Sample-Efficient Control with Directed Exploration in Discounted MDPs Under Linear Function Approximation

by

Raksha Kumar Kumaraswamy

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

An important goal of online reinforcement learning algorithms is efficient data collection to learn near-optimal behaviour, that is, optimizing the exploration-exploitation trade-off to reduce the sample-complexity of learning. To improve sample-complexity of learning it is essential that the agent directs its exploratory behaviour towards either visiting unvisited parts of the environment, or reducing uncertainty it may have with respect to the visited parts. In addition to such directed exploration, sample-complexity of learning can be improved by using a representation space that is amenable to online reinforcement learning. This thesis presents several algorithms that focus on these avenues for improving the sample-complexity of online reinforcement learning, specifically in the setting of discounted MDPs under linear function approximation.

A key challenge to direct effective online exploration is the learning of reliable uncertainty estimates. We address this by deriving high-probability confidence-bounds for value uncertainty estimation. We use these derived confidence-bounds to design two algorithms that direct effective online exploration; they differ mainly in their approach to directing exploration for visiting unknown regions of the environment. In the first algorithm we propose a heuristic to do so, whereas the second algorithm uses a more principled strategy based on optimistic initialization. The second algorithm is also a planning-compatible algorithm that can be parallelized, scaling sample-efficiency benefits with the compute resources afforded to the algorithm.

To improve sample-efficiency by utilizing representations that are amenable to online reinforcement learning, the thesis proposes a simple strategy for learning such representations offline. The representation learning algorithm encodes a property we call *locality*. Locality reduces interference in learning targets used by online reinforcement learning algorithms, consequently improving its sample-efficiency. The thesis shows that these learned representations also aid effective online exploration.

Overall, this thesis proposes algorithms for improving sample-efficiency of online reinforcement learning, motivates their utility, and evaluates their benefits empirically.

Preface

This thesis is an original work by the author.

Some central chapters of this thesis are based on co-authored conference publications.

- Chapter 4 is based on Kumaraswamy et al. (2018).
  Martha, Adam, and I derived the uncertainty estimates, and developed the algorithm together. Matthew and I worked on the experiments together. The paper was the result of a collaborative writing effort.
- Chapter 5 is based on Liu et al. (2019).
  Vincent, and I developed the idea, and worked on the experiments together. The paper was the result of a collaborative writing effort.

To my parents.

The truth is, most of us discover where we are headed when we arrive.

– Bill Watterson.

Acknowledgements

This document would not be possible without the many marvellous people I have been privileged to know and benefit from.

I am extremely fortunate to work under the guidance of my amazing advisors Martha White and Adam White. They have been a constant source of inspiration and kindness, and I sincerely thank them. Their approach to research and life has immeasurably influenced mine, for which I will always be grateful.

I thank my candidacy and thesis committee members Csaba Szepesvári, Levi Lelis, Nidhi Hegde, and Joelle Pineau, for their valuable time and feedback.

I thank my professors at Indiana University Bloomington who have been instrumental in this journey: Sriraam Natarajan, and Predrag Radivojac. They provided much of the initial boost and encouragement for my interest in research.

I would like to thank the members of RLAI and AMII communities – PIs, peers, colleagues, and friends. The rigorous, but not dogmatic, research atmosphere here has helped shape many things I value. The open and collaborative approach to research has made my experience a fun and fulfilling one.

I would like to acknowledge the people who helped create the content of, and refine the presentation of, this thesis: Lei Li, Matthew Schlegel, and Vincent Liu, for being incredible teammates and excellent collaborators; Khurram Javed, Zaheer Abbas, Abhishek Naik, Han Wang, and Suyog Chandramouli, for being superb copyeditors and reliable morale boosters.

I owe many thanks to wonderful friends, near and far. Your companionship has been invaluable through the roller-coaster that is graduate school; thank you for everything.

I owe much gratitude to my family, old and new, for all the patience, support, and love, while I have been away from home.

And finally, I owe much to Suyog, my kindred spirit and partner in life, for always facilitating and nurturing a happy headspace.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Learning to act by interacting with an environment is a promising strategy for designing intelligent artificial agents. The agent decides what actions to take based on its knowledge about the environment's state to achieve its goals. These actions can, in turn, affect the state of the environment. Reinforcement learning is a framework for learning to act by interacting, where the agent's goal is specified in terms of a scalar signal called the reward. The agent interacts with the environment to maximize the sum of future rewards. To learn efficiently, the agent must learn the optimal behaviour in as few interactions as possible, where optimal behaviour maximizes the sum of future rewards.

In order to learn the optimal behaviour through interactions, it is necessary that the agent incorporates *exploratory behaviour*, behaviour that may be suboptimal with respect to the experience it has accumulated. Exploratory behaviour is valuable for providing the agent with information that improves its knowledge of the environment, and consequently in achieving its objective. For example, let's say we have a driving robot that receives a reward for completing a route from point A to point B, and is learning online about *"how to be a good driving bot"*. A critical decision facing the robot during this learning phase, is whether it should *exploit* its current knowledge of the road network, assuming it knows the best path from point A to point B, or *explore* to improve its knowledge of the road network, which may reveal a better path from point A to point B. This dilemma is critical as: (1) the robot can under-explore, and therefore it may not learn the shortest path, failing to be a *good* driving bot, or (2) the robot can over-explore, learning the shortest path, but leading to an increase in the number of interactions necessary to complete the path, which, again, does not make it the ideal *good* driving bot.

Therefore, an important goal of online reinforcement learning algorithms is efficient experience gathering to learn near optimal behaviour: optimizing the exploration-exploitation trade-off to reduce the number of samples needed for learning (Sutton and Barto, 2018).

Towards this goal, several efficient strategies for exploration have been proposed when the number of distinct decision points – states of the environment – are small (Strehl and Littman, 2008; Szita and Szepesvari, 2010; Strehl et al., 2006). In the more general case where the number of states are large, and approximations are necessary to scale learning, there has been work under specific assumptions about the structure of the pre-defined approximator (Li et al., 2009; Grande et al., 2014). Additionally, with recent advances in Deep Reinforcement Learning there have been a number of approaches proposed for inducing effective exploratory behaviour when the approximator is learned online as well (Osband et al., 2016a; ODonoghue et al., 2017; Pathak et al., 2017; Burda et al., 2018). While these advances have resulted in significant empirical successes, a deeper understanding of how the proposed exploration methods interact with the inherent representation learning encoded by these non-linear architectures, is only beginning to be explored. Further, some approaches to exploration focus on evaluating the proposed algorithms in a separate phase, in which learning and exploration is turned off, which evaluates the question *how good is this fixed driving bot*. As such, these methods ignore the impact of exploration on the performance as the agent learns, in contrast to methods designed for learning near-optimal online behaviour, which evaluates the question *how good is this bot at learning to drive*.

In order to take another step towards sample-efficient online reinforcement learning algorithms, the goal of this thesis is to provide algorithms for a particular class of environments called discounted MDPs. In order to do so, the thesis first aims to identify subgoals that need to be achieved to facilitate sample-efficient online reinforcement learning. With the subgoals identified, the thesis proposes new algorithms that aim to achieve these subgoals, and consequently improve sample-efficiency of online reinforcement learning. As a first step, the work is focused on the simplest approximation setting for discounted MDPs — linear function approximation.

1.1 Objective

This thesis aims to provide an answer to the following question:

> How should an agent incorporate directed exploration and representation learning to improve the sample efficiency of online reinforcement learning under linear function approximation?

Here, directed exploration means to to take actions which are either optimal, or reflect the agent's choice to explore the consequence of the particular action in the state it is in.

This is different from strategies that do not direct exploration where actions are typically chosen randomly, for example dithering approaches like ϵ-greedy. Directed exploration is important to improve sample-efficiency of online reinforcement learning, as such an algorithm can stop exploring when exploration is not necessary, defaulting to optimal behaviour based on its experience. Additionally, in some environments a long sequence of particular actions is necessary to explore effectively. In these "needle in a haystack" environments, directed exploration strategies can drive behaviour to do so more reliably, resulting in information that is valuable for improving its behaviour. The probability of a dithering approach effectively exploring in such domains decays exponentially with the length of the exploratory path.

While incorporating a directed exploration strategy is one avenue to improve sample-efficiency of online reinforcement learning, there are other avenues that can improve it as well. One such promising avenue, in the function approximation case, is that of the generalization structure dictated by the representation. Many online reinforcement learning algorithms utilize bootstrapping — an idea that utilizes the algorithm's own estimates to design learning targets. The generalization structure of the representation space strongly influences the reliability of the learning targets, affecting the sample-efficiency of online reinforcement learning algorithms. Therefore, it is necessary to design representation learning algorithms that provide effective generalization for online reinforcement learning. A second avenue to improve sample-efficiency of online reinforcement learning is to design algorithms that are compatible with reusing previous experiences. This ability to reuse experiences, or use experience generated from a learned model, is also called planning. Planning can improve the sample-efficiency of online reinforcement learning as an agent can now use samples that are not based on online experience to improve its estimates and learning.

The main objective of this thesis is to design algorithms that achieve these goals in a particular instantiation of an online reinforcement learning algorithm under function approximation — the linear function approximation setting. The next section describes our approach to achieving this objective.

1.2 Approach

In order to address the question posed above, this thesis contains three key components: (1) identifying the goals for sample-efficient online learning and effective exploration, (2) algorithms that incorporate solution strategies to address these goals, and (3) a comprehensive empirical evaluation of the proposed solution strategy.

To identify key goals that an effective exploration algorithm should satisfy, we review the existing literature in this area of Reinforcement Learning. This helps motivate a class of algorithms that we think are particularly scalable, called *model-free confidence-based algorithms*. The first bottleneck to algorithmic advancement in this area is the propagation of uncertainty over the temporal aspect of the learning problem — that is, over the sequence of states experienced — as immediate decisions in states can affect consequences in future states. The second difficulty for this class of algorithms is directing exploration to visit unseen parts of the environment.

In order to propagate uncertainty over the temporal aspect, many algorithms (1) make assumptions about the structure of the representation space, or (2) make assumptions about the structure of the stochasticity in a problem, or (3) can be computationally expensive relying on an ensemble of estimators. Here, with limited assumptions, and a single estimator, we derive confidence sets that offer high probability bounds on the true value estimates for any policy of interest. Additionally, many algorithms do not explicitly account for the second difficulty — of directing exploratory behaviour to visit unseen parts of the environment. We introduce algorithms that explicitly account for this goal, improving the effectiveness of online exploration.

To improve sample-efficiency of online reinforcement learning algorithms, we also address two other key avenues. One, is the generalization structure of the representations used by online reinforcement learning algorithms. And the second, is ensuring compatibility of the algorithm's exploration strategy with sample-efficiency improving planning strategies like replay. While many approaches exist to learn representations, these representations can be inefficient for incremental online reinforcement learning. We identify a property which can aid incremental online reinforcement learning and design a representation learning strategy to achieve it. To address the second avenue, we propose a planning-compatible algorithm to improve sample-efficiency of online reinforcement learning.

1.3 Contributions

The key contributions of this thesis are the following.

An approach to value uncertainty estimation under linear function approximation. Estimating value uncertainty without making any assumptions about the structure of the representation, or the structure of the noise is a challenging problems. The thesis provides an approach for doing so under linear function approximation, assuming value estimation is done using the Least Squares Temporal Difference Learning algorithm. These uncertainty estimates result in high probability upper-confidence bounds, that can be used to guide future data collection to improve value estimation. Incorporating assumptions about the structure of stochasticity, it also provides a looser upper-confidence bound. This work was published in a refereed conference proceeding (Kumaraswamy et al., 2018).

An incremental on-policy control algorithm implementing directed exploration. The thesis proposes an incremental on-policy control algorithm that implements directed exploration utilizing the previously discussed uncertainty estimates. The estimated uncertainties are for a fixed policy, but here, they are tracked with respect to a changing policy to design a control algorithm. The algorithm is motivated using a simple approach to directed exploration, based on upper-confidence bounds, which can provide good online behaviour. This contribution improves upon many algorithms designed for the same setting, as validated with rigorous empirical experiments. This work was published in a refereed conference proceeding (Kumaraswamy et al., 2018).

An approach to offline representation learning to improve sample-efficiency of online reinforcement learning. The thesis identifies a key property called *locality* that can aid the sample-efficiency of online reinforcement learning algorithms. As this property is only available via hand-designed representations, the thesis proposes an algorithm for learning such representations, removing the need for hand-designing. In particular, the algorithm utilizes the widely used non-linear approximators, neural networks, making it scalable. The representations generated by this approach are compared to other strategies and their utility is validated empirically. This work was published in a refereed conference proceeding (Liu et al., 2019).

A planning-compatible online reinforcement learning algorithm that implements directed exploration. Planning strategies, like replay, are important to improve sample-efficiency of online reinforcement learning algorithms. Therefore, in order to implement a principled directed exploration strategy in a planning compatible algorithm, the thesis proposes a framework for doing so. Built on this framework, it proposes an algorithm to meet the criteria mentioned. The experimental results reflect the sample-efficiency gains, and effective directed exploration strategy employed by the proposed algorithm, when compared to other algorithms with similar goals. This work is currently under preparation for submission.

1.4 Overview

The thesis is structured as follows. We first provide a brief review of the background concepts (Chapter 2), following which we review exploration algorithms in the literature (Chapter 3). Then algorithms for on-policy control (Chapter 4), representation learning (Chapter 5), and planning-compatible online control (Chapter 6), are presented. We conclude with a chapter reviewing directions for future work (Chapter 7).

Chapter 2: Background

This chapter provides background on Reinforcement Learning and subtopics relevant to this thesis. In particular it reviews value functions under function approximation, and algorithms for learning them. It also presents the general idea of how online reinforcement learning algorithms learn optimal behaviour, and the types of objectives addressed for learning optimal behaviour. It concludes with a brief review of some theoretical tools used to evaluate such online reinforcement learning algorithms.

Chapter 3: Exploration

This chapter describes the approaches used to drive exploratory behaviour in reinforcement learning algorithms. It concludes with an overview of desirable attributes of an algorithm that directs exploration. These attributes motivate the contents of the following chapters.

Chapter 4: Incremental Control with Policy Evaluation Uncertainty Estimates

This chapter presents an approach to estimate upper-confidence bounds for the value estimates of a robust policy evaluation algorithm. The complexity of the derived approach to estimate these upper-bounds is quadratic, and therefore, a linear complexity variant is also described. The upper-bounds are used to design an incremental on-policy control algorithm. Designing this algorithm involves some approximations in the incremental setting, and therefore they are discussed in-depth. The chapter concludes with a rigorous empirical evaluation comparing the proposed algorithm to other approaches with similar goals.

Chapter 5: Representations for Online Control

This chapter identifies an important property of linear representations essential to promote sample-efficient online learning. It proposes a strategy to learning representations that encode this property. These representations are then investigated empirically in the online learning regime to evaluate their sample-efficiency. The proposed representation learning strategy is compared to many other representation learning strategies, some of which can promote a facet of the property described, but still fail to be effective. The chapter concludes with experiments that investigate how these representations impact exploration algorithms.

Chapter 6: Directed Exploration in Sample-Efficient Online Control

This chapter addresses the limitations of the algorithm proposed in Chapter 4. It first provides an overview of sample-efficient online control algorithms. The assumptions made and the ideas encoded in these algorithms are reviewed in order to facilitate developing such an algorithm for the discounted MDP setting under linear function approximation. Following this, the proposed algorithm is presented and empirically evaluated against other algorithms that address similar goals.

Chapter 7: Perspectives and Future Work

This chapter concludes the thesis by reviewing its main contributions and discussing their limitations, along with direction for future research.

1.5 Summary

In this chapter we introduced the problem area addressed by this thesis and its specific goals. We then described the approach followed in this thesis, and listed the main contributions. We concluded by listing the main contributions of this thesis, followed by an an overview of the following chapters in this thesis.

Chapter 2

Background

This chapter provides an introduction to Reinforcement Learning, and describes the necessary components that are helpful to understand this document. We will review: (1) the problem setting and its formulation, (2) value functions in Reinforcement Learning, and an approach to approximation, (3) a description of some algorithms that we utilize in this thesis, and (4) some specific details and definitions that help us characterize work related to this thesis.

2.1 Reinforcement Learning

Reinforcement learning is a learning paradigm for learning through interaction, via trial and error (Sutton and Barto, 2018). The paradigm consists of two entities: an *agent*, and an *environment*. Typically, the goal of the agent is to maximize a function of an external scalar signal, called the *reward*.

The agent interacts with the environment in discrete timesteps $t = 0, 1, 2, \dots$. At any timestep t, the environment can be described by its *state*, $s_t \in \mathcal{S}$, where $\mathcal{S}$ is a set of possible states. The agent, being privy to the current state of the environment (s_t), takes actions (a_t, makes decisions), causing a possible change in the state of the environment (s_{t+1}). The agent is then provided this new state, s_{t+1}, along with a scalar reward signal, r_{t+1}, that serves as feedback for its action a_t when the environment was in state s_t.

The interaction cycle produces a sequence of random states, actions and rewards:

$$S_0, A_0, S_1, R_1, A_2, S_2, R_2, \dots.$$

The dynamics of how the states and the rewards evolve over time are dictated by the *transition dynamics function*, denoted by P, and the *reward dynamics function*, denoted by R. An instantiation of $\mathcal{S}$, $\mathcal{A}$, P and R specifies an environment.

To maximize the sequence of scalar reward signal – also called the *return* — learning algorithms of the agent are designed to learn the *optimal policy* π^*. In general, a policy, π, is a mapping from states to a distribution over actions, $\pi : s \to \Delta(\mathcal{A})$. The optimal policy π^* is the policy that maximizes the return. This learning paradigm has seen many successes from agents learning to play games like Backgammon (Tesauro, 1994), Go (Silver et al., 2018), to arcade games in the ALE suite (Mnih et al., 2015).

2.1.1 Discounted Reinforcement Learning

The particular class of Reinforcement Learning problems addressed in this thesis are called Discounted Reinforcement Learning problems. Discounted Reinforcement Learning problems incorporate another variable called the *discount factor*, denoted by γ. This variable is also provided as feedback to the agent upon executing an action, changing the sequence of random variables generated due to the agent-environment interaction cycle to

$$S_0, A_0, S_1, R_1, \gamma_1, A_2, S_2, R_2, \gamma_2, \ldots.$$

Here, γ_{t+1} denotes the random variable the agent receives as feedback along with the scalar reward r_{t+1}, and it is in $[0, 1]$. The sequence of γ's controls the horizon of the return as the return, denoted by $G_t \in \mathbb{R}$, is defined as the following weighted cumulative sum

$$\begin{aligned} G_t &\overset{\text{def}}{=} R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+2} + \ldots \\ &= \sum_{i=0}^{\infty} \Big(\prod_{j=1}^{i} \gamma_{t+j} \Big) R_{t+1+i}. \end{aligned} \tag{2.1}$$

2.1.2 Markov Decision Processes

The Reinforcement Learning problem is formalized as a *Markov Decision Process* which is a quadruple denoted by $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R >$, where $\mathcal{S}$ denotes the possible states, and $\mathcal{A}$ corresponds to the actions an agent can take. P is the transition dynamics function which encodes the conditional distribution of transitioning to state s', from state s, upon taking action a — $P(s'|s, a)$. Similarly, R is the reward function which encodes the reward dynamics for transitioning from state s to state s' upon taking action a – $R(s, a, s')$. This reward can be deterministic or stochastic.

Discounted Markov Decision Processes: Discounted Markov Decision Processes are a modification of Markov Decision Processes that includes a function γ in its definition, and is used to model Discounted Reinforcement Learning problems. The function γ is again defined with respect to the triplet $< s_i, a_i, s_{i+1} >$, resulting in the random variable

γ_{i+1} that is used to define the discounted return in Equation (2.1). While γ_{i+1} can also be a stochastic variable, usually only deterministic variants of it are used.

2.1.3 Types of Discounted Environments

Typically, with the use of γ, the discount function, two different types of environments are used in Discounted Reinforcement Learning experiments: (1) episodic environments, and (2) continuing environments. Defining the γ function based on transition tuples helps unify both the problems and define them under the same interaction modelling framework (White, 2017).

The first type of environments, episodic environments, consist of interaction cycles that are defined by interaction trajectories, that may be of different lengths, called *episodes*. Such an environment consists of two sets of special states. The first set is called start states and is denoted by $\mathcal{S}_0$, with a distribution defined over it. All episodes start from a state sampled from this set. The second set of states is called the terminal state set denoted by $\mathcal{S}_T$. All episodes end when an agent transitions into a state in the set $\mathcal{S}_T$. The subsequent next state returned to the agent, upong transitiong to a state in $\mathcal{S}_T$ in the environment, is a new start state sampled form the set $\mathcal{S}_0$. This is because an agent cannot take any actions in a state belonging to $\mathcal{S}_T$, as the interaction cycle has *terminated*.

As episodes terminate, it is necessary to ensure returns G_t, for any (s, a) encountered during an episode, considers only the rewards obtained within the corresponding episode. This can be done by defining $\gamma(s, a, s') = 0$, for the transitions where taking action a in state s leads to a terminal state, resulting in s' being the start state of a new episode. The value of 0 for γ corresponding to such a transition automatically terminates the sequence of rewards after the episode while computing G_t. Typically, for episodic problems, γ for other transitions is set to be a constant value in $[0, 1)$.

The second type of environments, called continuing environments, are just a continuous chain of interactions. These environments can have returns that are unbounded, and therefore it is necessary to use $\gamma \in [0, 1)$. Typically, a constant γ, $\gamma_c \in [0, 1)$, is used.

2.2 Value Functions

A fundamental component of Reinforcement Learning is evaluating the value of taking an action a in a state s, denoted by $q^\pi(s, a)$, under policy π. The value $q^\pi(s, a)$ essentially denotes the expected return the agent can expect upon taking the action a in state s, and consequently acting according to the policy π, defined as

$$q^{\pi}(s,a) \overset{\text{def}}{=} \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a].$$

The expectation is taken under the dynamics dictated by the MDP, and the policy π. The values for all (s,a) pairs can then be represented as $\mathbf{q}^{\pi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, where each dimension corresponds to a (s,a) pair.

2.2.1 Function Approximation

When the number of states and actions is a small finite set, $q^{\pi}(s,a)$, $\forall(s,a)$ pairs, can be stored in a table. But in many real problems, this is not the case. Therefore, when the number of states, and/or actions are large, it is necessary to approximate these values. Only, then can reinforcement learning be scaled to large problems. Therefore, in large problems in order to approximate the values, a function approximator is used.

A function approximator is a state-action value function q, that assigns a value for all (s,a) pairs, $q_{\theta} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where θ corresponds to the parameters of the value function. In the simplest case learning such an approximated function can be thought of as the Regression problem in Machine Learning where the input x corresponds to (s,a) pairs, and the output y corresponds to a sample of the return G for the pair (s,a). Now, with many samples of y corresponding to an (s,a) pair, and many different (s,a) pairs and their corresponding y's, a function q_{θ} can be learned to make action-value predictions for possibly unseen (s,a) pairs, denoted by $\hat{q}_{\theta}(s,a)$. With sufficient training data, the goal is for $\hat{q}_{\theta}(s,a) \approx q^{\pi}(s,a)$, for policy π, making $\mathbf{q}^{\pi}_{\theta} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ the approximate values for policy π.

2.2.2 Linear Function Approximation

In the special case where the value-function is being approximated with linear functions with respect to an orthogonal set of basis $\{\phi_1, \ldots, \phi_d\}$, we are interested in learning $\mathbf{w} \in \mathcal{W} \subset \mathbb{R}^d$, where d represents the dimensionality of the linear space, and $d \ll |\mathcal{S}||\mathcal{A}|$. The value of a state-action pair is equal to $\hat{q}^{\pi}_{\mathbf{w}}(s,a) = \boldsymbol{\phi}(s,a)^{\top}\mathbf{w}$, where $\boldsymbol{\phi}(s,a)$ is the representation for state-action (s,a) in $\mathbb{R}^d$. The learned value estimates correspond to $\mathbf{q}^{\pi}_{\mathbf{w}} = \boldsymbol{\Phi}\mathbf{w}^{\pi}$, where $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times d}$ is a matrix that consists of the representations $\boldsymbol{\phi}(s,a)^{\top}$ for the various states and actions combinations along its rows, and $\mathbf{w}^{\pi}$ is the optimal $\mathbf{w} \in \mathcal{W}$.

There are many approximation schemes for constructing the $\boldsymbol{\phi}$'s corresponding to the (s,a) pairs, such as Fourier basis (Konidaris et al., 2011), Krylov basis (Parr et al., 2008)Radial Basis Functions (Sutton and Barto, 2018), etc. When $\mathcal{A}$ is finite, it is common to create feature representations based only on the state s, denoted by $\tilde{\boldsymbol{\phi}}(s)$. If the dimension of $\tilde{\boldsymbol{\phi}}(s)$ is d_s, the final feature vector $\boldsymbol{\phi}(s,a)$ is of dimension d, where $d = d_s \times |\mathcal{A}|$. That is, the final $\boldsymbol{\phi}(s,a)$ consist of $\tilde{\boldsymbol{\phi}}(s)$ in the sub-vector dedicated to action a, while the remaining part of the $\boldsymbol{\phi}(s,a)$ is set to 0.

In general, while a linear approximation scheme for approximating a complicated value function may seem simplistic, the prediction power of such an approximation scheme can be improved significantly by providing a good basis, $\{\phi_1, \ldots, \phi_d\}$, for approximating the value function linearly. This basis could be a non-linear encoding of the (s,a) pairs, providing a very effective abstraction scheme to scale learning via improved generalization. In this thesis we use two such widely used approximation schemes: Tile Coding and Neural Networks. As feature construction using Neural Networks are covered later, we will briefly review feature construction with Tile Coding next.

Tile Coding

Tile Coding is a feature construction scheme for generating a binary $\boldsymbol{\phi}$ given a (s,a) pair (Sutton and Barto, 2018). The feature construction scheme consists of a gridding unit called a tiling. Tilings essentially partition the complete input space into non-overlapping regions called tiles. Many tilings can be used with different offsets, providing variants of gridding configurations.

The feature representation reflects the tiles of the tilings. As each tiling partitions the space into non-overlapping regions or tiles, each input can fall only within a single tile, in a tiling. Consequently, the indicator for the corresponding tile in each tiling is set to 1 in the feature representation, reflecting the partition the input belongs to for the relevant tiling. Therefore, given n tilings, with c tiles each, the resulting feature vector is $d = c * n$ dimensional, and consists of n *active tiles* – tiles whose value equals 1. The features corresponding to other tiles are set to 0. Figure 2.1 shows a simple example to demonstrate Tile Coding's feature construction.

Tile Coding can further be designed to incorporate non-linear interactions in the input variables by creating tile coding schemes dedicated to a subset of them, or tile coding the resultant of a function that utilizes them, etc. It can also be simplified to tile code each dimension of the input independently. In such scenarios the resulting feature representation of the multiple tile coders employed can be stitched together, resulting in the final feature representation.

As the size of the feature representation depends on the number of tiles, instead of the

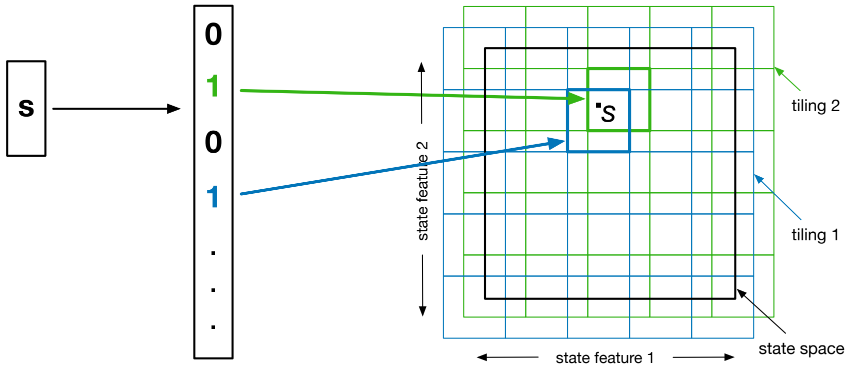

Figure 2.1: An example depicting how Tile Coding representations are generated. The input state $s \in \mathbb{R}^2$. The Tile Coding configuration uses 2 tilings of 25 tiles each, with 5 divisions in each dimension. The state s falls in a tile for each of the tilings used. The corresponding position in the feature vector is set to 1, with the rest of the positions being 0.

size of the input space, tile coding is an effective framework for reducing the dimensionality of the problem to scale learning. Additionally, as it is a sparse representation with only $n \ll d$ features active, it is also compatible with smarter linear estimation strategies like utilizing a summation over the relevant dimensions while computing dot products:

$$\phi(s,a)^\top \mathbf{w} = \sum_{i=0}^{d} \phi(s,a)[d] * \mathbf{w}[d] = \sum_{i=0}^{n} \mathbf{w}[\mathbf{t}[i]],$$

where, $\mathbf{t}$ represents a vector of length n, containing the feature number corresponding to the active tile for each of the n tilings, the active tiles.

2.3 Policy Evaluation

Given a policy π, learning the corresponding value function $\mathbf{q}^\pi$, or $\mathbf{q}^\pi_\theta$, is the problem of policy evaluation. It is a fundamental task in Reinforcement Learning as it aids in learning better behaviour via the policy iteration framework (Sutton and Barto, 2018).

As presented earlier, the value of a state corresponds to the expected cumulative discounted rewards into the future – the return G_t – after starting at state s with action a, and following π thereafter, with the dynamics dictated by the MDP's transition and reward dynamics denoted by P and R respectively. These values

$$q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma_{t+1}G_{t+1}|S_t = s, A_t = a],$$

when unrolled, correspond to

$$q^\pi(s,a) = \sum_{s'\in\mathcal{S}} P(s,a,s')R(s,a,s') + \sum_{s'\in\mathcal{S}} P(s,a,s')\gamma(s,a,s') \sum_{a'\in\mathcal{A}} \pi(s',a')q^\pi(s',a').$$

The exact values $q^\pi(s,a)$ satisfy the above equation for all state-action pairs.

In matrix form, the system can be written as

$$\mathbf{q}^\pi = \mathbf{r} + \mathbf{P}^\pi_\gamma \mathbf{q}^\pi,$$

where $\mathbf{q}^\pi$ in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ denotes the state-action values for all $\mathcal{S} \times \mathcal{A}$ pairs, $\mathbf{r} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ is the expected immediate reward

$$\mathbf{r}((s,a)) \overset{\text{def}}{=} \sum_{s' \in \mathcal{S}} P(s'|s,a) R(s,a,s'),$$

and $\mathbf{P}^\pi_\gamma \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|}$ is a sub-stochastic matrix that represents the transition process under π

$$\mathbf{P}^\pi_\gamma((s,a),(s'a')) \overset{\text{def}}{=} P(s'|,s,a)\gamma(s,a,s')\pi(a'|s').$$

The resulting linear system is

$$(\mathbf{I} - \mathbf{P}^\pi_\gamma)\mathbf{q}^\pi = \mathbf{r}.$$

This linear system can be solved analytically or iteratively to obtain the exact $\mathbf{q}^\pi$ values, and the associated *fixed point* for this linear system is denoted by the *Bellman evaluation operator* T_π. This operator is defined as

$$T_\pi \mathbf{q}^\pi \overset{\text{def}}{=} \mathbf{r} + \mathbf{P}^\pi_\gamma \mathbf{q}^\pi.$$

T_π is a monotonic and quasi-linear operator which is a contraction mapping in the L_∞ norm, which was later extended to p-norms by Munos (2003), with contraction rate γ_c in the constant-discounting regime (Bertsekas and Tsitsiklis, 1996). Therefore, repeated application of T_π to any random vector $\mathbf{q}$ converges to the optimal state-action value function $\mathbf{q}^\pi$ associated with π, for which

$$T_\pi \mathbf{q}^\pi = \mathbf{q}^\pi.$$

With the aid of this Bellman evaluation operator, Policy Evaluation algorithms aim to find the value function corresponding to a given policy π.

Policy Evaluation can be carried out online, while the agent is interacting with the environment, or offline, using interaction sequences generated by some policy in the environment. In either case the evaluation algorithms have access to samples of the form

$$< s_i, a_i, r_{i+1}, \gamma_{i+1}, s_{i+1}, a_{i+1} > .$$

Here, if offline, the actions a_i and a_{i+1} can be based on a different policy. In the online case, there are two options. In the simplest case where the policy being followed online is the policy we are interested in evaluating, the procedure is referred to as *on-policy policy evaluation*. Alternatively, if they differ, the regime is referred to as *off-policy policy evaluation*.

2.3.1 Temporal Difference Learning

In the simplest case, on-policy policy evaluation is possible because the value of a state-action pair can be written as

$$
\begin{aligned}
q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] \\
&= \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \big(\prod_{j=1}^{i} \gamma_{t+j}\big) R_{t+1+i} | S_t = s, A_t = a] \\
&= \mathbb{E}_{\pi}[R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \dots | S_t = s, A_t = a] \\
&= \mathbb{E}_{\pi}[R_{t+1} + \gamma_{t+1}(R_{t+2} + \gamma_{t+2} R_{t+3} + \dots) | S_t = s, A_t = a] \\
&= \mathbb{E}_{\pi}[R_{t+1} + \gamma_{t+1} G_{t+1} | S_t = s, A_t = a]. \qquad (2.2)
\end{aligned}
$$

Because $\mathbb{E}_{\pi}[G_{t+1} | S_t = s, A_t = a] = q^{\pi}(S_{t+1}, A_{t+1})$, the value of a state can written recursively as

$$q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma_{t+1} q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a].$$

Providing an estimate of G_t, using the estimate of $q^{\pi}(S_{t+1}, A_{t+1})$ is called *bootstrapping*.

Built on this idea of bootstrapping, Temporal Difference algorithms are a family of policy evaluation algorithms that are applicable for incremental policy evaluation. The basic update made by this family of algorithms utilizes the notion of *Temporal Difference error* (TD error) which corresponds to

$$
\begin{aligned}
\delta_t &\overset{\text{def}}{=} \delta(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1}, A_{t+1}) \\
&= R_{t+1} + \gamma_{t+1} q^{\pi}(S_{t+1}, A_{t+1}) - q^{\pi}(S_t, A_t), \qquad (2.3)
\end{aligned}
$$

to update the possibly incorrect estimate at $q^{\pi}(S_t, A_t)$ and improve the estimation of the value function. If the value function is approximated with a linear function approximator, the expected update to $\mathbf{w}$ corresponds to $\mathbb{E}[\delta_t \phi_t]$, where ϕ_t is the shorthand for $\phi(S_t, A_t)$. The resulting algorithm is also called *SARSA(0)* (Singh et al., 2000).

Traces

To improve the efficiency of Temporal Difference learning, an algorithm can incorporate *traces* (Sutton and Barto, 2018). Simplistically, traces are credit assignment tools that improve the sample-efficiency of online learning by using a different return as the learning target. Instead of the classical return G_t, traces utilize an alternative return called the λ-return, denoted by G_t^λ.

$$G_t^\lambda \overset{\text{def}}{=} (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n},$$

where,

$$G_{t:t+n} \overset{\text{def}}{=} R_{t+1} + \gamma_{t+1}R_{t+2} + \gamma_{t+1}\gamma_{t+2}R_{t+3} + \left(\Pi_{j=1}^{n}\gamma_{t+n}\right)Q^\pi(S_{t+n}, A_{t+n}),$$

is called the *n-step return*.

The corresponding value for a state-action pair is

$$q^\pi(s,a) = \mathbb{E}_\pi[G_t^\lambda | S_t = s, A_t = a],$$

and the corresponding TD-error utilizing Equation (2.3) is defined to

$$\delta_t^\lambda \overset{\text{def}}{=} \delta_t + \gamma_{t+1}\lambda\delta_{t+1}^\lambda. \tag{2.4}$$

But, here the λ-return target cannot be easily decomposed as in Equation (2.2), making δ_t^λ impossible to be estimated in the online case. Therefore, TD(λ) algorithms utilize another component called *eligibility traces*, $\mathbf{e}_t$, to enable incremental learning (Sutton and Barto, 2018). One update for the traces in the online case correspond to

$$\mathbf{e}_t = \gamma_t\lambda\mathbf{e}_{t-1} + \phi_t. \tag{2.5}$$

This update is called *accumulating traces*, and the corresponding expected update to $\mathbf{w}$ is equal to $\mathbb{E}[\delta_t\mathbf{e}_t]$. It can be shown that $\mathbb{E}[\delta_t^\lambda\phi_t] = \mathbb{E}[\delta_t\mathbf{e}_t]$ (Sutton and Barto, 2018), and the resulting algorithm is called *SARSA(λ)* (Singh et al., 2000).

2.3.2 Least Squares Temporal Difference Learning

In the special case where the value-function is approximated with linear functions with respect to an orthogonal set of basis $\{\phi_1, \ldots, \phi_d\}$, we can estimate closed-form solutions for the parameters of the value function $\mathbf{w} \in \mathcal{W} \subset \mathbb{R}^d$, instead of using the linear complexity SARSA algorithms described previously. The goal of both algorithms is to learn the optimal parameters $\mathbf{w}^\pi$ for evaluating the policy π.

In the linear approximation framework it is necessary that this optimal parameter satisfies the fixed-point equation $\Pi_\phi T_\pi \mathbf{\Phi w} = \mathbf{\Phi w}$, where Π_ϕ denotes a projection operator for the linear basis ϕ. This is because, the application of the Bellman operator to any $\mathbf{q} = \mathbf{\Phi w}$ in this space results in a new vector, $T_\pi \mathbf{q}$, which may not be in the span defined by the basis $\{\phi_1, \dots, \phi_d\}$. Therefore, a projection operation, $\mathbf{\Pi}$, is necessary to bring the resultant vector $T_\pi \mathbf{q}$ back to the representable space of functions (as spanned by the chosen basis).

