Ensuring the Safety of Reinforcement Learning Algorithms at Training and Deployment

Melrose Roderick

CMU-S3D-23-108

October 2023

Software and Societal Systems Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Zico Kolter, Chair
Jeff Schneider
Ruslan Salakhutdinov
Felix Berkenkamp (Bosch Center for AI)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Societal Computing.

This research was sponsored by Robert Bosch GMBH under award number 0087016732PCRPO0087023984 and the National Science Foundation Fellowship DGE1745016. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Reinforcement Learning, Environmental Sustainability, Robustness, Safe Exploration, Offline Reinforcement Learning

Abstract

Reinforcement learning (RL) has the potential to significantly improve the efficiency of many real-world control problems, including Tokomak control for nuclear fusion and power grid optimization. However, practitioners in these problem areas remain weary of applying RL to these problems. From my experience working with practitioners in Tokomak control, power grid optimization, and autonomous manufacturing, the primary concerns for applying RL to these domains are safety concerns: How do we maintain safety throughout training and during deployment of RL algorithms? In this thesis, we will discuss work we have done to address the challenges of ensuring safety of RL algorithms at training and deployment.

We start by with the problem of ensuring safety and robustness during deployment. In real world systems, external disturbances or other small changes to the system dynamics are inevitable and, thus, there is a need for controllers that are robust to these disturbances. When designing controllers for safety-critical systems, practitioners often face a challenging tradeoff between robustness and performance. While robust control methods provide rigorous guarantees on system stability under certain worst-case disturbances, they often yield simple controllers that perform poorly in the average (non-worst) case. In contrast, nonlinear control methods trained using deep learning have achieved state-of-the-art performance on many control tasks, but often lack robustness guarantees. We introduce a novel method to provide robustness guarantees to any deep neural-network-based controller trained using RL. We demonstrate our technique empirically improves average-case performance over other robust controllers while maintaining robustness to even worst-case disturbances.

Next, we discuss ensuring safety at training time. Traditionally, online RL algorithms require significant exploration to construct high-performing policies. These exploration strategies often do not take safety into account. Although a growing line of work in reinforcement learning has investigated this area of "safe exploration," most existing techniques either 1) do not guarantee safety during the actual exploration process; and/or 2) limit the problem to a priori known and/or deterministic transition dynamics with strong smoothness assumptions. Addressing this gap, we propose Analogous Safe-state Exploration (ASE), an algorithm for provably safe exploration in Markov Decision Processes (MDPs) with unknown, stochastic dynamics. Our method exploits analogies between state-action pairs to safely learn a near-optimal policy in a PAC-MDP (Probably Approximately Correct-MDP) sense. Additionally, ASE also guides exploration towards the most task-relevant states, which empirically results in significant improvements in terms of sample efficiency, when compared to existing methods.

Alternatively, RL can applied to offline datasets to safely learn control policies without ever taking a potentially dangerous action on the real system. A key problem in offline RL is the mismatch, or *distribution shift*, between the dataset and the distribution over states and actions visited by the learned policy. The main approach to correct this shift has been through importance sampling, which leads to high-variance gradients. Other approaches, such as conservatism or behavior-regularization, regularize the policy at the cost of performance. We propose a new approach for stable off-policy Q-Learning that builds on a theoretical result by Kolter [64]. Our method, Projected Off-Policy Q-Learning (POP-QL), is a novel actor-critic algorithm that simultaneously reweights off-policy samples and constrains the policy to prevent divergence and reduce value-

approximation error. In our experiments, POP-QL not only shows competitive performance on standard benchmarks, but also out-performs competing methods in tasks where the data-collection policy is significantly sub-optimal.

Another approach to offline RL is to used Model-Based methods. A unique challenge to Model-Based methods in the offline setting is the need to predict epistemic uncertainty – uncertainty deriving from lack of data samples. However, standard deep learning methods are unable to capture this type of uncertainty. We propose a new method, Generative Posterior Networks (GPNs), that uses unlabeled data to estimate epistemic uncertainty in high-dimensional problems. A GPN is a generative model that, given a prior distribution over functions, approximates the posterior distribution directly by regularizing the network towards samples from the prior. We prove theoretically that our method indeed approximates the Bayesian posterior and show empirically that it improves epistemic uncertainty estimation and scalability over competing methods.

Acknowledgments

During my PhD, I have had the immense pleasure to work with some amazing people that have really helped me throughout the process. I thank Vaishnavh Nagarrajan for providing me guidance early in my PhD process. It was incredibly helpful to have his input, both on research and navigating the PhD. I thank Priya Donti for not only being mentor, but also someone who shared and encouraged my passion for environmental sustainability. Her constant striving to have a positive social impact with her work is incredibly inspiring. I would also like to thank Felix Berkenkamp, whom I have had the pleasure to work with over the past few years. His insights and guidance have been incredibly helpful throughout that time. And finally, I would like to thank my advisor, Zico Kolter. On top of research insights and career advice, I am incredibly thankful for the freedom and encouragement to explore areas of research and applications I was most passionate about.

