \to \mathbb{R}^d$

Initialize $v_{\text{init}} \leftarrow \mathbf{1}, c \leftarrow \mathbf{1}, \beta \leftarrow 0.001$
Initialize $\mathbf{w}_{\text{var}} \leftarrow \mathbf{0}$, $\mathbf{w}_{\text{varInit}} \leftarrow \mathbf{1}$
while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  foreach *action* a **do**
    $u_a \leftarrow \sqrt{\left(1 - \frac{1}{\theta}\right)\left((\mathbf{w}^\top \boldsymbol{\phi}(S, a))^2 \, \|\boldsymbol{\phi}(S, a)\|^2_{\mathbf{w}_{\text{varInit}}}\right)}$
  end
  $A \leftarrow \arg\max_a \mathbf{w}^\top \boldsymbol{\phi}(S, a) + u_a$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    foreach *action* a **do**
      $u_a \leftarrow \sqrt{\left(1 - \frac{1}{\theta}\right)\left((\mathbf{w}^\top \boldsymbol{\phi}(S', a))^2 \, \|\boldsymbol{\phi}(S', a)\|^2_{\mathbf{w}_{\text{varInit}}}\right)}$
    end
    $A' \leftarrow \arg\max_a \mathbf{w}^\top \boldsymbol{\phi}(S', a) + u_a$
    Run UpdateWeights-UCLS
    $S \leftarrow S', A \leftarrow A'$
    Exit if *training budget exhausted*
  end
  Run UpdateWeights-UCLS
end

Algorithm 7: UpdateWeights-UCLS

$\delta \leftarrow R + \gamma \mathbf{w}^\top \boldsymbol{\phi}(S', A') - \mathbf{w}^\top \boldsymbol{\phi}(S, A)$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \boldsymbol{\phi}(S, A)$
$\delta_{\text{var}} \leftarrow \delta + \gamma \mathbf{w}_{\text{var}}{}^\top \boldsymbol{\phi}(S', A') - \mathbf{w}_{\text{var}}{}^\top \boldsymbol{\phi}(S, A)$
$\mathbf{w}_{\text{var}} \leftarrow \mathbf{w}_{\text{var}} + \alpha_{\text{var}} \delta_{\text{var}} \boldsymbol{\phi}(S, A)$
$\text{temp} \leftarrow v_{\text{init}}$
$v_{\text{init}} \leftarrow \max(v_{\text{init}}, \mathbf{w}_{\text{var}}{}_1^2, \ldots, \mathbf{w}_{\text{var}}{}_d^2)$
if *temp* $\neq v_{init}$ **then**
    $\mathbf{w}_{\text{varInit}} \leftarrow \mathbf{w}_{\text{varInit}} + (v_{\text{init}} - \text{temp})\mathbf{c}$
end
for *i such that* $\boldsymbol{\phi}(S, A)_i \neq 0$ **do**
    $\mathbf{c}_i \leftarrow (1 - \beta)\mathbf{c}_i$
    $\mathbf{w}_{\text{varInit}} \leftarrow (1 - \beta)\mathbf{w}_{\text{varInit}}$
end

Algorithm 8: BootstrapQ with neural network function approximation

Input: K value functions $\{Q_k\}_{k=1}^{K}$, each value function $\hat{q}_{\mathbf{w}_k} = \hat{f}_{\mathbf{w}_k} + \beta p_k$ parameterized by weights $\mathbf{w}_k \in \mathbb{R}^d$ initialized with Kaiming uniform initialization, where p_k is a randomized prior function
discount factor $\gamma \in [0, 1]$

Parameters: learning rate $\alpha > 0$
prior scale *beta*
mini-batch size n_{batch}
target network sync period T
replay buffer $\mathcal{B}$ with capacity N
number of environment steps per update n_{env}
number of optimization steps per update n_{opt}
Bernoulli masking distribution $M = Ber(p)$ with probability p