For linear systems, a simple weighted projection of the form $\Pi_\phi = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{D\Phi})^{-1}\mathbf{\Phi}^\top \mathbf{D}$ can be used, where $\mathbf{D} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|}$ is a diagonal matrix that weights the state-action pairs and controls the approximation error. Therefore, for any $\mathbf{q}$,

$$\Pi_\phi T_\pi \mathbf{q} = \underset{\mathbf{w} \in \mathcal{W}}{\operatorname{argmin}} ||\mathbf{w} - T_\pi \mathbf{q}||_2^2 = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{D\Phi})^{-1}\mathbf{\Phi}^\top \mathbf{D}(\mathbf{r} + \mathbf{P}_\gamma^\pi \mathbf{q}).$$

As $\Pi_\phi T_\pi \mathbf{\Phi w}^\pi = \mathbf{\Phi w}^\pi$ for the optimal $\mathbf{w}^\pi$, the equation can be expanded with respect to T_π and Π_ϕ, and manipulated to yield a linear system of the form

$$\mathbf{\Phi}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}_\gamma^\pi)\mathbf{\Phi w}^\pi = \mathbf{\Phi}^\top \mathbf{D r}.$$

In standardized notation, the components of the linear system correspond to

$$\mathbf{A} = \mathbf{\Phi}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}_\gamma^\pi)\mathbf{\Phi}, \qquad \mathbf{b} = \mathbf{\Phi}^\top \mathbf{D r}.$$

Therefore, the optimal $\mathbf{w} \in \mathcal{W}$ for any π, $\mathbf{w}^\pi$, can be estimated as

$$\mathbf{w}^\pi = \mathbf{A}^{-1}\mathbf{b},$$

and is termed as the *least-squares fixed-point approximation* to the true value function (Lagoudakis and Parr, 2003).

This formulation can be used incrementally for learning approximate value functions from samples and the algorithm is called *Least Squares Temporal Difference Learning* (LSTD) (Bradtke and Barto, 1996). Further, the algorithms can be extended to use λ-returns as well, leading to the LSTD(λ) algorithm (Boyan, 2002). Here, the linear system corresponds to

$$\mathbf{A}^\lambda = \mathbf{\Phi}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}^\lambda)\mathbf{\Phi}, \qquad \mathbf{b}^\lambda = \mathbf{\Phi}^\top \mathbf{D r}^\lambda,$$

where,

$$\mathbf{P}^\lambda = \mathbf{I} - (\mathbf{I} - \lambda \mathbf{P}_\gamma^\pi)^{-1}(\mathbf{I} - \mathbf{P}_\gamma^\pi), \qquad \mathbf{r}^\lambda = (\mathbf{I} - \lambda \mathbf{P}_\gamma^\pi)^{-1}\mathbf{r}.$$

If we have an interaction trajectory of length T of the form

$$s_0, a_0, r_1, \gamma_1, s_1, a_1, r_2, \ldots r_T$$

with actions a_t being chosen by a policy μ in the off-policy case, or the policy π in the on-policy case, the components of the linear system can be estimated as

$$\mathbf{A}_T = \frac{1}{T-1}\sum_{t=0}^{\top}[\mathbf{e}_t(\boldsymbol{\phi}_t - \gamma_{t+1}\boldsymbol{\phi}_{t+1})^\top], \qquad \mathbf{b}_T = \frac{1}{T-1}\sum_{t=0}^{\top}[\mathbf{e}_t r_{t+1}],$$

with,

$$\mathbf{e}_t \leftarrow \gamma_t \lambda \frac{\pi(s_t, a_t)}{\mu(s_t, a_t)}\mathbf{e}_{t-1} + \boldsymbol{\phi}_t,$$

$$\mathbf{e}_{-1} = \mathbf{0}, \text{ and } \gamma_0 \overset{\text{def}}{=} 0.$$

$\mathbf{e}$ is the eligibility trace, and the update accounts for the data being off-policy: if the actions were sampled by a policy μ, different from π, the policy being evaluated. In such a case, the best $\mathbf{w}$ given the data can be estimated as

$$\hat{\mathbf{w}}^\pi = \mathbf{A}_T^{-1}\mathbf{b}_T.$$

As the data structures $\mathbf{A}_T$ and $\mathbf{b}_T$ may not reflect the accurate expected values for a finite sample-size, in general $\hat{\mathbf{w}}^\pi \neq \mathbf{w}^\pi$, where $\mathbf{w}^\pi$ is the least-squares fixed-point for the λ-return of π, with weighting given by $\mathbf{D}_\mu$. In the limit,

$$\hat{\mathbf{w}}^\pi = \mathbf{w}^\pi \Leftrightarrow \mathbb{E}_{\mathbf{D}_\mu}[\delta_t \mathbf{e}_t] = 0.$$

2.4 Policy Iteration

Policy Iteration is a central idea in Reinforcement Learning, and is a procedure to find the optimal policy π^* using the principle of Dynamic Programming (Bellman, 1957). Classically, state value functions, $\mathbf{v}^\pi$, are evaluated, where,

$$v^\pi(s) = \sum_{s \in \mathcal{A}} \pi(a|s) q^\pi(s, a),$$

but here for ease of exposition, we will use state-action value functions, $\mathbf{q}^\pi$ instead. The algorithm assumes access to a perfect model of the MDP, which is usually assumed to have a finite state and action space. It consists of two phases – (1) a Policy Evaluation phase - this is the process of evaluating the $\mathbf{q}^\pi$ for a fixed deterministic policy π, and (2) a Policy Improvement phase - this is the process of obtaining a policy π', given the value function of π, such that $v^{\pi'}(s) \geq v^\pi(s), \forall s \in \mathcal{S}$.

The process starts with a randomly initialized π_0 and $\mathbf{q}^{\pi_0}$. The evaluation phase consists of repeated application of the previously described Bellman evaluation operator for evaluating $\mathbf{q}^{\pi_t}$, using the available model to get the expected targets. The improvement phase consists of creating an improved policy π_{t+1} that is greedy w.r.t. $\mathbf{q}^{\pi_t}$. The process is repeated until convergence, that is $\pi_t = \pi_{t+1}$. When function approximation is necessary to make learning tractable the algorithm is called Approximate Policy Iteration (Bertsekas and Tsitsiklis, 1996).

Algorithms that utilize this idea and only drive online reinforcement learning through the use of value functions are also called *value-based reinforcement learning* algorithms. Alternatively, some algorithms learn an explicit parametrized policy, like the parameterized value function. We do not utilize those class of algorithms in this thesis.

Generalized Policy Iteration

While Policy Iteration consists of two processes in two distinct phases, many online incremental learning algorithms are based on an approximation of the two phases such that the processes may not have converged, but are still influencing each other. This generalized process is called *Generalized Policy Iteration* (GPI) (Sutton and Barto, 2018).

An important attribute of GPI is that it does not rely on either process converging. Although convergence of GPI to the optimal policy has been proved only for asynchronous dynamic programming methods (that rely on a perfect model, but are not strict about the convergence of the processes) (Bertsekas, 1983), many empirically successful algorithms can be cast under this framework, such as Sarsa (Rummery and Niranjan, 1994), Q-learning (Watkins and Dayan, 1992), and LSPI (Lagoudakis and Parr, 2003).

2.5 Other MDP Related Details

In this section we provide an overview regarding other details about MDPs that is relevant when we characterize other work related to this thesis. Previously, we described Discounted MDPs, and and Discounted RL problem. In this section, we describe another dimension for classifying MDPs – based on its transition dynamics. After, we summarize all the learning problems modelled as MDPs in the Reinforcement Learning literature, and the tools used to evaluate the performance of these learning algorithms.

2.5.1 Types of MDPs

An MDP can encompass processes with extremely different transition dynamics. To aid in analysis, there are four fundamental types of MDPs based on the transition dynamics they encode (Kallenberg, 2011; Bartlett and Tewari, 2012). This classification arises based on the long-time behaviour of stationary deterministic policies, $\pi : \mathcal{S} \rightarrow \mathcal{A}$ in MDPs, resulting in a visitation structure in the MDPs' set of $\mathcal{S}$.

- **Ergodic** Every π induces a single recurrent class, that is, it is possible to reach any state from any other state in $\mathcal{S}$.
- **Unichain** Every π induces a single recurrent class, plus a possibly empty set of transient classes.
- **Communicating** For every $s_a, s_a \in \mathcal{S}$, there exists a π that can transition from s_1 to s_2.
- **Weakly Communicating** The state space $\mathcal{S}$ decomposes into a set of transient states, and another set that constitutes a communicating MDP.

This categorization enables characterizing the difficulty of reaching states in the MDP. While every stationary deterministic policy would reach the complete state space of an ergodic MDP, this is not so in a communicating MDP. Therefore, ideas of exploration can be incorporated in ergodic MDPs to improve sample efficiency gains, but the necessity of exploration is more evident, for instance, in a communicating MDP. In such a scenario, exploration plays the critical role of promoting visitation to unknown/unvisited parts of the MDP. Alternatively, even in ergodic MDPs there may exist parts of the MDP that are more stochastic than the other – requiring more samples to ensure effective learning. In such a scenario, exploration plays the crucial role of increasing visitation to these parts of the MDP to promote effective learning.

2.5.2 Types of Learning Problems in MDPs

Previously we provide a categorization of MDPs based on the transition dynamics. The reward dynamics on the other hand – which provide the learning signal for an agent – are not considered for it. But the reward dynamics of an MDP are critical as they set up the learning problem for an agent that lives in the given MDP. Therefore, given an MDP many different learning problems can be set up with respect to the reward function R, and the transition dynamics P of the MDP. From an optimization perspective, there is a gradation to the difficulty of learning problems. Below, we describe the main learning problems prevalent in the literature, in increasing order of difficulty.

- **Finite-Horizon RL** Given a starting state distribution over the MDP's state-space $\mathcal{S}$ from which s_0 is sampled, the agent's goal is to learn a policy π^* that maximizes the cumulative H-step reward $\mathbb{E}_\pi[\sum_{h=0}^{H} r_{h+1}]$, where r_{h+1} the reward received by the agent for the decision made at step h. H denotes the horizon of the problem after which the process terminates, following which the agent is reset.

- **Discounted RL** Given a starting state distribution over the MDP's state-space $\mathcal{S}$ from which s_0 is sampled, the agent's goal is to learn a policy π^* that maximizes the discounted return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma_{t+1}^{t} r_{t+1}]$, where γ_{t+1}, and r_{t+1} are the discount factor and the reward received by the agent for the decision made at timestep t. The learning problem here is characterized by a function γ, called the discount function which provides the values $\gamma_{i+1} \in [0, 1]$, and an absorbing state (or a set of absorbing states) in $\mathcal{S}$ called the *terminal state(s)*, for transitioning into which $\gamma_{i+1} = 0$, and the agent is reset. As some of the literature uses the term *episode* to also refer to the trajectories generated in finite-horizon learning problems, we will use the word *discounted* to refer to these class of problems.

- **Average-Reward RL** The learning problem here is similar to the finite-horizon setting where $H = \infty$, and there are no resets. The agent starts at a random state in the MDP and lives in it forever, with its experience dictated by the dynamics of the MDP and its decisions. In such a scenario the cumulative reward can be unbounded, and therefore, the goal of the agent is to learn π^* that maximizes a finite alternative – the average reward, defined as $\mathbb{E}_\pi\big[\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} r_{t+1}\big]$. Additionally, every average-reward problem can be cast as a discounted RL problem where γ is a constant function equal to $\gamma' \in (0, 1)$ such that $\gamma' \geq \gamma^*$, called the *critical discount rate* of this learning problem, which depends on the particular MDP.

The categorization of learning problems also provides a lens to assess the difficulty of the problem from an online learning perspective. The finite decision making points which constitute the optimization of a finite-horizon learning problem lends itself to tractable algorithmic design via *backward induction* (Kallenberg, 2011). Additionally, discounted and average reward learning problems can be cast as instances of finite-horizon learning problems by considering a horizon H that is large enough. This is because the difference between the performance of the optimal policy under such a finite-horizon formulation, and the optimal policy of the original problem – the *sub-optimality gap* – is inversely proportional to H (Wei et al., 2020).

Evaluating Performance of Learning Algorithms

While a learning algorithm's primary goal is maximizing the particular signal relevant to the learning problem, the algorithm does not have direct access to an *objective-form* that captures these signals. Therefore, as a surrogate goal, algorithms are designed to optimize two main types of objectives, optimizing which, is proportional to maximizing the primary objective (Szepesvári, 2010); they are: (1) minimize the regret, and (2) minimize sample complexity.

Before defining the surrogate goals it is useful to first describe how the performance of a policy π returned by any learning algorithm can be evaluated with a single number in the different learning problems. With $\mathcal{S}$ as the set of all states, if we assume V^π_x is the value function of policy π in learning problem x, μ_x is the start-state distribution over $\mathcal{S}$ in learning problem x (for fixed-horizon and discounted learning problems), μ_π is the stationary distribution induced by π over $\mathcal{S}$, then, the long-run performance of π — denoted by ρ^π_x, performance of π in learning problem x — can be evaluated as follows

$$\begin{aligned}
\rho^\pi_{FH} &= \sum_{s\in\mathcal{S}_0} \mu_{FH}(s)V^\pi_{FH}(s), && \text{for finite-horizon learning problems,}\\
\rho^\pi_{D} &= \sum_{s\in\mathcal{S}_0} \mu_{D}(s)V^\pi_{D}(s), && \text{for discounted learning problems,}\\
\rho^\pi_{AR} &= \sum_{s\in\mathcal{S}} \mu_{\pi}(s)E_\pi[r_{t+1}|S_t = s], && \text{for average-reward learning problems.}
\end{aligned}$$

The optimal policy π^*_x for every learning problem x in the MDP, from the space of all possible policies Π, can be defined as

$$\begin{aligned}
\pi^*_{FH} &= \operatorname*{argmax}_{\pi\in\Pi} \rho^\pi_{FH}, && \text{for finite-horizon learning problems,}\\
\pi^*_{D} &= \operatorname*{argmax}_{\pi\in\Pi} \rho^\pi_{D}, && \text{for discounted learning problems,}\\
\pi^*_{AR} &= \operatorname*{argmax}_{\pi\in\Pi} \rho^\pi_{AR}, && \text{for average-reward learning problems.}
\end{aligned}$$

With the above definitions, the first type of algorithms are designed to minimize regret. Given K episodes of interaction for finite-horizon problems, or K steps of interaction for average-reward problems, with respect to an algorithm, its regret for the learning problem x — denoted by $\mathcal{R}_x$ — is defined as

$$\mathcal{R}_{FH} = \sum_{k=1}^{K} \left(V_{FH}^{\pi^*}(s_1^k) - \sum_{h=1}^{H} r_{h+1}^k \right), \qquad \text{for finite-horizon learning problems,}$$

$$\mathcal{R}_{AR} = K\rho_{AR}^{\pi^*} - \sum_{k=1}^{K} r^{k+1}, \qquad \text{for average-reward learning problems.}$$

As far as we are aware, there has not been a definition for regret proposed in the literature for algorithms maximizing discounted return. One possible straightforward method for doing so is via the definition of regret for finite-horizon problems – where the H-step sum would need to be replaced by the discounted rewards experienced in the corresponding trajectory. Nonetheless, the goal of analysis of algorithms that minimize regret is to provide high probability upper-bounds on the algorithm's regret, utilizing parameters of the MDP.

In contrast, algorithms of the second type that aim to minimize sample complexity — called PAC-MDP algorithms, Probably Approximately Correct in the MDP — aim to bound the number of steps where the algorithm's policy is suboptimal compared to the optimal policy by more than a specified amount, using parameters of the MDP. That is, an algorithm is said to PAC-MDP if the number of steps, t, where $V^{\pi_t}(S_t) < V^{\pi^*}(S_t) - \epsilon$ is polynomial in $(\mathcal{S}, \mathcal{A}, 1/\epsilon, 1/\gamma, 1/\delta)$. Here, π_t is the policy of the algorithm at time-step t, and S_t is the state experienced at time-step t. As algorithms which are designed to minimize this objective have only considered the discounted RL problem setting, in literature, γ mentioned here is the constant value used by the γ function everywhere except for transitions leading to terminal states, and $1 - \delta$ is the probability with which this bound holds.

2.6 Summary

This chapter provided a review of basic concepts that will be useful for understanding the following chapters of this thesis. We described the problem setting, its formulation as MDPs, and the types of problems tackled in this thesis. Following this we reviewed value functions, function approximation, policy evaluation and policy iteration - the core reinforcement topics relevant to this thesis. Then we concluded by reviewing concepts that are used to characterize work related to this thesis – a classification of types of MDPs, other types of learning problems based on MDPs studied in the broader reinforcement learning literature, and theoretical tools used in literature to evaluate performance of learning algorithms.

Chapter 3

Exploration in Reinforcement Learning

This chapter provides a background about exploration algorithms in Reinforcement Learning. We particularly expand on algorithms that direct exploration to improve sample-efficiency of online reinforcement learning. The goal of the chapter is to provide an overview of the approaches to exploration, and conclude with what we believe is a scalable approach to exploration in the online reinforcement learning setting, along with desiderata for such an algorithm. In this thesis we compare to many algorithms described here, and design algorithms to meet the desiderata outlined.

In the next section, we will summarize approaches in literature which are motivated by the goal of directed exploration in order to learn optimal policies in MDPs where the states can be represented independently, tabular state-spaces. Following this, we will cover approaches which extend the ideas of tabular approaches to the case where they cannot be used, continuous state-spaces. We will conclude by highlighting desirable properties for directed exploration which motivates the development of the following chapters.

3.1 The Exploration-Exploitation Trade-Off

Single-agent online reinforcement learning is a problem where the goal of a sole agent is to learn optimal behaviour through interactions with the environment. A key challenge in this interactive style of learning is that of *exploration-exploitation trade-off*: balancing decisions to gain information – exploration – with that of behaving optimally based on current knowledge – exploitation.

Balancing exploration-exploitation is crucial as the data gathering process significantly impacts the optimality of the learned policies and values. The agent needs to balance the amount of time taking exploratory actions to learn about the world, versus taking actions to behave optimally - that is, maximize cumulative rewards. If the agent explores too conservatively, it could converge to a suboptimal policy; exploring too non-conservatively, however, results in many suboptimal decisions, that is, exploratory decisions which did not help improve the behaviour towards optimality. The goal of the agent is *data-efficient exploration*: to minimize how many samples are wasted in exploration, particularly exploring parts of the world that are known, while promoting exploration to provide coverage of unknown parts of the world in order to ensure convergence to the optimal policy.

To achieve such a goal, *directed exploration strategies* are key. Undirected strategies, where random actions are taken, such as in ϵ-greedy, are used commonly. In small domains these methods are guaranteed to find an optimal policy (Singh et al., 2000), because the agent is guaranteed to visit the entire space—but may take many many steps to do so. But in larger problems this is not guaranteed, and may take infinitely long. Therefore, while these strategies have the advantage of being simple, they are not data-efficient.

3.2 Directed Exploration in Tabular State-Spaces

Directed exploration strategies have largely been explored under the framework of "optimism in the face of uncertainty" (Kaelbling et al., 1996; Szepesvári, 2010). These can generally be categorized into count-based approaches and confidence-based approaches. Count-based approaches estimate the "known-ness" of a state, typically by maintaining counts for finite state-spaces (Kearns and Singh, 2002; Brafman and Tennenholtz, 2003; Szita and Szepesvari, 2010). These counts are essentially maximum likelihood estimates for the unknown parameters of the environment - the transition kernel $\mathbf{P}$, and reward kernel $\mathbf{R}$.

Confidence-based approaches, on the other hand, maintain interval estimates, and depend on variance of the target, not just on visitation frequency for states. Confidence-based approaches can be more data-efficient for exploration because the agent can better direct exploration where the estimates are less accurate. The majority of confidence-based approaches compute confidence intervals on model parameters (Strehl and Littman, 2004; Kaelbling, 1993; Auer and Ortner, 2006; Jaksch et al., 2010; Osband et al., 2013).

While the methods listed until now operate by utilizing a model to estimate $\mathbf{P}$, and are therefore model-based, a few algorithms encourage exploration in a model-free manner by utilizing a known value for $r_{\max}$, the maximum reward possible in the environment (Strehl et al., 2006; Szita and Lorincz, 2008). A particular work utilizes the notion of uncertainty in the value estimates themselves (Meuleau and Bourgine, 1999), and describes the difficulties with extending the local measures of uncertainty from the bandit literature to reinforcement learning.

3.3 Directed Exploration in Continuous State-Spaces

In most problems the state-space of the environment is large, and therefore the above listed methods are computationally infeasible. In such a framework, it is necessary to utilize a function approximator to ensure computationally tractable learning.

Counting-based approaches from finite state-spaces have been extended to continuous state-spaces when both (1) a fixed linear/non-linear basis, and (2) a learned non-linear basis (neural networks), is used for function approximation. Count-based approaches are extended by approaches that assume the fixed basis enforces a smoothness property with models (Kakade et al., 2003; Jong and Stone, 2007; Nouri and Littman, 2009; Li et al., 2009), and without (Pazis and Parr, 2013; Martin et al., 2017; Jung and Stone, 2010). Learned non-linear basis methods, on the other hand, utilize a notion of psuedo-counts to design bonuses that incentivize exploration in model-free algorithms. These bonuses are added on to rewards, leading them to be propagated to estimate optimistic value. The psuedo-counts are estimated via various strategies like density models (Bellemare et al., 2016), the successor representation (Machado et al., 2018), and feature aggregation (Ostrovski et al., 2017).

Confidence-based approaches with models are extended by similar smoothness assumptions with fixed representations (Ortner and Ryabko, 2012), and with Bayesian approaches for learning the models (Abbasi-Yadkori and Szepesvari, 2014; Osband and Van Roy, 2017). Although propagation of uncertainty is hard due to the long-term dependencies in reinforcement learning (Meuleau and Bourgine, 1999), computational tractability with models in continuous state-spaces can in-fact be more challenging.

Therefore, there have been more approaches that maintain confidence intervals on the value function for continuous state-spaces, by maintaining a distribution over value functions to estimate an upper-bound (White and White, 2010; Grande et al., 2014), or by maintaining a randomized set of value functions from which to sample (Osband et al., 2016b,a; Plappert et al., 2017; Moerland et al., 2017; ODonoghue et al., 2017; Osband et al., 2018). Though significant steps forward, these approaches have limitations particularly in terms of computational efficiency.

DGPQ (Grande et al., 2014) requires updating Gaussian Processes, which is cubic in the number of basis vectors for the Gaussian Process. RLSVI (Osband et al., 2016b) is relatively efficient, maintaining a Gaussian distribution over parameters with Thompson sampling to get randomized values. Similarly, BSP (Osband et al., 2018) maintains a posterior over value functions by using an ensemble. Both the algorithms, RLSVI and BSP, use a staged approach which does not allow for value estimates to be updated online, as the value function is fixed per episode to gather an entire trajectory of data. Moerland et al. (2017), on the other hand, sample a new parameter vector from the posterior distribution each time an action is considered, which is expensive. The bootstrapping approaches can be efficient, as they simply have to store several value functions, either for training on a bootstrapped subset of samples, as in Bootstrapped DQN (Osband et al., 2016a), or for maintaining a moving bootstrap around the changing parameters themselves, as in UCBootstrap (White and White, 2010). For both of these approaches, however, it is unclear how many value functions would be required, which could be large depending on the problem.

Alternatively, ODonoghue et al. (2017) formulate an Uncertainty Bellman operator for propagating uncertainty - like the propagation of values via the regular Bellman operator, which is computationally efficient. Although the paper shows promising empirical evidence for the method, these uncertainty estimates are with respect to the policy being followed - instead of the optimal policy - and may therefore be suboptimal for covering the continuous state-space.

More recently, inspired by information-directed sampling (IDS) strategies in bandit algorithms (Russo and Roy, 2014), Nikolov et al. (2018) propose an algorithm called C51-IDS. Information-directed strategies assume access to a function called the *information gain function* utilizing which they compute a regret-information ratio. This ratio is inversely proportional to information gain – higher the gain, lower the ratio. The regret-information ratio is used for decision making, selecting actions that lead to the lowest regret, and highest information gain. This approach is a little different from other count-based and confidence-based methods whose action selection is guided by implementing the principle of optimism in the face of uncertainty.

C51-IDS (Nikolov et al., 2018) builds on the combination of Bootstrap DQN (Osband et al., 2016a) — which uses and ensemble of value functions — and C51 (Bellemare et al., 2017) — which estimates return distributions, instead of expected returns — to compute the regret-information ratio. It is one approach to implementing IDS for exploration in online reinforcement learning. While IDS itself as an approach seems promising, it is still new and under-explored in the reinforcement learning setting. One difficulty for designing exploration algorithms based on IDS is computing the regret-information ratio in a computationally tractable manner.

3.4 Desiderata for Directed Exploration

Given the overview of existing approaches to directed exploration in literature, we believe *model-free confidence-based methods* are particularly scalable. This is especially because learning effective models in the approximation setting is still a hard problem with active on-going research. Additionally, accurately evaluating visitation counts can be challenging in the function approximation case. Keeping this in mind, to be an effective model-free confidence-based algorithm, we think the algorithm needs to account for three key factors. While these factors may not be the complete list of attributes of an effective model-free confidence-based algorithm, we believe it identifies key necessary attributes, which serve as the motivation for the contents of this thesis.

First, the key difficulty, as highlighted in Meuleau and Bourgine (1999), is the *propagation of uncertainty over the long-term dependencies*. If this uncertainty propagation bottleneck can be overcome - either via augmenting the reward function or by an explicit second learner - then such an approach would be promising. An approach that propagates uncertainty correctly can induce exploratory behaviour that can trade-off immediate reward for reducing uncertainty in the future, informing learning.

Second, to ensure effectiveness of directed exploration strategies in online control, it is necessary that the *function space being used is amenable to sample-efficient learning*. For example, with linear function approximation it is necessary that the generalization structure of the representations used facilitates the learning algorithm and its updates. Similarly, with such representations under nonlinear function approximation, sample-efficiency of online learning can be improved without using strategies like target networks (Ghiassian et al., 2020). Additionally, such amenability to online reinforcement learning can also help improve the sample-efficiency of exploration strategies.

Third, to effectively implement the idea of optimism in the face of uncertainty under function approximation, it is necessary that *the optimal value function, $\mathbf{q}^*$, is within the set used by the directed exploration strategies*. This is necessary to guide exploratory behaviour that encompasses the complete state space, as, if the agent inherently undervalues any part of the state space with respect to its optimal value, then it cannot implement the idea of optimism faithfully. With confidence-interval based. methods, the set used by the directed exploration strategy can be large including $\mathbf{q}^*$, narrowing slowly based on the online experience.

Therefore, with the three desiderata described above: (1) propagation of uncertainty over the long-term dependencies, (2) function space being used is amenable to sample-efficient learning, and (3) the optimal value function, $\mathbf{q}^*$, is within the set used by the directed exploration strategies, the rest of this thesis proposes solutions towards each of them. In Chapter 4 we derive a method for efficient propagation of uncertainties in the Policy Evaluation setting, and adapt these uncertainty estimates to design an incremental control algorithm, addressing desiderata (1). Chapter 5 addresses desiderata (2), where we highlight a desirable attribute of representation basis for improving sample-efficiency in the incremental setting, and propose a method to learn such a basis. Chapter 6 addresses desiderata (3) by proposing an exploration strategy that incorporates a solution strategy to address it.

3.5 Summary

This chapter provided an overview of algorithms that utilize directed exploration strategies. It discussed algorithms both in the tabular and continuous state-space settings, and outlined limitations of proposed solutions. Finally, it provided a list of desiderata for effective directed exploration in the online control setting which guides the content of further chapters in this thesis.

Chapter 4

Incremental Control with Policy Evaluation Uncertainty Estimates

In this chapter, we present a method for directing exploration in the online incremental on-policy setting. The previous discussion highlighted that model-free confidence-based methods are particularly scalable. A key bottleneck to design such a method is effective uncertainty propagation (Meuleau and Bourgine, 1999). If this bottleneck can be overcome, however, such a method is promising for the online setting. A typical solution to design such a strategy with value-based reinforcement learning algorithms is to estimate uncertainties about value estimates.

In the online setting, the policy used for decision making by the agent is constantly changing. Nevertheless, at each point decision point the behaviour is dictated by the value estimates of a policy. If these value estimates are accurate the agent can act greedily with respect to them and improve its online behaviour. However, since these value estimates are learned online, they are *estimates*, and it is likely they are not accurate – that is, the agent is uncertain about them. If this uncertainty in the value estimate can be estimated online as well, then the agent could utilize these estimated value and uncertainty estimates to guide better behaviour to improve value estimation.

There are two key components to such a process when used online: (1) accurate value estimation, and (2) accurate uncertainty estimation. It is important that the value estimation algorithm is good, as this can significantly reduce any uncertainty arising from just poor learning. One such good online value estimation algorithm for a stationary problem is the Least Squares Temporal Difference Learning (LSTD) algorithm. LSTD, at each step, solves a linear system to estimate the values of a policy. If the uncertainty around these value estimates can be evaluated, then these uncertainty estimates can be used to guide exploration in the online control setting.

In this chapter, we develop such an algorithm. We first derive a method for estimating confidence intervals around value estimates learned with Least Squares Temporal Difference Learning (LSTD) for a fixed policy π. Following this, we present an algorithm for online incremental on-policy control with LSTD, called Upper-Confidence Least Squares (UCLS), and evaluate it empirically.

Contributions

Contribution 1: A method to estimate confidence-intervals around the value estimates provided by the LSTD algorithm. The goal is to guarantee with high probability that the true values are within the range provided by the estimated values plus/minus the uncertainty estimates. Our idea is to estimate the variance of the value estimate given the linear representations used by the algorithm. Then, by characterizing this variance, uncertainty estimates around the value estimates can be derived. We provide two forms of such an upper-confidence bound, one that makes a simplifying assumption of uniform noise across the state space (as would be common in Machine Learning), and another that does not, resulting in the assumption that the noise is a state-dependent function which is more natural in MDPs.

Contribution 2: A quadratic complexity algorithm that utilizes these derived upper-confidence bounds to guide exploration during online incremental on-policy control, and a linear complexity extension of the same. The algorithms are empirically evaluated and compared to other algorithms that estimate upper-confidence bounds to drive exploratory behaviour as well, albeit with different machinery. The empirical evaluation also highlights the benefits of utilizing state-dependent context without making the simplifying assumption of uniform noise in the MDP to guide online exploration.

4.1 Confidence Intervals for LSTD(λ)

Consider the goal of obtaining a confidence interval around value estimates learned incrementally by LSTD(λ) for a policy π. The goal is to guarantee, with probability $1 - p$ for a small $p > 0$, that the confidence interval around this estimate contains the value $\boldsymbol{\phi}^\top \mathbf{w}^\pi$ given by the optimal $\mathbf{w}^\pi \in \mathcal{W}$, where $\boldsymbol{\phi}$ is the state-action representation. To estimate such an interval without parametric assumptions, we use Chebyshev's inequality which—unlike other concentration inequalities like Hoeffding or Bernstein—does not require independent samples.

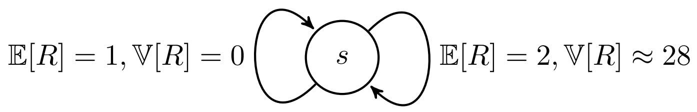
$\mathbb{E}[R] = 1, \mathbb{V}[R] = 0$
s
$\mathbb{E}[R] = 2, \mathbb{V}[R] \approx 28$

Figure 4.1: One-state world, where one action has higher variance than the other; the reward here is uniformly sampled from within the set $\{-5, -2, 2, 5, 10\}$.

To use this inequality, we need to determine the variance of the estimate $\boldsymbol{\phi}^\top \mathbf{w}$; the variance of the estimate, given $\boldsymbol{\phi}$, is due to the variance of the weights. Let $\mathbf{w}^\pi$ be the solution corresponding to the TD fixed-point for the λ-return for a fixed policy π. To characterize the noise for this optimal estimator, let ν_t be the TD-error for the optimal weights $\mathbf{w}^\pi$, where

$$r_{t+1} = (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^\top \mathbf{w}^\pi + \nu_t \qquad \text{with } \mathbb{E}[\nu_t \mathbf{e}_t] = 0, \tag{4.1}$$

where the expectation is with respect to the stationary distribution of the policy used for generating the data.

This noise ν_t is incurred from the variability in the reward, the variability in the transition dynamics and potentially the capabilities of the function approximator. A common assumption — when using linear regression for contextual bandits (Li et al., 2010) and for reinforcement learning (Osband et al., 2016b) — is that the variance of the target is a constant value σ^2 for all contexts $\boldsymbol{\phi}$. Such an assumption, however, is likely to produce larger confidence intervals than necessary. For example, consider a one-state world with two actions, as shown in Figure 4.1, where one action has a high variance reward and the other has a lower variance reward. A global sample variance will encourage both actions to be taken many times. For data-efficient learning however, the agent should take the high-variance action more, and only needs a few samples from the low-variance action.

We derive a confidence interval for LSTD(λ) without making the simplifying assumption of a global variance, and instead allow for the variance of ν_t to be context-dependent. We provide this result in Theorem 1.

Theorem 1 (Contextual upper-confidence bounds for LSTD)**.** *Let* $\bar{\boldsymbol{\nu}}_T \overset{\text{def}}{=} \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \nu_t$ *and* $\mathbf{w}_T = \mathbf{A}_T^+ \mathbf{b}_T$ *where* $\mathbf{A}_T^+$ *is the pseudoinverse of* $\mathbf{A}_T$*. Let* $\boldsymbol{\epsilon}_T^\pi \overset{\text{def}}{=} (\mathbf{A}_T^+ \mathbf{A}_T - \mathbf{I})\mathbf{w}^\pi$ *reflect the degree to which* $\mathbf{A}_T$ *is not invertible. Assume that the following are all finite:* $\mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi]$*,* $\mathbb{V}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi]$ *and all state-action features* $\boldsymbol{\phi}$*. With probability at least* $1 - p$*, given* *state-action features $\boldsymbol{\phi}$,*

$$\boldsymbol{\phi}^\top \mathbf{w}^\pi \leq \boldsymbol{\phi}^\top \mathbf{w}_T + \sqrt{\tfrac{p+1}{p}}\sqrt{\boldsymbol{\phi}^\top \mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}]\boldsymbol{\phi}} + O\left(\mathbb{E}[(\boldsymbol{\phi}^\top \boldsymbol{\epsilon}_T^\pi)^2]\right). \tag{4.2}$$

Proof: First we compute the mean and variance for our learned parameters. Because $r_{t+1} = (\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top \mathbf{w}^\pi + \nu_t$,

$$\begin{aligned}
\mathbf{w}_T &= \left(\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top\right)^{-1} \left(\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t r_{t+1}\right) \\
&= \mathbf{A}_T^+ \left(\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t((\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top \mathbf{w}^\pi + \nu_t)\right) \\
&= \mathbf{A}_T^+ \mathbf{A}_T \mathbf{w}^\pi + \mathbf{A}_T^+ \left(\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \nu_t\right) \\
&= \mathbf{w}^\pi + \mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi.
\end{aligned}$$

This estimate has a small amount of bias, that vanishes asymptotically. But, for a finite sample,

$$\mathbb{E}\left[\mathbf{A}_T^+ \left(\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \nu_t\right)\right] \neq \mathbb{E}[\mathbf{A}_T^+]\mathbb{E}\left[\tfrac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \nu_t\right] = \mathbf{0}.$$

Further, because $\mathbf{A}_T$ may not be invertible, there is an additional error $\boldsymbol{\epsilon}_T^\pi$ term which will vanish with enough samples, i.e., once $\mathbf{A}_T$ can be guaranteed to be invertible.

For covariance, because

$$\begin{aligned}
\mathbf{w}_T - \mathbb{E}[\mathbf{w}_T] &= \left(\mathbf{w}^\pi + \mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi\right) - \mathbb{E}\left[\mathbf{w}^\pi + \mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi)\right] \\
&= \mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi - \mathbb{E}\left[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi\right],
\end{aligned}$$

the covariance of the weights is

$$\mathbb{V}[\mathbf{w}_T] = \mathbb{V}\left[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi\right].$$

Now we can use concentration inequalities to bound the deviation of our estimate using the variance. While many concentration inequalities require independent samples – which are not available in this framework – Chebyshev's inequality does not. Chebyshev's inequality states that for a random variable X, if the $\mathbb{E}[X]$ and $\mathbb{V}[X]$ are bounded, then for any $\epsilon \geq 0$:

$$\Pr\left(|X - \mathbb{E}[X]| < \epsilon\sqrt{\mathbb{V}[X]}\right) \geq 1 - \frac{1}{\epsilon^2}.$$

If we set $\epsilon = \sqrt{1/p}$, then this gives

$$\Pr\left(|X - \mathbb{E}[X]| < \sqrt{\tfrac{1}{p}}\sqrt{\mathbb{V}[X]}\right) \geq 1 - p.$$

Now we have characterized the variance of the weights, but what we really want is to characterize the variance of the value estimates. Notice that the variance of the value-estimate, for state-action $\boldsymbol{\phi}$ is

$$\begin{aligned}
\mathbb{V}[\boldsymbol{\phi}^\top \mathbf{w}_T|\boldsymbol{\phi}] &= \mathbb{E}[\boldsymbol{\phi}^\top \mathbf{w}_T \mathbf{w}_T^\top \boldsymbol{\phi}|\boldsymbol{\phi}] - \mathbb{E}[\boldsymbol{\phi}^\top \mathbf{w}_t|\boldsymbol{\phi}]^2 \\
&= \boldsymbol{\phi}^\top \left(\mathbb{E}[\mathbf{w}_T\mathbf{w}_T^\top] - \mathbb{E}[\mathbf{w}_T]\mathbb{E}[\mathbf{w}_T]^\top\right)\boldsymbol{\phi} \\
&= \boldsymbol{\phi}^\top \mathbb{V}[\mathbf{w}_T]\boldsymbol{\phi}.
\end{aligned}$$

Therefore, the variance of the estimate is characterized by the variance of the weights. With high probability,

$$\begin{aligned}
\left|\boldsymbol{\phi}^\top\mathbf{w}_T - \boldsymbol{\phi}^\top\mathbf{w}^\pi\right| &= \left|\boldsymbol{\phi}^\top(\mathbf{w}_T - \mathbb{E}[\mathbf{w}_T]) + \boldsymbol{\phi}^\top(\mathbb{E}[\mathbf{w}_T] - \mathbf{w}^\pi)\right| \\
&\leq \left|\boldsymbol{\phi}^\top(\mathbf{w}_T - \mathbb{E}[\mathbf{w}_T])\right| + \left|\boldsymbol{\phi}^\top(\mathbb{E}[\mathbf{w}_T] - \mathbf{w}^\pi)\right| \\
&\leq \frac{1}{\sqrt{p}}\sqrt{\boldsymbol{\phi}^\top\mathbb{V}\left[\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi\right]\boldsymbol{\phi}} + \left|\boldsymbol{\phi}^\top\mathbb{E}[\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi]\right| \qquad (4.3) \\
&= \frac{1}{\sqrt{p}}\sqrt{\boldsymbol{\phi}^\top\left(\mathbb{E}\left[\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T\bar{\boldsymbol{\nu}}_T^\top\mathbf{A}_T^{+\top} + \boldsymbol{\Sigma}_T^\pi\right] - \boldsymbol{\mu}_T^\pi\boldsymbol{\mu}_T^{\pi\top}\right)\boldsymbol{\phi}} + \sqrt{\boldsymbol{\phi}^\top\boldsymbol{\mu}_T^\pi\boldsymbol{\mu}_T^{\pi\top}\boldsymbol{\phi}}, \qquad (4.4)
\end{aligned}$$

where Equation (4.3) uses Chebyshev's inequality, and the last step is a rewriting of Equation (4.3) using the definitions

$$\boldsymbol{\mu}_T^\pi \overset{\text{def}}{=} \mathbb{E}[\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T + \boldsymbol{\epsilon}_T^\pi],$$

and

$$\boldsymbol{\Sigma}_T^\pi \overset{\text{def}}{=} \mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T\boldsymbol{\epsilon}_T^{\pi\top} + \boldsymbol{\epsilon}_T^\pi(\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T)^\top + \boldsymbol{\epsilon}_T^\pi\boldsymbol{\epsilon}_T^{\pi\top}.$$

To simplify Equation (4.4), we need to determine an upper bound for the general formula $c\sqrt{a^2 - b^2} + b$ where $a \geq b \geq 0$. Because $p < 1$, we know that $c = \sqrt{1/p} \geq 1$. Therefore, the extremal points for b, $b = a$ and $b = 0$, both result in an upper bound of ca. Taking the derivative of the objective, gives a single stationary point in-between $[0, a]$, with $b = \frac{a}{\sqrt{c^2+1}}$. The value at this point evaluates to be $a\sqrt{c^2+1}$. Therefore, this objective is upper-bounded by $a\sqrt{c^2+1}$.