I would also like to thank my friends and family who have helped and encouraged me throughout the process. I would like thank my brother, William Roderick, without whom I would not have gotten to where I am. From career and academic advice, to simply always being there when I need to talk, I am incredible thankful to him. I thank both my parents, George Roderick and Rosemary Gillespie, for their constant encouragement and support throughout the process. It was also nice to always have an extra set of eyes on a paper! I would also like to thank Melanie Dieterlen. I was lucky to have great rock climbing buddy whenever I needed a break from PhD work. Finally, my pandemic pod, Tom Magelinski, Jessica Howard, Matthew Ho, Andrew Kuznetsov, and Morgan Evans. The Pandemic was not an easy time to be in a PhD program, but having you all around made it much easier.

There are many missing people from this list who have helped me along the way, in ways both big and small. To all my friends and family, thank you so much!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement Learning (RL) is a sub-field of Machine Learning (ML) that has the potential to massively improve efficiency of autonomous systems, including autonomous manufacturing and power grid optimization. Throughout my Ph.D. work, I have been particularly motivated by RL applications in environmental sustainability, specifically the problem of Tokomak control for nuclear fusion. A Tokomak reactor is a specific type of nuclear fusion reactor that magnetically suspends hydrogen atoms in a toroidal chamber. When these hydrogen atoms are accelerated around the torus at high enough speeds, they can fuse, forming helium, along with an large amount of energy [33]. While enormous progress has been made in Tokomak design, there is still a fundamental unsolved problem that stands in the way of these systems being viable power generators: How do we control the hydrogen atoms inside the chamber? The collection of hydrogen atoms, or "plasma," must maintain a specific helical structure in order for them to be accelerated to high enough speeds. Unfortunately, as the plasma heats up, it becomes more challenging to keep the plasma in the specific structure; in other words, the system becomes more unstable. When this structure of the plasma is broken, the resulting event is called a "disruption", where suddenly all the energy of the high-temperature plasma is very suddenly released into the walls of the Tokomak enclosure, potentially causing significant damage. On top of that, at higher temperatures, the physical properties of plasma are not well understood and, thus, we cannot simulate the dynamics of the system. This is a real problem for Tokomak reactors. For example, ITER, a Tokomak reactor under construction in France, would only be able to withstand a small number of unmitigated full-power disruptions before it would need significant repairs.

ML has already been applied to this problem with remarkable success. Kates-Harbeck et al. [60], for example, used a deep learning system to predict when a disruption event was going to occur with a couple hundred miliseconds warning. This tool allows the system to significantly mitigate the costs of a disruption event by quickly injecting particles into the chamber and, thus, cooling the plasma down before the disruption occurs. This tool, however, still does not have 100% success rate, meaning inevitably there will be unmitigated disruptions. Moreover, even if the disruptions are mitigated, they still cause the power-generating reaction to stop and Tokomak would need to restart the power-intensive process of heating up the plasma. These disruptions would significantly reduce the efficiency of these reactors. Ideally, we could use ML to control

the plasma in real-time and avoid these disruption events.

This Tokomak control problem seems to be a perfect application for RL. RL is a subfield of ML focused on learning an optimal control policy for a control system. We can use RL to learn to actuate the magnets in a Tokomak in order to control the plasma. In fact, Degrave et al. [28] illustrated that a controller for a Tokomak reactor trained using RL can out-performed controllers made by humans. Like most RL algorithms, this controller was trained using a simulator. However, as we discussed earlier, the physical dynamics of high-temperature plasmas are not well understood and, thus, these methods only work for relatively low-temperature plasmas.[1] Moreover, the physics behind what causes most types of disruption events is also not well understood, so these controllers are just as vulnerable to these types of disruptions.

An alternative approach to using a simulator would be to run an RL algorithm on the Tokomak reactor itself. This would allow the algorithm to learn from its mistakes and, potentially, perform well even when the physics is not perfectly understood. However, allowing an RL algorithm to explore on the Tokomak reactor introduces serious safety concerns: a high-temperature disruption could significantly damage the Tokomak enclosure. So how can we safely learn a controller to operate a Tokomak?

This challenge of safety when applying RL to real-world problems is not unique to Tokomak control. Throughout my Ph.D., I have collaborated with people working in Tokomak control, electrical power grid optimization, autonomous manufacturing, and autonomous driving. While more optimized controllers could vastly improve the efficiency of each of these areas, practitioners are very weary to apply RL to these safety-critical domains due to safety concerns. And for good reason. RL is notoriously brittle to small changes in the environment during deployment and most often randomly explores the environment at training time. In order for RL to be successfully applied on these real-world challenges, these safety concerns need to be addressed.

In this thesis we introduce various methods to improve the safety of RL algorithms both at training time and during deployment. We start with the deployment stage. At mentioned before, policies learned using RL are notoriously brittle to small changes in the environment. However, in real-world settings, small disturbances are inevitable and, thus, practitioners are reluctant to employ RL. To address this concern, we introduce a new method to provably guarantee robustness for deep neural network policies trained using RL. To the best of our knowledge, this is the first technique to make robust control guanantees on non-linear, neural-network policies.