Iteration $t \leftarrow 0$
Target networks $\mathbf{w}'_j \leftarrow \mathbf{w}_j, j \in \{1, \ldots, K\}$
while *training budget not exhausted* **do**
  $S \leftarrow$ initial state of the episode
  Sample a value function to act with $k \sim \text{Uniform}(1, \ldots, K)$
  $A \leftarrow \arg\max_a \hat{q}_{\mathbf{w}_k}(S, a)$
  while *episode not terminates* **do**
    Take action A, observe R, S'
    $A' \leftarrow \arg\max_a \hat{q}_{\mathbf{w}_k}(S, a)$
    Sample bootstrap mask $m \sim M$
    Add transition (S, A, R, S', m) to $\mathcal{B}$
    Run UpdateWeights-DQN
    $S \leftarrow S', A \leftarrow A'$
    $t \leftarrow t + 1$
    Exit if *training budget exhausted*
  end
  Run UpdateWeights-BootstrapQ
end

Algorithm 9: BootstrapQ update with neural networks

if $len(\mathcal{B}) > n_{batch}$ *and* $t \mod n_{env} == 0$ **then**
  for $i = 1, \ldots, n_{update}$ **do**
    Sample a minibatch $\{(S_i, A_i, R_i, S'_i, m_i)\}_{i=0}^{n_{\text{batch}}}$ from $\mathcal{B}$
    for $j = 1, \ldots, K$ **do**
      Perform one gradient descent step to update $\mathbf{w}_j$ according to Equation 2.2, using only transitions with $m_i == 1$ for this value function sample
    end
  end
end
if $t \mod T == 0$ **then**
  Sync target network $\mathbf{w}'_j \leftarrow \mathbf{w}_j, j \in \{1, \ldots, K\}$
end

Appendix C

Hyperparameter settings of representative agents

In this appendix we describe the hyperparameter settings of the baselines and representative exploration methods. Descriptions and pseudocode for each method are introduced in Chapter B. Unless otherwise specified we set the replay buffer $\mathcal{B}$'s capacity $N = 50000$, minibatch size $n_{\text{batch}} = 128$, target network sync period $T = 128$, number of environment steps between successive weight updates $n_{\text{env}} = 4$, number of optimization steps per update $n_{\text{opt}} = 1$, and discount factor $\gamma = 0.99$. For nonlinear Q-value estimate updates, we use the Adam optimizer (Kingma et al. 2017) with default parameters $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon_{\text{Adam}} = 10^{-8}$. As for linear Q-value estimate updates, both Adam in the default setting and stochastic gradient descent (SGD) optimizers are swept. We present detailed hyperparameter choices for each algorithm in subsequent sections. For the same hyperparameter, we typically sweep over a wider range for nonlinear agents, as they tend to achieve their best performance with much more diverse values than themselves in linear settings.

C.1 Baselines

For the Q-learning agent, we sweep its learning rate α as described in Chapter 5. Unless otherwise specified, the learning rate α for any Q-value estimate updates of a representative exploration method takes the same set of values.

The Softmax Actor-Critic agent consists of two components: the actor, which defines the agent's policy through a preference value function estimate and a softmax distribution over preferences that injects stochasticity to action selection, and the critic, which reduces the variance in the actor's gradient estimates through a value function estimate. The critic updates its value function estimates with a learning rate α, and the actor updates the policy with a learning rate β. We note that in the linear setting both the actor and the critic are parameterized with the same tilecoding features, while in the nonlinear setting, each component is parameterized with its own neural network, whose architecture is the same as the one for Q-learning except the critic net's output layer. We sweep α and β separately, and each of them takes values from the same set for Q-learning's learning rate.

C.2 Undirected Methods

The ϵ-greedy agent is a Q-learning agent that takes ϵ-greedy actions. we sweep over $\epsilon \in \{0.0078125, 0.015625, 0.03125, 0.0625, 0.125, 0.25, 0.5\}$. Only constant ϵ is supported for linear agents, whereas both constant and linearly decaying ϵ are supported. The linear decaying scheduling starts with $\epsilon = 1$ — i.e., the policy is purely random at the beginning of the learning phase — and anneals ϵ linearly during the first 10k timesteps, until ϵ reaches a pre-specified ϵ_0, where ϵ_0 takes the same values as the values for constant ϵ-greedy agents.