Now for $a^2 = \boldsymbol{\phi}^\top\mathbb{E}\left[\mathbf{A}_T^+\bar{\boldsymbol{\nu}}_T\bar{\boldsymbol{\nu}}_T^\top\mathbf{A}_T^{+\top} + \boldsymbol{\Sigma}_T^\pi\right]\boldsymbol{\phi}$, the term involving $\boldsymbol{\phi}^\top\mathbb{E}\left[\boldsymbol{\Sigma}_T^\pi\right]\boldsymbol{\phi}$ should quickly disappear, since it is only due to the potential lack of invertibility of $\mathbf{A}_T$ while T

si small. This term is equal to $\mathbb{E}\left[2(\boldsymbol{\phi}^\top \mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T)(\boldsymbol{\phi}^\top \boldsymbol{\epsilon}_T^\pi) + (\boldsymbol{\phi}^\top \boldsymbol{\epsilon}_T^\pi)^2\right]$, which results in the additional $O(\mathbb{E}[(\boldsymbol{\phi}^\top \boldsymbol{\epsilon}_T^\pi)^2])$ in the bound.

■

Why does the uncertainty reduce to 0?

If we assume $\mathbf{A}_T$ is invertible, the big-O term in Equation (4.2) above would disappear. Consequently, the magnitude of the upper-bounds is controlled by a term proportional to $\mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}]$. In this term, $\bar{\boldsymbol{\nu}}_T$ is a vector that accumulates the product of TD-error and traces, $\bar{\boldsymbol{\nu}}_T = \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \nu_t$.

As $T \to \infty$, the expectation concentrates, improving the estimation of $\mathbf{w}_T$. We know that at the optimal weights, $\mathbf{w}^\pi$, $\mathbb{E}[\mathbf{e}_t \nu_t] = 0$. Therefore, as $T \to \infty$, $\mathbf{w}_T \to \mathbf{w}^\pi$, and $\bar{\boldsymbol{\nu}}_T \to \mathbf{0}$. Consequently, the expected covariance $\mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}] \to \mathbf{0}$, resulting in the uncertainty estimate reducing to 0.

Additionally, if one were to use an estimated covariance $\hat{\mathbb{E}}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}]$, instead of the true covariance $\mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}]$, the uncertainty estimates would still reduce to 0 with increasing T. The reasoning is similar: as $T \to \infty$, $\mathbf{w}_T \to \mathbf{w}^\pi$, resulting in $\bar{\boldsymbol{\nu}} \to \mathbf{0}$, and consequently $\hat{\mathbb{E}}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}] \to \mathbf{0}$.

Global-variance Upper-Bounds for LSTD

We also derive the confidence interval assuming a global variance in Corollary 1, to provide a comparison. Again, if $\mathbf{A}_T$ is invertible, the big-O term in Equation (4.5) would disappear.

Corollary 1 (Global-variance bounds for LSTD). *Assume that ν_t are i.i.d., with mean zero and bounded variance σ^2. Let $\bar{\mathbf{e}}_T = \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t$ and assume that the following are finite: $\mathbb{E}[\boldsymbol{\epsilon}_T^\pi]$, $\mathbb{V}[\boldsymbol{\epsilon}_T^\pi]$, $\mathbb{E}[\mathbf{A}_T^+ \bar{\mathbf{e}}_T \bar{\mathbf{e}}_T^\top \mathbf{A}_T^{+\top}]$ and all state-action features $\boldsymbol{\phi}$. With probability at least $1-p$, given state-action features $\boldsymbol{\phi}$,*

$$\boldsymbol{\phi}^\top \mathbf{w}^\pi \le \boldsymbol{\phi}^\top \mathbf{w}_T + \sigma\sqrt{\tfrac{p+1}{p}}\sqrt{\boldsymbol{\phi}^\top \mathbb{E}[\mathbf{A}_T^+ \bar{\mathbf{e}}_T \bar{\mathbf{e}}_T^\top \mathbf{A}_T^{+\top}]\boldsymbol{\phi}} + O\left(\mathbb{E}[(\boldsymbol{\phi}^\top \boldsymbol{\epsilon}_T^\pi)^2]\right). \tag{4.5}$$

Proof: The result follows similarly, with some simplifications due to global-variance:

$$\begin{aligned}
\mathbb{E}\left[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T\right] &= \mathbb{E}\left[\mathbb{E}\left[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \middle| S_0, ..., S_T\right]\right] \\
&= \mathbb{E}\left[\mathbf{A}_T^+ \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{e}_t \mathbb{E}\left[\nu_t \middle| S_0, ..., S_T\right]\right] \\
&= \mathbf{0}, \\
\mathbb{E}[\mathbf{A}_T^+ \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}] &= \sigma^2 \mathbb{E}[\mathbf{A}_T^+ \bar{\mathbf{e}}_T \bar{\mathbf{e}}_T^\top \mathbf{A}_T^{+\top}].
\end{aligned}$$

■

4.1.1 Linear Complexity Confidence Intervals for LSTD(λ)

The uncertainty estimates derived above have quadratic computational complexity. But alternatively, they can be approximated with linear computational complexity for a fixed policy. As the covariance matrix is the outer-product of the solution to a least-squares system, $\mathbf{A}_T^{-1}\bar{\boldsymbol{\nu}}_T$, its solution $\mathbf{w}_{\text{var}}$, and can be estimated incrementally as:

$$\mathbf{w}_{\text{var}t+1} = \mathbf{w}_{\text{var}t} + \alpha\delta_{\text{var}t}\mathbf{e}_t,$$

where, $\delta_{\text{var}t} = \delta_t + \gamma_{t+1}\boldsymbol{\phi}_{t+1}^\top\mathbf{w}_{\text{var}t} - \boldsymbol{\phi}_t^\top\mathbf{w}_{\text{var}t}$. Therefore, for a given policy, the true action-values satisfy the following:

$$\boldsymbol{\phi}^\top\mathbf{w}^\pi \leq \boldsymbol{\phi}^\top\mathbf{w}_T + \sqrt{\tfrac{p+1}{p}}\sqrt{\boldsymbol{\phi}^\top\mathbf{w}_{\text{var}T}\mathbf{w}_{\text{var}T}^\top\boldsymbol{\phi}}. \tag{4.6}$$

Similarly, a linear variant for Corollary 1 can be obtained, and it consists of an outer-product of a least-squares system whose solution is given by $\mathbf{A}_T^{-1}\bar{\mathbf{e}}_T$.

4.2 On-Policy Control

The classical idea of GPI (Sutton and Barto, 2018) is adapted here with LSTD value estimates for designing an *on-policy* control algorithm. The characteristic feature of an on-policy algorithm is simply that the learning goal of the agent corresponds to its behaviour – that is, the agent learns about the policy it is following. Therefore, if the policy it is following improves, its value estimates improve; if the value estimates improve, the policy it is following improves – the crux of GPI. In such a scenario, it is important to ensure the agent's behaviour is neither so exploitative that it learns a suboptimal policy, nor so exploratory that it takes a lot of suboptimal decisions. The goal here, then, is *data-efficient exploration*: taking exploratory actions which are the most beneficial for improving on the current policy, if the policy can be improved, else behaving according to the current policy. To achieve such a goal, *directed exploration strategies* are key.

Towards designing such a strategy consider having access to confidence-intervals which estimate a radius of uncertainty $\hat{u}(S_t, A_t)$ around value estimates $\hat{q}(S_t, A_t)$. Let the action selection be greedy w.r.t. to these optimistic values, $\operatorname{argmax}_a \hat{q}(S_t, a) + \hat{u}(S_t, a)$, which provides a high-confidence upper bound on the best possible value for that action. If the $\hat{u}(S_t, A_t)$ estimates are optimistic with respect to the optimal values $q^*(S_t, A_t)$ in expectation, then, similar to online learning, we can guarantee that greedy-action selection according to upper-confidence values will converge to the optimal policy, (1) if the confidence interval radius shrinks to zero, (2) if the algorithm to estimate action-values for a policy converges to the corresponding actions, and (3) if the upper-confidence estimates are optimistic with respect to the optimal values in expectation. This simple, general idea, is presented below as a theorem.

4.2.1 Optimistic Values Theorem

Let π^* be the reference (optimal or approximately optimal) policy, q^* the true action value for that policy, and $d : \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$, an evaluation density that dictates the relative importance of state-action pairs. This d can be related to the trajectory of the optimal policy π^*, but generically allows specification of any density, such as one putting all weight on a set of start states or such as one that is uniform across states and actions to ensure equal importance for any point in the state-action space.

Here, we provide a motivational theorem that shows that if an agent (1) estimates two quantities - action-values, and confidence-interval radius, (2) behaves greedily with respect to the resulting upper-confidence bounds, and (3) its estimated quantities satisfy some assumptions under the evaluation density d, the agent can be expected to learn the approximate q^* under the evaluation density d.

Let $\tilde{q}_t = \hat{q}_t + \hat{u}_t$ be the estimated action-values plus the confidence interval radius $\hat{u}_t$ on time step t, to get the estimated upper-confidence bound which the agent uses to select actions. Let π_t be the policy induced by greedy action selection on $\tilde{q}_t$.

Assumption 1 (Expected Optimism)**.** *At some point $T > 0$, the action-values at every step $t \geq T$ are optimistic in expectation: $\mathbb{E}[\tilde{q}_t(S, A)] \geq \mathbb{E}[q^*(S, A)]$, with expectation according to density d.*

Assumption 2 (Shrinking Confidence Interval Radius)**.** *The expected value under d of the confidence interval radius $\hat{u}_t$ goes to zero: $\mathbb{E}[\hat{u}_t(S, A)] \leq f(t)$ for some non-negative function f with $f(t) \rightarrow 0$.*

Assumption 3 (Convergent Action Values)**.** *The expected value under d of the estimated action-values $\hat{q}_t$ approach the expected value of the true action-values for policy π_t: $|\mathbb{E}[\hat{q}_t(S, A) - q^{\pi_t}(S, A)]| \leq g(t)$ for some non-negative function g with $g(t) \rightarrow 0$.*

Given the three key assumptions, the theorem below is straightforward to prove. However, these three conditions are fundamental, and do not imply each other. For example, Assumption 1 and 2 do not imply Assumption 3, because the confidence interval radius could decrease to zero, and $\hat{q}_t$ can still be optimistic in expectation and an over-estimate of values that correspond to a suboptimal policy. Assumption 1 and 3 do not imply Assumption 2, because $\hat{q}_t$ could converge to the policy corresponding to acting greedily w.r.t. $\tilde{q}_t$, but $\hat{u}_t$ may never fade away. Then, $\tilde{q}_t$ could still be optimistic in expectation, but the policy π_t could be suboptimal because it is acting greedily according to inaccurate, inflated estimates of value $\tilde{q}_t$.

Theorem (Optimistic Values Theorem)**.** *Under Assumptions 1, 2 and 3,*

$$\mathbb{E}[q^*(S,A)] - \mathbb{E}[q^{\pi_t}(S,A)] \leq f(t) + g(t),$$

and as $t \to \infty$, $\mathbb{E}[q^*(S,A)] - \mathbb{E}[q^{\pi_t}(S,A)] \to 0$.

Proof: Consider the difference across states and actions

$$\begin{aligned}\mathbb{E}[q^*(S,A) - q^{\pi_t}(S,A)] &= \mathbb{E}[q^*(S,A) - \tilde{q}_t(S,A)] + \mathbb{E}[\tilde{q}_t(S,A) - q^{\pi_t}(S,A)] \\ &\leq \mathbb{E}[\tilde{q}_t(S,A) - q^{\pi_t}(S,A)]\end{aligned}$$

because $\mathbb{E}[q^*(S,A) - \tilde{q}_t(S,A)] \leq 0$ by Assumption 1. By Assumptions 2 and 3,

$$\mathbb{E}[\tilde{q}_t(S,A) - q^{\pi_t}(S,A)] = \mathbb{E}[\hat{q}_t(S,A) - q^{\pi_t}(S,A)] + \mathbb{E}[\hat{u}_t(S,A)] \leq g(t) + f(t).$$

Additionally as $f(t) \to 0$, and $g(t) \to 0$, as $t \to \infty$, $\mathbb{E}[q^*(S,A)] - \mathbb{E}[q^{\pi_t}(S,A)] \to 0$ as $t \to \infty$, completing the proof. ■

This result is abstract such that the three assumptions could be satisfied in a variety of ways. The first assumption would need propagation of optimism as done by many methods which use the principle of optimism in the face of uncertainty in tabular RL. We hypothesize that the last two assumptions could be addressed with a two-timescale analysis, with $\hat{u}_t$ updating more slowly than $\hat{q}_t$. This would reflect an iterative approach, where the optimistic values are essentially held fixed—such as is done in Delayed Q-learning (Grande et al., 2014)—and q^{π_t} estimated, before then adjusting the optimistic values. The updates to $\hat{q}_t$, then, would be updated on a faster timescale, converging to q^{π_t}, and the upper confidence radius $\hat{u}_t$ updating on a slower timescale. If Assumption 1 does not hold on the other hand, the regret can be high, but the behaviour policy $\tilde{q}_t$ would still converge to policy π_t if Assumptions 2 and 3 are satisfied.

The result is inspired by a well known result in the literature: a Bayesian regret bound for the algorithm RLSVI, provided by Osband et al. (2016b). The work characterizes the expected Bayesian regret experienced by an RLSVI agent in a finite-horizon tabular state-space setting, which is dependent on the size of the state and action spaces. The episodic regret is calculated considering a policy that remains stationary for the length of the episode, H, the finite-horizon, where a stationary policy for the episode implies $\pi_x = \pi_{x+1} = \dots \pi_{x+H}, \forall x \in \{x | x \in \mathbb{N}, (x + H - 1)\%H == 0\}$. The expected episodic regret looks similar to the result provided above with d as the starting state distribution for the finite-length episodes.

Additionally, the Optimistic Values Theorem result also looks to be similar to another well-known result: the Performance Difference Lemma provided by Kakade and Langford (2002). While they results look similar because they both present an approach to evaluate the difference between the value of two policies under a distribution, the two results are different because the result provided by Performance Difference Lemma does not imply the Optimistic Values Theorem result.

This result is useful because it suggests that algorithms that use such a confidence-interval based approach to exploration with optimism can be well-behaved if the assumptions discussed are met. The confidence intervals derived in the previous section are epistemic confidence intervals for LSTD, which for a fixed policy π, would meet the requirements of Assumption 2, Shrinking Confidence Interval Radius. Assumption 3, Convergent Action Values, can be satisfied with a good policy evaluation algorithm to ensure reliable value function estimation, and a very small policy improvement step. Fortunately, LSTD meets this criteria of being a good policy evaluation algorithm. If Assumption 1, Expected Optimism, can be satisfied, then Theorem 1 can bound the regret of an algorithm which uses these confidence intervals for exploration. Motivated by this, we present the first-step taken towards designing such a strategy to exploration in the next section.

4.2.2 UCLS: Control with LSTD using Upper-Confidence Bounds

In order to design a directed exploration strategy we utilize the uncertainty estimates that were derived in Section 4.1 in the algorithm we call Upper-Confidence-Least-Squares (UCLS). It uses the incrementally estimated upper-confidence bounds – shown below again for reference (with the big-O terms omitted) – for guiding on-policy exploration

$$\boldsymbol{\phi}^\top \mathbf{w}^\pi \leq \boldsymbol{\phi}^\top \mathbf{w}_T + \sqrt{\tfrac{p+1}{p}} \sqrt{\boldsymbol{\phi}^\top \mathbb{E}[\mathbf{A}_T^{+} \bar{\boldsymbol{\nu}}_T \bar{\boldsymbol{\nu}}_T^\top \mathbf{A}_T^{+\top}] \boldsymbol{\phi}}.$$

These upper-confidence bounds include the true covariance which we do not have access to online. Therefore, instead, we utilize the online estimated covariance as a stand-in, while trading in some of the guarantees provided by the bound. Further, the upper-confidence bounds are derived for a fixed policy π. In the online control setting, under the GPI framework, the key idea is to slowly estimate both the values and the uncertainty estimates, under a changing policy that acts greedily with respect to the upper-confidence bounds. Effectively tracking these upper bounds with a changing policy effectively incurs some approximations, and they are discussed below.

(1) We are not evaluating a fixed policy; rather, the policy is changing. The estimates $\mathbf{A}_T$ and $\mathbf{b}_T$ will therefore be out-of-date. As is common for LSTD with control, we use an exponential moving average to estimate $\mathbf{A}_T$, $\mathbf{b}_T$ and the upper-confidence bound. The exponential moving average uses $\mathbf{A}_T = (1-\beta)\mathbf{A}_{T-1} + \beta \mathbf{e}_T(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top$, for some $\beta \in [0,1]$. If $\beta = 1/T$, then this reduces to the standard sample average; otherwise, for a fixed β, such as $\beta = 0.01$, more recent samples have a higher weight in the average. As exponential average is unbiased, the derived uncertainty bounds will still hold.

(2) We cannot obtain samples of the noise

$$\nu_t = r_{t+1} + \gamma_{t+1}\boldsymbol{\phi}_{t+1}^\top \mathbf{w}^* - \boldsymbol{\phi}_t^\top \mathbf{w}^*$$

which is the TD-error for the optimal value function parameters $\mathbf{w}^*$. Instead, we use δ_t as a proxy. This proxy results in an upper bound that is too conservative—too loose—because δ_t is likely to be larger than ν_t. This is likely to ensure sufficient exploration, but may cause more exploration than needed. The moving average $\bar{\boldsymbol{\nu}}_t = \bar{\boldsymbol{\nu}}_{t-1} + \beta_t(\delta_t \mathbf{e}_t - \bar{\boldsymbol{\nu}}_{t-1})$ should help mitigate this issue, as older δ_t are likely larger than more recent ones.

(3) The covariance matrix $\mathbf{C}$ estimating $\mathbb{E}[\mathbf{A}_T^{-1}\bar{\boldsymbol{\nu}}_T\bar{\boldsymbol{\nu}}_T^\top\mathbf{A}_T^{-1}]$ could underestimate covariances, depending on a skewed distribution over states and depending on the initialization. This is particularly true in early learning, where the distribution over states is skewed to be higher near the start state. To see why, let $\mathbf{a} = \mathbf{A}_T^{-1}\bar{\boldsymbol{\nu}}_T$. The covariance estimate $\mathbf{C}_{ij} = \mathbb{E}[\mathbf{a}_i\mathbf{a}_j]$ corresponds to feature i and j. The agent begins in a certain region of the space, and so features that only become active outside of this region will be zero, providing samples $\mathbf{a}_i\mathbf{a}_j = 0$. As a result, the covariance is artificially driven down in unvisited regions of the space by accumulating updates of 0. Further, if the initialization to the covariance $\mathbf{C}_{ii}$ is an underestimate, a visited state with high variance will artificially look more optimistic than an unvisited state. We propose two simple approaches to this issue: updating $\mathbf{C}$ based on locality and adaptively adjusting the initialization to $\mathbf{C}_{ii}$. Each covariance estimate $\mathbf{C}_{ij}$ for features i and j should only be updated if the sampled outer-product is relevant, with the agent in the region where i and j are active. To reflect this locality, each $\mathbf{C}_{ij}$ is updated with the $\mathbf{a}_i\mathbf{a}_j$ only if the eligibility traces is non-zero for i and j. To adaptively update the initialization, the maximum observed $\mathbf{a}_i^2$ is stored, as c_{max}, and the initialization c_0 to each

$\mathbf{C}_{ii}$ is retroactively updated using

$$\mathbf{C}_{ii} = \mathbf{C}_{ii} - (1-\beta)^{c_i} c_0 + (1-\beta)^{c_i} c_{\text{max}},$$

where, c_i is the number of times $\mathbf{C}_{ii}$ has been updated; as though at initialization $\mathbf{C}_{ii} = c_{\text{max}}$ was used. Additionally, because we are in the online setting, we utilize the empirical covariance instead of the true covariance: that is, $\mathbf{C}$ tracks estimated $\hat{\mathbb{E}}[\mathbf{A}_T^{-1}\bar{\boldsymbol{\nu}}_T\bar{\boldsymbol{\nu}}_T^\top\mathbf{A}_T^{-1}]$, instead of $\mathbb{E}[\mathbf{A}_T^{-1}\bar{\boldsymbol{\nu}}_T\bar{\boldsymbol{\nu}}_T^\top\mathbf{A}_T^{-1}]$.

(4) To improve the computational complexity of the algorithm, we propose an alternative, incremental strategy for estimating $\mathbf{w}$, that takes advantage of the fact that we already need to estimate the inverse of $\mathbf{A}$ for the upper bound. In order to do so, we make use of the summarized information in $\mathbf{A}$ to improve the update, but avoid directly computing $\mathbf{A}^{-1}$ as it may be poorly conditioned. Instead, we maintain an approximation $\mathbf{B} \approx \mathbf{A}^{-\top}$ that uses a simple gradient descent update, to minimize $\|\mathbf{A}^\top\mathbf{B}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t\|_2^2$. If $\mathbf{B}$ is the inverse of $\mathbf{A}^\top$, then this loss is zero; otherwise, minimizing it provides an approximate inverse. This estimate $\mathbf{B}$ is useful for two purposes in the algorithm. First, it is clearly needed to estimate the upper-confidence bound. Second, it also provides a pre-conditioner for the iterative update $\mathbf{w} = \mathbf{w} + \mathbf{G}(\mathbf{b} - \mathbf{A}\mathbf{w})$, for preconditioner $\mathbf{G}$. The optimal preconditioner is in fact the inverse of $\mathbf{A}$, if it exists. We use $\mathbf{G} = \mathbf{B}^\top + \eta\mathbf{I}$ for a small $\eta > 0$ to ensure that the preconditioner is full rank.

With these four criteria incorporated, UCLS utilizes the estimated upper-confidence bounds to guide incremental on-policy exploration. The pseudocode for UCLS is presented in Algorithm 2.

Algorithm 1 GetOptimisticAction($\boldsymbol{\phi}_{s,\cdot}$)

```
$u_a \leftarrow \sqrt{\left(1+\frac{1}{p}\right)\boldsymbol{\phi}_{s,a}^\top \mathbf{C}\boldsymbol{\phi}_{s,a}} \quad \forall a \in \mathcal{A}$
$a = \mathrm{argmax}_{a\in\mathcal{A}}\, \boldsymbol{\phi}_{s,a}^\top \mathbf{w} + u_a$
**return** $a$
```

Algorithm 2 UCLS(λ)

```
$\mathbf{A} \leftarrow \mathbf{0}$, $\mathbf{b} \leftarrow \mathbf{0}$, $\mathbf{e} \leftarrow \mathbf{0}$, $\mathbf{w} \leftarrow \mathbf{0}$
$\mathbf{B} \leftarrow \mathbf{I}$, $\mathbf{C} \leftarrow \mathbf{I}$, $\bar{\boldsymbol{\nu}} \leftarrow \mathbf{0}$, $\mathbf{c} \leftarrow \mathbf{1}$
$p = 0.1$, $\eta = 10^{-4}$, $\beta = 0.001$, $c_{\text{max}} = 1.0$
$\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
$a \leftarrow$ GetOptimisticAction($\boldsymbol{\phi}_{s,\cdot}$)
**repeat**
    Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
    $a' \leftarrow$ GetOptimisticAction($\boldsymbol{\phi}_{s',\cdot}$)
    $\delta \leftarrow r + (\gamma\boldsymbol{\phi}_{s',a'} - \boldsymbol{\phi}_{s,a})^\top \mathbf{w}$
    $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \boldsymbol{\phi}_{s,a}$
    $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta r\mathbf{e}$
    $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\mathbf{e}(\boldsymbol{\phi}_{s,a} - \gamma\boldsymbol{\phi}_{s',a'})^\top$
    ▷ Update $\mathbf{B} \approx \mathbf{A}^{-\top}$
    $\alpha = \min\left\{1.0, \frac{0.01}{||\mathbf{A}||_F^2 ||\boldsymbol{\phi}_{s,a}||_2^2 + 1.0}\right\}$
    $\mathbf{B} \leftarrow \mathbf{B} - \alpha\mathbf{A}(\mathbf{A}^\top\mathbf{B}\boldsymbol{\phi}_{s,a} - \boldsymbol{\phi}_{s,a})\boldsymbol{\phi}_{s,a}^\top$
    ▷ Update $\mathbf{C}$
    $\bar{\boldsymbol{\nu}} \leftarrow (1-\beta)\bar{\boldsymbol{\nu}} + \beta\delta\mathbf{e}$
    $\mathbf{a} \leftarrow \mathbf{B}^\top\bar{\boldsymbol{\nu}}$
    temp $= c_{\text{max}}$
    $c_{\text{max}} = \max(c_{\text{max}}, \mathbf{a}_1^2, \ldots, \mathbf{a}_d^2)$
    **if** temp $\neq c_{\text{max}}$ **then**                                        ▷ Adjust initialization
        $\mathbf{C}_{ii} \leftarrow \mathbf{C}_{ii} + \mathbf{c}_i(c_{\text{max}} - \text{temp}), \forall i$
    **for** $i$ such that $\mathbf{e}_i \neq 0$ **do**
        $\mathbf{c}_i = \mathbf{c}_i(1-\beta)$
        **for** $j$ such that $\mathbf{e}_j \neq 0$ **do**
            $\mathbf{C}_{ij} \leftarrow (1-\beta)\mathbf{C}_{ij} + \beta\mathbf{a}_i\mathbf{a}_j$
    ▷ Update $\mathbf{w}$
    $\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{B}^\top + \eta\mathbf{I})(\mathbf{b} - \mathbf{A}\mathbf{w})$
    $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{\mathbf{s}',a'}$ and $a \leftarrow a'$
**until** agent done interaction with environment
```

Global Variance UCB

We also adapt the bounds of Corollary 1 which assumes the noise in value estimates to be i.i.d. variables to design another algorithm called Global Variance-UCB (GV-UCB). To estimate a global variance σ^2, it is possible that the noise may not be 0-mean during the learning process. We account for this by estimating mean of ν_t as well. We know $\nu_t \sim \mathcal{N}(\bar{\nu}_t, \sigma_t^2)$. Therefore:

$$\begin{aligned} \bar{\nu}_{t+1} &= E[r_{t+1}] - E[\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}]^\top \mathbf{w}_t, \\ \bar{\nu^2}_{t+1} &= E[r_{t+1}^2] - 2E[r_{t+1}(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})]^\top \mathbf{w}_t \\ &\qquad + \mathbf{w}_t^\top E[(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^\top]\mathbf{w}_t. \end{aligned}$$

These expected values are maintained incrementally. Utilizing this, $\sigma_{t+1}^2 = \bar{\nu^2}_{t+1} - \bar{\nu}_{t+1}^2$. We refer to Global variance UCB as GV-UCB. The pseudocode for GV-UCB given in Algorithm 4.

Algorithm 3 GetOptimisticActionGlobal($\phi_{s,\cdot}$)

```
$u_a \leftarrow \sigma\sqrt{\left(1+\frac{1}{p}\right)\phi_{s,a}^\top \mathbf{C}\phi_{s,a}} \quad \forall a \in \mathcal{A}$
$a = \text{argmax}_{a\in\mathcal{A}}\, \phi_{s,a}^\top \mathbf{w} + u_a$
**return** $a$
```

Algorithm 4 GV-UCB(λ)

```
$\mathbf{A} \leftarrow \mathbf{0}$, $\mathbf{b} \leftarrow \mathbf{0}$, $\mathbf{e} \leftarrow \mathbf{0}$, $\mathbf{w} \leftarrow \mathbf{0}$,
$\mathbf{B} \leftarrow \mathbf{I}$, $\mathbf{C} \leftarrow \mathbf{I}$, $\bar{\mathbf{z}} \leftarrow \mathbf{0}$
$p = 0.01$,$\eta = 10^{-4}$, $\beta = 0.001$
$\sigma = 1.0$, $\bar{r} = 0.0$, $\bar{r^2} = 100.0$, $\bar{\mathbf{d}} \leftarrow \mathbf{0}$, $\bar{\mathbf{d}_\mathbf{r}} \leftarrow \mathbf{0}$, $\bar{\mathbf{D}} \leftarrow \mathbf{0}$
$\phi_{s,\cdot} \leftarrow$ initial state-action features, for any action
$a \leftarrow$ GetOptimisticActionGlobal($\phi_{s,\cdot}$)
**repeat**
    Take action $a$ and observe $\phi_{s',\cdot}$ and $r$, and $\gamma$
    $a' \leftarrow$ GetOptimisticActionGlobal($\phi_{s',\cdot}$)
    $\delta \leftarrow r + (\gamma\phi_{s',a'} - \phi_{s,a})^\top \mathbf{w}$
    $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \phi_{s,a}$
    $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta r\mathbf{e}$
    $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\mathbf{e}(\phi_{s,a} - \gamma\phi_{s',a'})^\top$
    ▷ Update $\mathbf{C}$
    $\bar{\mathbf{z}} \leftarrow (1-\beta)\bar{\mathbf{z}} + \beta\mathbf{e}$
    $\mathbf{a} \leftarrow \mathbf{B}^\top\bar{\mathbf{z}}$
    **for** $i$ such that $\mathbf{e}_i \neq 0$ **do**
        **for** $j$ such that $\mathbf{e}_j \neq 0$ **do**
            $\mathbf{C}_{ij} \leftarrow (1-\beta)\mathbf{C}_{ij} + \beta\mathbf{a}_i\mathbf{a}_j$
    ▷ Update $\sigma$
    $\bar{r} \leftarrow (1-\beta)\bar{r} + \beta r$
    $\bar{r^2} \leftarrow (1-\beta)\bar{r^2} + \beta r^2$
    $\bar{\mathbf{d}} \leftarrow (1-\beta)\bar{\mathbf{d}} + \beta(\phi_{s,a} - \gamma\phi_{s',a'})$
    $\bar{\mathbf{d}_\mathbf{r}} \leftarrow (1-\beta)\bar{\mathbf{d}_\mathbf{r}} + \beta r(\phi_{s,a} - \gamma\phi_{s',a'})$
    $\bar{\mathbf{D}} \leftarrow (1-\beta)\bar{\mathbf{D}} + \beta(\phi_{s,a} - \gamma\phi_{s',a'})(\phi_{s,a} - \gamma\phi_{s',a'})^\top$
    $\bar{\nu} = \bar{r} - \bar{\mathbf{d}}^T\mathbf{w}$
    $\bar{\nu^2} = \bar{r^2} - 2\bar{\mathbf{d}_\mathbf{r}}^T\mathbf{w} + \mathbf{w}^\top\bar{\mathbf{D}}\mathbf{w}$
    $\sigma = \sqrt{\bar{\nu^2} - \bar{\nu}^2}$
    ▷ Update $\mathbf{w}$ and $\mathbf{B} \approx \mathbf{A}^{-\top}$
    $\alpha = \min\left\{1.0, \frac{0.01}{||\mathbf{A}||_F^2||\phi_{s,a}||_2^2+1.0}\right\}$
    $\mathbf{B} \leftarrow \mathbf{B} - \alpha\mathbf{A}(\mathbf{A}^\top\mathbf{B}\phi_{s,a} - \phi_{s,a})\phi_{s.a}^\top$
    $\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{B} + \eta\mathbf{I})(\mathbf{b} - \mathbf{A}\mathbf{w})$
    $\phi_{s,a} \leftarrow \phi_{\mathbf{s}',a'}$ and $a \leftarrow a'$
**until** agent done interaction with environment
```

4.2.3 UCLS-L: Estimating Upper-Confidence Bounds for Linear TD in Control

Just as UCLS utilizes the policy evaluation upper-bound of LSTD for control, with a slowly changing control policy, UCLS-L utilizes the policy evaluation upper-bound of linear TD for control. At each step, UCLS-L, given in Algorithm 6, uses a stochastic update to estimate mean action-values, and their corresponding contextual-variance estimates. These stochastic updates use fixed, and if necessary are different, step-sizes (α, and α_{var} respectively), instead of a closed-form solution as done by UCLS. The rate of change of the policy in UCLS-L is controlled by the step-size, unlike in UCLS which utilizes weighted forms of experience samples in $\mathbf{A}$ and $\mathbf{b}$. Therefore, UCLS-L can be sensitive to the step-sizes, but adapt more quickly to a changing feature-space. Further, in order to account for underestimates of variances, UCLS-L uses another vector $\mathbf{w}_{\text{varInit}}$, in a similar spirit as UCLS's retroactive initialization of covariance estimates. Additionally, as these upper-bounds are estimated incrementally, they can be quite loose, specifically so in the linear framework. Therefore, instead of choosing the best parameter p, we can choose a parameter $\bar{p} = \sqrt{1 + \frac{1}{p}}$: the loss of theoretical interpretation of the upper-bound is traded-off for better empirical performance.

Similarly, a linear variant of GV-UCB can also be derived.

Algorithm 5 GetOptimisticActionLinear($\boldsymbol{\phi}_{s,\cdot}$)

```
$u_a \leftarrow \sqrt{\left(1+\frac{1}{p}\right)\left((\boldsymbol{\phi}_{s,a}^\top \mathbf{w}_{\text{var}})^2 + ||\boldsymbol{\phi}_{s,a}||^2_{\mathbf{I}\mathbf{w}_{\text{varInit}}}\right)} \quad \forall a \in \mathcal{A}$
$a = \text{argmax}_{a\in\mathcal{A}}\, \boldsymbol{\phi}_{s,a}^\top \mathbf{w} + u_a$
**return** $a$
```

Algorithm 6 UCLS-L(λ)

```
$p = 0.1$, $\beta = 0.001$, $v_{\text{init}} = 1.0$, $\alpha = 0.01$, $\alpha_{\text{var}} = 0.1$
$\mathbf{w} \leftarrow \mathbf{0}$, $\mathbf{w}_{\text{var}} \leftarrow \mathbf{0}$, $\mathbf{w}_{\text{varInit}} \leftarrow \mathbf{1} * v_{\text{init}}$, $\mathbf{c} \leftarrow \mathbf{1}$
$\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
$a \leftarrow$ GetOptimisticActionLinear($\boldsymbol{\phi}_{s,\cdot}$)
**repeat**
    Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
    $a' \leftarrow$ GetOptimisticActionLinear($\boldsymbol{\phi}_{s',\cdot}$)
    $\delta \leftarrow r + (\gamma\boldsymbol{\phi}_{s',a'} - \boldsymbol{\phi}_{s,a})^\top \mathbf{w}$
    $\delta_{\text{var}} \leftarrow \delta + (\gamma\boldsymbol{\phi}_{s',a'} - \boldsymbol{\phi}_{s,a})^\top \mathbf{w}_{\text{var}}$
    $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \boldsymbol{\phi}_{s,a}$
    ▷ Update $\mathbf{w}_{\text{var}}$ and $\mathbf{w}_{\text{varInit}}$
    $\mathbf{w}_{\text{var}} \leftarrow \mathbf{w}_{\text{var}} + \alpha_{\text{var}}\delta_{\text{var}}\mathbf{e}$
    temp $= v_{\text{init}}$
    $v_{\text{init}} = \max(v_{\text{init}}, {\mathbf{w}_{\text{var}}}_1^2, \ldots, {\mathbf{w}_{\text{var}}}_d^2)$
    **if** temp $\neq v_{\text{init}}$ **then**                                        ▷ Adjust initialization
        ${\mathbf{w}_{\text{varInit}}}_i \leftarrow {\mathbf{w}_{\text{varInit}}}_i + \mathbf{c}_i(v_{\text{init}} - \text{temp}), \forall i$
    **for** $i$ such that $\mathbf{e}_i \neq 0$ **do**
        $\mathbf{c}_i = \mathbf{c}_i(1-\beta)$
        ${\mathbf{w}_{\text{varInit}}}_i \leftarrow (1-\beta) * {\mathbf{w}_{\text{varInit}}}_i, \forall i$
    ▷ Update $\mathbf{w}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$
    $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{\mathbf{s}',a'}$   and   $a \leftarrow a'$
**until** agent done interaction with environment
```

4.3 Evaluation of UCLS

We conducted several experiments to (1) investigate the benefits of UCLS's and UCLS-L's directed exploration against other methods that use confidence intervals for action selection, and (2) to contrast the advantage of contextual variance estimates (Theorem 1) over global variance estimates (Corollary 1) for directed exploration.

We compare to DGPQ (Grande et al., 2014), UCBootstrap (White and White, 2010), RLSVI (Osband et al., 2016b) and our extension of LSPI-Rmax to an incremental setting (Li et al., 2009). All the algorithms are proposed for online control using confidence intervals, or counting strategies, with linear function approximation. DGPQ is a method that utilizes two estimators: one built on Gaussian Processes to estimate confidence intervals, and the second a linear estimator to estimate optimistic values. UCBootstrap is a method that evaluates confidence intervals via statistical bootstrapping strategies, whereas RLSVI is a method that utilizes Bayesian Linear Regression to evaluate confidence intervals. LSPI-Rmax is a batch-based algorithm built on LSTD, that utilizes Rmax style bonuses to estimate direct exploration; here, we adapt the algorithm to be compatible to the incremental setting. We also include Sarsa with ϵ-greedy, with ϵ optimized over an extensive parameter sweep. Though ϵ-greedy is not a generally practical algorithm, particularly in larger worlds, we include it as a baseline.