Next, we discuss various techniques for ensuring safety at training time, both online and offline. In the online setting, we need to ensure the agent never takes an unsafe action during exploration. To achieve this, we introduce a new safe exploration method called Analogous Safe Exploration (ASE), which is uniquely able to handle the more realistic setting of stochastic dynamics while maintaining safety and optimality. In the offline setting, we can ensure safety by never taking potentially dangerous actions on the real system, but instead using a fixed dataset collected a priori. We propose a new method for improving offline RL performance for TD methods and an

[1]These RL controllers have not been tested on high-temperature plasmas, so we do not know for sure if they would work or not at higher temperatures. But, since the dynamics change at those temperatures, we would not expect these controllers to perform well at higher temperatures.

epistemic uncertainty approximation method for improving offline RL performance for offline Model-Based methods. Finally, we discuss the benefits and drawbacks of each of these approaches and for which real-world environments they are best suited.

1.1 Contributions

1. Chapter 3 introduces a method for providing robust control guarantees to any RL algorithm. Using Lyaponov analysis, we construct the set of all possible robustly stabalizable linear controllers. This provides us with the set of all possible actions at every step that are guaranteed to result in a robustly stable policy. By projecting a non-linear control policy onto this set, we can guarantee the robustness of the resulting policy. We prove that this process does indeed result in a robustly stabalizable control policy. We evaluate this method against standard robust control techniques and non-robust RL algorithms on a simulated quadrotor and microgrid domain. Through these experiments, we show that this method improves the average-case performance over the other robust control techniques and remains stable even under worst-case disturbances. This technique opens the door for RL techniques to be applied in real-world, safety-critical control systems.

2. Chapter 4 introduces Analogous Safe-state Exploration (ASE), a novel safe exploration algorithm that is uniquely able to capture three properties simultaneously: (1) ASE can be applied on domains with stochastic dynamics, (2) ASE is provably safe and optimal, and (3) ASE is a guided exploration algorithm, improving sample-efficiency over more naive exploration algorithms. We prove that ASE never takes an unsafe action throughout the training process with high probablity and is optimal in the PAC-MDP sense. We compare our method against both unsafe and exploration algorithms on a safety-critical grid-world domain and platformer domain. We shows ASE improves sample efficiency over more naive safe exploration algorithms while maintaining safety throughout training.

3. Chapter 5 presents a new method for performing off-policy TD learning. We consider the contraction mapping condition, first proposed in Kolter [64], that guarantees convergence and bounded error of TD learning. We propose a new sampling projection that allows for a much more computationally efficient algorithm, allowing us to apply this algorithm on high-dimensional, deep RL domains. Secondly, we extend this projection to additionally consider policy optimization. We propose a new offline Q-Learning algorithm that jointly projects the policy and sampling distribution onto the contraction mapping set. This new algorithm significantly reduces Q-approximation error and improves policy performance over standard offline Q-learning. We evaluate this method on the a standard offline RL benchmarks, the D4RL tasks [35]. While our approach does not out-perform the state of the art methods in all tasks, we illustrate our method outperforms all other methods when the dataset is severely sub-optimal. Our results illustrates the power of the contraction mapping condition and the potential of this new class of RL techniques.

4. Chapter 6 introduces Gaussian Posterior Networks (GPNs) that uses unlabeled data to estimate epistemic uncertainty in high-dimensional problems. A GPN is a generative

model that, given a prior distribution over functions, approximates the posterior distribution directly by regularizing the network towards samples from the prior. We prove theoretically that our method indeed approximates the Bayesian posterior under mild assumptions. We evaluate out-of-distribution detection performance of our method and baseline methods on classification and regression tasks. Our experiments show that GPNs improves epistemic uncertainty estimation and scalability over competing methods.

1.2 Summary of Publications

Priya L. Donti, Melrose Roderick, Mahyar Fazlyab, and J. Zico Kolter. "Enforcing Robust Control Guarantees within Neural Network Policies." *International Conference on Learning Representations*. 2021.

Melrose Roderick, Vaishnavh Nagarajan, and Zico Kolter. "Provably safe PAC-MDP exploration using analogies." *International Conference on Artificial Intelligence and Statistics*. 2021.

Melrose Roderick, Gaurav Manek, Felix Berkenkamp, and J. Zico Kolter. "Projected Off-Policy Q-Learning (POP-QL) For Stabilizing Offline Reinforcement Learning." Under Review.

Chapter 2

Background and Preliminaries

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a sub-field of machine learning concerned with learning an optimal (or approximately optimal) behavior, or "policy", for a specific control problem. Concretely, a policy is a function that maps a "state" of the world to an action to take in that state. A control problem is any type of problem where an "agent" can take actions that change the dynamics of the environment. For example, in an Atari video, the agent is the player, the states are the images on the screen, and the actions are the inputs to the gaming controller. There are many different examples of control problems in the real world, from autonomously stabalizing a quadrotor drone to controlling a Tokomak fusion reactor.

A key aspect of RL that sets it apart from traditional "planning" is the assumption that the dynamics of the system are unknown. If the dynamics of the system are perfectly known a-priori, it is much easier to construct an optimal policy using a control planning algorithm[1] Instead, in RL, we usually assume the dynamics of the environment are unknown.