NoisyNet utilizes noisy linear layers instead of an ordinary fully connected layer. At each layer with input x, the output $y = wx + b$ is calculated using $w = \mu^w + \sigma^w \odot \xi^w$ and $w = \mu^b + \sigma^b \odot \xi^b$, where $\mu^w, \mu^b, \sigma^w, \sigma^b$ are trainable parameters, ξ^w and ξ^b are noise variables sampled from a standard Gaussian distribution after each step of optimization, and $\odot$ denotes element-wise multiplication. Exploratory behaviour is induced from those noise in the parameter space. In our implementation we use factorised Gaussian noise to sample ξ^w and ξ^b (Fortunato et al. 2019). Each element of μ^w or μ^b is initial-

ized by sampling from an uniform distribution $\mathcal{U}[-\frac{1}{\sqrt{p}}, \frac{1}{\sqrt{p}}]$, and each element of σ^w or σ^b is initialized to a constant $\frac{\sigma_0}{\sqrt{p}}$, where p is the input dimension to the corresponding noisy linear layer, and σ_0 is set to 0.5.

C.3 Bonus-Based Methods

Count-based methods maintain a visitation count estimates $\hat{N}(s)$ at state s (or $\hat{N}(s,a)$ at state-action pair (s,a)), and augment the extrinsic reward r_e by a bonus that captures the uncertainty using the count estimate: $r = r_e + \beta \frac{1}{\sqrt{\hat{N}(s)}}$, where β is the intrinsic reward scaling factor. We sweep over $\beta \in \{0.005, 0.01, 0.05, 0.1, 0.5\}$ for the linear agent, and $\beta \in \{0.001, 0.01, 0.1, 1, 10\}$ for the nonlinear agent.

Random network distillation (RND) estimates uncertainty using a prediction network and a target network, both of which are initialized with orthogonal matrix initialization (Saxe et al. 2014) and the target network is held fixed throughout the whole learning phase. In the linear setting, the 'network' is realized by the inner product of the feature vector and the weight vector. For simplicity, in the linear setting, we initialize the prediction network with constant 0 and the target network with random normal samples. Each of the prediction network and the target network encodes the state to a k-dimensional representation vector. The learning rate α_{pred} for updating the prediction network takes the same values as α for Q-value estimates. We sweep the representation dimension $k \in \{2, 5, 10, 20\}$. The intrinsic reward is the L2 norm of the error between the target network's output and the prediction network's output, normalized by the standard deviation of the intrinsic return, and then scaled by a constant β. We sweep over $\beta \in \{0.005, 0.01, 0.05, 0.1, 0.5\}$ for the linear agent, and $\beta \in \{0.001, 0.01, 0.1, 1, 10\}$ for the nonlinear agent.

C.4 Directly Estimating Value Uncertainty

The optimistic initialization agent is the same as the greedy Q-learning base agent, except that its Q-value estimate is initialized to an inflated value. We

sweep over its initial value $w \in \{-1, 1, 10, 20, 50, 100\}$.

IEQL+ maintains approximate upper confidence bounds on the Q-value. The intrinsic reward is constructed with a standard deviation parameter $\sigma_{\max}$ and a confidence interval size parameter θ: $r_i = \frac{\sigma_{\max} Z_{\theta/2}}{\sqrt{n}}$, where $Z_{\theta/2}$ is the value at which the cumulative distribution function of the standard normal distribution has value $1-\frac{\theta}{2}$. We sweep over $\sigma_{\max} \in \{0.001, 0.005, 0.01, 0.015\}$, and $\theta \in \{0.5, 0.75, 0.9, 0.95, 0.99\}$.

UCLS also maintains upper confidence bounds on the Q-value. The upper confidence bounds are constructed with a variance estimate. The learning rate α_{var} for updating the variance estimate takes values $\alpha_{\text{var}} \in \{0.005, 0.01, 0.05, 0.1, 0.5\}$. The weights associated with the variance estimates are also scaled by a parameter $p = \sqrt{1 - \frac{1}{\theta}}$, and we sweep over $p \in \{0.05, 0.5, 1, 5, 10\}$.

BootstrapQ learns an ensemble of $K > 1$ Q-value functions, each of which is the sum of a trainable action-value function and a fixed random prior function scaled by a parameter β. A Bernoulli masking distribution $M = Ber(p)$ is used to determine whether a transition tuple is used to train each value function. We sweep the ensemble size $K \in \{2, 5, 10, 20\}$, the masking probability $p \in \{0.5, 1\}$, and the scaling factor $\beta \in \{0, 0.1, 1, 10\}$, where $\beta = 0$ is the setting where the randomized prior function is omitted.