We experiment with the following classical hard exploration problems: (1) Sparse Mountain Car (Sutton and Barto, 2018) - an episodic task where an agent needs to drive up a hill from the valley, and receives a reward only at the end of the episode, (2) Puddle World (Sutton and Barto, 2018) - an episodic task where an agent needs to learn to navigate to a goal while avoiding puddles which result in high negative rewards, and River Swim (Szita and Lorincz, 2008) - a standard continuing exploration benchmark where an agent faces the difficult choice between choosing to swim upstream to a place of high reward, or choosing to swim downstream to a more easily accessible low reward. More details regarding the algorithms, their pseudocode, the parameter sweeps, and descriptions of the domains can be found in the Appendix A.4.

We investigate a learning regime where the agents are allowed a fixed budget of interaction steps with the environment, rather than allowing a finite number of episodes of unlimited length. Our primary concern is sample-efficiency, and hence, early learning performance. Thus each experiment is restricted to 50,000 steps, with an episode cutoff (in Sparse Mountain Car and Puddle World) at 10,000 steps. We average over 100 runs in

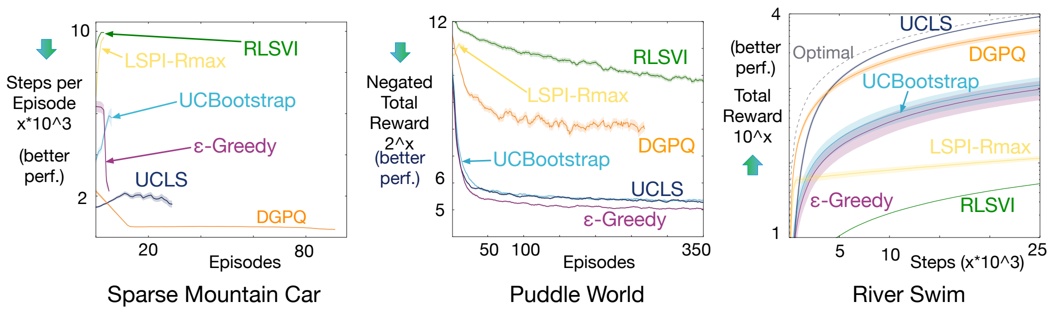

Figure 4.2: A comparison of speed of learning of directed exploration algorithms in Sparse Mountain Car, Puddle World and River Swim. In plots (a) and (b) lower on y-axis are better, whereas in (c) curves higher along y-axis are better. Sparse Mountain Car and Puddle World are episodic problems with a fixed experience budget. Note RLSVI did not show significant learning after 50,000 steps. The RLSVI result in Puddle World uses a budget of 1 million. The shaded region represents 95% confidence interval, and are mostly small as the results are averaged over an extensive number of runs.

River Swim and 200 runs for the other domains. The parameters for UCLS are fixed across the domains, whereas for the competitors are swept for each domain independently.

Lastly, all the algorithms except DGPQ use the same representation: (1) Sparse Mountain Car - 8 tilings of 8x8, hashed to a memory space of 512, (2) River Swim - 4 tilings of granularity 32, hashed to a memory space of 128, and (3) Puddle World - 5 tilings of granularity 5x5, hashed to a memory space of 128. DGPQ uses its own kernel-based representation with normalized state information.

4.3.1 Online Performance

Our first set of results compare the online performance of UCLS to other competitors. Figure 4.2 shows the early learning results across all three domains. In all three domains UCLS achieves the best final performance. In Sparse Mountain Car, UCLS learns faster than the other methods, while in River Swim DGPQ learns faster initially. UCBootstrap and UCLS learn at a similar rate in Puddle World, which is a cost-to-goal domain. UCBootstrap, and bootstrapping approaches generally, can suffer from insufficient optimism, as they rely on sufficiently optimistic or diverse initialization strategies (White and White, 2010; Osband et al., 2016a). LSPI-Rmax and RLSVI do not perform well in any of the domains. DGPQ does not perform as well as UCLS in Puddle World, and exhibits high variance compared with the other methods. In Puddle World, UCLS goes on to finish 1200 episodes in the alloted budget of steps, whereas in River Swim both UCLS and DGPQ get close to the optimal policy by the end of the experiment. The DGPQ algorithm uses the maximum

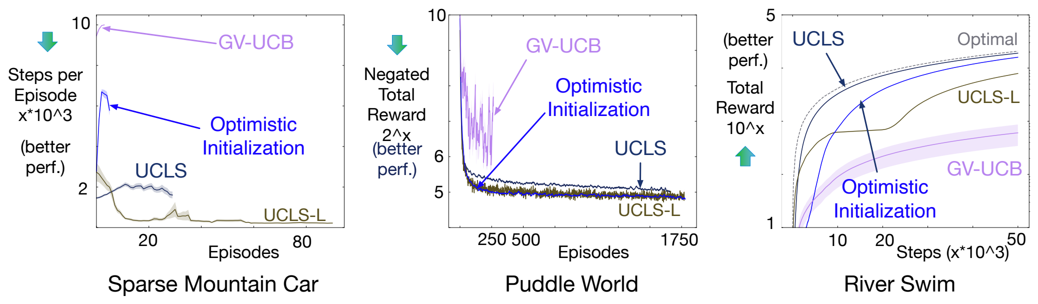

Figure 4.3: A comparison of speed of learning of UCLS in Sparse Mountain Car, Puddle World and River Swim to algorithms similar to it and Optimistic Initialization. In plots (a) and (b) lower on y-axis are better, whereas in (c) curves higher along y-axis are better. Sparse Mountain Car and Puddle World are episodic problems with a fixed experience budget. The shaded region represents 95% confidence interval, and are mostly small as the results are averaged over an extensive number of runs.

reward (Rmax) to initialize the Gaussian processes. In Sparse Mountain Car this effectively converts the problem back into the traditional -1 per-step formulation, explaining the good learning performance. In Puddle World an initialization of 0 corresponds to optimistic initialization, and therefor ϵ-greedy performs well.

In our second set of results we compare UCLS to other variants of itself, alongwith the simple exploration strategy Optimistic Initialization. We include UCLS-L, the linear complexity variant of UCLS, and GV-UCB, the algorithm derived using Corollary 1 result. For both UCLS-L and GV-UCB, we sweep the parameter used to scale the uncertainty estimate, along with the learning rates α and α_{var} for UCLS-L. The initialization for Optimistic Initialization is also swept from a wide range.

The results are presented in Figure 4.3. Sarsa with optimistic initialization performs remarkably well in these domains, except in Sparse Mountain Car.

UCLS-L does reasonably well in all the domains. While it experiences more regret in Puddle World, and River Swim during early learning, by the end of the steps budget, it learns the optimal policy. In Sparse Mountain Car, surprisingly, UCLS-L learns much faster and a better policy than UCLS. This can be attributed to the fact that the parameter p in UCLS was not swept, whereas in UCLS-L we did sweep to find the best parameter to scale the variance estimate. As the domain is a sparse-reward domain, the variance estimates play a significant role in influencing exploratory behaviour, and therefore optimizing for p would improve UCLS' performance. Nonetheless, these results show UCLS-L to be a promising algorithm for linear complexity based control. With the loss of contextual variance estimates GV-UCB

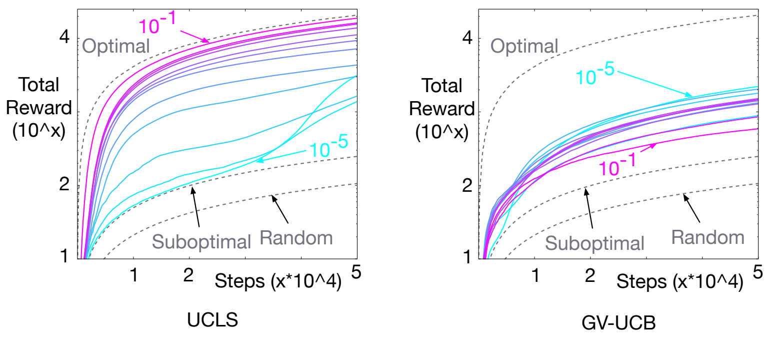

Figure 4.4: The effect of the confidence parameter p on the policy, in River Swim, using context-dependent variance (UCLS) and global variance (GV-UCB). The values for p are $\{10^{-5}, [1, 2, \ldots, 9] \times 10^{-3}, 10^{-2}, 10^{-1}\}$.

explores the complete state space more thoroughly, and therefore performs poorly.

4.3.2 Sensitivity to p and Benefits of Contextual Uncertainty

Next we investigated the impact of the confidence level $1 - p$, on the performance of UCLS in River Swim. The confidence interval radius is proportional to $\sqrt{1 + 1/p}$; smaller p should correspond to a higher rate of exploration. In Figure 4.4, smaller p resulted in a slower convergence rate, but all values eventually reach the optimal policy.

To investigate the benefit using contextual variance estimates over global variance estimates within UCLS, we also show the effect of various p values on the performance of GV-UCB. While UCLS converges to the optimal policy with different values of p, albeit at different rates, GV-UCB on the other hand, results in significant over-estimates of variance across the state space, resulting in poor online performance. The explicit upper confidence bound given by UCLS does not suffer from this, and sufficiently explores the domain to converge to an optimal policy without excessively exploring. For regions where there is low variance, the upper-confidence-bound converges more quickly to zero, whereas it remains higher in regions of uncertainty. Therefore, contextual variance estimates provide the flexibility of variable convergence based on the variance of the region, and global variance estimates decay too slowly.

To empirically reinforce the utility of contextual confidence interval radius (CIR) over global CIR, we evaluate the policies obtained by UCLS and GV-UCB after 50,000 learning steps in River Swim and present the results in Figure 4.5. In the left plot it can be seen that UCLS(M) and UCLS(M+CIR) perform almost as well as the optimal policy, whereas both versions of GV-UCB are still sub-optimal in many parts of the state space. Additionally, the

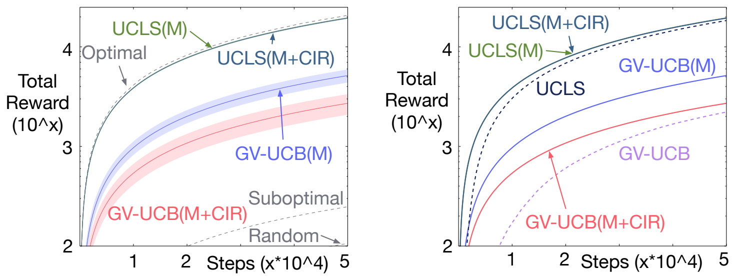

Figure 4.5: Policy evaluation plots comparing variations of final policy obtained by UCLS ($p = 0.1$) and GV-UCB ($p = 10e{-}5$) after 50,000 learning steps in River Swim. Policies with (M) indicate greedy policy w.r.t. mean estimates, whereas policies with (M+CIR) indicate greedy policies w.r.t. (mean + CIR) estimates.

overlap of UCLS(M) and UCLS(M+CIR) indicates that contextual CIR fades faster than global CIR, and is a more data-efficient exploration strategy. The right plot helps contrast the final policies obtained to the actual control policy used during learning, indicated by just GV-UCB and UCLS.

4.4 Summary

This chapter addressed a long-standing problem in policy evaluation: of estimating uncertainty in the values estimated by an algorithm. The estimated uncertainties provide high probability bounds on the range of the optimal value estimate of any state-action for the policy. These uncertainties are characterized under two situations: (1) when uniform noise is assumed across the state-action space, and (2) when the noise is allowed to be context dependent with respect to the representation of the state-action pairs.

Given these uncertainty estimates for a policy, they can be utilized within an approximate policy iteration framework for control. But, in practice, good online policies can be learned even under imperfect policy evaluation. Therefore, we build on generalized policy iteration to present an incremental strategy that utilizes these learned uncertainty estimates and directs exploration in an on-policy control algorithm. Although the algorithm utilizes the empirical covariance estimated in place of the true covariance while estimating uncertainties, we empirically show that (1) it performs well, if not better, when compared to other approaches to exploration based on upper-confidence bounds, and (2) the utility of its uncertainty estimates that do not make a simplifying assumption of uniform noise across the state-action space.

Chapter 5

Representations for Online Control

In this chapter we shift focus from how to explore for sample-efficient online control to what representations improve sample-efficiency of learning in online control. Representations play a critical role in promoting the sample-efficiency of online learning algorithms, specifically in reinforcement learning, even with linear function approximators. This is because, the structure of generalization encoded by the representations dictate the targets that are used for learning, via bootstrapping, in online reinforcement learning algorithms.

Many algorithms learn this generalization structure implicitly, online, by building on the breakthrough work that utilizes neural networks for feature construction (Mnih et al., 2015). However, these algorithms incorporate strategies such as experience replay and target networks to aid online learning. Such strategies not only require additional storage space for an algorithm, but they slow down learning as they use outdated value estimates, via target networks, to stabilize online learning. Alternatively, learning representations offline, which are effective with fast-learning online reinforcement learning algorithms under linear function approximation, is still a challenging problem.

There has been a lot of work on learning representations that aid value estimation for a fixed policy (Mahadevan and Maggioni, 2007; Parr et al., 2008; Konidaris et al., 2011). While such representations have been successful for policy evaluation, their success as applied to the control setting of reinforcement learning has been limited compared to the success of a particular representation in the control setting, *tile coding* (Sutton and Barto, 2018). This is possibly because a significantly useful attribute of tile coding has been its ability to facilitate online policy improvement with stable learning targets, even under the extreme case of learning purely from incremental samples, as is done in the SARSA algorithm.

As tile coding is a hand-designed representation, it is not particularly scalable. Although recent work has utilized it a pre-processing step towards improving sample-efficiency of online control with deep neural networks (Ghiassian et al., 2020), the need to hand-design the tile coder used in the pre-processing step still exists. Alternatively, this chapter proposes a representation learning strategy that is designed to capture the sample-efficiency improving characteristic of tile coding, removing the need for any hand-designing.

We will first describe desirable properties of representations for sample-efficient online control, and then present a method for learning them. Thereafter, we will demonstrate the empirical benefits of the proposed approach, and analyze its properties along with reasons for the obtained benefits. We will compare this representation learning strategy with other strategies that may result in representation spaces with similar properties, before concluding with experiments that evaluate if the learned representations benefit exploration algorithms.

Contributions

Contribution 1: A method to learn representations that improve sample-efficiency of incremental online reinforcement learning algorithms. In particular, this method identifies a key characteristic of hand-designed representations that have been successful in online control with linear function approximation, and designs a regularizer to encode such a characteristic in learned representations.

Contribution 2: An empirical evaluation of the proposed representation learning algorithm which: (1) evaluates its effectiveness for online control, and analyses the reason for it, (2) compares it to other similar representation learning strategies, and (3) evaluates how the property encoded by the representation learning algorithm can aid exploration in online reinforcement learning algorithms.

5.1 Locality in Control

In control, an agent updates its value function at every timestep. While model-based approaches can be incorporated to improve sample-efficiency (Sutton et al., 2012), in the purely incremental setting an agent can update its learning with just the sample it sees at a particular timestep t, $\{S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1}\}$. Such incremental updating is an important characteristic of successful control algorithms like SARSA (Rummery and Niranjan, 1994) which use temporal difference updates, that is, the target for timestep t, $Y_{t+1} = R_{t+1} + \gamma_{t+1} Q(S_{t+1}, A_{t+1})$ where A_{t+1} is the action the agent will take in the

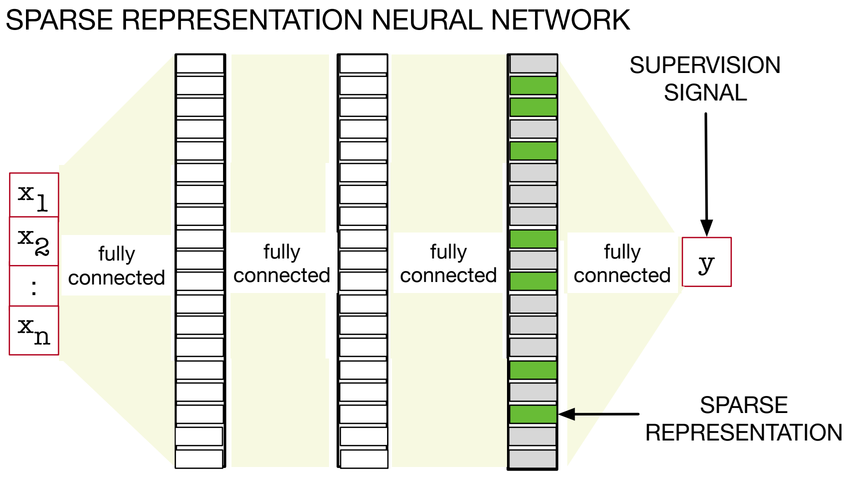

Figure 5.1: A neural network with dense connections producing a sparse representation: Sparse Representation Neural Network (SR-NN). The green squares indicate active (non-zero) units, making a sparse last hidden layer where only a small percentage of units are active. This contrasts a network with sparse connections—which is often also called sparse. Sparse connections remove connections between nodes, but are likely to still produce a dense representation.

next state – also called on-policy, as the agent is learning about the policy it is using to behave.

A particular characteristic of linear function approximation scheme, where algorithms like SARSA have been effective, is that the representations are all local, like tile coding (Sutton, 1996). Such local representations enable the agent to learn predictions for that local region, without affecting the non-local regions with respect to the function. For instance, if an agent is learning to drive, it can benefit from a representation that reflects its context – such as, is it driving in a residential area or an interstate – as likely learning by distinguishing between the two is simpler than learning based on a representation that conflates the two. Additionally, such a representation can help prevent forgetting or interference (McCloskey and Cohen, 1989; French, 1991), by only updating local weights, as opposed to dense representations where any update would modify many weights – thereby providing more stable targets for temporal-difference based methods like SARSA.

When such representations are learned, using classes of function approximators like neural networks, it is also important that the learned representations generalize effectively. While these representations do not need to have binary features like tile coding, it is beneficial that the representation for an input is distributed across multiple features or attributes, promoting generalization in the compact representation. This property has been described in the literature as a representation that is distributed (Bengio, 2009; Bengio et al., 2013).

Such properties can be well captured by learning sparse representations: those for which only a few features are active for a given input (Figure 5.1). Enforcing sparsity promotes identification of key attributes, because it encourages the input to be well-described by a small subset of attributes. Sparsity, then, promotes locality, because local inputs are likely to share similar attributes (similar activation patterns) with less overlap to non-local inputs. In fact, many hand-crafted features are sparse representations, including tile coding (Sutton, 1996; Sutton and Barto, 2018), radial basis functions and sparse distributed memory (Kanerva, 1988; Ratitch and Precup, 2004). Other useful properties of sparse representations — which can be seen as projecting data into a higher-dimensional space — include invariance (Goodfellow et al., 2009; Rifai et al., 2011); decorrelated features per instance (Földiák, 1990); improved computational efficiency for updating weights in the predictor, as only weights corresponding to active features need to be updated; and enabling linear separability in the high-dimensional space (Cover, 1965), which facilitates the learning of a simple linear predictor. Further, such sparse, distributed representations have been observed in the brain (Olshausen and Field, 1997; Quian Quiroga and Kreiman, 2010; Ahmad and Hawkins, 2015).

5.2 Distributional Regularizes for Sparsity

In this section, we describe how to use Distributional Regularizers to learn sparse representations with neural networks. The idea was originally introduced for neural networks with Sigmoid activations in an unpublished set of notes Ng (2011), but had not been systematically explored. We also introduce the idea of a Set Distributional Regularizer, which enables sparse representations to be learned with neural networks.

The goal of using Distributional Regularizers is to encourage the distribution of each hidden node—across samples—to match a desired target distribution. In a neural network, we can view the hidden nodes, $Y_1, \ldots, Y_d$, as random variables, with randomness due to random inputs. Each of these random variables Y_j has a distribution $p_{\hat{\beta}_j(\theta)}$, where the parameters $\hat{\beta}_j(\theta)$ of this distribution are induced by the weights $\boldsymbol{\theta}$ of the neural network:

$$p_{\hat{\beta}_j(\theta)}(y) = \int_{s \in \mathcal{S}} p(s) p(\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(s) = y) ds.$$

This provides a distribution over the values for the feature $\boldsymbol{\phi}_{j,\boldsymbol{\theta}}(s)$, across inputs s. A Distributional Regularizer is a KL divergence $KL(p_\beta || p_{\hat{\beta}_j(\boldsymbol{\theta})})$ that encourages this distribution to match a desired target distribution p_β with parameter β.

Such a regularizer can be used to encourage sparsity, by selecting a target distribution that has high mass or density at zero. Consider a Bernoulli distribution for activations, with $Y_j \in \{0, 1\}$. Using a Bernoulli target distribution with $\beta = 0.1$, giving $p_\beta(Y = 1) = 0.1$, encodes a desired activation of 10%. As another example, for continuous nonnegative Y_j, the target distribution can be set to an exponential distribution $p_\beta(y) = \beta^{-1}\exp(-y/\beta)$, which has highest density at zero with expected value β. Setting $\beta = 0.1$ encourages the average activation to be 0.1 and increases density on $y = 0$.

The efficacy of this regularizer, however, is tied to the parameterization of the network, which should match the target distribution. Therefore, for a ReLU activation, which has a range $[0, \infty)$ an exponential distribution is suitable, but for a Sigmoid activation, giving values between $[0, 1]$, a Bernoulli is reasonably appropriate. Additionally, the parametrization should be able to set activations to zero. The ReLU activation naturally enables zero values (Glorot et al., 2011), by pushing activations to negative values. The addition of a Distributional Regularizer simply encourages this natural tendency, and is more likely to provide sparse representations. Activations under Sigmoid and tanh are more difficult to encourage to zero because they require highly negative input values or input values exactly equal to 0.5, respectively, to set the hidden node to zero. For these reasons, we advocate for ReLU for the sparse layer, with an exponential target distribution.

Finally, we modify this regularizer to provide a Set Distributional Regularizer, which does not require an exact level of sparsity to be achieved. It can be difficult to choose a precise level of sparsity, making the Distributional Regularizer prone to misspecification. Rather, the actual goal is typically to obtain *at least* some level of sparsity. For this modification, we specify that the distribution should match any of a set of target distributions Q_β, giving a *Set KL*: $\min_{p \in Q_\beta} KL(p||p_{\hat{\beta}_j(\boldsymbol{\theta})})$. Generally, this Set KL can be hard to evaluate. However, as we show below, it corresponds to a simple clipped KL-divergence for certain choices of Q_β, importantly including for exponential distributions where $Q_\beta = \{p_{\tilde{\beta}} \,|\tilde{\beta} \leq \beta\}$.

Theorem 2 (Set KL as a Clipped-KL). *Let p_η be a one-dimensional exponential family distribution with the natural parameter η, $B = [\eta_1, \eta_2]$ be a convex set in the natural parameter space and $Q_B = \{p_\eta : \eta \in B\}$. Then the Set KL divergence*

$$SKL(Q_B||p_\eta) \overset{\text{def}}{=} \min_{p \in Q_B} KL(p||p_\eta),$$

is (a) non-negative (b) convex in η and (c) corresponds to a simple clipped form

$$SKL(Q_B||p_\eta) = \begin{cases} KL(p_{\eta_2}||p_\eta), & \textit{if } \eta > \eta_2, \\ KL(p_{\eta_1}||p_\eta), & \textit{if } \eta < \eta_1, \\ 0, & \textit{else.} \end{cases}$$

Proof: For exponential families, the KL divergence correspond to a Bregman divergence (Banerjee et al., 2005):

$$KL(p_{\eta_1}||p_\eta) = D_F(\eta||\eta_1),$$

for a convex potential function F that depends on the exponential family. Hence, we have

$$SKL(Q_B||p_\eta) = \arg\min_{\tilde{\eta}\in B} D_F(\eta||\tilde{\eta}).$$

If $\eta \in B$, this minimum over Bregman divergences is clearly zero. If $\eta < \eta_1$ and $\eta > \eta_2$, we have to consider the minimization. The Bregman divergence is not necessarily convex in the second argument. Instead, we can rely on convexity of the set B. Taking the derivative of $D_F(\eta||\tilde{\eta})$ wrt $\tilde{\eta}$, we get

$$\begin{aligned} \frac{d}{d\tilde{\eta}} D_F(\eta||\tilde{\eta}) &= \frac{d}{d\tilde{\eta}} \left[F(\eta) - F(\tilde{\eta}) - (\eta - \tilde{\eta}) \frac{d}{d\tilde{\eta}} F(\tilde{\eta}) \right] \\ &= -\frac{d}{d\tilde{\eta}} F(\tilde{\eta}) + \frac{d}{d\tilde{\eta}} F(\tilde{\eta}) - (\eta - \tilde{\eta}) \frac{d^2}{d\tilde{\eta}^2} F(\tilde{\eta}) \\ &= -\frac{d^2}{d\tilde{\eta}^2} F(\tilde{\eta})(\eta - \tilde{\eta}). \end{aligned}$$

Now because F is convex, $-\frac{d^2}{d\tilde{\eta}^2} F(\tilde{\eta})$ is always negative. The derivative, then, is negative when $\tilde{\eta} < \eta$, indicating $\tilde{\eta}$ should be increased to decrease $D_F(\eta||\tilde{\eta})$. Similarly, when $\tilde{\eta} > \eta$, the derivative is positive, indicating $\tilde{\eta}$ should be decreased to decrease $D_F(\eta||\tilde{\eta})$. This derivative, then, points $\tilde{\eta}$ to the boundaries when $\eta \notin B$, respectively to the boundary points closest to η. ■

Corollary 2 (SKL for Exponential Distributions). *For p_β an exponential distribution, with natural parameter $\eta = -\beta^{-1}$, and $B = (0, \beta]$, then*

$$SKL(Q_B||p_{\hat{\beta}}) = \begin{cases} \log\hat{\beta} + \frac{\beta}{\hat{\beta}} - \log\beta - 1, & \textit{if } \hat{\beta} > \beta, \\ 0, & \textit{else.} \end{cases}$$

Proof: For $B = (0, \beta]$ and $\eta = -\beta^{-1}$, $B = (-\infty, -1/\beta], \eta \in B$. For exponential distribution

$$KL(p_\eta||p_{\hat{\eta}}) = \log(-\eta) + \frac{\hat{\eta}}{\eta} - \log(-\hat{\eta}) - 1.$$

Therefore,

$$SKL(Q_B||p_{\hat{\eta}}) = \begin{cases} \log(-\eta) + \frac{\hat{\eta}}{\eta} - \log(-\hat{\eta}) - 1, & \text{if } \hat{\eta} > -1/\beta, \\ 0, & \text{else.} \end{cases}$$

■

We use the SKL in Corollary 2, to encode a sparsity level of at least β, rather than exactly β, for the last layer in a two-layer neural network with ReLU activations. We call the representations learned with this regularizer as SR-NN, which stands for Sparse Representation learned by a Neural Network.

The overall loss function is

$$J_{SR-NN}(\theta) = J(\theta) + \lambda_{SKL} \sum_{i}^{d} SKL(Q_B||p_{\hat{\eta}_i}),$$

where $J(\theta)$ is the vanilla objective function, d is the dimension of representation, and λ_{SKL} controls the weight of the sparsity regularization. We include pseudocode for optimizing the regularized objective J_{SR-NN} with the SKL for ReLU activation with Exponential Distributions in Algorithm 7. This algorithm is not used online. Instead, we learn the representation for a batch of data, offline. The pseudocode is given for this offline batch setting.

Algorithm 7 Optimizing the regularized objective for SR-NN

```
1: Initialize neural networks weights based on He initialization (He et al., 2015): for each
   layer $l$ and each element $ij$ of the weight matrix $\mathbf{W}^{(l)}_{ij} \sim \mathcal{N}(0, \frac{2}{n_l})$ and $\mathbf{b}^{(l)} = \mathbf{0}$ where
   $n_l$ the number of input nodes for layer $l$.
2: while not converge to a minimum do
3:     Draw $m$ i.i.d. samples $\{y_1, ..., y_m\}$ from the true data distribution, and do the for-
       ward pass for the neural network
4:     For $j = 1, ..., k$, compute $\hat{\beta}_j = \sum_{i=1}^{m} y_{ij}/m$ and the gradient:
           $\frac{\partial KL(p_\beta||p_{\hat{\beta}_j})}{\partial \hat{\beta}_j} = (\frac{1}{\hat{\beta}_j} - \frac{\beta}{\hat{\beta}_j^2})\mathbb{1}[\hat{\beta}_j > \beta]$
5:     Update each weight $\boldsymbol{\theta} \in \{\forall l, \mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$ with the gradient:
           $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \lambda_{KL} \sum_{j=1}^{k} \frac{\partial KL(p_\beta||p_{\hat{\beta}_j})}{\partial \hat{\beta}_j} \frac{\partial \hat{\beta}_j}{\partial \boldsymbol{\theta}}$
```

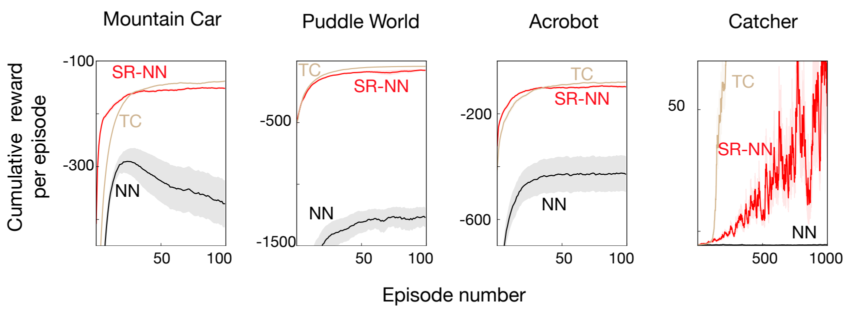

Figure 5.2: Learning curves for SARSA(0) comparing SR-NN, Tile Coding and vanilla NN in the four domains. The shaded region represents 95% confidence interval. In all the domains, higher y-axis is better, and the x-axis denotes the episode number.

5.3 The Utility of Sparsity for Control

We now look at the empirical benefits of such learned sparse representations over just learned representations. We compare SR-NN to tile coding, a static representation, as a baseline to compare to, as it known to perform very well in the benchmark RL domains we experiment with (Sutton and Barto, 2018).

We evaluate control performance on four benchmark domains: Mountain Car, Puddle World, Acrobot and Catcher. All domains are episodic, with discount set to 1 until termination. We choose these domains because they are well-understood. A priori, it would be expected that a standard action-value method, like SARSA, with a two-layer neural network, should be capable of learning a near-optimal policy in all domains.

The experimental set-up is as follows. To extract a representation with a neural network, to be used for control, we pre-train the neural network on a batch of data with a mean-squared temporal difference error (MSTDE) objective and the applicable regularization strategies. The training data consists of trajectories generated by a fixed policy that explores much of the space in the various domains. For the SR-NN, we use our distributional regularization strategy. This learned representation is then fixed, and used by a fully incremental SARSA(0) agent for learning a control policy, where only the weights $\mathbf{w}$ on the last layer are updated. The meta-parameters for the representation along with the agent were swept in a wide range, and chosen based on control performance. The aim is to provide the best opportunity to the more sensitive regular feed-forward network (NN) to learn on these problems.

We choose this two-stage training regime to remove confounding factors in difficulties of training neural networks incrementally. Our goal here is to identify if a sparse representation can improve control performance, and if so, why. The networks are trained with an

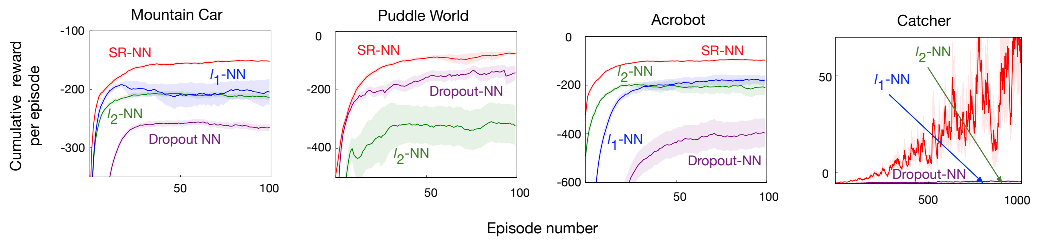

Figure 5.3: Learning curves for SARSA(0) comparing SR-NN to the regularized representations. The shaded region represents 95% confidence interval. All representations except ℓ_1-NN in Puddle World could reach goals more than 70 times out of 100. ℓ_1 does poorly in Puddle World, and is not visible.

objective for learning values, on a large batch of data generated by a policy that covers the space; the learned representations are capable of representing the optimal policy. We investigate their utility for the fully incremental learning framework. Additional details about the domains, ranges and objectives are provided in Appendix B.2.

The learning curves for the four domains, with Tile-Coding (TC), SR-NN and NN, are shown in Figure 5.2. Both SR-NN and NN used two-layers, of size $[32, 256]$, with ReLU activations. The NNs performs surprisingly poorly, in some case increasing and then decreasing in performance (Mountain Car), and in others failing altogether (Catcher). In all the benchmark RL domains, the baseline sparse representation, TC, performs well, as expected. Specifically in Catcher, TC learns a close-to-optimal policy as the representation is powerful. The learned SR-NN performs as well as TC in all domains except Catcher, where it is effective for learning, whereas NN performs really poorly in all domains, and does not learn anything in Catcher. Both SR-NN and NN representations were trained in the same regime, with similar representational capabilities. Yet, the sparsity of SR-NN enables the SARSA(0) agent to learn, where the regular feed-forward NN does not. We investigate this effect further in the next sets of experiments, to better understand the phenomenon.

To determine if the main impact of the sparse representation is simply from regularization, preventing overfitting, we tested several regularization strategies for the neural network. These include ℓ_2 and ℓ_1 on the weights of the network (ℓ_2-NN and ℓ_1-NN respectively) and Dropout on the activation (Srivastava et al., 2014) (Dropout-NN). The ℓ_1 regularizer actually encourages weights to go to zero, reducing the number of connections, but does not necessarily provide a sparse representation. In Figure 5.3, we can see that regularization is unlikely to account for the improvements in control. SR-NN performs well across all domains, whereas none of the regularization strategies consistently perform well. ℓ_1-NN and ℓ_2-NN perform well in Mountain Car during early learning, but fail in other do-

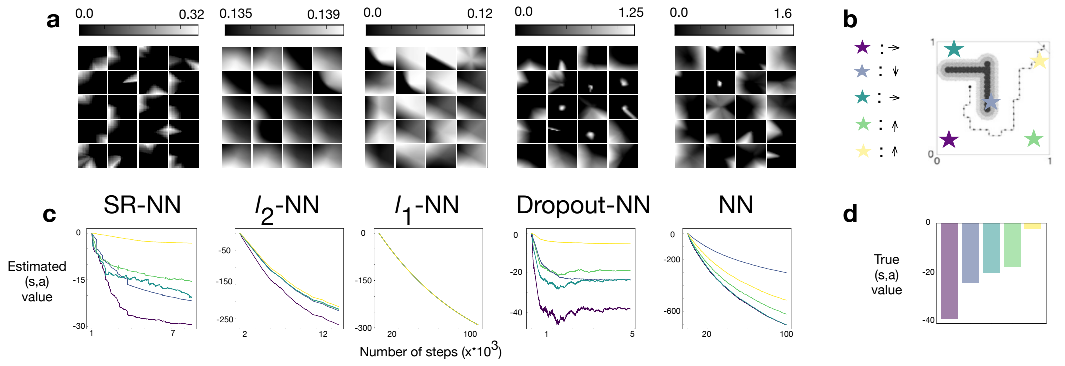

Figure 5.4: A study in Puddle World to investigate the effect of locality during on-policy control. (**a**) The activation maps for 20 randomly chosen neurons for different representations - each cell in the heatmap corresponds to the complete Puddle World state-space. The activation maps, and magnitude of activation of SR-NN are visibly sparser, and lower, when compared to Dropout and NN. ℓ_1 and ℓ_2 are quite dense in terms of activation area, whereas the magnitudes are really low - due to regularization of the network weights. (**b**) A visualization of the domain, denoting the selected state-action pairs used in the analysis. (**c**) The estimated state-action values for the selected configurations during on-policy control with SARSA(0) ($\epsilon = 0.1$), while utilizing the specific representation of interest. (**d**) The true state-action values for the selected configurations with an $\epsilon = 0.1$-optimal policy, estimated from 100k Monte Carlo rollouts.

mains. Dropout-NN performs poorly in all domains except Puddle World. Interestingly, in this one domain, Dropout-NN appears to have learned a sparse representation, based on the heatmap shown in Figure 5.4. It has been observed that Dropout can at times learn sparse representations (Banino et al., 2018), but not consistently, as corroborated here.

We next investigate the hypothesis that locality is preventing catastrophic interference. We first investigate the locality of the representations, as well as examining the bootstrap values over time. We show results for Puddle World first, as it is an interpretable two-dimensional domain, and then show the results for Mountain Car.

Figure 5.4(a) shows the activation map of randomly selected hidden neurons with the different networks in Puddle World. We can see that each hidden neuron in SR-NN only responds to a local region of the input space, while some hidden neurons in NN respond to a large part of the space. Consequently, when one state is updated in a part of the space with the NN representation, it is more likely to significantly shift the values in other parts of the space, as compared to the more local SR-NN. The ℓ_2-NN, and ℓ_1-NN representations do not exhibit any discernible locality properties. Dropout-NN does achieve some degree of locality in this domain, as mentioned earlier.