2.1.1 The Markov Decision Process (MDP)

RL problems are most commonly formalized using a Markov Decision Process (MDP). An MDP is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition dynamics, signifying the probability of taking an action and transitioning between two states, and $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, how good is a specific state-action pair, and $\gamma \in [0, 1)$ is the discount factor. The goal in this formulation is to find a probabilistic policy, $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, that maximizes the expected discounted return

$$\mathbb{E}_{p,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0\right] \tag{2.1}$$

[1]There are cases where RL is still useful in settings where the dynamics are known a-priori, such as when state-action space is very large, so exhaustive search is intractable.

This formulation takes use of what is often referred to as the "Markov Assumption", which is the assumption that the dynamics are conditionally independent of the history given the current state and action,

$$p(s_{t+1} \mid (s_t, a_t), (s_{t-1}, a_{t-1}), \ldots, (s_0, a_0)) = p(s_{t+1} \mid s_t, a_t) \tag{2.2}$$

While this assumption is often violated in practice, for example when the state is not fully observable, this assumption is very useful for finding tractable RL algorithms. Additionally, the algorithms that take advantage of this assumption often find useful policies even when the assumption is violated.

Markov Reward Process For some of the theory discussed in this thesis, we assume a fixed policy. Under a fixed policy, an MDP reduces to a much simpler Markov Reward Process (MRP). An MRP is defined as the 4-tuple $(\mathcal{S}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $p : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_+$ and $r : \mathcal{S} \to \mathbb{R}$ are the transition and reward functions, and $\gamma \in (0, 1)$ is the discount factor.

In cases where the state space is discrete and finite, we can use the MRP matrix notation. In this setting, the state-space can be indexed as $\mathcal{S} = \{1, \ldots, n\}$. In matrix notation, we use matrices P and R, to represent the functions p and r, where each row corresponds to a state.

2.1.2 Categories of RL Algorithms

RL algorithms fall into three main categories (or a mixture these categories): Policy Gradient (PG), Temporal Difference (TD), and Model-Based (MB) methods. Each of these categories of algorithms have their benefits and drawbacks.

Policy Gradient PG methods involve performing Markov "rollouts" of a parameterized policy to estimate the expected policy return, then performing gradient updates to maximize the return. Concretely, let π_θ be a parameterized policy. The policy gradient can be written as:

$$\nabla_\theta \mathbb{E}_{p,\pi} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] = \mathbb{E}_{p,\pi} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t) \right] \tag{2.3}$$

where T is the trajectory length.

Given enough samples, policy gradient methods tend to have the highest asymptotic performance compared to TD and Model-Based methods. However, there are two key downsides to PG approaches. Firstly, in order to compute the policy gradient new trajectories need to be sampled every gradient step. For these reason PG methods are significantly less sample efficient compared to TD and Model-Based methods. Secondly, the variance of the gradient updates grows with the trajectory length T, which can be a problem for longer-horizon problems.

The variance of the PG updates can be reduced by introducing an advantage function [103]. This advantage function, often trained using a TD approach, estimates how much higher the return is over taking some action a, over following the current policy. This advantage function is then added to the reward. Most modern PG gradient make use an advantage function.

To address the sample challenge, some approaches, such as Trust Region Policy Optimization (TRPO) [96] and Proximal Policy Optimization (PPO) [97], make conservative estimates of the PG in order to make multiple gradient steps with a single set of rollouts. This can significantly reduce sample efficiency.

Temporal Difference TD methods make use of Markov assumption to estimate "cost-to-go" of a given policy, the expected discounted return starting a specified state (or state-action pair). Recall that, under the Markov assumption, the expected discounted of a given state (or state-action pair) is conditionally independent of the state history given the current state (or state-action pair). Thus, the action-value function, or Q-function, given by:

$$Q^{\pi}(s,a) = \mathbb{E}_{p,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) \,|\, s_0 = s, a_0 = a\right] \tag{2.4}$$

is the fixed point of the following equation,

$$Q^{\pi}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot \,|\, s,a), a' \sim \pi(\cdot \,|\, s')}\left[Q^{\pi}(s',a')\right] \tag{2.5}$$

This equation is known as the Bellman equation.

Now, if we can compute the Q-function, the optimal policy can be computed by finding the action that maximizes the Q-function at every step:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{a \sim \pi(\cdot \,|\, s)} Q^{\pi}(s,a) \tag{2.6}$$

This algorithm is known as Q-learning.

In simple MDPs with small, discrete state-action spaces, we can simply solve the Q-function using dynamic programming. However, when the state-action space is large and continuous, we need to approximate the Q-function. This can be done with a variety of methods [46, 81, 114].

While TD methods significantly improve sample efficiency over PG methods primarily due off-policy learning. Off-policy involves using samples from a different policy to update the Q-function of the current policy. This allows TD methods to re-use samples many times. However, as the distribution of samples gets further from the distribution of state and actions the current policy will see, this leads to significant problems, which we describe later in Section 2.3.2.

Model-Based Methods The final category of methods is Model-Based methods. With Model-Based methods, the goal is to approximate the transition dynamic function, p, directly. Once an accurate approximation of p is obtained, we can take advantage of the large body of planning algorithms to compute an optimal policy.

Because model updates are independent of the data-collection policy, model-based methods tend to be the most sample efficient methods. However, they come with a significant drawback – modeling error. Unlike traditional regression, errors in model learning have a compounding effect on the value estimation error. This is because the model is called at every step of planning.

2.1.3 Exploration

Another critical challenge for RL is exploration. As the state-action space grows, how can the agent gather enough data to find the optimal policy? This process of gathering data of the state-action space is known as exploration.

One of the most common exploration techniques is random exploration. Notable examples of random exploration are ϵ-greedy, where the agent takes actions greedy actions $1 - \epsilon$ proportion of the time and takes a random action ϵ proportion of the time, and entropy-maximizing techniques, such as Soft Actor-Critic [46], which augment the reward with an policy entropy term to encourage exploration.

However, in environments with sparse rewards, random exploration tends to perform poorly. An alternative approach, often referred to as optimistic exploration or "optimism in the face of uncertainty", is to assume a given state-action pair has high return until it has been sufficiently explored. One of the first of these approaches is R-Max [22], which exhaustively explores the state-action space by assuming every state has maximum reward until it has been explored some number n times. Other optimistic exploration algorithms are able to improve sample efficiency over R-Max. Model-Based Interval Estimation (MBIE) [99], for example, maintains uses Hoefding bounds to maintain confidence intervals over the transition dynamics. With these confidence intervals, MBIE can maximize over both the policy and transition dynamics to maximize discounted return, resulting in a policy that receives high return over the best-case dynamics. However, as the state-action space grows, exhaustive exploration becomes impractical. Using the idea that similar states exhibit similar behavior, some approaches [13] get around this issue by approximating how often a given state or similar states have been explored.

2.1.4 Provably Efficient and Optimal RL

How can we prove that an RL algorithm will result in an optimal policy? Most commonly, this is done using sample efficiency bounds. Sample efficiency bounds for RL fall into two main categories: 1) regret [56] and 2) PAC (Probably Approximately Correct) bounds [31, 101]. Regret bounds bound the total regret, or the difference in reward the agent received versus what the optimal policy would have received. The longer the algorithm takes to converge to the optimal policy, the higher the regret bounds. PAC bounds are slightly different. While there are multiple formulations of the PAC bounds for RL, in this thesis we focus on the PAC-MDP [101] framework. PAC-MDP bounds bound the number of ϵ-suboptimal steps taken by the learning agent.

In this thesis, we use the PAC formulation for our analyses. PAC-MDP bounds have been shown for many popular exploration techniques, including R-Max [22] and a slightly modified Q-Learning [100]. As mentioned above, while R-Max is PAC-MDP, it explores the state-action space exhaustively, which can be inefficient in large domains. MBIE [99], on the other hand, is also PAC-MDP optimal and outperforms the sample efficiency of R-Max.

2.2 RL for Safety-Critical Applications

RL has been applied very successfully in simulated problems, such as video games [15, 81, 117] or controlled robotics settings [1, 48]. However, there has been a surprising lack of RL applications on real-world control environments, such as autonomous driving, autonomous manufacturing, or control of Tokamak fusion reactors. One of the key reasons for this lack of applicability is concerns related to safety. Many real-world control problems are "safety-critical" systems, where a single action, either during training or deployment, can result in significant harm. RL exploration algorithms generally do not take safety considerations into account and policies learned using RL algorithms are notoriously brittle to changes to the environment. Thus, in order to apply RL in safety-critical environments, we need to be able to safely learn policies and improve the robustness of the learned policies.

2.2.1 At Training Time – Safe RL

Safe RL is the a subfield of RL concerned with learning control policies while maintaining some notion of safety throughout the training process. The two most common notions of safety are predefined unsafe state-action pairs that should be avoided or "returnability", which marks state-action pairs as safe if there exists a path to return the agent back to the starting location.

Many safe RL techniques require sufficient prior knowledge to guarantee safety a priori. Risk-aware control methods [17, 32, 87], for example, can compute safe control policies even in situations where the state is not known exactly, but require the dynamics to be known a priori. Similar works [52, 91] allow for learning unknown dynamics, but assume sufficient prior knowledge to determine safety information before exploring. In domains where safety barriers are easy to construct, for example in slow-moving robotics domains, these methods work very well. However, as the system dynamics become more complex, the prior knowledge required for these approaches is challenging to construct a priori.

If we wish to preserve safety while exploring an environment with unknown dynamics, we must have a way to infer the safety of unexplored, potentially dangerous state-action pairs. To do this, assumptions must be made on the environment. Moldovan and Abbeel [82], for example, assume that the agent knows a priori a function that can compute the transition dynamics given observable attributes of nearby state-action pairs. However, the most common approach is to make some regularity assumption about the dynamics in order to transfer knowledge between "similar" state-action pairs.

State-action similarities have been used outside of the safety literature in order to improve computation time of planning and sample complexity of exploration. Bisimulation seeks to aggregate states into groupings of states that have similar dynamics or similar Q-values [2, 43, 59, 111]. These state aggregations allow for more efficient planning and exploration [59]. Other work has used pseudo-counts to learn approximate state aggregations [108]. However, these methods are not amenable to environments where similarities cannot easily partition the state space, such as situations where the similarity between two states is proportional to their distance.