To show the interference (or lack of interference) of bootstrap targets used during con-

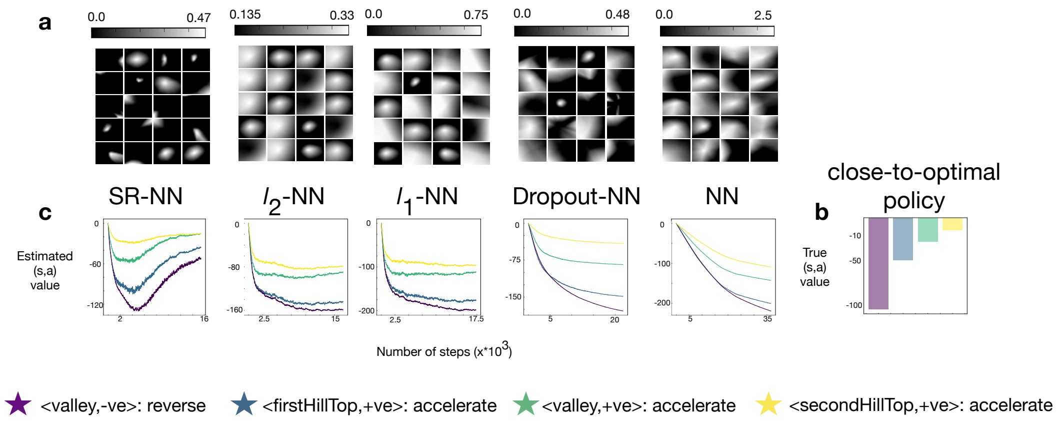

Figure 5.5: A study in Mountain Car to investigate the effect of locality during on-policy control. The chosen 4 state-action pairs are denoted using the following format in the legend – ⟨car-position,car-velocity⟩:action. (**a**) The activation maps for 20 randomly chosen neurons for different representations - each cell in the heatmap corresponds to the complete Mountain Car state-space. (**b**) The true state-action values for the selected configurations with an $\epsilon = 0.1$-optimal policy, estimated from 10k Monte Carlo rollouts. (**c**) The estimated state-action values for the selected configurations during on-policy control with SARSA(0) ($\epsilon = 0.1$), while utilizing the specific representation of interest.

trol, we select five states and evaluate their action-values for the optimal action over the course of learning. These states are distributed across the observation space, depicted in Figure 5.4(b). The bootstrap estimates, that correspond to the algorithm settings for the learning curves, are plotted in Figure 5.4(c). We can see that the relative ordering of the value estimates is maintained with SR-NN and Dropout-NN, which were the two NNs effective for on-policy control, and that their values converge to near the true values (given in Figure 5.4(d)). The other representations, on the other hand, have very poor estimates.

We now present the bootstrap values comparing SR-NN to different regularization strategies, and NN in Mountain Car in Figures 5.5. We do not visualize the location in the domain here. We include the bootstrap values and the heatmaps in Figures 5.5(a) and Figures 5.5(b). Here we see that while the range of values and the relative ordering of action-vaues estimated by various representations are similar, from Figure 5.5(c), the heatmaps reflect that the more localized activations result for the neurons learned with SR-NN, Figure 5.5(b).

Finally, we report additional measures of locality, to determine if the successful methods are indeed sparse. The heatmaps provide some evidence of locality, but are more qualitative than quantitative. We report two qualitative measures: *instance sparsity* and *activation overlap*. Instance sparsity corresponds to the percentage of active units for each input. A

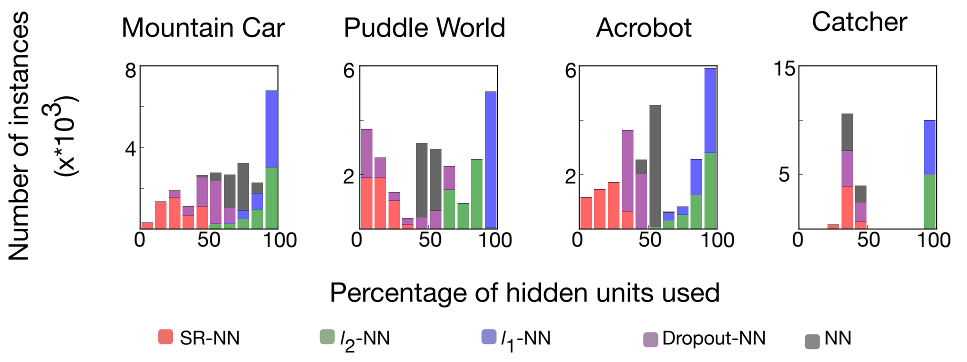

Figure 5.6: Instance sparsity comparing SR-NN to the regularized variants and vanilla NN. The percentage evaluation is designed to disregard units that are never active across all samples in the batch (dead units).

sparse representation should be instance sparse, where most inputs use a few active units. As shown in Figure 5.6, SR-NN has consistently low instance sparsity across all four domains, with slightly higher level in Catcher, potentially explaining the noisy behaviour in that domain. Once again, Dropout-NN is noticeably more instance sparse on Puddle World, but less so on other domains. The NN representation has some instance sparsity, likely due to simply using ReLU activation. Interestingly, ℓ_1-NN and ℓ_2-NN actually produced less instance sparsity.

Activation overlap, introduced by French (1991), reflects the amount of shared activation between any two inputs. We consider a variant of activation overlap that measures the number of shared activation between two representations, $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$, for two samples, $\mathbf{x}_1$, and $\mathbf{x}_2$:

$$\text{overlap}(\phi(\mathbf{x}_1),\phi(\mathbf{x}_2)) = \sum_j \mathbb{1}[(\phi_j(\mathbf{x}_1) > 0) \wedge (\phi_j(\mathbf{x}_2) > 0)].$$

We measure the activation overlap of the five chosen states, distributed across Puddle World and Mountain Car. If the overlap between two representations is zero, the interference would be zero. Updating the value function with respect to one state, therefore, would not affect the other state's value. Table 5.1 shows the average overlap, and once again, a similar trend emerges where, SR-NN has significantly less overlap in both the domains (about 8 and 16.8), with Dropout-NN showing the next least overlap in PW (with about 30), and l_2-NN showing the next least overlap in MC (with about 72.5).

Overall, these results provide some evidence that (a) sparse representations can improve control performance in an incremental control setting, (b) these sparse representations appear to provide locality and (c) this locality reduces interference and improves accuracy of bootstrap values in SARSA(0).

	SR-NN	ℓ_2-NN	ℓ_1-NN	Dropout-NN	NN
PW	**8.8**	111.5	142.5	31.2	54.0
MC	**16.8**	112.3	109.5	72.5	106.5

Table 5.1: Activation overlap in Puddle World and Mountain Car. The numbers are the average overlap over all pairs of selected states.

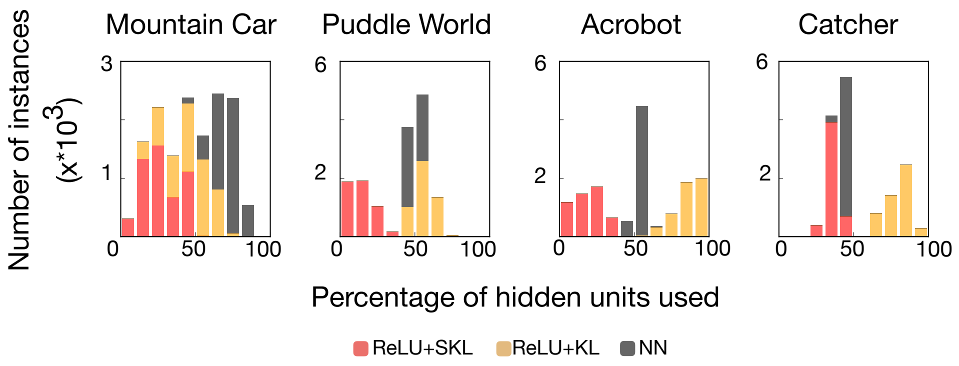

Figure 5.7: Instance sparsity as evaluated on a batch of test data comparing ReLU+KL and ReLU+SKL to NN. While ReLU+KL can make representations denser than just NN, ReLU+SKL always results in sparser representations.

5.4 Evaluation of Distributional Regularizers

In this section, we investigate the efficacy of Distributional Regularizers for obtaining sparsity. There are a variety of possible choices with Distributional Regularizers, including activation function and corresponding target distribution and using a KL versus a Set KL. Here, we present results of investigating some of these combinations, particularly focusing on the difference in sparsity and performance when using (a) KL versus SKL; (b) Sigmoid (with a Bernoulli target distribution) versus ReLU (with an Exponential target distribution); and (c) previous strategies to obtain sparse representations versus the proposed variant of the Distributional Regularizer.

5.4.1 Comparing KL to Set KL

In the first set of experiments, we compare the instance sparsity of KL to Set KL, with ReLU activations and Exponential Distributions (ReLU+KL and ReLU+SKL). Figure 5.7 shows the instance sparsity for representations learned with ReLU activations with Exponential Distributions (ReLU+KL and ReLU+SKL), and for the NN without regularization. Interestingly, ReLU+KL actually reduces sparsity in several domains, because the optimization encouraging an exact level of sparsity is quite finicky. ReLU+SKL, on the other hand, sig-

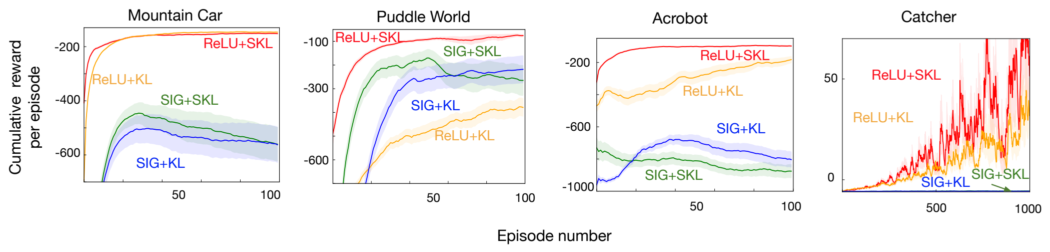

Figure 5.8: Learning curves for SARSA(0) with different Distributional Regularizers. The shaded region represents 95% confidence interval. ReLU networks utilize exponential distribution, whereas Sigmoid networks utilize Bernoulli distribution.

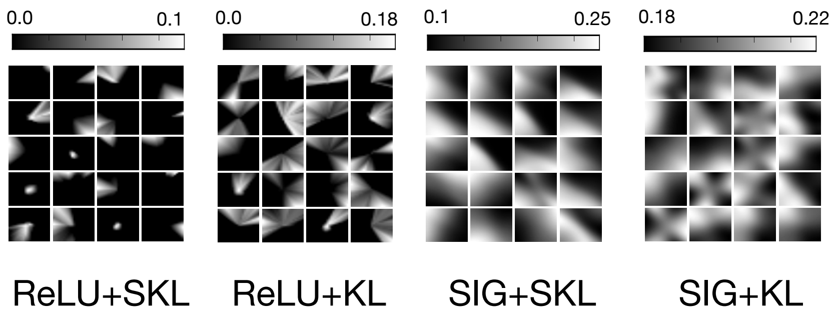

Figure 5.9: Heatmaps of activations with different Distributional Regularizers in Puddle World. Each square represents the activation of a different neuron across the state space, and corresponds to the complete 2D state-space.

nificantly improves instance sparsity over the NN. This instance sparsity again translates into control performance, where ReLU+KL does noticeably worse than ReLU+SKL across the four domains in Figure 5.8. Despite the poor instance sparsity, ReLU+KL does actually seem to provide some useful regularity, that does allow some learning across all four domains. This contrasts the previous regularization strategies, ℓ_2, ℓ_1 and Dropout, which all failed to learn on at least one domain, particularly Catcher.

5.4.2 Comparing Sigmoid to ReLU

In the next set of experiments, we compare Sigmoid (with a Bernoulli target distribution) versus ReLU (with an Exponential target distribution). With both KL and Set KL they result in combinations ReLU+KL, ReLU+SKL, SIG+KL, and SIG+SKL. We expect Sigmoid with Bernoulli to perform significantly worse—in terms of sparsity levels, locality and performance—because the Sigmoid activation makes it difficult to truly get sparse representations. This hypothesis is validated in the learning curves in Figure 5.8 and the heatmaps for Puddle World in Figure 5.9. SIG+KL and SIG+SKL perform poorly across domains,

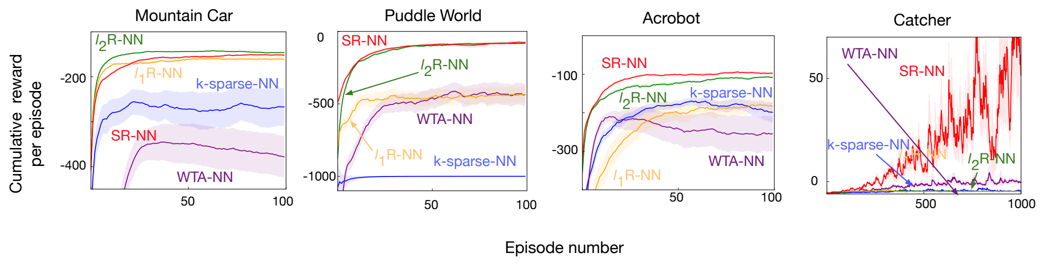

Figure 5.10: Learning curves for SARSA(0) comparing SR-NN to previous proposed sparse representations learning strategies. All strategies except SR-NN fail to learn online in Catcher, resulting in low visibility in the plot.

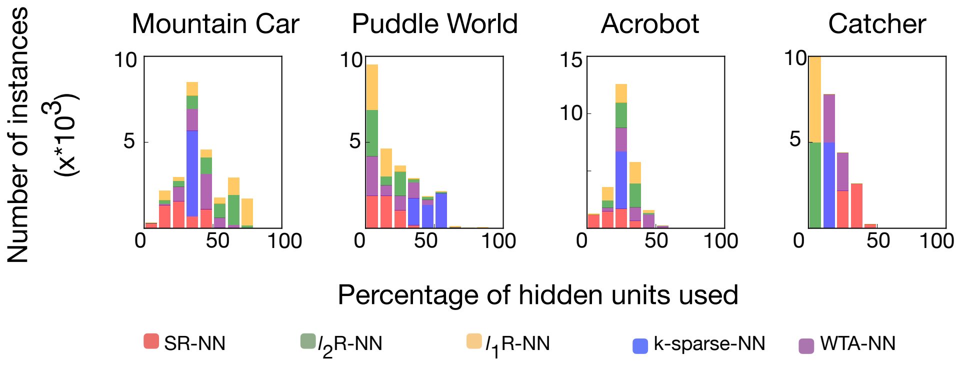

Figure 5.11: Instance sparsity comparing SR-NN to previous proposed sparse representations learning strategies. The shaded region represents 95% confidence interval. All the representation learning strategies learn relatively sparse representations, with SR-NN being the most sparse, expect in Catcher. Although SR-NN representations are the least sparse among all in Catcher, they lead to the best online performance.

even in Puddle World, where they achieved their best performance. Unlike ReLU with Exponential, here the Set KL seems to provide little benefit. The heatmaps in Figure 5.9 show that both versions, SIG+KL and SIG+SKL, cover large portions of the space, and do not have local activations for hidden nodes. In fact, SIG+KL and SIG+SKL use all the hidden nodes for all the samples across domains, resulting in no instance sparsity.

5.4.3 Comparing to Other Sparse Representation Learning Strategies

Next, we compare to other alternatives for learning sparse representations. One simple strategy is using ℓ_1 and ℓ_2 regularization on activation (denoted by ℓ_1R-NN and ℓ_2R-NN respectively). These approaches seem natural to achieve sparsity, but there is little literature investigating their utility (Bengio et al., 2013). Another strategy is enforcing sparsity explicitly with thresholds. k-sparse auto-encoders guarantee instance sparsity by keeping only the top-k largest activations for each node (Makhzani and Frey, 2013) (denoted by k-sparse-NN), whereas Winner-Take-All (WTA) autoencoders keeps the top k% activations per node across instances during training, to promote sparse activations of the node over time (Makhzani and Frey, 2015) (denoted by WTA-NN). We include learning curves and instance sparsity for these methods, for a ReLU activation, in Figures 5.10 and 5.11. Results for the Sigmoid activation are included in Appendix B.1. Neither WTA-NN nor k-sparse-NN are effective. We found that k-sparse-NN was prone to dead units, and often truncates non-negligible value. Surprisingly, ℓ_2R-NN performs comparably to SR-NN in all domains but Catcher, whereas ℓ_1R-NN is effective only during early learning in Mountain Car. From the instance sparsity plots in Catcher, we see that ℓ_1R-NN and ℓ_2R-NN produce highly sparse (2%-3% instance sparsity), potentially explaining its poor performance. While similar instance sparsity was effective in Puddle World, this is unlikely to be true in general. This was with considerable parameter optimization for the regularization parameter.

5.5 Sparsity and Exploration Algorithms

We next explore how sparsity impacts exploration algorithms. We utilize the SR-NN and NN representations with two different exploration algorithms: UCLS, presented in Chapter 4, and RLSVI (Osband et al., 2016b). UCLS is a purely incremental control algorithm whose exploration strategy relies on estimating upper-confidence bounds using learned uncertainty estimates. This uncertainty estimation procedure can benefit from local/sparse representations as the locality information can improve the algorithm's ability to track visited part of the state space online, in turn, aiding exploration. RLSVI is a batch method to exploration that uses Bayesian uncertainty estimates. These two methods, therefore, represent the extremes of the data usage spectrum: one purely incremental relying on tracking, and the other, utilizing all the data to estimate closed-form solutions. Nonetheless, both methods can be sensitive to the degree of generalization in the approximation space used.

We experiment with two different forms of UCLS – (1) a UCB variant as presented in Chapter 4 utilizing an upper-confidence bound, and (2) a Thompson sampling variant that utilizes the estimated value, $\hat{q}(s, a)$, and uncertainty, $\hat{u}(s, a)$, to sample value estimates from a normal distribution $\mathcal{N}(\hat{q}(s, a), \beta\hat{u}(s, a))$, where β controls the scale of the uncertainty. We believe the Thompson sampling variant can be more effective when the representations are dense. The relevant parameters of all three algorithms are selected based on a sweep.

We present the results comparing the two representations and the three algorithms (two variants of UCLS, and one RLSVI) in Figure 5.12. In all domains UCLS variants with SR-

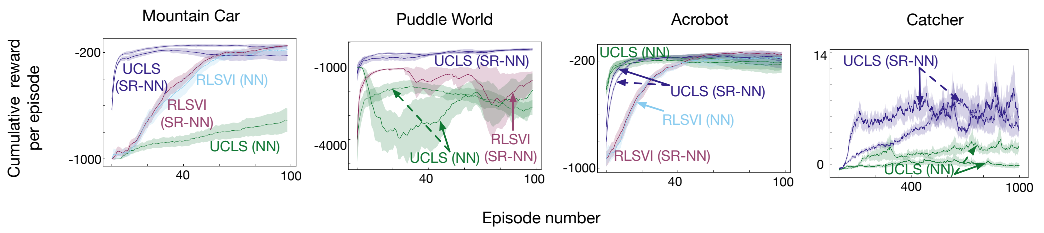

Figure 5.12: Learning curves for the exploration algorithms comparing SR-NN and NN representations. The shaded region represents 95% confidence interval. The dashed variants of UCLS lines utilize Thompson sampling. The UCB variant of UCLS(NN) is unstable in Mountain Car for the parameter range swept, whereas RLSVI(NN) is unstable in Puddle World; therefore they are not present in the graph. RLSVI with both NN and SR-NN representations in Catcher learns policies where the episodes do not terminate – the optimal policies. And therefore, are not present in the graph.

NN representations exhibit good online performance. UCLS variants with SR-NN exhibit high variance in Catcher, which can likely be reduced with more runs or a finer parameter sweep. RLSVI on the other hand performs similarly with SR-NN and NN representations in Mountain Car and Acrobot, possibly because it is a batch algorithm. In Catcher, RLSVI performs really well with the closed-form solution. With both NN and SR-NN representations, RLSVI has many runs where episodes do not terminate in a budget of 1M steps, leading to a very high cumulative reward per episode: 3 of 30 runs where RLSVI uses NN representations exhibit this behaviour, whereas 22 of 30 runs where RLSVI uses SR-NN representations exhibit this behaviour. Therefore, they are not included in the plot. RLSVI with NN representations fail in Puddle World, whereas UCLS with UCB and NN representations fail in Mountain Car.

Nonetheless, overall, these results suggest that the locality information provided by SR-NN representations is important for achieving the exploration goals of algorithms like UCLS that track visitation in state space online.

5.6 Summary

This chapter proposed a strategy for learning representations that improve sample-efficiency of online reinforcement learning algorithms, called SR-NN. SR-NNs utilize a regularizer to promote the representation function to include locality, like tile coding, which helps reduce interference of bootstrap targets during online learning. SR-NNs are evaluated on a suit of microworlds, and contrasted against other representation learning strategies, along with the benefits it can provide for algorithms designed address online exploration.

Chapter 6

Directed Exploration in Sample-Efficient Online Control

Off-policy strategies like replay are crucial for improving sample-efficiency of online control algorithms. As the availability of hardware for storage and compute becomes ubiquitous, algorithms that can benefit from multiple replay based updates will scale in sample-efficiency as well. However, not all the exploration strategies discussed are compatible with such sample-efficient replay updates, like, Kumaraswamy et al. (2018); White and White (2010).

Second, surprisingly, the distinction between the two objectives necessary for effective exploration – maintaining out-of-sample epistemic uncertainty, and estimating in-sample epistemic uncertainty – are not explicitly considered in many methods. While some methods consider the idea of out-of-sample epistemic uncertainty explicitly, like DORA (Choshen et al., 2018) via exploration-values, others consider the idea of in-sample epistemic uncertainty quantification primarily, like Bootstrap DQN (Osband et al., 2016a).

Third, some methods that maintain a distribution over value functions follow a staged-approach to exploration where the value function is fixed per episode, which does not allow for value estimates to be updated online (Osband et al., 2016b,a, 2018). This delays learning, and therefore reduces the potential sample-efficiency benefits. Further, the ideal frequency for policy updates is unclear if the domain is continuing.

Designing a sample-efficient online control algorithm with a compatible exploration strategy has been a long standing goal of reinforcement learning research. In the approximation setting, designing such an effective exploration strategy inevitably places a strong requirement on the function approximator used. Additionally, it is also necessary that the exploration strategy accounts for the two goals of exploration – visiting unknown parts of the MDP, and reducing uncertainty with respect to the visited part of the MDP – while being compatible with off-policy strategies like replay. There are many algorithms that are designed to address these two goals, either explicitly (Grande et al., 2014) or implicitly (Osband et al., 2016b, 2018), but they have limitations that makes it challenging to use them online. DGPQ (Grande et al., 2014) utilizes a Gaussian Process for learning for which specifying an effective kernel can be hard. RLSVI (Osband et al., 2016b) and Randomized Prior for Bootstrap DQN (Osband et al., 2018) are algorithms that maintain a distribution over value functions, whose online learning schedule can be hard to design, as discussed previously.

In this chapter, we propose an algorithm to address this problem setting. We first review the limitations of UCLS, particularly the reasons why it is not a satisfactory algorithm for this online setting. Then we provide an overview of existing online control algorithms and the key assumptions they make in the approximation setting. Following this, we present an architecture for effective exploration in the online setting, and an algorithm that is an instantiation of the architecture called Online Optimistic Value Iteration (OOVI). We conclude this chapter with an empirical investigation of the algorithm.

Contributions

Contribution 1: A classification of algorithms designed for sample-efficient incremental control, and an overview of the important ideas that makes them effective algorithms. An identification of two sources of epistemic uncertainty that is important for guiding effective exploration in the online setting, and an algorithm called OOVI that accounts for these goals. The algorithm addresses the limitations of UCLS while retaining its benefits, and can be implemented to scale with the availability of computational resources.

Contribution 2: An empirical evaluation of OOVI with linear function approximation that compares it with other online control algorithms, and evaluates (1) the benefits of replay updates, (2) the importance of the two sources of epistemic uncertainty it estimates, and (3) its sensitivity to the availability of data for planning purposes.

6.1 Limitations of UCLS

In Chapter 3, Section 3.4, we reviewed the three attributes we would like in a control algorithm. They are: (1) propagation of uncertainty, (2) the function-space being used is amenable to sample-efficient learning, and (3) the optimal value function $\mathbf{q}^*$ is within the set of functions considered by the exploration strategy of the online control algorithm. While the first two attributes were incorporated with reasonable solution strategies described in Chapter 4 and Chapter 5, the third attribute remains, and is crucial to promote learning near-optimal behaviour. In particular, although the performance of our proposed control algorithm, UCLS, was shown to be empirically promising in the linear function approximation setting, it is unclear whether it learns near-optimal behaviour, in general, outside the set of experiments tested.

As UCLS may not satisfy the third desirable attribute, it is possible that the regret bound presented by the Optimistic Values Theorem, Theorem 4.2.1, does not apply to it. In this case, UCLS violates Assumption 1 of Optimistic Values Theorem – Expected Optimism. Assumption 1 utilizes the estimates $\tilde{q}_t$, which are defined as $\tilde{q}_t = \hat{q}_t + \hat{u}_t$. While satisfying Assumption 3 – Convergent Action-Values – which UCLS does, an algorithm can satisfy Assumption 1 with the aid of the confidence interval radius term, $\hat{u}_t$. For effective exploration, this confidence interval term that estimates the epistemic uncertainty, $\hat{u}_t$, typically needs to incorporate two sources of information. They are:

1. *in-sample epistemic uncertainty* - that estimates any uncertainty with respect to the data seen – the visited parts of the MDP, and

2. *out-of sample epistemic uncertainty* - that estimates any uncertainty with respect to any data not seen – the unvisited parts of the MDP.

While UCLS utilizes a heuristic for ensuring effective out-of sample epistemic uncertainty, it also conflates the two sources of information into a single term. This makes it hard to verify if UCLS satisfies Assumption 1.

Arguably, to improve sample-efficiency of online learning, it is necessary to incorporate planning strategies. The simplest form of planning updates rely on utilizing transition tuples of the form $< s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1} >$. If the algorithm is designed to learn a model, then such transition tuples can be sampled from the model. But, learning effective models in the approximation case is a hard problem and an active area of research. Alternatively, more simplistically, if storage complexity of an algorithm is not a concern, then one could use an *experience replay buffer*. The buffer is essentially a part of the available storage in the algorithm that is used for saving the transition tuples experienced online. Transition tuples can then be sampled from this experience replay buffer to simulate an experience which the agent can utilize for planning updates. These planning updates can be used to improve both value estimation, and uncertainty estimation. But UCLS' exploration strategy is not compatible with such planning updates.

Algorithm property		Learning problem		
Representation	Model?	Finite-horizon	Discounted	Average-reward
Tabular	Yes	UCRL, PSRL, UCRL2, REGAL	MBIE, MORMAX, OIM	UCRL, UCRL2, REGAL , R-max, E^3
	No	RLSVI	DQL, MBIE-EB	
Linear	No	LSVI-UCB, ELEANOR	DGPQ, **?**	FOPO

Table 6.1: A table summarizing approaches to sample-efficient online control in the various online reinforcement learning settings. The table also categorizes them based on two properties – the representation space they are designed for, and whether they are model-based or not. All finite-horizon algorithms and average-reward algorithms have associated regret bounds, whereas the discounted algorithms have associated PAC sample complexity guarantees. The red box highlights the category of problem setting for which proposing an algorithm is the focus of the rest of this chapter.

UCLS' exploration strategy is designed to be effective when the algorithm utilizes only the incremental online samples to learn. This is because UCLS maintains both in-sample and out-of-sample epistemic uncertainties through the term $\hat{u}_t$, and UCLS' in-sample epistemic uncertainties reflect uncertainties with respect to its value estimates. If one were to improve in-sample epistemic uncertainty estimation using replay with the term $\hat{u}_t$ – it can rapidly reduce to 0 if the algorithm happens to have accurate value estimates for any evaluated policy π_t. This, in-turn, leads to a lack of effective out-of-sample epistemic uncertainty, and consequently, failure of effective directed exploration. Therefore, there is a need for an exploration strategy which improves upon UCLS, by being effective under sample-efficient planning strategies such as replay, while maintaining the desirable attributes of UCLS, such as, its in-sample epistemic uncertainty estimation component.

6.2 An Overview of Online Control Algorithms

In control, the goal of an agent is to maximize some form of long-term reward accumulation (Puterman, 2014). When the interaction cycle between the agent and the environment is of a fixed length, the goal can be modelled as maximizing the cumulative sum of rewards – giving us *finite-horizon RL*. Otherwise an agent can either optimize infinite-horizon long-term cumulative discounted reward for a subset of the states in the MDP – giving us *discounted-reward RL*, or infinite-horizon average reward for all states in the MDP – giving us *average-reward RL*.

We will now review online control algorithms that have theoretical guarantees for these problem settings. In particular, we identify the assumptions behind these methods, and the central ideas that allow for these theoretical guarantees, so as to develop a new algorithm inspired by these theoretically-sound ideas. Table 6.1 provides a categorization of existing algorithms for value-based online control with both tabular and linear representations. The algorithms are also classified based on whether they are model-based or model-free.

Tabular Representations: Early work in the area of online control algorithms focused on the average-reward setting, the foundational ideas from which are still the basis of new algorithms in the approximation setting. Some model-based algorithms such as E^3 (Kearns and Singh, 2002), and R-max (Brafman and Tennenholtz, 2003) assume the existence of a highly rewarding reachable state, called the"Garden of Eden" state, and utilize counts of visitation for exploration, whereas, other algorithms uch as UCRL (Auer and Ortner, 2006), UCRL2 (Jaksch et al., 2010), and REGAL (Bartlett and Tewari, 2012) utilize confidence intervals. The latter methods are also designed for MDPs with increasing difficulty of transition dynamics – unichain, communicating, and weakly communicating, respectively.

For discounted problems, inspired by confidence intervals of UCRL, Strehl and Littman (2008) proposed MBIE, whereas inspired by R-max, Szita and Szepesvari (2010) proposed MORMAX. Additionally, Szita and Lorincz (2008) introduced OIM based on the principle of optimistic initialization. Both MORMAX and OIM incorporate the idea of "Garden of Eden" state to induce out-of-sample epistemic uncertainty. As these methods are all model-based, extracting a policy relies on using the *extended value iteration* algorithm (Strehl and Littman, 2008; Jaksch et al., 2010) in its planning phase.

To avoid the expensive planning necessary in model-based algorithms, some breakthrough model-free algorithms were proposed for discounted MDPs. DQL (Strehl et al., 2006) utilizes optimistic initialization for inducing out-of-sample epistemic uncertainty. It also tracks visitation times and update flags, which helps ensure that updates made to the value functions are based on well-estimated targets, therefore stabilizing learning. Additionally, Strehl and Littman (2008) propose MBIE-EB — a model-free extension to MBIE, where EB stands for Exploration Bonus. The algorithm augments the reward function with bonuses that encourage exploration, and are inversely proportional to visitation counts.

Confidence-interval based average-reward algorithms described above can also be adapted to the finite-horizon problem setting (Neu and Pike-Burke, 2020). While these methods utilize frequentist confidence intervals, Bayesian model-based methods are explored by Osband et al. (2013) in PSRL, whereas Bayesian model-free methods are explored by Osband et al. (2016b) in RLSVI. Additionally, Bayesian RLSVI, along with ideas for exploration based on MBIE's exploration bonuses discussed previously, have motivated many model-free approaches to exploration in the deep reinforcement learning setting (Osband et al., 2016a; Ostrovski et al., 2017; Osband et al., 2018).

These many algorithms have been designed with different theoretical goals. The frequentist methods of both finite-horizon and average-reward settings are designed to minimize regret — the difference between the optimal policy's performance and the online policy's performance. On the other hand, Bayesian algorithms are designed to minimize Bayesian regret — the difference between the performance of the optimal policy conditioned on the seen data and the online policy's performance. Other algorithms have been designed to be PAC-MDP — that is, algorithms which produce policies that are Probably Approximately Correct in the MDP, within a polynomial, in the parameters of the MDP, number of steps.

Linear Function Approximation: The algorithms discussed so far have been designed for tabular representations, but, arguably, tabular representations are data inefficient. A widely used, and well-studied approximator is the linear approximator. For such an approximator to be effective it is crucial that the representations they build on facilitate learning. To characterize this facilitation it is necessary to define properties of these representations (Zanette et al., 2020). Here, the most widely used assumptions about the representations for a given MDP, in decreasing order of generality, are: (i) the representation can represent Q^* – *realizability*, (ii) the representation has low error w.r.t. the Bellman optimality operator – *low inherent Bellman error*, and (iii) the dynamics of the MDP are linear in the representation – *linear MDP*. Below we briefly present these assumptions and discuss algorithms built on them.

First, while the realizability assumption is the most general, it is also the hardest assumption to design sample-efficient algorithms under – even for the non-online (batch) setting (Xie and Jiang, 2020). To the best of our knowledge, existing online algorithms have only been proposed under stronger assumptions.

Second, the assumption of low inherent Bellman error was originally proposed in the setting where the model is available, by Munos and Szepesvári (2008). Zanette et al. (2020) adapted it to the online finite-horizon setting and developed a sample-efficient online algorithm. Intuitively, the low inherent Bellman error assumption states that the approximation space, under the representation used, has low projection error for the value functions generated by the Bellman optimality operator T. For a general online setting with a single learner, if the goal is to learn $\mathbf{w} \in \mathcal{W} \subset \mathbb{R}^d$, the inherent Bellman error can be evaluated as

$$\sup_{\mathbf{w}' \in \mathcal{W}} \inf_{\mathbf{w} \in \mathcal{W}} \sup_{(s,a) \in (\mathcal{S} \times \mathcal{A})} |\boldsymbol{\phi}(s,a)^\top \mathbf{w} - (T\mathbf{q}_{\mathbf{w}'})(s,a)|,$$
$$\text{where, } (T\mathbf{q}_{\mathbf{w}'})(s,a) = R(s,a) + \mathbb{E}_{s' \sim P(s,a,.)}\big[\gamma(s,a,s') \max_{a'} (\mathbf{q}_{\mathbf{w}'})(s',a')\big],$$
$$\text{and, } (\mathbf{q}_{\mathbf{w}'})(s',a') = \boldsymbol{\phi}(s',a')^\top \mathbf{w}'.$$

Given, $\mathbf{w} \in \mathcal{W}$, the corresponding approximate values for $(\mathcal{S} \times \mathcal{A})$ will be $\mathbf{q}_{\mathbf{w}}$, where $\mathbf{q}_{\mathbf{w}} \in \mathcal{Q} \subset \mathbb{R}^n$, and n denotes the number of state-action pairs. If the inherent Bellman error is equal to 0, then for any $\mathbf{q} \in \mathcal{Q}$, $T\mathbf{q} \in \mathcal{Q}$. In general, if the inherent Bellman error of the approximation class is small, then any error propagated with repeated applications of the Bellman operator will be bounded, and the algorithm will not be prone to divergence online.

Third, is the linear MDP assumption introduced by Jin et al. (2019) for the finite-horizon setting. This is a stronger assumption that expects the MDP's dynamics, P and R, to be linear in the representation. If we assume the dynamics are the same at every step of the horizon they can be written as

$$P(s,a,s') = \boldsymbol{\phi}(s,a)^\top \boldsymbol{\mu}(s'), \text{ and, } R(s,a) = \boldsymbol{\phi}(s,a)^\top \boldsymbol{\theta}_r,$$
$$\text{where, } \boldsymbol{\mu} : \mathcal{S} \to \mathbb{R}^d \text{ is an unknown representation for the states,}$$
$$\text{and, } \boldsymbol{\theta}_r \in \mathbb{R}^d \text{ is an unknown vector.}$$

If this assumption is satisfied, the value of all policies π are representable (Zanette et al., 2020).

Sample-Efficient Linear Function Approximation for Finite-Horizon RL: Sample-efficient finite-horizon algorithms have been proposed under the last two assumptions. As the problem is finite-horizon, these algorithms treat the learning problem at each step of the horizon as an independent linear regression problem. Under the linear MDP assumption, Jin et al. (2019) introduce LSVI-UCB, whereas under the low inherent Bellman error assumption, Zanette et al. (2020) introduce ELEANOR.

At each step of the horizon, to induce optimism, LSVI-UCB uses bonuses corresponding to the linear estimators' uncertainty for state-action representations at the next step of the horizon; these bonuses are also called *local* bonuses (Neu and Pike-Burke, 2020). This strategy is effective under the linear MDP assumption.

These local bonuses can be non-linear in the representations and are therefore not compatible with the more general low inherent Bellman error assumption. ELEANOR uses a different strategy to estimate bonuses for inducing optimism; the bonuses it uses are also called *global* bonuses (Neu and Pike-Burke, 2020). In this strategy the most optimistic policy, and the corresponding value function are computed by taking into account the complete dynamics of the finite-horizon problem. The variables in this optimization include both the value function parameters, and the bonus parameters. This explicit method of computing bonuses is crucial for inducing optimism that is both compatible with this more general assumption, and capable of inducing effective exploratory behaviour. But, due to the explicit optimization objective, it is a difficult algorithm to implement currently.

Sample-Efficient Linear Function Approximation for Average-Reward RL: Inspired by the linear MDP assumption, Wei et al. (2020) propose an algorithm called FOPO for the average-reward case. As an average-reward learning problem does not consist of a finite number of steps like finite-horizon problems, the algorithm is designed to update the online policy only when the visitation structure of the representation space being used changes significantly — that is, it depends on the determinant of the online covariance matrix. All analysis, in both the finite-horizon algorithms described previously and the average-reward algorithm FOPO, focus on the regret experienced by the algorithms.

Sample-Efficient Linear Function Approximation for Discounted-Reward RL: For the discounted setting, Grande et al. (2014) introduce the algorithm DGPQ, that can be considered a linear function approximation algorithm. It extends DQL to the approximation setting utilizing two different approximators: (1) a non-parametric Gaussian Process to measure in-sample epistemic uncertainty, and (2) a set of linear basis vectors to promote out-of-sample epistemic uncertainty, and therefore optimism, in the value estimates. The algorithm has PAC guarantees, but is computationally expensive because it relies on GPs. Additionally it requires an effective kernel for the Gaussian Process, and an effective basis for the optimistic out-of-sample epistemic uncertainty estimates. Therefore, there is still a need for an effective, computationally tractable algorithm in discounted MDPs with linear function approximation.

6.3 Online Optimistic Value Iteration

One approach to ensuring systematic exploration with model-free algorithms is to initialize the agent's value estimates to be optimistic. This can be achieved by initializing the state-action value function to predict the maximum possible value, or higher, in all state-action pairs. The initialization can be done either by regressing towards such a target with uniformly sampled state-action pairs, or by naively initializing the value function parameter's components to be high if the features can be guaranteed to be always positive. But such optimistic initialization, along with being incompatible with non-stationary problems, does not consider the information present in aliasing caused by the function approximator used. Any information based on the aliasing structure can help improve sample efficiency of online exploration significantly, and is therefore useful from an exploration perspective. Further, it is unclear how one would maintain such initialized optimism without overwriting the initialization with estimates based on experienced data.

While these are the limitations of optimistic initialization if it were to be the sole strategy for effective exploration, it encodes a desirable quality that has been at the centre of successful approaches to effective exploration in tabular state-spaces: the ability to incorporate out-of-sample epistemic uncertainty. Incorporating out-of-sample epistemic uncertainty promotes only first visits in state-action space. But it does not necessarily encourage the right consequent visits. Consequent visits can be sample-efficient if they are guided by the confidence of the agent in its estimates. Therefore, in-sample epistemic uncertainty can be estimated and used to guide consequent visits.

In model-free algorithms, where the agent is learning without utilizing a model, there are a few avenues to estimate confidence-intervals for the action values. While some approaches maintain a distribution over plausible optimal value functions themselves, like RLSVI (Osband et al., 2016b), Bootstrap DQN (Osband et al., 2016a) and Randomized Priors (Osband et al., 2018), others estimate the uncertainty in the current value estimates with respect to the current policy being followed. Tang et al. (2020) estimate a non-parametric upper-bound for the estimate at any state-action pair by assuming that the function class considered is Lipschitz continuous, whereas Kumaraswamy et al. (2018) estimate a parametric uncertainty to quantify a confidence-interval around estimated action-values, and (ODonoghue et al., 2017) propagate uncertainty based on visitation in feature-space. Regardless of the specific approach to quantify uncertainty in value estimates, the estimated uncertainties can be utilized to guide online exploratory behaviour.

In order to capture the two sources of information for effective exploration in the sample-efficient online setting, we propose to utilize three different estimators –

1. value function estimator, $\mathbf{w}$ - the regular value function estimator that is used by the agent for decision making online.

2. in-sample epistemic uncertainty estimator, $\mathbf{w}^{\text{e-in}}$ - an epistemic uncertainty estimator

Notation	Description
$\pi_{\mathbf{x}}$	denotes a policy greedy in values estimated using parameters $\mathbf{x}$
π	the target policy, $\pi_{\mathbf{w}^o}$
$\mathbf{w}$	parameters used for estimating the approximate value function for policy π
$\mathbf{w}^o$	intermediate parameters used to estimate an optimistic value function for policy $\pi_{\mathbf{w}}$
$\pi_{(\mathbf{x},\mathbf{y})}$	denotes a policy greedy in values estimated using parameters $\mathbf{x}$
	plus values estimated using parameters $\mathbf{y}$
μ	the behaviour policy, $\pi_{(\mathbf{w},\mathbf{w}^{\text{e-out}})}$
$\mathbf{w}^{\text{e-in}}$	parameters used for estimating in-sample epistemic uncertainty in estimates of $\mathbf{w}$ for policy π
$\mathbf{w}^{\text{e-out}}$	parameters used for estimating out-of sample epistemic uncertainty
$\mathbf{q}_{\mathbf{x}}$	values estimated using parameters $\mathbf{x}$
$T_{\pi_{\mathbf{x}}}\mathbf{y}$	applying the Bellman operator using policy $\pi_{\mathbf{x}}$ with the targets designed using $\mathbf{y}$
$T^{\mathbf{0}}_{\pi_{\mathbf{x}}}\mathbf{y}$	applying the Bellman operator using policy $\pi_{\mathbf{x}}$ with the targets designed using $\mathbf{y}$ and 0 reward

Table 6.2: A table summarizing the notation used in the OOVI pseudocode, Algorithm 8.

that can be used for estimating in-sample epistemic uncertainty in the value estimates of $\mathbf{w}$.

3. out-of-sample epistemic uncertainty estimator, $\mathbf{w}^{\text{e-out}}$ - the estimator that is initialized to be optimistic in order to promote out-of-sample epistemic uncertainty.

Given this architecture demarcating the three estimators and their specific goals with respect to effective online exploration, we propose an algorithm built on this separation that is compatible with replay called Online Optimistic Value Iteration (OOVI).

Algorithm 8 Online Optimistic Value Iteration (OOVI)

```
1: $\mathcal{B} \leftarrow [\emptyset]$, $\mathbf{w} \leftarrow \mathbf{0}$, $\mathbf{w}^{\text{e-in}} \leftarrow \mathbf{0}$, $\mathbf{w}^{\text{e-out}} \leftarrow v_{\max}$ with respect to $\mathcal{S}$, $\beta \leftarrow$ in-sample
   epistemic uncertainty scaling parameter
2: $\triangleright \mu(\boldsymbol{\phi}_{s,\cdot}) \leftarrow \pi_{(\mathbf{w},\mathbf{w}^{\text{e-out}})}$
3: $\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
4: $a \leftarrow \mu(\boldsymbol{\phi}_{s,\cdot})$
5: repeat
6:     Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
7:     Store $[\boldsymbol{\phi}_{s,a}, r, \gamma, \boldsymbol{\phi}_{s',\cdot}]$ in $\mathcal{B}$
8:     Update $\mathbf{w}^o$ to estimate $T_{\pi_{\mathbf{w}}}\mathbf{q}_o$, where $\mathbf{q}_o$ includes $\mathbf{w}^{\text{e-in}}$
9:     $\pi \leftarrow \pi_{\mathbf{w}^o}$
10:    Update $\mathbf{w}$ to estimate $T_{\pi}\mathbf{q}_{\mathbf{w}^o}$
11:    Update $\mathbf{w}^{\text{e-in}}$ to estimate uncertainty in $\mathbf{w}$'s estimates for $\pi$
12:    Update $\mathbf{w}^{\text{e-out}}$ to estimate $T^{\mathbf{0}}_{\pi_{\mathbf{w}}}\mathbf{q}_{\mathbf{w}^{\text{e-out}}}$
13:    $a \leftarrow \mu(\boldsymbol{\phi}_{s',\cdot})$, and $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{\mathbf{s}',a}$
14: until agent finishes interaction with environment
```

6.3.1 Outline of OOVI

OOVI is built on the framework described previously, with specific choices for (1) estimating the two estimators – $\mathbf{w}^{\text{e-in}}$ and $\mathbf{w}^{\text{e-out}}$, and (2) a 2-stage value iteration update that incorporates the estimated $\mathbf{w}^{\text{e-in}}$ to promote optimistic values and guide online exploration.

Table 6.2 provides a description of the notation used by OOVI. The pseudocode using this notation for OOVI is provided in Algorithm 8. As OOVI is compatible with replay, we can use two different policies in the algorithm – (1) a policy that drives the learning updates, called the target policy, denoted by π, and (2) a policy that drives the behaviour of the agent online, called the behaviour policy, denoted by μ.

A brief overview of OOVI is as follows. The algorithm estimates the values $\mathbf{w}_{t+1}$ for the policy π_{t+1}, the target policy for timestep t. The policy π_{t+1} is greedy in optimistic values that are obtained via the parameters $\mathbf{w}^o_t$. $\mathbf{w}^o_t$ is an intermediary set of parameters that incorporates any in-sample epistemic uncertainty estimated by $\mathbf{w}^{\text{e-in}}_t$. The epistemic uncertainty estimates reflect the uncertainty of value estimates made by the previous value function $\mathbf{w}_t$, for the target policy in timestep $t-1$, π_t. The behaviour policy, μ_{t+1} is greedy with respect to the more optimistic values that includes both estimates of $\mathbf{w}_{t+1}$ and the out-of-sample epistemic uncertainty $\mathbf{w}^{\text{e-out}}_{t+1}$, which is also tracked online. This is necessary in order to promote behaviour that visits parts of the state-space that are not visited yet, and therefore, are not accounted for by the in-sample epistemic uncertainty estimator $\mathbf{w}^{\text{e-in}}_t$.

To explain the algorithm more clearly, we now go over the pseudocode of OOVI. At every timestep t, OOVI makes the following updates.

[*Lines 8, 9 and 10*]: These lines together estimate a value function $\mathbf{w}_{t+1}$ for the new target policy π_{t+1}. This policy is greedy in the value function that incorporates any in-sample epistemic uncertainty estimated by $\mathbf{w}^{\text{e-in}}_t$. Estimating this new value function requires two main steps and an intermediate assignment step for the new target policy, π_{t+1}. It also requires an intermediate set of parameters which we will denote by $\mathbf{w}^o$. The parameters $\mathbf{w}^o_t$ dictate the new policy π_{t+1}.

[*Line 8*]: In step one, $\mathbf{w}^o_t$ incorporates any in-sample epistemic uncertainty that exists for policy that is greedy in $\mathbf{w}_t$, denoted by $\pi_{\mathbf{w}_t}$. This is done by updating towards $T_{\pi_{\mathbf{w}_t}}\mathbf{q}_o$ which uses either (1) upper-confidence bound, or (2) Thompson sampling (Thompson, 1933) to incorporate the epistemic uncertainty.

$$(T_{\pi_{\mathbf{w}_t}}\mathbf{q}_o)(s,a) = R(s,a) + \mathbb{E}_{s'\sim P(s,a,.),a'\sim\pi_{\mathbf{w}_t}(s',.)}\big[\gamma(s,a,s')(\mathbf{q}_o)(s',a')\big],$$

where, if using upper-confidence bound

$$(\mathbf{q}_o)(s', a') = \boldsymbol{\phi}_{s,a}^\top \mathbf{w} + \beta * |\boldsymbol{\phi}(s', a')^\top \mathbf{w}^{\text{e-in}}|,$$

else, with Thompson sampling is

$$(\mathbf{q}_o)(s', a') = \mathcal{N}(\boldsymbol{\phi}_{s,a}^\top \mathbf{w}, \beta * |\boldsymbol{\phi}(s', a')^\top \mathbf{w}^{\text{e-in}}|).$$

If we use Thompson sampling to estimate $\mathbf{q}_o$, then along with (s', a') the estimate itself is a random variable, sampled from the distribution $\mathcal{N}(\boldsymbol{\phi}_{s,a}^\top \mathbf{w}, \beta * |\boldsymbol{\phi}(s', a')^\top \mathbf{w}^{\text{e-in}}|)$.

[Line 9]: This line just assigns the new target policy, π_{t+1} as the policy that is greedy in the estimates of value function with parameters $\mathbf{w}_t^o$, the optimistic value function.

[*Line 10*]: Then in step two, a new value function for π_{t+1}, $\mathbf{w}_{t+1}$, is estimated by using $T_{\pi_{t+1}} \mathbf{q}_{\mathbf{w}_t^o}$. The targets are designed by utilizing value function estimates with parameters $\mathbf{w}^o$, denoted by $\mathbf{q}_{\mathbf{w}^o}$.

[*Line 11*]: Any in-sample epistemic uncertainty in $\mathbf{w}_t^{\text{e-in}}$ for the values estimated by $\mathbf{w}_t$ has been incorporated into $\mathbf{w}_t^o$ in Line 8. The following policy evaluation step in Line 10 results in an estimate, $\mathbf{w}_{t+1}$, of the parameters for the value function of the target policy π_{t+1}. Therefore, it is necessary to re-estimate any in-sample epistemic uncertainty in the values estimated by $\mathbf{w}_{t+1}$. OOVI does so by utilizing the linear complexity strategy described in Section 4.2.3 of Chapter 4.

Simplistically, this epistemic uncertainty estimation involves learning a value function, utilizing the parameters $\mathbf{w}^{\text{e-in}}$. This value function uses the temporal difference errors in the values estimated by $\mathbf{w}$ as the reward, instead of the real reward obtained from the environment. Given a transition tuple $< s_i, a_i, s_{i+1}, r_{i+1}, \gamma_{i+1} >$, the temporal difference error in the values estimated by $\mathbf{w}$ for a policy π is

$$\delta_{i+1} = r_{i+1} + \gamma_{i+1} \boldsymbol{\phi}(s_{i+1}, a_{i+1})^\top \mathbf{w} - \boldsymbol{\phi}(s_i, a_i)^\top \mathbf{w},$$

where a_i is sampled based on policy π, the value function of which is estimated by $\mathbf{w}$. These temporal difference errors are used as reward to estimate the parameters $\mathbf{w}^{\text{e-in}}$. Therefore, given $\mathbf{w}_{t+1}$, the estimated value function for π_{t+1}, $\mathbf{w}_{t+1}^{\text{e-in}}$ estimates any in-sample epistemic uncertainty.

[*Line 12*]: Finally, it is necessary to decay away any out-of-sample epistemic uncertainty that has been accounted for by the in-sample epistemic uncertainty estimates. As any in-sample epistemic uncertainty is incorporated in the value function $\mathbf{w}_{t+1}$, the out-of sample epistemic uncertainty is decayed away with respect to the policy greedy in it, denoted by $\pi_{\mathbf{w}_{t+1}}$. Therefore, at timestep t, $\mathbf{w}_{t+1}^{\text{e-out}}$ is estimated by updating $\mathbf{w}_t^{\text{e-out}}$ towards $T^{\mathbf{0}}_{\pi_{\mathbf{w}_{t+1}}}\mathbf{q}_{\mathbf{w}^{\text{e-out}}}$ defined as

$$(T^{\mathbf{0}}_{\pi_{\mathbf{w}_{t+1}}}\mathbf{q}_{\mathbf{w}_t^{\text{e-out}}})(s,a) = \mathbb{E}_{s'\sim P(s,a,.),a'\sim\pi(s',.)}\left[\gamma(s,a,s')(\mathbf{q}_{\mathbf{w}_t^{\text{e-out}}})(s',a')\right].$$

[Line 2]: Given these sequence of updates, the behaviour, μ_{t+1}, is designed to be greedy with respect to the sum of values estimated by $\mathbf{w}_{t+1}$ and $\mathbf{w}_{t+1}^{\text{e-out}}$, $\pi_{(\mathbf{w}_{t+1},\mathbf{w}_{t+1}^{\text{e-out}})}$. This is because over timesteps, any in-sample epistemic uncertainty is incorporated in $\mathbf{w}$, requiring any exploratory behaviour to be driven by its estimates combined with any remaining out-of-sample epistemic uncertainty, estimated by $\mathbf{w}^{\text{e-out}}$.

6.3.2 Ideas Incorporated in OOVI

With the main components of OOVI described, we will now describe the key ideas that aided its design.

Out-Of-Sample Epistemic Uncertainty in OOVI: The core contribution of out-of-sample epistemic uncertainty estimator, $\mathbf{w}^{\text{e-out}}$, for driving exploratory behaviour cannot be learned from data experienced – it needs to be *anticipated*. Many tabular state-space algorithms achieve this by utilizing the idea of *"Garden of Eden State"*, for instance MOR-MAX (Szita and Szepesvari, 2010). OOVI uses such a strategy, by regressing $\mathbf{w}^{\text{e-out}}$ to a high value v_{max} with respect to the input space $\mathcal{S}$.

Using the regression strategy is different from initializing the parameters of $\mathbf{w}^{\text{e-out}}$ to v_{max}. Specifically, for linear representations, regressing with respect to the underlying state-space used ensures the out-of-sample epistemic uncertainty utilizes the generalization structure of the representation space. If the generalization structure uses the linear representation space non-uniformly, regressing can ensure the induced out-of-sample epistemic uncertainty is with respect to the underlying MDP states. Additionally for both linear and non-linear representations, such a regression based initialization strategy ensures that high values are associated with all parts of the underlying state space, which may not necessarily be true without regression as the features can be both positive and negative.

In-Sample Epistemic Uncertainty in OOVI: To estimate in-sample epistemic uncertainty in the model-free case, OOVI utilizes the replay compatible strategy proposed in Section 4.2.3 of Chapter 4. These in-sample epistemic uncertainty estimates can be used with Thompson sampling to provide stochastically optimistic values for state-action pairs (Osband et al., 2016b). This strategy introduces a parameter β that helps controls the scale of in-sample epistemic uncertainty utilized to promote optimistic estimates.

Optimistic Value Iteration: In general, in the online scenario the agent is interested in satisfying two different goals: (1) learning the optimal policy's value function, and (2) learning good exploratory behaviour that aids in learning the optimal policy's value function. If the algorithm incorporates off-policy learning strategies the agent can utilize two different policies to achieve these goals. The first is called the target policy, denoted by π, whose goal is learning the optimal policy's value function. And the second is called the behaviour policy, denoted by μ, whose goal is to effectively explore the domain. And in the online setting, once the exploration goals of the domain are met by μ, the online behaviour should naturally transition to the learned π, with or without an explicit policy switch.

In order to define what the target policy π should be for OOVI, let us consider what the estimates for timesteps t and $t+1$ would correspond to. In both the timesteps we would estimate the uncertainty in the values estimated by $\mathbf{w}$, corresponding to target policy π, using $\mathbf{w}^{\text{e-in}}$. $\mathbf{w}_t^{\text{e-in}}$ estimated for $\mathbf{w}_t$ with respect to π_t in timestep t, is overwritten by $\mathbf{w}_{t+1}^{\text{e-in}}$ estimated for $\mathbf{w}_{t+1}$ with respect to π_{t+1} in timestep $t+1$. Therefore, there is a need to carry forward any information present in $\mathbf{w}_t^{\text{e-in}}$ before it is overwritten.

In order to carry forward this information, we utilize an intermediate set of parameters denoted by $\mathbf{w}^o$. $\mathbf{w}_{t+1}^o$ is a value function that essentially incorporates any uncertainty present in $\mathbf{w}_t$, as evaluated by $\mathbf{w}_t^{\text{e-in}}$, to estimate an upper-bound value function for the greedy policy dictated by $\mathbf{w}_t$. We call $\mathbf{w}_{t+1}^o$ the optimistic value function as it incorporates any in-sample epistemic uncertainty in $\mathbf{w}_t^{\text{e-in}}$ that is relevant to the greedy policy in $\mathbf{w}_t$. Now, this optimistic value function $\mathbf{w}_{t+1}^o$ dictates the target policy π_{t+1}. π_{t+1} can be a completely greedy or a soft improvement over $\mathbf{w}^o$ and is used to evaluate a new value function $\mathbf{w}_{t+1}$ and any uncertainty around its estimates in $\mathbf{w}_{t+1}^{\text{e-in}}$.

This sequence of updates over time-steps dictates how the agent learns the optimal value function for any given experience. If any estimated in-sample epistemic uncertainty in $\mathbf{w}^{\text{e-in}}$ is not relevant with respect to the underlying greedy policy dictated by $\mathbf{w}$, this is not incorporated in $\mathbf{w}^o$, and consequently is not the focus of the target policy π which aims to learn the optimal value function.

As the value estimates include the in-sample epistemic uncertainty components, the behaviour policy μ_{t+1} can be greedy with respect to the estimates of $\mathbf{w}_{t+1}$ and $\mathbf{w}_{t+1}^{\text{e-out}}$. Here, the out-of-sample epistemic uncertainty of $\mathbf{w}_t^{\text{e-out}}$ is decayed to estimate $\mathbf{w}_{t+1}^{\text{e-out}}$ based on the greedy policy of $\mathbf{w}_{t+1}$. This choice is made as in the absence of any out-of-sample epistemic uncertainty that is the policy that the behaviour would naturally follow.

Additionally, as $\mathbf{w}^{\text{e-out}}$ decays to 0, the behaviour would naturally follow the greedy policy induced by $\mathbf{w}$, the target policy. This behaviour may further generatxe data to reduce any incorporated epistemic uncertainty, before transitioning to greedy behaviour with respect to learned optimal value function.

Assumptions about the Function Approximator in OOVI: As OOVI is based on off-policy learning and does not use any additional strategies to mitigate variance issues in off-policy scenario (Ghiassian et al., 2018), it is necessary that the learning updates made by OOVI are stable online. Towards this end, we make two assumptions about the function space that will be used by OOVI.

First, it is desirable that the function space used by OOVI has Low Inherent Bellman Error (LIBE), where Inherent Bellman error (Munos and Szepesvári, 2008) of a linear function space is defined as follows:

$$\eta = \sup_{\mathbf{w}'\in\mathcal{W}} \inf_{\mathbf{w}\in\mathcal{W}} \sup_{(s,a)\in(\mathcal{S}\times\mathcal{A})} |\boldsymbol{\phi}(s,a)^\top \mathbf{w} - (T\mathbf{q}_{\mathbf{w}'})(s,a)|,$$
$$\text{where, } (T\mathbf{q}_{\mathbf{w}'})(s,a) = R(s,a) + \mathbb{E}_{s'\sim P(s,a,.)}\big[\gamma(s,a,s')\max_{a'}(\mathbf{q}_{\mathbf{w}'})(s',a')\big],$$
$$\text{and, } (\mathbf{q}_{\mathbf{w}'})(s',a') = \boldsymbol{\phi}(s',a')^\top \mathbf{w}'.$$

If the function space has LIBE, then $\eta \leq \epsilon$, where ϵ denotes the worst case approximation error for the function space. While the greedy Policy Improvement operator here incorporates the expected discounted next state value, Zanette et al. (2020) adapt the definition to the online setting in finite-horizon MDPs, where the expectation in the definition is replaced by the samples experienced online. Such a similar modification can be considered here, except for the discounted setting. Any function space, whether linear or not, has an associated Inherent Bellman Error. In this work we assume this is low, although we do not explicitly utilize the associated η value pertaining to the function space in our algorithm.

The second assumption we make about the function space used by OOVI is as follows. Under function approximation in the model-free case, encouraging first-visits to unknown parts of the MDP corresponds to maintaining sufficient out-of-sample epistemic uncertainty in the corresponding parts of the representation space. Maintaining such effective out-of-sample epistemic uncertainty can be challenging if the approximation space generalizes aggressively between states too much. Therefore, such an endeavour to maintain effective out-of-sample epistemic uncertainty, and explore effectively, can benefit from using local representations. Therefore, OOVI is designed to be used with representations that have low inherent Bellman error for the MDP, while encoding good locality information.

6.4 Evaluation of OOVI

In this section we explore the following aspects of OOVI empirically: (1) general online performance as compared to other online control algorithms, (2) the advantage of more replay updates, (4) the utility of the each source of epistemic uncertainty used by OOVI, and (3) the sensitivity of OOVI to the size of the online buffer used.

We compare OOVI to RLSVI (Osband et al., 2016b), Randomized Prior for Bootstrap-DQN (denoted henceforth by BSP, for Bootstrap DQN with Prior) (Osband et al., 2018), and UBE (ODonoghue et al., 2017). All the algorithms are replay compatible algorithms that utilize a form of uncertainty to drive exploratory behaviour online. RLSVI and BSP utilize a Bayesian approach to exploration; the former utilizing Bayesian linear regression to estimate uncertainty, and the latter an ensemble to estimate uncertainty. UBE estimates uncertainty by utilizing a Bellman-style update for propagating local variance estimates. We also compare OOVI to variants of UCLS, which are not replay compatible but can utilize traces, as this can help us evaluate the utility of replay vs. traces. In-depth descriptions of each algorithm can be found in Appendix C.3, and implementation details, along with sweep details can be found in the Appendix C.4.

We utilize the same domains as in Chapter 4 — Sparse Mountain Car, Puddle World and River Swim — with the key difference being the representations used. As we would like representations that have Low Inherent Bellman Errors (LIBE), we use learned sparse representations, representations that are produced by SR-NN with the learning objective being next state and reward prediction. The policies used to generate the training data for SR-NN are competitive hand-coded policies in all the domains. To promote LIBE we utilize stochastic versions of the domains first, along with a deterministic version of Sparse Mountain Car, which is the original form of the domain (Sutton and Barto, 2018). We use representations of 256 dimension in all domains, a real valued sparse vector produced by SR-NNs. More details about the experimental setup for learning representations can be found in Appendix C.4.

6.4.1 Evaluating Online Performance

We investigate a learning regime similar to Chapter 4 – the agent is allowed a fixed budget of interaction steps with the environment. As our goal is to evaluate the effectiveness of the exploration strategy, we do not use cutoffs in the episodic problems. We utilize a budget of 100k steps. RLSVI's weights are recalculated using all experienced transitions at the

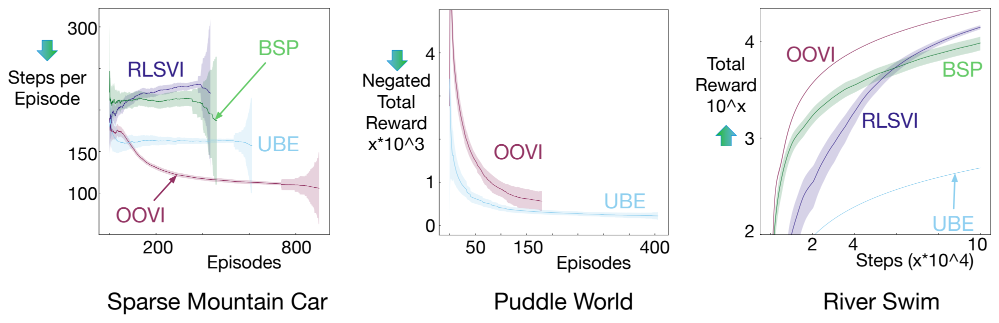

Figure 6.1: Learning performance in the three stochastic domains comparing OOVI to other online control algorithms. In the first two plots lower y-axis is better, and in the last higher y-axis is better. The experiments are run with a fixed budget of steps with no episode cutoffs. The x-axis for the episodic domains is chosen based on the median number of episodes completed. This leads to the confidence intervals towards the end of the learning curves being larger as data from fewer runs is available for computing these statistics.

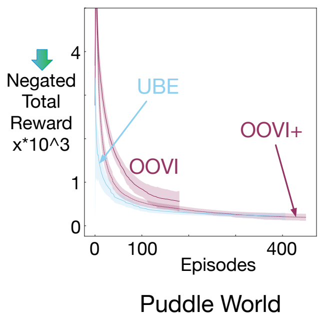

Figure 6.2: Learning performance comparing UBE to a version of OOVI with more replay updates. OOVI with more replay updates is plotted with a dashed line and denoted by OOVI+. OOVI is also visualized to contrast the benefit of more replay updates.

beginning of an episode in Puddle World and Sparse Mountain Car, and every 5,000 steps in River, similar to BSP's learning schedule. All algorithms utilize an online buffer the size of the experiment, unless specified otherwise. The parameters for OOVI and the competitors are selected from a large parameter sweep which averages over 3 runs. OOVI uses 10 planning steps in all the experiments unless specified otherwise. The results shown here are averaged over 50 independent runs for the best parameter configuration. The shaded region represents 95% confidence interval. The range of the x-axis, if episodic, reflects the median number of episodes completed across runs, leading to statistics later in the experiment being average over fewer runs, and consequently larger confidence intervals.

Comparing to other control algorithms: Our first experiment compares OOVI to other control algorithms. Figure 6.1 shows the results for online performance. In all three domains OOVI is either competitive with the other control algorithms, or outperforms the

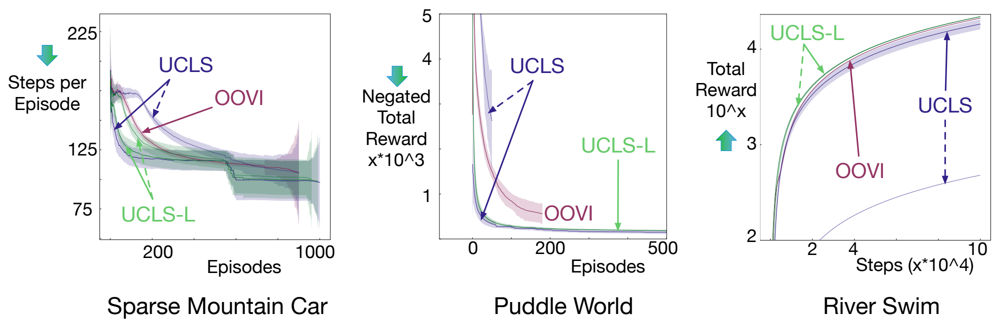

Figure 6.3: Learning performance in the three stochastic domains comparing OOVI to variants of UCLS: quadratic complexity UCLS and linear complexity UCLS-L. UCLS variants with solid lines incorporate traces with $\lambda = 0.9$, whereas the dotted line variants do not. UCLS (with traces) goes on to finish about 750 episodes in Puddle World, whereas UCLS-L (with traces) finished about 1750 episodes.

algorithms. In Sparse Mountain Car OOVI outperforms the other algorithms whereas in Puddle World UBE outperforms OOVI significantly. OOVI's poor performance can possibly be attributed to the algorithm requiring more planning updates. Therefore, a version with more planning updates, denoted by OOVI+, is evaluated in Figure 6.2 and it matches up to the performance of UBE. Both RLSVI and BSP fail to finish a single episode in Puddle World. In River Swim OOVI outperforms all the algorithms by a large margin. RLSVI, BSP and UBE discover the upstream reward but experience a large regret online.

Comparing to UCLS variants: Our next experiment compares OOVI to variants of UCLS from Chapter 4. Specifically, we compare to both UCLS and UCLS-L. While OOVI utilizes off-policy replay to improve sample-efficiency, the on-policy, incremental algorithms like UCLS utilize traces. Therefore, here we compare to two variants of UCLS and UCLS-L: (1) with $\lambda = 0.9$ (with traces), and (2) with $\lambda = 0.0$ (without traces).

Figure 6.3 shows the results for online performance of all the UCLS variants and OOVI. There is no statistical difference between all the variants of UCLS and OOVI. Both UCLS and UCLS-L with traces outperform OOVI in Puddle World, whereas the without traces versions perform poorly, with UCLS-L without traces failing. In Riverswim both variants of UCLS-L and UCLS with traces performs similar to OOVI, whereas UCLS without traces performs poorly. UCLS-L consists of two step-sizes that can independently control the mean and epistemic uncertainty update rate, thereby possibly aiding UCLS-L retain its out-of-sample epistemic uncertainty longer. In general, UCLS and UCLS-L with traces (denoted by solid lines) always outperform or similarly, to their no traces counterparts (denoted by dashed lines).

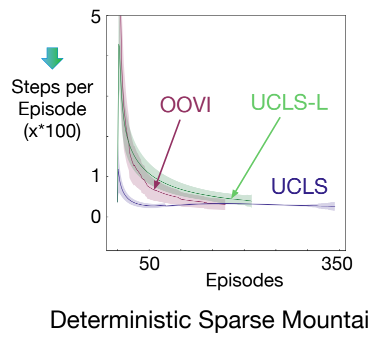

Figure 6.4: Learning curves comparing all the online control algorithms, variants of UCLS, and OOVI in Deterministic Sparse Mountain Car. The online control algorithms RLSVI, BSP, and UBE, along with UCLS variants with $\lambda = 0$ are not visible as they fail to finish any episodes and learn in this domain.

Exploration in Hard Deterministic Problems: As the reward is sparse in Sparse Mountain Car (1 upon reaching the goal, 0 otherwise), it is surprising that all algorithms finish the first episode early in Figure 6.1(left). This is possibly because the domain is stochastic, unlike the classic version which is deterministic, making it easier for all algorithms. Therefore we experiment with the classic version of Sparse Mountain Car here, using representations that are re-learned with respect to the deterministic dynamics. Figure 6.4 shows the performance of all the algorithms, including UCLS variants for completeness. Other online control algorithms and UCLS variants without traces fail to finish episodes in the domain, whereas UCLS with with traces performs better than OOVI. The deterministic nature of the domain along with UCLS' summarized model information may help maintain more accurate estimates of epistemic uncertainty, aiding in the above performance. OOVI with more planning updates should obtain the same benefit. Nonetheless, the main take away of the experiment is that OOVI's exploration strategy is more viable in deterministic hard exploration problems when compared to other control algorithms.

These results indicate that OOVI is a competitive online algorithm, whose directed exploration strategy is promising to tackle hard exploration problems. While some parameters of OOVI have been swept, some of its parameters like number of planning steps and size of the online buffer have not. These parameters can be tuned in a domain specific manner to improve online performance. Next, we evaluate how changing these parameters impacts the performance of OOVI.

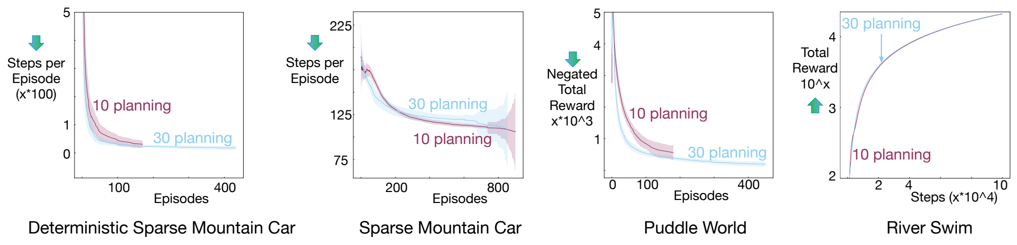

Figure 6.5: Learning curves evaluating the benefit of increasing replay in all the domains for OOVI. *10 planning* denotes OOVI with 10 replay steps, whereas *30 planning* denotes OOVI with 30 replay steps. In the first three plots lower y-axis is better, and higher y-axis is better in last.

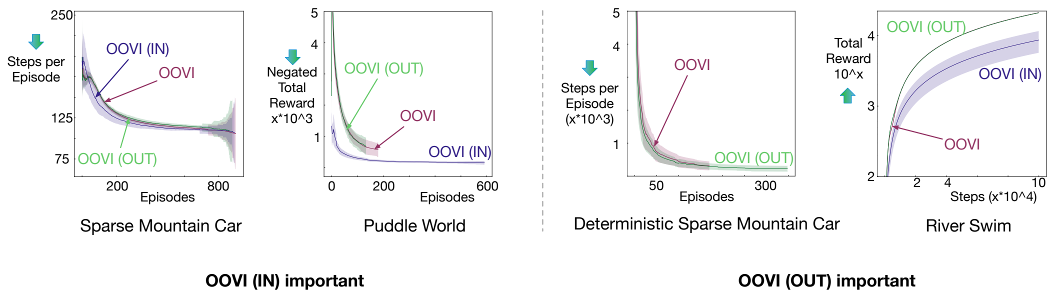

Figure 6.6: Learning curves evaluating the importance of the two different epistemic uncertainty components used by OOVI. *OOVI (IN)* represents version of OOVI utilizing only in-sample epistemic uncertainty, whereas *OOVI (OUT)* represents version of OOVI with only out-of-sample epistemic uncertainty.

6.4.2 Evaluating the Benefit of Replay Updates in OOVI

In this section we explore if more replay updates can benefit OOVI. The increase of replay should would improve the estimation of all the parameters – $\mathbf{w}^o$, $\mathbf{w}$, $\mathbf{w}^{\text{e-in}}$, and $\mathbf{w}^{\text{e-out}}$ – leading to better online sample-efficiency. In the previous experiments we used 10 replay steps, but in this section we increase it to 30 and evaluate how this impacts performance.

The results comparing OOVI with 10 planning steps to OOVI with 30 planning steps are presented in Figure 6.5 for all the four domains including the deterministic version of Sparse Mountain Car (shown on the extreme left of the plot). With more planning steps the online performance of OOVI improves across all the domains. Although more replay steps can be computationally expensive, implementation of OOVI can be parallelized to improve the time complexity of utilizing more replay steps.

6.4.3 Evaluating OOVI's Epistemic Uncertainty Components

OOVI utilizes two different components to promote online exploration – $\mathbf{w}^{\text{e-in}}$ to estimating in-sample epistemic uncertainty, and $\mathbf{w}^{\text{e-out}}$ for estimating out-of-sample epistemic uncertainty. In this section we evaluate the role of each component to promote effective exploratory online behaviour in the domains experimented. We do so by experimenting with two versions of OOVI – (1) OOVI (OUT) - a version of OOVI that utilizes only the out-of-sample epistemic uncertainty component, and (2) OOVI (IN) - a version of OOVI that utilizes only the in-sample epistemic uncertainty component.

We hypothesize that different epistemic uncertainty components play a significant role in these domains. In Sparse Mountain Car and Puddle World, which are uniformly stochastic domains, in-sample epistemic uncertainty may play a more significant role. Alternatively, in Deterministic Sparse Mountain Car, where the domain is deterministic, out-of-sample epistemic uncertainty may play a more significant role. And likewise, in River Swim, where the transition dynamics are stochastic but skewed towards the downstream less rewarding region of the chain-like domain, out-of-sample epistemic uncertainty may be more important.

The results for the experiments are shown in Figure 6.6. While OOVI (IN) improves marginally over OOVI in Sparse Mountain Car, it improves significantly in Puddle World, indicating that the out-of-sample epistemic uncertainty provided by $\mathbf{w}^{\text{e-out}}$ can likely be decayed more aggressively in this domain for this representation. Alternatively, OOVI (IN) experiences higher online regret compared to OOVI in River Swim and fails to finish any episodes in Deterministic Sparse Mountain Car.

6.4.4 Evaluating OOVI's Sensitivity to the Size of the Online Buffer

In this section we explore the sensitivity of OOVI to the size of the online buffer. As more data offers a better coverage of the dynamics of the problem, it should likely also improve estimation of all the parameters in OOVI – $\mathbf{w}^{o}$, $\mathbf{w}$, $\mathbf{w}^{\text{e-in}}$, and $\mathbf{w}^{\text{e-out}}$. Therefore, here we hypothesize that a decrease in the online buffer size should degrade online performance.

In the previous experiments OOVI stored all the data experienced online (100% data). Here we compare it to using a circular buffer that retains only 10% of the online data. The results are shown in Figure 6.7. While the online performance does degrade in Sparse Mountain Car and Puddle World as hypothesized, surprisingly it improves in Deterministic Sparse Mountain Car and River Swim.

This improvement in performance is possibly because the 10% online buffer concentrates updates rapidly decaying out-of-sample epistemic uncertainty more aggressively, and estimating in-sample epistemic uncertainty towards the on-policy distribution. Therefore, if true, with a smaller buffer, for example 1% buffer, the online performance should improve.

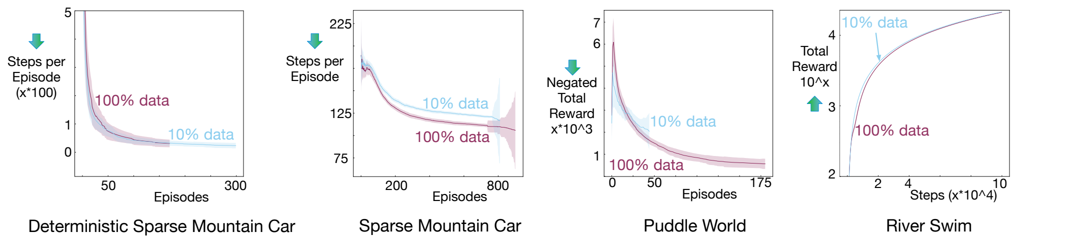

Figure 6.7: Learning curves evaluating OOVI's sensitivity to the size of the online buffer. *100% data* denotes a buffer of size 100k, equal to the length of the experiment. *10% data* denotes a buffer of size 10k.