Other works make assumptions about the regularity of the transition or safety functions of the

environment, which allows them to model the uncertainty in these functions using Gaussian Processes (GPs) [6, 14, 115, 118]. Then, by examining the worst-case estimate of this model, they guarantee safety on continuous environments by examining the worst-case estimate of the model. However, there are two key downsides to these approaches. First, (with the exception of [118]), these approaches are not reward-directed, and instead focus on only exploring the state-action space as much as possible. Second, although the GP-based methods [6, 14, 115, 118] help capture uncertainty, they do not model inherent stochasticity in the environment and instead assume the true transition function to be deterministic.

2.2.2 At Deployment Time – Robust Control

Robust control is concerned with the design of feedback controllers for dynamical systems with modeling uncertainties and/or external disturbances [11, 130], specifically controllers with guaranteed performance under worst-case conditions. Many classes of robust control problems in both the time and frequency domains can be formulated using linear matrix inequalities (LMIs) [21, 65]; for reasonably-sized problems, these LMIs can be solved using off-the-shelf numerical solvers based on interior-point or first-order (gradient-based) methods. However, providing stability guarantees often requires the use of simple (linear) controllers, which greatly limits average-case performance.

In contrast, RL (and specifically, deep RL) is not restricted to simple controllers or problems with uncertainty bounds on the dynamics. Instead, deep RL seeks to learn an optimal control policy, represented by a neural network, by directly interacting with an unknown environment. These methods have shown impressive results in a variety of complex control tasks (e.g., [7, 81]); see [25] for a survey. However, due to its lack of safety guarantees, deep RL has been predominantly applied to simulated environments or highly-controlled real-world problems, where system failures are either not costly or not possible.

Efforts to address the lack of safety and stability in RL fall into two main categories. The first tries to combine control-theoretic ideas, predominantly robust control, with the nonlinear control policy benefits of RL (e.g., [3, 26, 30, 34, 51, 58, 75, 78, 83, 92, 122, 128]). For example, RL has been used to address stochastic stability in H_∞ control synthesis settings by jointly learning Lyapunov functions and policies in these settings [51]. As another example, RL has been used to address H_∞ control for continuous-time systems via min-max differential games, in which the controller and disturbance are the "minimizer" and "maximizer" [83]. Previous work, however, has not been applied to the broad class of class of robust control settings outside the H_∞ setting.

The second category of methods uses Constrained Markov Decision Processes (C-MDPs). These methods seek to maximize a discounted reward while bounding some discounted cost function [4, 8, 109, 124]. While these methods do not require knowledge of the cost functions a-priori, they only guarantee the cost constraints hold during test time. Additionally, using C-MDPs can yield other complications, such as optimal policies being stochastic and the constraints only holding for a subset of states.

2.3 Offline RL

An alternative approach to safely learning a control policy in a safety-critical system is to never interact with the system itself. Offline RL considers the problem of learning a control policy from a fixed offline dataset. There are many real-world environments, such as autonomous driving and autonomous manufacturing, where there exists a large amount of sub-optimal control data. Can this data be used to learn an optimal policy, or a policy that out-performs the data-collection policy?

There are some key advantages to offline RL over online RL. The first is safety. Since the dataset is fixed, the agent never needs to take potentially dangerous actions in the environment to explore and gather data. The second is cost. Real world data collection on control system is a costly process, both in terms of time and money. Since all the data is assumed to be already collected, there is no need to perform this costly process. However, there are challenges to offline RL that are not present in standard online RL.

In offline RL, if we wish to learn a policy that improves performance over the data-collection policy, we will need to learn a policy that differs from the data-collection policy. The difference between the learned policy and data-collection policy will result in a *distribution shift* between the dataset and the visitation distribution of the policy. Thus, to update the learned policy, we will need to perform *off-policy* learning, using data collected with a different policy than the learned policy.

Each of the three categories is able to perform off-policy learning to different degrees. As alluded to earlier, PG methods are on-policy algorithms. Thus, the policy must remain very close to the data-collection policy to avoid divergence. TD methods and Model-Based methods, on the other hand, are both able to perform updates off-policy; however, not without their own challenges.

The main challenges to performing off-policy RL are *support mismatch* and *projected-TD instability*, a problem unique to TD methods. Support mismatch refers to the problem where the support of the data distribution does not the visitation distribution of the learned policy. If there is no training data in a given area of the state-action space, the errors in the model or policy can be arbitrarily large. This results in very poor performing policy. Projected-TD instability describes the instability inherent to off-policy TD learning with function approximation, known as the *deadly triad*.

2.3.1 Support Mismatch and Epistemic Uncertainty

When the support of the data distribution and the policy visitation distribution do not match, there is not enough data to accurately predict the performance of the policy, whether using TD or Model-Based methods. Uncertainty deriving from lack of data samples is known as *epistemic uncertainty*. Accurately estimating epistemic uncertainty can help construct policies that either avoid areas of high epistemic uncertainty or act conservatively in the face of uncertainty. However, standard deep networks are not able to predict epistemic uncertainty.

There are two primary methods of formalizing epistemic uncertainty: (1) as a Gaussian Process

(GP) or (2) as a Bayesian Inference problem. The GP framework is often used in practice, specifically in low-dimensional, low-data settings where the problem can be solved exactly [55, 113]. Many GP approximation methods have been proposed to improve scalability [120, 121], but each have their drawbacks. We highlight Spectral-normalized Neural Gaussian Processes (SNGP) [76] as one of the best GP approximation methods for high dimensional problems. While SNGP shows impressive out of distribution prediction performance, we show in our experiments that the learned posterior does not adequately capture the true posterior.