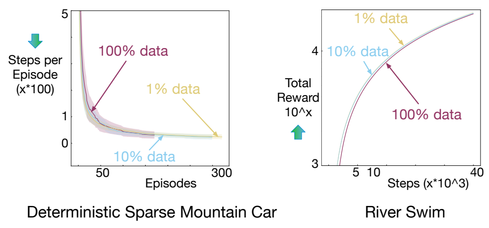

Figure 6.8: Learning curves evaluating OOVI's online performance with respect to smaller buffers in two domains. *100% data* utilizes all the data experienced online, whereas *10% data* utilizes a circular buffer of 10k, and *1% data* utilizes a circular buffer of 1k.

Figure 6.8 shows that 1% buffer does improve over 10% buffer in both Deterministic Sparse Mountain Car and River Swim. It is possible that a more extreme version OOVI which utilizes only online samples may improve performance further. Such a version would be more similar to UCLS, except differing in their approach to estimating optimistic values.

Overall these results show OOVI can be sensitive to the size of the online buffer – in domains where out-of-sample. epistemic uncertainty is critical, smaller buffers may improve online performance.

6.5 Summary

This chapter provided an overview of online control algorithms and key ideas that make them effective. Then, to address the limitation of UCLS it proposes a new algorithm called OOVI. OOVI utilizes the different parameters to estimate the two sources of epistemic uncertainty: in-sample and out-of-sample. It incorporates these uncertainty estimates to guide on-policy control by utilizing a modified value iteration style update. The proposed algorithm is empirically evaluated by comparing to other online exploration algorithms that utilize uncertainty to guide exploration, and other properties of the algorithm are also explored.

Chapter 7

Perspectives and Future Work

In this chapter, we summarize the main contributions of this thesis, and discuss their limitations. We provide some directions for future work which we think will be beneficial for sample-efficient online reinforcement learning, before concluding.

7.1 Summary of Contributions

This thesis set out to answer the following question:

> How should an agent incorporate directed exploration and representation learning to improve the sample efficiency of online reinforcement learning under linear function approximation?

In the end, this dissertation presents a partial answer to this ambitious question, while making progress in three key areas. First, we provide a scalable approach to estimate value uncertainty. These value uncertainty estimates are derived for the LSTD(λ) algorithm and provide high-probability finite-sample bounds for the optimal values. Second, we provide an approach to offline representation learning that improves the sample-efficiency of online reinforcement learning under linear function approximation. The representations learned are shown to also aid online exploration. Third, we provide online control algorithms that incorporate directed exploration strategies for improving sample-efficiency. One is an incremental on-policy algorithm that utilizes a heuristic for effective exploration, while the second incorporates a more principle approach to effective exploration and improves upon the first by being compatible with planning updates. Next, we summarize these key contributions in more detail.

Value Uncertainty Estimation

Propagation of uncertainty in the approximation setting has been an open research question for many years. Though various promising approaches have made progress by making assumptions about the stochastic structure of the environment, or the representation space used, these assumptions may not be valid generally in environments and representations used in practice. In this thesis, we present a scalable approach to propagate uncertainty under simpler assumptions, like the invertibility of a covariance matrix. Our approach is promising because it provides high-probability finite-sample bounds with respect to the optimal value of any policy under linear function approximation.

Additionally, as the uncertainty estimates are derived for the LSTD(λ) algorithm, they are compatible with traces, retaining the sample-efficiency benefits of the policy evaluation algorithm. These value uncertainty estimates have been used to guide online exploration effectively under the generalized policy iteration framework in two algorithms proposed here. We present a quadratic complexity confidence-bound, and a more computationally scalable linear complexity confidence-bounds. The linear complexity bound, while more computationally efficient, retains the utility of the quadratic-complexity bound for directing online exploration.

Representations for Online Sample-Efficiency

Arguably, before the recent success of reinforcement learning with neural networks, successful applications of online reinforcement learning were limited to representations which encoded locality, such as tile-coding. As online reinforcement learning algorithms utilize bootstrapping, locality is crucial to good learning targets and improve sample-efficiency. We provide a simple strategy to learn representations that encode this property using neural networks. Motivated by the successful hand-designed representation tile-coding, we utilize sparsity as a surrogate for locality.

We show that the learned representations, SR-NNs, improves sample-efficiency of online reinforcement learning algorithms when compared to other representations that have the same representational capacity. This is possible because SR-NNs reduce interference in bootstrap learning targets. Our experiments also show that the locality encoded by SR-NNs is also useful to guide online exploration.

Directed Exploration for Online Sample-Efficiency

Directing effective exploration while optimizing behaviour online involves two critical goals: (1) visiting parts of the space which the agent is uncertain about, and (2) visiting parts of the space an agent does not know about. This thesis presents a heuristic strategy, and a more principled approach, to incorporate these goals in the online reinforcement learning setting. The heuristic is used to design an on-policy directed exploration algorithm that is compatible with traces, called UCLS. The framework is used to design an algorithm called OOVI that is replay-compatible, providing online sample-efficiency by incorporating planning updates.

UCLS is an algorithm that adapts the LSTD algorithm to the control setting by incorporating its uncertainty estimates for addressing goal (1), and a heuristic for addressing goal (2). We demonstrate that UCLS improves upon other algorithms that estimate value uncertainty to drive online exploration. OOVI utilizes the proposed uncertainty estimation procedure to estimate in-sample epistemic uncertainty that addresses goal (1), and it utilizes an optimistic initialization based component for effective out-of-sample epistemic uncertainty to address goal (2). The algorithm can be parallelized with more compute to scale in sample-efficiency benefits. Extensive empirical comparison to algorithms with similar goals shows that OOVI's design, which explicitly accounts for the two goals of exploration, is indeed more effective in hard exploration microworlds.

7.2 Limitations and Future Work

In this section we look at the limitations of the contributions, and present some directions for future work that can improve upon these limitations and explore promising research directions.

λ-returns and Variance Issues in the Off-Policy Case

An important goal while designing OOVI was to maintain compatibility with planning, as planning is key for improving sample-efficiency. Another approach to improve sample-efficiency, which we have not used with OOVI, are λ-returns. Our uncertainty estimation algorithm is compatible with traces which are used to estimate λ-returns, but OOVI utilizes single-step targets where $\lambda = 0$.

In general, off-policy value estimation is a problem that is susceptible to high variance issues. Additionally, while sample-efficiency of learning can be improved via λ-returns, the utilization of traces in the off-policy case can exacerbate this issue if the behaviour policy is very different from the target policy. But, algorithms like Tree-Backup (Precup, 2000) can be used to alleviate the issue.

Tree-Backup cannot be integrated with OOVI trivially. First, the target policy and the behaviour policy in OOVI are constantly changing. Second, the policies considered currently are deterministic, which can exacerbate variance issues. Therefore, incorporating traces through smart variance-reduction strategies such as Tree-Backup may not be sufficient. An avenue to explore for incorporating traces effectively may be utilizing target policies that are soft-improvements, rather than greedy improvements, along with behaviour that is stochastic with respect to the estimated optimistic values. This can possibly reduce the number of planning steps needed, while providing additional online sample-efficiency benefits.

Theoretical Challenges

Optimistic Values Theorem was an important motivation that served the design of UCLS. Although motivated by it, a limitation of UCLS was highlighted in Chapter 6, as follows.As UCLS utilizes a heuristic strategy to promote out-of-sample epistemic uncertainty, it is possible that it does not satisfy the assumptions made by the theorem, making the optimality of the policy learned by UCLS unclear. OOVI was designed to address this limitation, along with being an algorithm that is compatible with planning.

UCLS uses $\hat{q}(S, A)$ as the value estimates, and $\hat{u}(S, A)$ as the uncertainty estimates. The uncertainty estimates $\hat{u}(S, A)$ are solely required to characterize both out-of-sample and in-sample epistemic uncertainty, and consequently, for providing any necessary optimism. In order to improve upon UCLS, OOVI utilizes an explicit (1) out-of-sample epistemic uncertainty estimator to promote optimism with respect to the unvisited parts of the MDP, $\hat{u}^o(S, A)$, and (2) an in-sample epistemic uncertainty estimator $\hat{u}^i(S, A)$ to account for any uncertainty in the visited parts of the MDP. The estimates made by $\hat{u}^i(S, A)$ are incorporated in its value estimator $\hat{q}(S, A)$, to promote partially optimistic value estimates, making it different from UCLS. $\hat{q}(S, A)$ are then combined with the $\hat{u}^o(S, A)$ estimates to provide the complete optimistic value estimates.

Therefore, the Optimistic Values Theorem, which provides a motivational regret bound for the setup used by UCLS, cannot be directly applied to OOVI (as OOVI utilizes a value estimator $\hat{q}(S, A)$ that incorporates some degree of optimism). Consequently, despite promising empirical results, there is still a need for some theoretical characterization of the policies learned by OOVI.

Online Representation Learning

In general, it is difficult to learn representations that lend themselves to effective linear value estimation. The representations used in this thesis were effective because they were trained using data generated by reasonable policies in each environment. But designing reasonable policies for effective offline representation learning is challenging in and of itself. Therefore, it is necessary that either the representation learning strategies used are online, or the value functions utilize non-linear approximators, which can be more powerful when the representations used are not effective under linear approximation.

The proposed strategy for representation learning, SR-NN, is a strategy that utilizes a stationary policy. While the representations produced by it are effective for online reinforcement learning, where a policy is constantly changing, learning an effective representation with SR-NN can be hard in the online case. This hypothesis is based on preliminary experiments where such a strategy fails to be effective. A possible reason for this failure is that the proposed regularizer is not easy to optimize under a changing data distribution. Additionally, we show that the representations produced by SR-NNs, which are the last hidden layer of an offline trained neural network, are effective for providing stable bootstrap targets in the case of linear function approximation. But it is possible that an online representation learning component, or a non-linear value function approximator, may need more aggressive locality enforcing regularizers, possibly in all the intermediate layers. Another simple avenue to use SR-NNs effectively in the online case may be to interleave representation learning and reinforcement learning itself — where during representation learning the policy is fixed — but the schedule for interleaving such a procedure, and its sample-efficiency is unclear.

Effectiveness Under Non-Linear Approximation

This thesis is focused on the linear function approximations setting. But as discussed in the previous section, naturally, the next step is exploring OOVI with a non-linear function approximator. There are two key challenges to such an extension: (1) initializing and maintaining effective out-of-sample epistemic uncertainty, and (2) ensuring function spaces with low-inherent Bellman error.

I believe addressing both the challenges requires non-linear networks that encode locality. While locality is important for initializing and maintaining effective out-of-sample epistemic uncertainty, the reason I believe it is also important for promoting function spaces with low inherent Bellman error is as follows. If a representation does encode locality, it can improve the sample-efficiency of online reinforcement learning, reducing the magnitude of errors experienced online. The magnitude of errors experienced online has a direct relationship to the magnitude of the inherent Bellman error. Therefore, enforcing locality in a non-linear approximator may be key to addressing both the challenges. Nonetheless, the fundamental question remains: how do we train non-linear approximators during online reinforcement learning to encode locality? This is an interesting direction for future work.

7.3 Summary

This thesis identifies key ideas necessary for directed exploration and sample-efficient online learning under linear function approximations, and proposes solutions which incorporate them. This chapter closed this thesis by summarizing the key contributions, discussing their limitations, and proposing some interesting directions for future research.

Bibliography

Abbasi-Yadkori, Y. and Szepesvari, C. (2014). Bayesian optimal control of smoothly parameterized systems: The lazy posterior sampling algorithm. In *Uncertainty in Artificial Intelligence*.

Ahmad, S. and Hawkins, J. (2015). Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory.

Auer, P. and Ortner, R. (2006). Logarithmic online regret bounds for undiscounted reinforcement learning. *Advances in Neural Information Processing Systems*.

Banerjee, A., Merugu, S., Dhillon, I. S., and Ghosh, J. (2005). Clustering with bregman divergences. *Journal of Machine Learning Research*.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*.

Bartlett, P. L. and Tewari, A. (2012). Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. *arXiv preprint arXiv:1205.2661*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *CoRR*.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Bertsekas, D. P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming*.

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

Boyan, J. A. (2002). Least-squares temporal difference learning. *Machine Learning*.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*.

Brafman, R. and Tennenholtz, M. (2003). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Choshen, L., Fox, L., and Loewenstein, Y. (2018). DORA the explorer: Directed outreaching reinforcement action-selection. *CoRR*.

Cover, T. M. (1965). Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Trans. Electronic Computers*.

Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*.

French, R. M. (1991). Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Annual Cognitive Science Society Conference*.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., and White, A. (2018). Online off-policy prediction. *arXiv preprint arXiv:1811.02597*.

Ghiassian, S., Rafiee, B., Lo, Y. L., and White, A. (2020). Improving performance in reinforcement learning by breaking generalization in neural networks. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*.

Goodfellow, I., Lee, H., Le, Q. V., Saxe, A., and Ng, A. Y. (2009). Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems*.

Grande, R., Walsh, T., and How, J. (2014). Sample efficient reinforcement learning with gaussian processes. In *International Conference on Machine Learning*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning, lecture 6a, overview of mini-batch gradient descent.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *The Journal of Machine Learning Research*.

Jin, C., Yang, Z., Wang, Z., and Jordan, M. I. (2019). Provably efficient reinforcement learning with linear function approximation. *arXiv preprint arXiv:1907.05388*.

Jong, N. and Stone, P. (2007). Model-based exploration in continuous state spaces. *Abstraction, Reformulation, and Approximation*.

Jung, T. and Stone, P. (2010). Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *Machine Learning: ECML PKDD*.

Kaelbling, L. P. (1993). *Learning in embedded systems*. MIT press.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*.

Kakade, S., Kearns, M., and Langford, J. (2003). Exploration in metric state spaces. In *International Conference on Machine Learning*.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *Proceedings of the19th International Conference on Machine Learning*.

Kallenberg, L. (2011). Markov decision processes. *Lecture Notes. University of Leiden*, pages 2–5.

Kanerva, P. (1988). *Sparse Distributed Memory*. MIT Press.

Kearns, M. J. and Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Konidaris, G., Osentoski, S., and Thomas, P. (2011). Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*.

Kumaraswamy, R., Schlegel, M., White, A., and White, M. (2018). Context-dependent upper-confidence bounds for directed exploration. *Advances in Neural Information Processing Systems*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *World Wide Web Conference*.

Li, L., Littman, M. L., and Mansley, C. R. (2009). Online exploration in least-squares policy iteration. In *International Conference on Autonomous Agents and Multiagent Systems*.

Liu, V., Kumaraswamy, R., Le, L., and White, M. (2019). The utility of sparse representations for control in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2018). Count-based exploration with the successor representation. *CoRR*.

Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*.

Makhzani, A. and Frey, B. (2013). k-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

Makhzani, A. and Frey, B. (2015). Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*.

Martin, J., Sasikumar, S. N., Everitt, T., and Hutter, M. (2017). Count-based exploration in feature space for reinforcement learning. In *International Joint Conference on Artificial IntelligenceI*.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation*.

Meuleau, N. and Bourgine, P. (1999). Exploration of multi-state environments - local measures and back-propagation of uncertainty. *Machine Learning*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*.

Moerland, T. M., Broekens, J., and Jonker, C. M. (2017). Efficient exploration with double uncertain value networks. In *Advances in Neural Information Processing Systems*.

Munos, R. (2003). Error Bounds for Approximate Policy Iteration.

Munos, R. and Szepesvári, C. (2008). Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*.

Neu, G. and Pike-Burke, C. (2020). A Unifying View of Optimism in Episodic Reinforcement Learning. *arXiv.org*.

Ng, A. (2011). Sparse autoencoder. *CS294A Lecture notes*.

Nikolov, N., Kirschner, J., Berkenkamp, F., and Krause, A. (2018). Information-directed exploration for deep reinforcement learning. *CoRR*.

Nouri, A. and Littman, M. L. (2009). Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems*.

ODonoghue, B., Osband, I., Munos, R., and Mnih, V. (2017). The uncertainty bellman equation and exploration.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*.

Ortner, R. and Ryabko, D. (2012). Online regret bounds for undiscounted continuous reinforcement learning. In *Advances in Neural Information Processing Systems*.

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized Prior Functions for Deep Reinforcement Learning. *NeurIPS*.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016a). Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*.

Osband, I., Russo, D., and Van Roy, B. (2013). (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*.

Osband, I. and Van Roy, B. (2017). Why is posterior sampling better than optimism for reinforcement learning? In *International Conference on Machine Learning*.

Osband, I., Van Roy, B., and Wen, Z. (2016b). Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *International Conference on Machine Learning*.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*. PMLR.

Pazis, J. and Parr, R. (2013). Pac optimal exploration in continuous space markov decision processes. In *AAAI Conference on Artificial Intelligence*.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv.org*.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Quian Quiroga, R. and Kreiman, G. (2010). Measuring sparseness in the brain: Comment on bowers (2009).

Ratitch, B. and Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. In *Machine Learning: ECML PKDD*.

Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Inter. Conf. on Machine Learning*.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK.

Russo, D. and Roy, B. V. (2014). Learning to optimize via information directed sampling. *CoRR*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*.

Singh, S. P., Jaakkola, T. S., Littman, M. L., and Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*.

Strehl, A. and Littman, M. (2004). Exploration via model based interval estimation. In *International Conference on Machine Learning*.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). Pac model-free reinforcement learning. In *International Conference on Machine Learning*.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*.

Sutton, R., Szepesvári, C., Geramifard, A., and Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In *Conference on Uncertainty in Artificial Intelligence*.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. P. (2012). Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*.

Szepesvári, C. (2010). Algorithms for Reinforcement Learning. *Algorithms for Reinforcement Learning*.

Szita, I. and Lorincz, A. (2008). The many faces of optimism. In *International Conference on Machine Learning*.

Szita, I. and Szepesvari, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds. In *International Conference on Machine Learning*.

Tang, Z., Feng, Y., Zhang, N., Peng, J., and Liu, Q. (2020). Off-policy interval estimation with lipschitz value iteration. *CoRR*.

Tange, O. (2021). Gnu parallel 20210622 ('protasevich').

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25.

van Seijen, H. and Sutton, R. (2015). A deeper look at planning as learning from replay. In *International Conference on Machine Learning*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*.

Wei, C.-Y., Jafarnia-Jahromi, M., Luo, H., and Jain, R. (2020). Learning infinite-horizon average-reward mdps with linear function approximation. *arXiv preprint arXiv:2007.11849*.

White, M. (2017). Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*.

White, M. and White, A. (2010). Interval estimation for reinforcement-learning algorithms in continuous-state domains. In *Advances in Neural Information Processing Systems*.

Xie, T. and Jiang, N. (2020). Batch Value-function Approximation with Only Realizability. *arXiv.org*.

Zanette, A., Lazaric, A., Kochenderfer, M., and Brunskill, E. (2020). Learning Near Optimal Policies with Low Inherent Bellman Error. *arXiv.org*.

Appendix A

Incremental Control with Policy Evaluation Uncertainty Estimates Appendix

A.1 Comparing UCLS and DGPQ

We include plots showing best and worst runs for UCLS and DGPQ — the two closest competitors — to show the variance of each algorithm Figure A.1. While UCLS exhibits relatively high variance in Sparse Mountain Car, DGPQ exhibits high variance across all the domains.

A.2 Issues with LSTD for control

Although LSTD is a policy evaluation algorithm, it has been used successfully in many control problems. Here, we explore why it has been successful in this setting.

LSTD is a more data-efficient algorithm than its incremental counterpart TD, and typi-

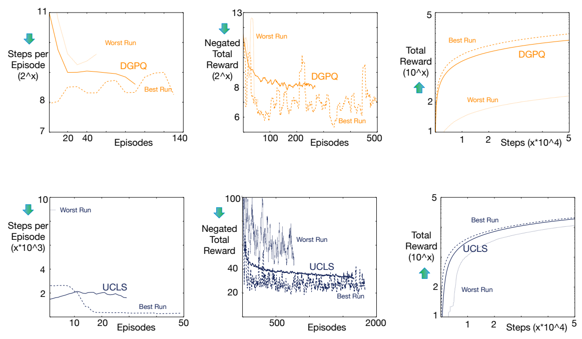

Figure A.1: Learning curves comparing best and worst runs for DGPQ and UCLS. DGPQ is the top row, whereas UCLS is the bottom row. From left to right: Sparse Mountain Car, Puddle World, River Swim.

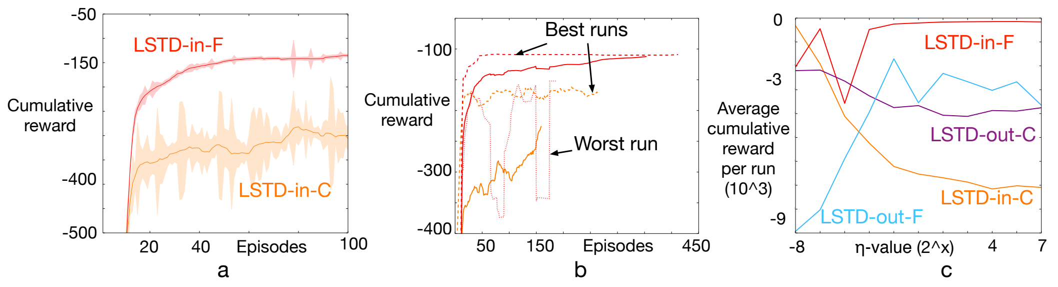

Figure A.2: Learning performance in Mountain Car for LSTD-in and LSTD-out with η kept constant through learning and η fading with time. η kept constant is denoted by (-C), and η fading with time is denoted by (-F). (a) Early learning curves for LSTD-in. This plot does not include LSTD-out as it performed too poorly to be visible. (b) Learning curves for LSTD-in with best and worst runs. LSTD-in-C's worst run performed too poorly to be visible. (c) Parameter sensitivity for both variants LSTD-in and LSTD-out to η/η_r.

cally performs quite well in policy evaluation. This is primarily due to TD only using each sample once for a stochastic update with a tuned stepsize parameter. In the case of control, LSTD performs surprisingly well without ϵ-greedy exploration and lack of an optimism strategy. We highlight here the inadvertent use of the regularization parameter as a form of optimism for LSTD in control, and empirically show when this strategy fails leading us to UCLS as a sound approach in using LSTD in control.

In practice, the inverted matrix $\mathbf{A}^{-1}$ is often directly maintained using a Sherman-Morrison update, with a small regularizer η added to the matrix $\mathbf{A}$ to guarantee invertibility (Szepesvári, 2010).

There are two objectives that can be solved when dealing with an ill-conditioned system $\mathbf{A}\mathbf{w} = \mathbf{b}$. The most common is to use Tikohonov regularization solving, referred to here as *LSTD-out*:

$$\min_{\mathbf{w}} \|\mathbf{A}\mathbf{w} - \mathbf{b}\|_2^2 + \eta_r \|\mathbf{w}\|_2^2.$$

Another approach is to solve the system:

$$\min_{\mathbf{w}} \|(\mathbf{A} + \eta\mathbf{I})\mathbf{w} - \mathbf{b}\|_2^2.$$

The second approach is implicitly what is solved when a Sherman-Morrison update is used for $\mathbf{A}^{-1}$, with a small regularizer η added to the matrix $\mathbf{A}$ to guarantee invertibility. This approach is referred to here as *LSTD-in*. When $\eta = 0$, both approaches are solving $\|\mathbf{A}\mathbf{w} - \mathbf{b}\|_2^2$, which may have infinitely many solutions if $\mathbf{A}$ is not full rank. While the Tikohonov regularization strategy is more common, the second approach is useful for enabling use of the incremental Sherman-Morrison update to facilitate maintaining $\mathbf{A}^{-1}$ directly.

Another choice in regularizing the ill-conditioned system is in how η decays over time. A small fixed η can be used as a constant regularizer, even as the number of samples increases, because the true $\mathbf{A}$ may be ill-conditioned. However, more regularization could also be used at the beginning and then decayed over time. The incremental Sherman-Morrison update implicitly decays η proportionally to $1/t$.

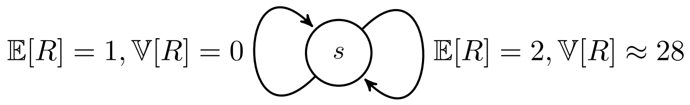
$\mathbb{E}[R] = 1, \mathbb{V}[R] = 0$
s
$\mathbb{E}[R] = 2, \mathbb{V}[R] \approx 28$

Figure A.3: One-state world, where the optimal action (right) has high-variance; the reward here is uniformly sampled from within the set $\{-5, -2, 2, 5, 10\}$. LSTD, with $\epsilon = 0$ and η large, fails in this world, unlike the cost-to-goal problems.

We conducted an empirical study using LSTD without an ϵ-greedy exploration strategy in two domains: Mountain Car and a new One-State world. One-State world—depicted in Figure A.3—simulates a typical setting where sufficient exploration is needed: one outcome with low variance and lower expected value and one outcome with high variance and higher expected value. For an algorithm that does not explore sufficiently, it is likely to settle on the suboptimal action, but more immediately rewarding low-variance outcome. This world simulates a larger continuous navigation task from White and White, 2010. We include results for both systems described above and consider a fading version (shown by *-F*) or a constant regularization parameter (shown by *-C*).

Figure A.2 shows results for the four different LSTD strategies in Mountain Car. The Tikohonov regularization, with η_r, is unable to learn an optimal policy in this domain, whereas with either constant or fading η, the agent can learn an optimal policy. This is surprising, considering we use neither randomized exploration nor optimistic initialization. The parameter sensitivity curve, shown in Figure A.2(c), indicates η and η_r needs to be sufficiently large as time passes in order to find an optimal policy.

Next, we show that neither regularization strategy with fading η is effective in the One-State world. The optimal strategy is to take the Right action, to get an expected reward of 2 under a higher variance for obtaining rewards. All of the LSTD variants fail for this domain, because η no longer plays a role in encouraging exploration. To verify that a directed exploration strategy helps, we experiment with ϵ-greedy exploration, with $\epsilon = 0.1$, decayed by a factor of 0.2 every 100 steps (shown in Figure A.4). With ϵ-greedy, and small

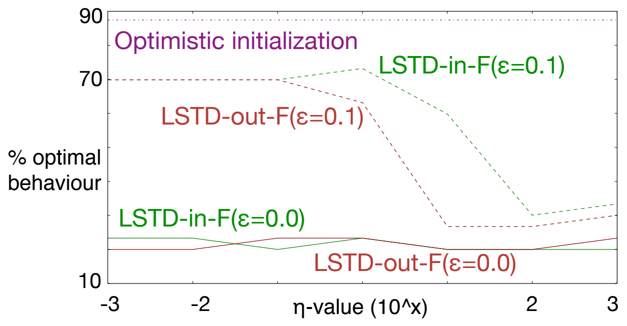

Figure A.4: η-sensitivity in 1-State world with various LSTD updates. Sarsa with optimistic initialization $\alpha = 0.001$ is used as a baseline. The y-axis represents percentage optimal behaviour, where optimal behaviour is choosing to go right, in 20k steps (averaged over 30 runs). Sarsa with optimistic initialization is highly sensitive to the step-size chosen. With other stepsizes (not shown in figure), it reduces its values too quickly, and fails a significant percentage of the time. The best stepsize is chosen here to show near-optimal performance is possible in the domain.

values of η_r and η, the policy converges to the optimal action, whereas it fails to with higher values of η_r and η.

These results suggest that η's role in exploration has obscured our understanding of how to use LSTD for control. LSTD, with sufficient optimism does seem to reach optimal solutions, and unlike Sutton et al. (2008), we did not find any issues with forgetting. This further explains why there have been previous results with small ϵ for LSTD in cost-to-goal problems, that nonetheless still obtained the optimal policy (van Seijen and Sutton, 2015). Therefore, in developing UCLS, we more explicitly add optimism to LSTD, and ensure η is strictly used as a regularization parameter (to ensure well-conditioned updates).

A.3 Details about Other Algorithms

A.3.1 Bootstrapped Upper-Confidence Bounds (UCBootstrap)

The strategy for action selection which utilizes bootstrapped confidence intervals, as proposed by White and White (2010), is given in Algorithm 9. This action selection strategy can be used in conjunction with any learning algorithm to guide on-policy control. The algorithm requires a window of recent $\mathbf{w}$'s. The window can be maintained with a circular queue. The window is updated after each learning step of the main algorithm, resulting in a new $\mathbf{w}_t$ in the queue. The original UCBootstrap paper proposed both a global and a sparse updating mechanism, where only the global approach was theoretically justified. The sparse mechanism was used to reduce the number of parameters stored, particularly by taking advantage of tile-coding representations. We found in our experiments that the global approach worked just as well as the sparse approach, and so we include only the simpler, theoretically justified algorithm.

Algorithm 9 UCBootstrap($\boldsymbol{\phi}_{s,\cdot}$) select action from state features $\boldsymbol{\phi}_{s,\cdot}$ at time t

```
$l$ = block length, $B$ = number of bootstrap resamples, $w$ = number (window) of value
functions weights to store and confidence level $\alpha$
examples: $l = 10$, $B = 50$, $w = 100$, $\alpha = 0.05$
  $M \leftarrow \lfloor w/l \rfloor$        ▷ num of length $l$ blocks to sample with replacement and concatenate
  for each action $a$ do
      $Q_N \leftarrow \{\mathbf{w}_{t-w}^\top \boldsymbol{\phi}_{s,a}, \ldots, \mathbf{w}_{t-1}^\top \boldsymbol{\phi}_{s,a}\}$
      $\bar{Q}_N \leftarrow$ mean($Q_N$)        ▷ The mean value for this $(s, a)$, given the window of recent
  weights
      Blocks = $\Big\{\{[Q_N[0], \ldots, Q_N[l\text{-}1]\}, \{[Q_N[1], \ldots, Q_N[l]\},$
                         $\ldots, [Q_N[w\text{-}l], \ldots, Q_N[w\text{-}1]]\Big\}$
      for all $i = 1$ to $B$ do
          for all $j = 1$ to $M$ do
              $A_j^* \leftarrow$ random block from Blocks (chosen with replacement)
          $A \leftarrow (A_1^*, A_2^*, \ldots, A_M^*)$                            ▷ Concatenate blocks
          $T_i^* = \frac{1}{lM} \sum_{k=1}^{lM} A[k]$ ▷ $i$th bootstrap estimate is the mean of the $M$ concatenated
  blocks
      $T \leftarrow$ sort($\{T_1^*, \ldots, T_B^*\}$)                                  ▷ ascending order
      $j \leftarrow \lfloor \frac{B\alpha}{2} + \frac{\alpha+2}{6} \rfloor$        ▷ $j$ is the position of the critical samples to help estimate the
  continuous sample quantile
      $r \leftarrow \frac{B\alpha}{2} + \frac{\alpha+2}{6} - j$                              ▷ $r$ is the remainder
      $T_{\alpha/2}^* \leftarrow (1 - r)T_j^* + rT_{j+1}^*$                       ▷ the $\alpha/2$ sample quantile
      $u_a \leftarrow 2\bar{Q}_N - T_{\alpha/2}^*$
  $a = \text{argmax}_{a \in \mathcal{A}} u_a$
  return $a$
```

A.3.2 Delayed Gaussian Process Q-Learning (DGPQ)

Another approach to exploration is found in a model-free algorithm using gaussian processes named Delayed-GPQ (DGPQ) (Grande et al., 2014). The pseudocode for DGPQ is in Algorithm 10.

Algorithm 10 DGPQ($k(\cdot,\cdot), d(\cdot,\cdot), L_Q, Env, \mathcal{A}, R_{max}, s_0, \gamma, \sigma^2, \sigma^2_{tol}, \epsilon$)

```
$k(\cdot,\cdot), d(\cdot,\cdot)$ are typically the RBF w/ bandwidth = $\sigma^2$ and euclidean distance respectively.
$L_Q$ correlates with exploration.
$\mathcal{A}$ is the set of possible actions.
$\gamma$ is the discount factor.
$\sigma^2_{tol}$ is the tolerance of induced variance of using a new point to update a GP
  $\hat{Q}(s,a) \overset{\text{def}}{=} \min($
          $V_{max},$
          $\min_{(s_i,a) \in \hat{Q}_a.BV} \{[\hat{\mu}_i + L_Q d((s,a),(s_i,a))]\}$
          $)$
  for $a \in A$ do
      $\hat{Q}_a.BV = \emptyset$
      $GP_a = GP.init(\mu = \frac{Rmax}{1-\gamma}, k(\cdot,\cdot))$
  for $t \in [0,T]$ do
      $a_t = \operatorname{argmax}_a \hat{Q}(s,a)$
      //take action $a_t$ in state $s_t$, observe $(s_{t+1}, r_t)$
      $(s_{t+1}, r_t) = Env(s_t, a_t)$
      $q_t = r_t + \gamma \max_a \hat{Q}(s_{t+1})$
      $\sigma_1^2 = GP_{a_t}.variance(s_t)$
      //If the new sample is not well covered by $GP_{a_t}$
      if $\sigma_1^2 > \sigma^2_{tol}$ then
          $GP_{a_t}.update(s_t, q_t)$
      $\sigma_2^2 = GP_{a_t}.variance(s_t)$
      //If the $GP_{a_t}$ now well covers a previously unknown state and the new approximation
  is $2\epsilon$ less than what is found in $\hat{Q}$ (i.e. is a less optimistic estimate).
      if $\{\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2\}$ and
       $\left\{\hat{Q}_{a_t}(s_t) - GP_{a_t}.mean(s_t) > 2\epsilon\right\}$ then
          $\mu = GP_{a_t}.mean(s_t) + \epsilon$
          $\hat{Q}_{a_t}.BV.add((s_t, a_t), \mu)$
          for $((s_j, a_t), \mu_j) \in \hat{Q}_{a_t}.BV$ do
              if $\mu_j \leq \mu + L_Q d((s_t, a_t), (s_j, a_t))$ then
                  $\hat{Q}_{a_t}.BV.delete(((s_j, a_t), \mu_j))$
          //To prevent slow learning or halted learning reset the current GPs and initialize
  to the current estimates.
          $\forall a \in A, GP_a = GP.init(\hat{\mu} = \hat{Q}_a, k(\cdot,\cdot))$
```

A.3.3 Least Squares Policy Iteration - Rmax (LSPI-Rmax)

LSPI-Rmax Li et al. (2009) combines LSPI Lagoudakis and Parr (2003) with Rmax Brafman and Tennenholtz (2003) for online control in continuous state-spaces. Exploration is encouraged by determining the *knowness* of a transition, utilizing kernels. LSPI algorithm is designed for a batch setting, where the LSTD solution is computed in closed form for staged batches of data. However, because it accumulates optimistic values, it can be simply converted into an online algorithm using incremental updates to the matrix $\mathbf{A}$ and $\mathbf{b}$, as done in Li et al. (2009). We summarize this extension in pseudocode as Algorithm 11.

Until states become known, the algorithm estimates action-values that predict the maximum possible return; once a state becomes known, it starts to use actual rewards sampled from the environment. To estimate the *knowness* of a state under function approximation, we use feature counts. Each state has a set of active features; the active feature with the minimum count reflects an upper bound on the number of times that this state has been seen. Once a states active features have been seen frequently enough, it becomes known.

Algorithm 11 Incremental LSPI-Rmax(m)

```
$\mathbf{A} \leftarrow \mathbf{0}$, $\mathbf{b} \leftarrow \mathbf{0}$, $\mathbf{e} \leftarrow \mathbf{0}$, $\mathbf{w} \leftarrow \mathbf{0}$,
$\mathbf{B} \leftarrow \mathbf{I}$, $\mathbf{c} \leftarrow \mathbf{0}$
$\eta = 10^{-4}$, $\beta = 0.001$, $\lambda = 0$, $G_{\text{max}} = r_{\text{max}}/(1-\gamma)$ if continuing or $\gamma \neq 1$, else $G_{\text{max}} = r_{\text{max}} h$ for a predicted maximum episode length (e.g., $h = 10000$).
$\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
$a \leftarrow$ greedy action according to value estimates given by $\boldsymbol{\phi}_{s,a}^\top \mathbf{w}$
repeat
  Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
  $a' \leftarrow$ greedy action according to value estimates given by $\boldsymbol{\phi}_{s',a'}^\top \mathbf{w}$
  $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \boldsymbol{\phi}_{s,a}$
  if IsKnown$(s, a)$ then
    if IsKnown$(s')$ then
      $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\mathbf{e}(\boldsymbol{\phi}_{s,a} - \gamma\boldsymbol{\phi}_{s',a'})^\top$
      $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta r\mathbf{e}$
    else
      $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\boldsymbol{\phi}_{s,a}\boldsymbol{\phi}_{s,a}^\top$
      $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta(r + \gamma G_{\text{max}})\boldsymbol{\phi}_{s,a}$
  else
    $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\boldsymbol{\phi}_{s,a}\boldsymbol{\phi}_{s,a}^\top$
    $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta G_{\text{max}}\boldsymbol{\phi}_{s,a}$
  for $\forall \tilde{a} \in A \backslash a$ do
    if !IsKnown$(s, \tilde{a})$ then
      $\mathbf{A} \leftarrow (1-\beta)\mathbf{A} + \beta\boldsymbol{\phi}_{s,\tilde{a}}\boldsymbol{\phi}_{s,\tilde{a}}^\top$
      $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta G_{\text{max}}\boldsymbol{\phi}_{s,\tilde{a}}$
  $\mathbf{c} \leftarrow \mathbf{c} + \boldsymbol{\phi}_{s,a}$
  $\alpha = \min\left\{1.0, \frac{0.01}{||\mathbf{A}||_F^2 ||\boldsymbol{\phi}_{s,a}||_2^2 + 1.0}\right\}$
  $\mathbf{B} \leftarrow \mathbf{B} - \alpha\mathbf{A}(\mathbf{A}^\top\mathbf{B}\boldsymbol{\phi}_{s,a} - \boldsymbol{\phi}_{s,a})\boldsymbol{\phi}_{s,a}^\top$
  $\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{B} + \eta\mathbf{I})(\mathbf{b} - \mathbf{A}\mathbf{w})$
  $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{\mathbf{s}',a'}$ and $a \leftarrow a'$
until agent done interaction with environment
```

Algorithm 12 IsKnown(s, a)

```
// Uses the minimum count of the features for a state, to decide if $s, a$ is known
// If $a$ not given, sums over all $a$
$m = 5$
if $a$ not given then
  $\mathbf{f} \leftarrow \sum_a \mathbf{c}(\boldsymbol{\phi}_{s,a}) \in \mathbb{R}^d$
else
  $\mathbf{f} \leftarrow \mathbf{c}(\boldsymbol{\phi}_{s,a}) \in \mathbb{R}^d$
if $\min(\mathbf{f}) > m$ then
  return "Known"
else
  return "Not Known"
```

A.3.4 Randomized Least Squares Value Iteration (RLSVI)

RLSVI (Osband et al., 2016b) is an algorithm that maintains a distribution over the possible value functions. The value functions are assumed to be linearly parametrized. While the main algorithm proposed uses a finite-horizon assumption, a modified version proposed in the Appendix of the paper does not, and this is the version used in the experiments here. The pseudocode is provided here for completeness in Algorithm 14.