The Bayesian Inference formulation, considers a different problem: given a prior distribution over functions and some labeled data, the goal is to construct a posterior distribution in the Bayesian sense. Because this posterior distribution is usually intractable to compute exactly, approximate sampling methods are used instead [18, 110]. For high-dimensional problems, Bayesian Neural Networks (BNNs) [19] and Bayesian Dropout [19] are alternative methods for approximating Bayesian Inference on neural networks.

Neural network ensembling is a method that predicts out-of-distribution data well in practice, requires very little fine-tuning and, under the right regularization, approximates samples from the Bayesian posterior He et al. [54], Pearce et al. [90]. However, these methods have one main drawback, namely that each new sample from the posterior requires training a new network from scratch. Our method, on the other hand, seeks to construct a generative posterior model using similar regularization techniques to Pearce et al. [90], allowing for quick sampling from the posterior. Epistemic Neural Networks Osband et al. [88] are a concurrent work to ours that, while not motivated by the Bayesian Inference problem, arrived at a very similar technique to ours. The key difference between our method and Epistemic NNs is in the regularization: our method requires unlabeled data from the test-distribution for our regularization, but results in improved posterior sampling.

2.3.2 The Deadly Triad: Off-Policy TD-Learning with Function Approximation

First described by Tsitsiklis and Van Roy [114], the use of TD learning, function approximation, and off-policy sampling together, known as the *deadly triad* [102, p. 264], can cause severe instability or divergence. This instability is caused by projecting TD-updates onto our linear basis,[2] which can result in TD-updates that increase value error and, in some cases, diverge.

Recall that using TD-learning to solve for the state-value function, V, requires finding the fixed point to the Bellman equation. In matrix form, this can be written as

$$V = R + \gamma PV. \tag{2.7}$$

If the state-action space is continuous (or simply very large), we will not be able to exactly compute the value function. Instead, we approximate the value function as $V(s) \approx w^\top \phi(s)$, where $\phi : \mathcal{S} \to \{x \in \mathbb{R}^k : \|x\|_2 = 1\}$ is a fixed normalized basis function and we estimate parameters $w \in \mathbb{R}^k$. In matrix notation, we write this as $V \approx \Phi w$. In the off-policy setting, the

[2]This challenge remains present in the deep RL setting.