Algorithm 13 ComputeVF($s_1, a_1, \dots, s_T, a_T, \theta_t, \lambda > 0, \sigma > 0, \gamma$)

```
// Estimate posterior mean $\bar{\theta}_{t+1}$, and posterior variance $\Sigma_{t+1}$ with Bayesian Linear regression
// Sample $\theta_{t+1}$ from the distribution
$\mathbf{A} \leftarrow \begin{bmatrix} \boldsymbol{\phi}_{s_1,a_1}^\top \\ \vdots \\ \boldsymbol{\phi}_{s_T,a_T}^\top \end{bmatrix}$
$\mathbf{b}_i \leftarrow \begin{cases} r_i + \gamma \max_a \boldsymbol{\phi}_{s_{i+1},\cdot}\theta_{t+1}, & \text{if } s_i \text{ is not terminal} \\ r_i, & \text{otherwise} \end{cases}$
$\bar{\theta}_{t+1} \leftarrow (\mathbf{A}^\top\mathbf{A} + \lambda\sigma^2\mathbf{I})^{-1}\mathbf{A}^\top\mathbf{b}$
$\Sigma_{t+1} \leftarrow \sigma^2(\mathbf{A}^\top\mathbf{A} + \lambda\sigma^2\mathbf{I})^{-1}$
Sample $\theta_{t+1} \sim \mathcal{N}(\bar{\theta}_{t+1}, \Sigma_{t+1})$
return $\theta_{t+1}$
```

Algorithm 14 RLSVI

```
$\mathbf{w} \leftarrow \mathbf{0}, \gamma_f \leftarrow \gamma, \lambda > 0, \sigma > 0$
$\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
$a \leftarrow$ greedy action according to value estimates given by $\boldsymbol{\phi}_{s,a}^\top\mathbf{w}$
repeat
    Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
    Store $s, a, s', r$
    if $\gamma == 0$ then
        $\mathbf{w} \leftarrow$ ComputeVF($s_1, a_1, \dots, s_T, a_T, \theta_t, \lambda, \sigma, \gamma_f$)
    $a' \leftarrow$ greedy action according to value estimates given by $\boldsymbol{\phi}_{s',a'}^\top\mathbf{w}$
until agent done interaction with environment
```

A.4 Experimental Details

The parameters for UCLS are fixed. The parameters for other algorithms based on LSTD – UCBootstap, LSPI-Rmax – and the linear complexity algorithms – Optimistic Initialization, ϵ-greedy – are swept in the following range:

$$\begin{aligned}
\eta \in \quad & \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000\}, \\
\alpha \in \quad & \{0.00078125, 0.0015625, 0.003125, 0.00625, 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8\}, \\
r_{\text{max}} \in \quad & \{0, 1, 10, 100, 1000, 10000\}, \\
\epsilon \in \quad & \{0.01, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60\}, \\
\lambda \in \quad & \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.91, \\
& 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0\}.
\end{aligned}$$

The parameters for RLSVI are swept in the range:

$$\begin{aligned}
\lambda &\in \{0.01, 0.1, 1, 10, 100\}, \\
\sigma &\in \{0.01, 0.1, 1, 10, 100\}.
\end{aligned}$$

The parameters for DGPQ are swept in the range:

$$\begin{aligned}
\sigma^2 &\in [0.001, 0.5], \\
\sigma_{tol}^2 &\in [0.01, 0.1], \\
\epsilon &\in [0.01, 0.1], \\
L_Q &\in [1, 20].
\end{aligned}$$

Appendix B

Representations for Online Control Appendix

B.1 More Results

B.1.1 Control Curves

We perform the evaluation of sparsity inducing networks with Sigmoid activation. Figure B.7 shows the performance of Sarsa(0) with representations learned by different networks. k-sparse and WTA performs well in Puddle World, however, none of these representations are effective across all domains.

The learning curves for various k-sparse networks with distributional regularizers are in Figure B.8. It suggests that k-sparse (ReLU+k+SKL) provides no improvement over just using distributional regularizer for ReLU activation (SR-NN).

B.1.2 Activation Heatmaps

The activation heatmaps for randomly selected neurons (excluding dead neurons) in Mountain Car with different regularization stratergies are shown in Figure B.1, and with differnt Distributional Regularization designs are shown in Figure B.2. Heatmaps for sparsity inducing networks with ReLU activations and Sigmoid activation, for Mountain Car and Puddle

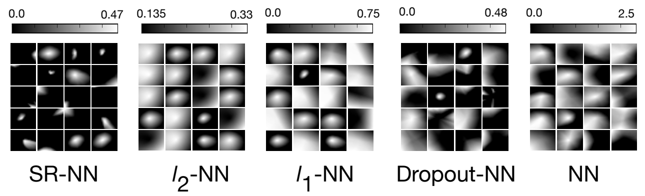

Figure B.1: Heatmaps of activations comparing SR-NN to different regularization strategies and NN in Mountain Car.

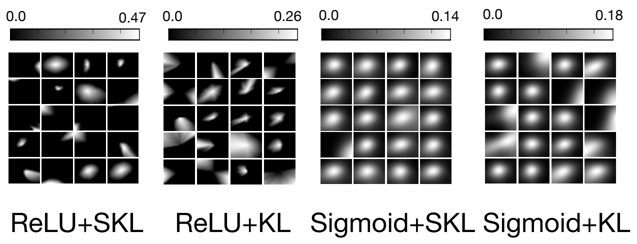

Figure B.2: Heatmaps of activations with different Distributional Regularizers in Mountain Car.

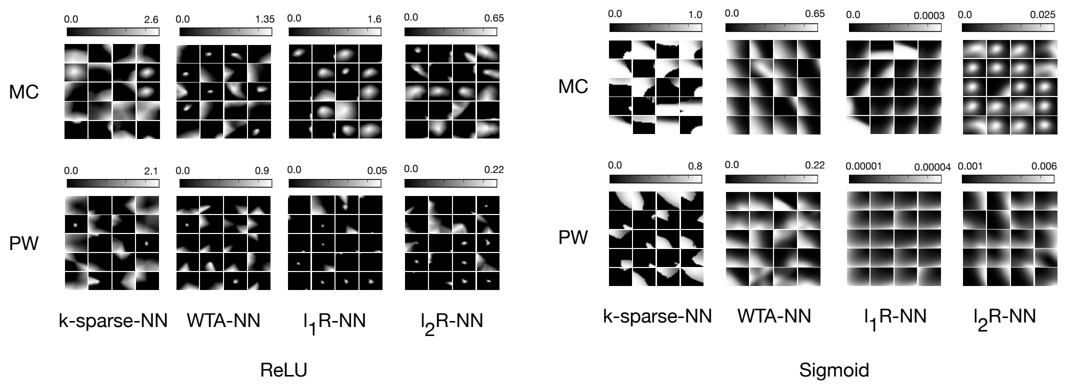

Figure B.3: Heatmaps of activations for nodes from other networks which aim to generate sparse representations (ReLU and Sigmoid activation).

	Mountain Car	Puddle World
SR-NN	16.8	8.8
ℓ_2-NN	112.3	111.5
ℓ_1-NN	109.5	142.5
Dropout-NN	72.5	31.2
NN	106.5	54.0
ReLU+KL	36.8	71.4
SIG+SKL	256.0	256.0
SIG+KL	256.0	256.0
k-sparse-NN	36.6	61.8
WTA-NN	24.8	6.5
ℓ_2R-NN	30.0	3.8
ℓ_1R-NN	10.5	0.4

Table B.1: Activation overlap in Mountain Car and Puddle World. The numbers are the average overlap over all pairs of selected states.

World are shown in Figure B.3.

B.1.3 Activation Overlap

We show the overlap of representations learned by different networks in Table B.1 for Mountain Car and Puddle World. ℓ_2R-NN and ℓ_1R-NN have low overlap values. However, the regularizers tend to push many neurons to be activated for a really small region to reduce penalty as shown in Figure B.3. SR-NN, on the other hand, learns a more distributed representation.

B.2 Experimental Details

In general, we advocate for learning the representation incrementally, for the task faced by the agent. However, for our experiments, we learned the representations first to remove confounding factors. We detail that learning regime here.

The problem of learning a good representation $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s, a)$ in the case of finite actions can be transformed to learning a good representation of the form $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$, and using that to represent the action-value function as:

$$\hat{Q}_{\mathbf{w},\boldsymbol{\theta}}(s, a) \stackrel{\text{def}}{=} \boldsymbol{\phi}_{\boldsymbol{\theta}}(s)^\top \mathbf{w}_a.$$

Here, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$ is the linear representation of the state s, which is used in conjunction with the linear predictor $\mathbf{w}_a$ to estimate action-values for action a across the state space. Under a given policy, like the action-values $Q^\pi(s,a)$, corresponding state-values, $V^\pi(s)$, are:

$$V^\pi(s) \overset{\text{def}}{=} \mathbb{E}[G_t | S_t = s],$$
$$\text{where, } G_t = R_{t+1} + \gamma_{t+1} G_{t+1}.$$

An easy objective to train connectionist networks with simple backpropagation is the Mean Squared Temporal Difference Error (MSTDE) Sutton (1988). For a given policy, the MSTDE is defined as:

$$\sum_{s \in \mathcal{S}} \mathbf{d}(s) \mathbb{E}[\delta_t^2 | S_t = s],$$
$$\text{where, } \delta_t \overset{\text{def}}{=} R_{t+1} + \gamma_{t+1} \boldsymbol{\phi}_{\boldsymbol{\theta}}(S_{t+1})^\top \mathbf{w}_v - \boldsymbol{\phi}_{\boldsymbol{\theta}}(S_t)^\top \mathbf{w}_v.$$

Here, $\mathbf{d}$ denotes the stationary distribution over the states induced by the given policy, and $\boldsymbol{\theta}$ and $\mathbf{w}_v$ are parameters that can be estimated with stochastic gradient descent. Therefore, given experience generated by a policy that explores sufficiently in an environment, a strong function approximator (a dense neural network) can be trained to estimate useful features, $\boldsymbol{\phi}_{\boldsymbol{\theta}}(s)$. These features can then be used for estimating action-values in on-policy control for learning the (close-to) optimal behaviour policy in the environment.

Representation Learning Data

In Mountain Car, we use the standard energy pumping policy with 10% randomness. In Puddle World, by a policy that chooses to go North with 50% probability, and East with 50% probability on each step. The data in Acrobot is generated by a near-optimal policy. In Catcher, the agent chooses to move toward the apple with 50% probability, and selects a random action with 50% probability on each step; and gets only 1 life in the environment.

Tile Coding

We compare to Tile Coding (TC) representation, a well-known sparse representation, as the baseline. TC uses overlapping grids on the observation space, to convert a continuous space to a discrete dimensional space. The representations generated by it are sparse and distributed based on a static hashing technique. We experiment with several configurations for the fixed representation, particularly with grid-sizes(N) in $\{4, 8, 16\}$ and number of tilings (D) in $\{8, 16, 32\}$. We use a hash size of 8192, which is significantly larger than the largest feature size of 256, as used in the other learned representation models we compare to. The results shown in Figure 5.3 are for the best configuration of the static tile-coder after a sweep.

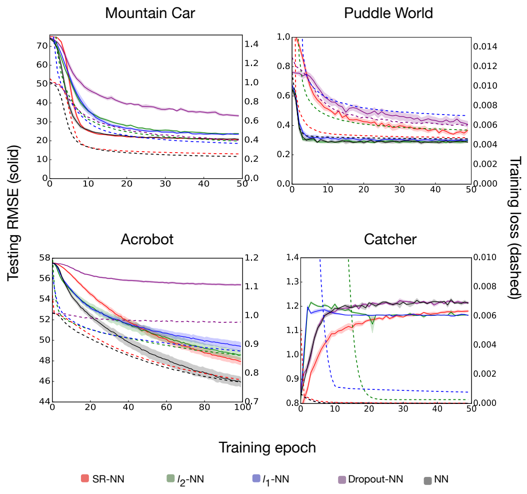

Figure B.4: Learning curves during the training of neural networks with different regularization methods.

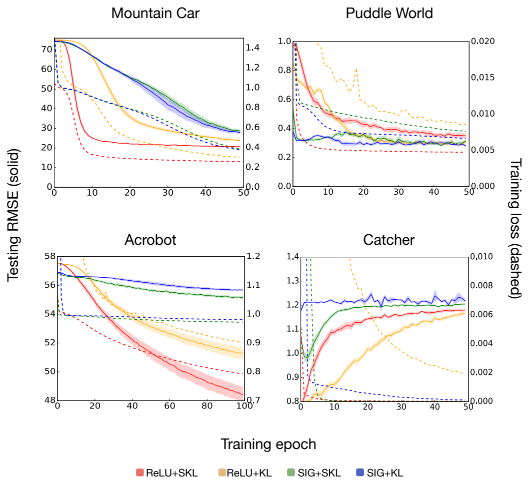

Figure B.5: Learning curves during the training of neural networks with different distributional regularizers.

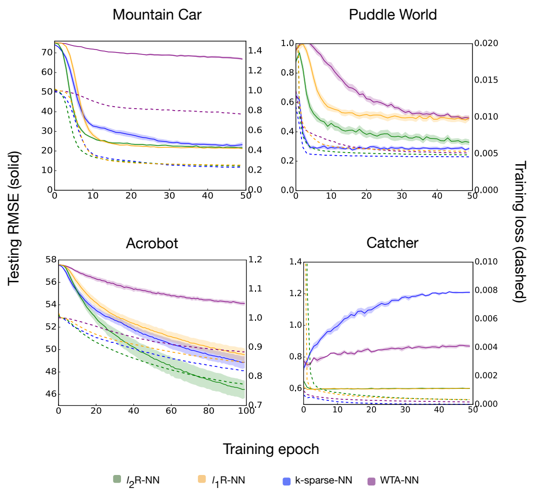

Figure B.6: Learning curves during the training of neural networks with different sparsity inducing approaches.

Training neural networks

Architecture and optimizer: We used neural networks with two hidden layers. The first layer 32 hidden units. The second layer, which is the representation layer used for prediction, has 256 units. We optimized the neural network weights using Adam optimization (Kingma and Ba, 2014) with a batch size of 64. The neural network weights are initialized based on He initialization (He et al., 2015). That is, the neural networks weights are initialized with zero-mean Gaussian distribution with variance equals to $2/n_l$, where n_l is the number of input nodes for layer l.

Representation hyperparameters: The range of grid search for the representation hyperparameters are as follows:

$$
\begin{aligned}
\lambda_{KL} &\in \{0.1, 0.01, 0.001\},\\
\beta &\in \{0.05, 0.1, 0.2\},\\
\lambda_{NN} \text{ for } \ell_1 \text{ and } \ell_2 &\in \{0.1, 0.01, 0.001, 0.0001\},\\
\text{dropout probability } p &\in \{0.1, 0.2, 0.3, 0.4, 0.5\},\\
k \text{ for k-sparse} &\in \{16, 32, 64, 128\},\\
k \text{ for WTA} &\in \{6.25\%, 12.5\%, 25\%, 50\%\}.
\end{aligned}
$$

Algorithmic choices: For k-sparse networks, only the top-k hidden units in the representation layer are activated. We also use scheduling of sparsity level described in the original paper Makhzani and Frey (2013). If used in conjunction with a distributional regularizer, the top-k nodes are chosen before application of the distributional regularizer. For dropout, given the form of the supervision goal (MSTDE), the same dropout mask is chosen to generate the representation for both states S_{t+1} and S_t[1] – this preserves dropouts role as regularizer w.r.t. the target, and promotes diversity in learning.

Grid search evaluation metric: The learned representations are then used for on-policy control in Sarsa(0) with fixed $\epsilon = 0.1$. The value function for Sarsa is initialized with zero-mean Gaussian distribution with small variance. For sparse representations, we use semi-gradient Sarsa with step decay learning rate. For dense representations, we use adaptive learning rate method RMSprop (Hinton et al., 2012). The initial learning rate for Sarsa(0) is swept in the set:

$$\alpha_0 \in \{0.1, 0.04, 0.01, 0.004, 0.001, 0.0004, 0.0001\}.$$

All the sweeps for selecting the representation learning hyper-parameters across domains use 50 epochs and 10 runs.

Grid search exploration algorithms: The learned representations are used with online control algorithms UCLS and RLSVI. The parameters of UCLS and RLSVI are swept in the following range:

$$\begin{aligned} p \text{ Thompson sampling} &\in \{1.0, 0.1, 0.01\}, \\ p \text{ UCB} &\in \{0.1, 0.5, 0.9\}, \\ \text{regularization parameter } \eta &\in \{1e-0, 1e-1, 1e-2\}, \\ \text{step-size } \alpha &\in \{1e-5, 1e-4, 1e-3, 1e-2, 1e-1\}. \end{aligned}$$

Learning curves: The chosen hyper-parameters are used to train a good representation (saturated testing loss – 100 epochs for Acrobot, and 50 epochs for other domains), following which it is used for on-policy control with Sarsa(0). While the control performance is focused on in the main paper, the learning curve during the representation training phase is shown in Figures B.4, B.5 and B.6. The metric on the y-axis is the Root Mean Squared Error (RMSE), which is evaluated as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{\boldsymbol{\phi} \in \mathbf{X}_{test}} (\hat{V}(\boldsymbol{\phi}) - V^*(\boldsymbol{\phi}))^2}{\mathbf{X}_{test}}},$$

[1]We have experimented with different dropout masks for S_{t+1} and S_t, and the result suggests that it is not able to learn good representations even for prediction across all domains.

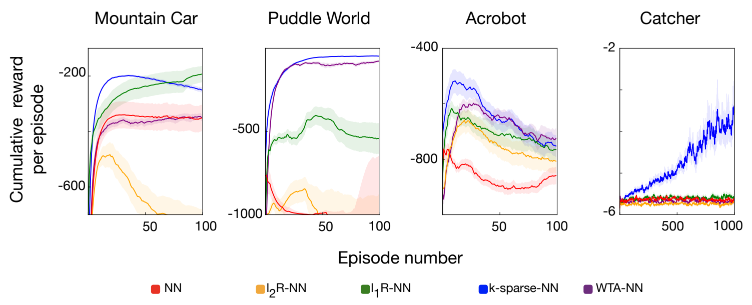

Figure B.7: Learning curves for Sarsa(0) comparing various sparsity inducing networks with Sigmoid activation.

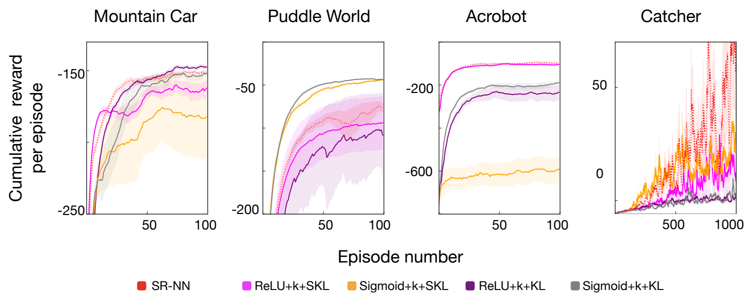

Figure B.8: Learning curves for Sarsa(0) comparing various variants of distributional regularizers with k-sparse.

where, $\mathbf{X}_{test}$ is the set of test states for which the representations have been extracted, $\hat{V}(\phi)$ is the estimated value of state ϕ and $V^*(\phi)$ is the true value of state ϕ computed using Monte Carlo rollouts. The number of test states are 5000 for benchmark domains and 1000 for Catcher. Most algorithms converge to a good solution within 50 epochs in Mountain Car, Puddle World and Catcher, and 100 epochs in Acrobot as shown in the curves. The curves are for representation purposes and only averaged over 5 runs. All learning curves for Sarsa(0) are averaged over 30 runs, and are plotted with exponential moving average ($\beta = 0.1$).

Appendix C

Directed Exploration in Sample-Efficient Online Control Appendix

C.1 Confidence Intervals for Value Estimates

In the problem of online value-based model-free control in reinforcement learning, we are interested in estimating the value function of the optimal policy π^*, as this value function can be used to guide behaviour that is optimal. But, for learning such a value function online we need to ensure effective exploration.

In online tabular model-based RL, effective exploration is achieved by utilizing policies which implement the idea of optimism in the face of uncertainty (OFU). A predominant approach to do so is by using confidence intervals. Algorithms construct confidence intervals around transition/reward dynamics parameters, and find the most optimistic MDP in this confidence set. For such an MDP the optimal policy's value function is obtained. This complete procedure of finding the optimal value function for the most optimistic MDP in the set is called the extended value iteration algorithm. The behaviour is then greedy with respect to the value estimates of this optimistic optimal value function (called so, as it is optimal for an optimistic MDP). Because we are in the tabular setting, there are no policies whose value functions cannot be represented. And as the confidence intervals shrink, the behaviour improves – and the agent gradually transitions from exploring, to balancing exploration-exploitation, to exploiting.

Now, arguably, learning confidence intervals for model-based strategies in the approximation setting is a hard problem. Therefore, from here on, we consider the case where an agent learns confidence intervals directly for the value estimates learned by the agent in the approximation setting. As discussed before for such confidence intervals to be effective they need to include two components: (1) out-of-sample epistemic uncertainty, and (2) in-sample. epistemic uncertainty. To derive such confidence intervals, simplistically, we assume the agent has infinite memory and can store all the transition tuples it has experienced online.

Ignoring how the confidence intervals for value estimates are derived, one can define an optimistic policy using them. A straightforward way to do so is a policy that acts greedily with respect to the optimistic value estimates – the upper-bound value estimates with respect to the estimated values and the confidence intervals. The value function of this policy can be learned by using these optimistic value estimates as the target with the Bellman optimality operator. But in the online approximation setting with limited data, depending on the generalization structure of the function approximator, such a naive strategy can lead to erroneous updates. Further, recovering from such erroneous updates can be hard, given the online nature of the algorithm. Therefore, it is necessary to define assumptions under which such a strategy can be used. The two assumptions relevant here are that of (1) linear MDP, and (2) low inherent Bellman error.

Linear Regression: In regression the errors in learning are due to two sources: (1) the bias of the approximation, and (2) the variance of the targets. The bias of the approximation, also called the *approximation error*, can be reduced by increasing the approximation power of the function class used. The variance of the targets, also called *estimation errors*, can be reduced by increasing the number of samples used to estimate the target. In linear approximation the class of functions is defined by the representation basis.

In the online case both approximation and estimation errors are controlled by the distribution of the data available – more samples can reduce the estimation errors of the targets, whereas the distribution of samples influences the approximation error. From the perspective of online RL, where samples are of the form $< \boldsymbol{\phi}_{s,a}, r, \gamma, \boldsymbol{\phi}_{s',\cdot} >$, the estimation error is controlled by the number of samples with transition tuples conditioned on (s, a), and the approximation error is controlled by the distribution of Π_ϕ, the projection operator defined by the distribution of (s, a).

C.1.1 Optimism with Linear MDP

If we denote the value function estimated at time t as $\mathbf{w}_t$, the value function of an improved policy can be computed as $\mathbf{w}_{t+1} = \Pi_\phi T_\pi \mathbf{q}_{\mathbf{w}_t}$. Here π can be a greedy or soft improvement with respect to values of $\mathbf{w}_t$.

For any state-action pair, the target to be estimated by $\mathbf{w}_{t+1}$ is equal to

$$\mathbb{E}[r|s,a] + \mathbb{E}[\boldsymbol{\phi}'^{\top}\mathbf{w}_t|s,a],$$

where expectations are with respect to the dynamics of the MDP and the greedy policy with respect to $\mathbf{w}_t$.

If the MDP is linear in the approximation used — it is a linear MDP — then

$$\begin{aligned}\mathbb{E}[r|s,a] &= \boldsymbol{\phi}(s,a)^{\top}\theta_r,\\ \mathbb{E}[\boldsymbol{\phi}'^{\top}\mathbf{w}_t|s,a] &= \sum_{(s',a')\in(\mathcal{S}\times\mathcal{A})} P(s,a,s')\pi(s',a')\boldsymbol{\phi}(s',a')^{\top}\mathbf{w}_t\\ &= \sum_{\substack{(s',a')\in\\(\mathcal{S}\times\mathcal{A})}} \boldsymbol{\phi}(s',a')^{\top}\boldsymbol{\mu}(s')\pi(s',a')\boldsymbol{\phi}(s',a')^{\top}\mathbf{w}_t.\end{aligned}$$

Therefore, the best $\mathbf{w} \in \mathcal{W}$ with respect to (s,a) corresponds to

$$\mathbf{w} = \theta_r + \sum_{(s',a')\in(\mathcal{S}\times\mathcal{A})} \boldsymbol{\mu}(s')\pi(s',a')\boldsymbol{\phi}(s',a')^{\top}\mathbf{w}_t.$$

As a result, for any (s,a) there exists a $\mathbf{w} \in \mathcal{W}$ that can estimate its value accurately. Further, this remains true even if there are any estimation errors due to a lack of transition samples conditioned on (s,a).

Now, if we include non-linear components to provide optimistic targets for evaluating an optimistic policy – that is, $\mathbf{w}_{t+1} = \Pi_{\phi}T_{\pi}\mathbf{q}_{\mathbf{w}_t,\mathbf{U}_t}$, where $(\mathbf{q}_{\mathbf{w}_t,\mathbf{U}_t})(s',a') = \boldsymbol{\phi}(s',a')^{\top}\mathbf{w}_t + \sqrt{\boldsymbol{\phi}(s',a')^{\top}\mathbf{U}_t^{-1}\boldsymbol{\phi}(s',a')}$. With a similar procedure, the best $\mathbf{w} \in \mathcal{W}$ with respect to (s,a) corresponds to

$$\mathbf{w} = \theta_r + \sum_{(s',a')\in(\mathcal{S}\times\mathcal{A})} \boldsymbol{\mu}(s')\pi(s',a')\left(\boldsymbol{\phi}(s',a')^{\top}\mathbf{w}_t + \sqrt{\boldsymbol{\phi}(s',a')^{\top}\mathbf{U}_t^{-1}\boldsymbol{\phi}(s',a')}\right).$$

Again, this remains true even if there are any estimation errors.

Therefore, for every (s,a) there is a value function in $\mathcal{W}$, even under estimation errors, that can accurately represent any evaluated target. Further, the optimistic bonuses do not have to linear in the features. With the approximation weighting defined in Π_{ϕ}, $\mathbf{w}_{t+1} \in \mathcal{W}$ can be estimated effectively. This is the assumption used for designing the LSVI-UCB algorithm (Jin et al., 2019), and the reason for its efficient online guarantees when the assumption is satisfied.

C.1.2 Optimism with Low Inherent Bellman Error

The target with non-linear bonuses for a given (s, a) pair, without any assumptions would equal

$$\mathbb{E}[r|s,a] + \sum_{(s',a')\in(\mathcal{S}\times\mathcal{A})} P(s,a,s')\pi(s',a')\left(\boldsymbol{\phi}(s',a')^\top \mathbf{w}_t + \sqrt{\boldsymbol{\phi}(s',a')^\top \mathbf{U}_t^{-1}\boldsymbol{\phi}(s',a')}\right).$$

Let $\mathbf{o}_t(s',a') = \boldsymbol{\phi}(s',a')^\top \mathbf{w}_t + \sqrt{\boldsymbol{\phi}(s',a')^\top \mathbf{U}_t^{-1}\boldsymbol{\phi}(s',a')}$ denote the optimistic targets, and let $\mathbf{o} \in \mathbb{R}^n$ be the optimistic target vector with components corresponding to (s, a) pairs. If $\mathbf{o} = \boldsymbol{\Phi}\mathbf{w}$, for some $\mathbf{w} \in \mathcal{W}$, then the optimistic targets are representable with $\mathbf{w} \in \mathcal{W}$. Therefore, under the low inherent Bellman error assumption, the expected error is bounded. If there are any estimation errors, as the expected error is bounded, they will be small and well-behaved as more of the space is explored. If there is no such $\mathbf{w} \in \mathcal{W}$, and $\mathbf{o}$ is used as the optimistic target, then any error due to estimation in $\mathbf{o}$ can amplify depending on the approximation weighting. This process can lead to uncontrollable errors, resulting in poor behaviour, and/or even divergence.

Therefore, it is necessary to use optimistic targets that can be represented with some $\mathbf{w} \in \mathcal{W}$ with the low inherent Bellman Error. In order to do so, the algorithm Eleanor (Zanette et al., 2020) proposes an optimization objective that jointly optimizes over the value function and confidence interval parameters. The result is an optimistic value function that guides online behaviour.

C.1.3 Optimism in OOVI

The objective proposed by Eleanor provides an optimistic value function, $\mathbf{w}_{t+1}$ along with confidence interval radius $\mathbf{u}_t$ that was utilized to estimate $\mathbf{w}_{t+1}$. The uncertainty estimates are: (1) policy-independent and based on the dynamics across all the horizons, and (2) linear in the representations used.

The epistemic uncertainty estimates derived in Chapter 4 take into account the dynamics of the discounted MDP, but are with respect to the policy at time-step t, π_t. They are therefore limited as: (1) they do not include the anticipatory component, and (2) they are non-linear in the representations used.

Comparing to optimism in UCLS: As discussed in Chapter 6, UCLS' exploration strategy is not compatible with planning updates. This is because UCLS' exploration strategy is designed to maintain optimism through confidence interval radius $\hat{u}_t$, which with planning updates can result in fast decay of anticipatory optimism, and therefore poor exploration. OOVI rectifies this by (1) splitting the two optimism components into two independent learners, (2) introducing updates that retain estimated in-sample epistemic uncertainty under planning before they are replaced by an improved policy's uncertainty, and (3) decaying out-of-sample epistemic uncertainty proportional to policies estimated. This strategy for learning, along with behaviour that is greedy in the optimistic value estimates made by $\mathbf{w}$, combined with the out-of-sample value estimates of $\mathbf{w}^{\text{e-out}}$ — where $\mathbf{w}$ retains the in-sample epistemic uncertainties of the previously evaluated policies — results in directed exploration that is compatible with sample-efficient planning strategies. A drawback of OOVI is that it cannot easily be cast under the Optimistic Values Theorem framework.

C.2 Confidence Intervals for Value Functions

Confidence intervals can also be over value functions, instead of over value estimates, as utilized by RLSVI (Osband et al., 2016b), and BSP (Osband et al., 2018). While RLSVI is a finite-horizon algorithm with theoretical guarantees for tabular MDPs, it differs from the OFU strategies discussed in the previous section based on the following: it utilizes OFU in the value function space .

RLSVI maintains confidence intervals over value function parameters, and utilizes Thompson sampling to sample value functions from the space. As the space of value functions is expected to contain $\mathbf{q}^*$, this strategy explores utilizing optimism directly in the value function space. This is different from approaches that utilize OFU by learning optimistic value estimates via bonuses.

A key advantage may be that strategies that rely on optimism in value function space do not have to be limited by the generalization structure of the approximation space being used as they do not utilize possibly non-representable bonuses to implement the principle of OFU. On the other hand, because the value functions are sampled, it may be necessary to utilize the sampled value function for a significant number of steps to have effective directed exploration. In finite-horizon problems where the length of an interaction cycle is well-defined, a sampled policy can be used for this length. But it is unclear how one would faithfully and effectively implement such a strategy in discounted MDPs.

C.3 Details about Other Algorithms

C.3.1 Randomized Prior for Bootstrap DQN (BSP)

An approach to exploration that is designed to extend Bayesian Linear Regression from the context of RLSVI with linear value functions, to the deep reinforcement learning setting is the algorithm called BSP (Osband et al., 2018). The pseudocode for BSP is in Algorithm 15.

Algorithm 15 Randomized Prior for Bootstrap DQN (BSP)

```
1:  $K$ = number of ensembles, $\mu$ = mean, $\lambda$ = scale
2:  $\mathbf{w}_i \leftarrow \mathbf{0}$, $\mathbf{p}_i \leftarrow \mathbf{0}$, and $\mathcal{B}_i \leftarrow [\emptyset]$, $\forall i \in \{1, \ldots, N\}$
3:  $n \leftarrow \mathcal{U}[0, K]$, chosen ensemble
4:  $\triangleright \pi(\boldsymbol{\phi}_{s,.}) \leftarrow \max_{a \in \mathcal{A}} \boldsymbol{\phi}_{s,a}^\top \mathbf{w} + \boldsymbol{\phi}_{s,a}^\top \mathbf{p}$
5:  $\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
6:  $a \leftarrow \pi(\boldsymbol{\phi}_{s,\cdot})$
7:  **repeat**
8:      Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
9:      **for all** $i = 1$ to $K$ **do**
10:         **if** $\mathcal{U}(0, 1) < 0.5$ **then**
11:             Store $[\boldsymbol{\phi}_{s,a}, r, \gamma, \boldsymbol{\phi}_{s',\cdot}]$ in $\mathcal{B}_i$
12:     **for all** $i = 1$ to $K$ **do**
13:         $\mathcal{D} \leftarrow$ sampled minibatch from $\mathcal{B}_i$
14:         Optimize $\mathbf{w}_i$ with semi-gradients to minimize
                $\sum_{\mathcal{D}}(r + \gamma \max_{a' \in \mathcal{A}}(\mathbf{w}_i, \mathbf{p}_i)(\boldsymbol{\phi}_{s',.}) - (\mathbf{w}_i, \mathbf{p}_i)(\boldsymbol{\phi}_{s,a}))^2$
15:     **if** $\gamma == 0$ **then**
16:         $n \leftarrow \mathcal{U}[0, K]$, chosen ensemble
17:     $a \leftarrow \pi(\boldsymbol{\phi}_{s',.})$, and $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{s',a}$
18: **until** agent finishes interaction with environment
```

C.3.2 Uncertainty Bellman Equation (UBE)

An approach to exploration that is designed to propagate uncertainty via a Bellman Equation style update is called UBE (ODonoghue et al., 2017). The pseudocode for UBE is in Algorithm 16.

Algorithm 16 Uncertainty Bellman Equation (UBE)

```
1:  $\lambda =$ scale, $\beta =$ Thompson sampling hyper-parameter
2:  $\mathbf{w} \leftarrow \mathbf{0}$, $\mathbf{u} \leftarrow \mathbf{0}$, $\Sigma_a \leftarrow \lambda\mathbf{I}$, $\mathcal{B} \leftarrow [\emptyset]$
3:  $\triangleright \pi(\boldsymbol{\phi}_{s,.}) \leftarrow \max_{a \in \mathcal{A}} \mathcal{N}(\boldsymbol{\phi}_{s,a}^\top \mathbf{w}, \beta * \mathcal{N}(0, \sqrt{\boldsymbol{\phi}_{s,a}^\top \mathbf{u}}))$
4:  $\boldsymbol{\phi}_{s,\cdot} \leftarrow$ initial state-action features, for any action
5:  $\boldsymbol{\phi}_s \leftarrow$ initial state features
6:  $a \leftarrow \pi(\boldsymbol{\phi}_{s,\cdot})$
7:  repeat
8:      Take action $a$ and observe $\boldsymbol{\phi}_{s',\cdot}$ and $r$, and $\gamma$
9:      Store $[\boldsymbol{\phi}_{s,a}, \boldsymbol{\phi}_s, r, \gamma, \boldsymbol{\phi}_{s',\cdot}]$ in $\mathcal{B}_i$
10:     $a' \leftarrow \pi(\boldsymbol{\phi}_{s',\cdot})$
11:     $y \leftarrow \boldsymbol{\phi}_s^\top \Sigma_a \boldsymbol{\phi}_s + \gamma^2 \boldsymbol{\phi}_{s',a'}{}^\top \mathbf{u}$
12:     Update $\mathbf{u}$ to minimize $(y - \boldsymbol{\phi}_{s,a}{}^\top \mathbf{u})^2$ with semi-gradients
13:     $\mathcal{D} \leftarrow$ sampled minibatch from $\mathcal{B}$
14:     Optimize $\mathbf{w}$ with semi-gradients to minimize
            $\sum_{\mathcal{D}} (r + \gamma \max_{a' \in \mathcal{A}} \mathbf{w}^\top \boldsymbol{\phi}_{s',\cdot} - \mathbf{w}^\top \boldsymbol{\phi}_{s,a})^2$
15:     $\Sigma_a \leftarrow \Sigma_a - (\Sigma_a \boldsymbol{\phi}_{s,a} \boldsymbol{\phi}_{s,a}^\top \Sigma_a)/(1 + \boldsymbol{\phi}_{s,a}^\top \Sigma_a \boldsymbol{\phi}_{s,a})$
16:     $a \leftarrow a'$, and $\boldsymbol{\phi}_{s,a} \leftarrow \boldsymbol{\phi}_{s',a'}$
17: until agent finishes interaction with environment
```

C.4 Experimental Details

Representation learning parameters: For learning the representation with SR-NN we use mini-batch training with a minibatch size of 64. We use an online buffer of size 1000. The neural network is a two-layer network of size [32,256] resulting in a 256 dimensional representation. We sweep over using and not using a bias in the network. The network is trained to predict next state and reward. The weight of SR-NN loss, its sparsity coefficient, and the weights of the state and reward prediction losses are swept in the following set:

$$\begin{aligned}
\lambda_{SKL} &\in \{0.001, 0.01, 0.1\},\\
\beta &\in \{0.05, 0.1, 0.2\},\\
\lambda_S &\in \{1.0, 1.5, 2.0, 2.5, 3.0\},\\
\lambda_R &\in \{1.0, 0.1, 0.01\}.
\end{aligned}$$

Parameter sweep: The relevant parameters for all algorithms are swept in the following set for 3 runs:

$$\begin{aligned}
p, \text{Thompson sampling} &\in \{1.0, 0.1, 0.01\},\\
p, \text{UCB} &\in \{0.1, 0.5, 0.9\},\\
\text{regularization parameter } \lambda &\in \{1e-0, 1e-1, 1e-2\},\\
\text{step-size } \alpha &\in \{1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 2e-1\},\\
\text{variance scaling parameter } \beta &\in \{1e-0, 1e-1, 1e-2\}.
\end{aligned}$$

$\mathbf{w}^{\text{e-out}}$ of OOVI is initialized to estimate 100 as the out-of-sample epistemic uncertainty with respect to the underlying state-space.