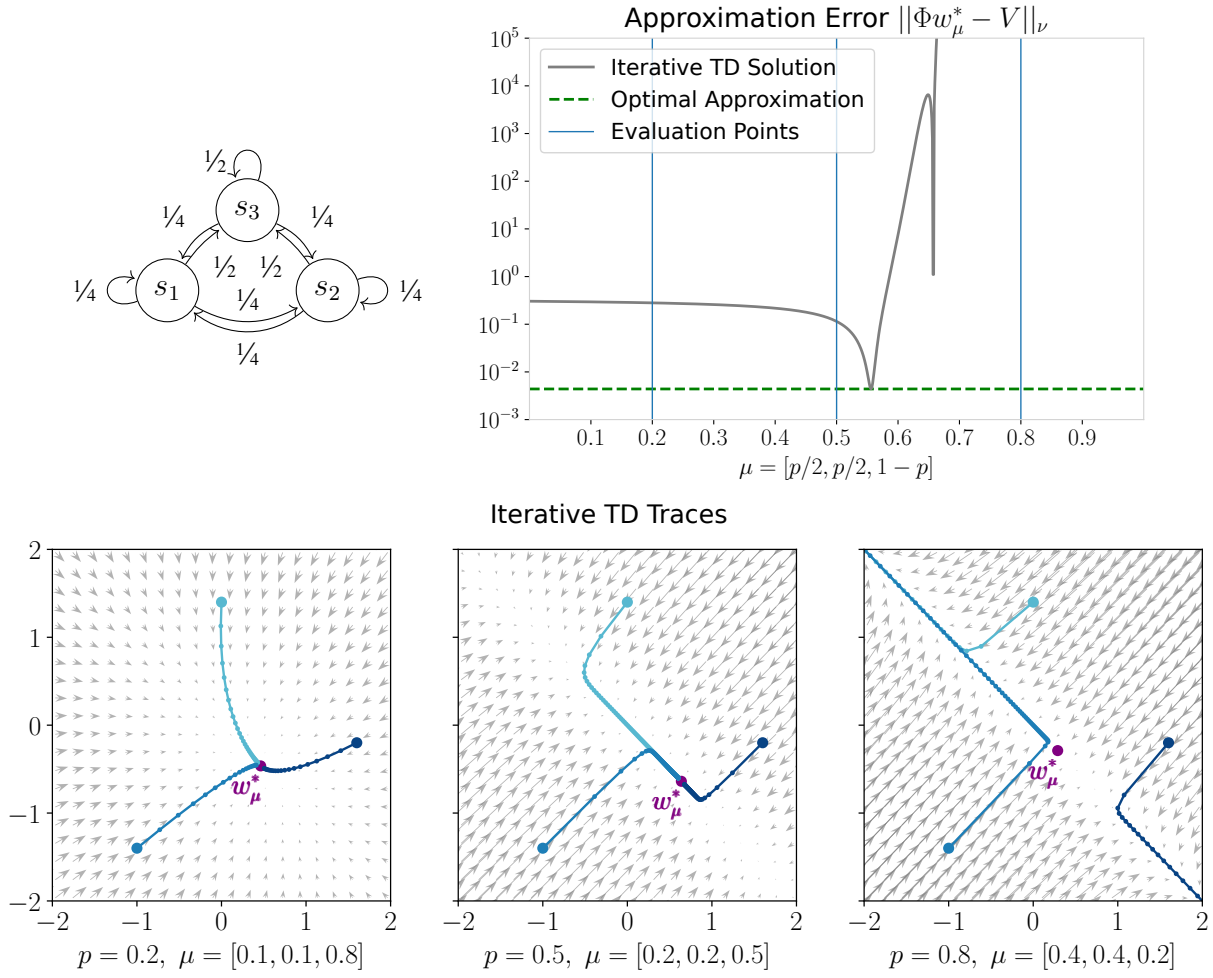


Figure 2.1: The three-state MRP by Manek and Kolter [80] (top-left), a plot of Q-function approximation error over different sampling distributions using Iterative TD and TD traces for at three different evaluation sampling distributions. We can see that performing off-policy, projected TD updates off-policy can lead to divergence.

sampling distribution μ differs from the stationary distribution ν. In this setting, the temporal difference (TD) solution is the fixed point of the projected Bellman equation:

$$\Phi w^\star = \Pi_\mu(R + \gamma P \Phi w^\star), \tag{2.8}$$

where $\Pi_\mu = \Phi(\Phi^\top D_\mu \Phi)^{-1}\Phi^\top D_\mu$ is the projection onto the column space of Φ weighted by the data distribution μ through the matrix $D_\mu = \mathrm{diag}(\mu)$. This projection may be arbitrarily far from the true solution so that the error may be correspondingly large. Moreover, the fixed point of the projected Bellman equation does not equal the optimal approximation, $\arg\min_w \|\Phi w - V\|_\mu$.

To illustrate how the deadly triad can lead to divergence of TD updates, we use the simple three-state MRP introduced in Manek and Kolter [80] (Figure 2.1). In this example, the value function is given by $V = [1,\ 1,\ 1.05]^\top$, with discount factor $\gamma = 0.99$, reward function $R = (I - \gamma P)V$,

and basis Φ where

$$\Phi = \begin{bmatrix} 1 & 0 \\ 0 & -1. \\ 1/2(1.05+\epsilon) & -1/2(1.05+\epsilon) \end{bmatrix} \tag{2.9}$$

The basis includes the representation error term $\epsilon = 10^{-4}$.

For illustration purposes, we select the family of distributions $\mu = (p/2, p/2, 1-h)$ parameterized by $p \in [0, 1]$. This characterizes the possible distributions of data that we will use for the TD updates. The on-policy distribution corresponds to $p = 0.5$ and the approximation error is minimized where $p \approx 0.55$. However, when $p > 0.55$, the TD updates begin to increase in approximation error and, eventually, diverge. Figure 2.1 also plots TD-Learning traces for three different sampling distributions. We can see when $p = 0.8$ (right), the TD updates point away from the fixed point solution.

2.3.3 Current Approaches to Offline RL

There are many methods that address the challenges of off-policy RL, most of which fall into two main categories. The first is importance sampling (IS). First proposed by Precup et al. [93], IS methods for RL approximate the on-policy distribution by reweighting samples with the ratio of the on-policy and data distribution densities. The challenge with this approach is the high variance in the updates of the re-weighting terms, which grows exponentially with the trajectory length. Many approaches have looked at methods for reducing this variance [41, 49, 74, 77, 84, 85]. Emphatic-TD [57, 106, 129] and Gradient-TD [105] are other importance sampling approaches that are provably stable in the full-support linear case. One critical challenge with IS methods is that they do not address support mismatch and, thus, tend to perform poorly on larger scale problems.

The second category of methods involves regularizing the policy towards the data-policy. This policy regularization can be done explicitly by ensuring the learned policy remains "close" to the data collection policy or implicitly through conservative methods. Explicit policy regularization can be achieved by penalizing KL-divergence between the learned policy and data policy [37, 123] or by regularizing the policy parameters [79]. However, even in small-scale settings, policy-regularization methods can be shown to diverge [80]. Conservative methods, on the other hand, involve making a conservative estimate of the value function, thus creating policies that avoid low-support regions of the state-action space. In the online learning setting, one of the most common conservative methods is Trust Region Policy Optimization (TRPO) [96]. However, TRPO does not extend well to the fully offline setting since the value estimates tend to be overly conservative as the learned policy diverges from the data policy. One of the most successful algorithms for offline RL is Conservative Q-Learning (CQL, Kumar et al. [69]), which adds a cost to out-of-distribution actions. Other methods use conservative value estimates with ensembles [67] or model-based approaches [63, 127]. One downside to policy regularized methods (explicit or implicit) is that regularization can reduce policy performance when the data policy is sub-optimal, which we demonstrate in our experiments.

There are a few notable approaches that do not fit into either of these categories. Kumar et al. [68] present DisCor, which reweights the sampling distribution such that the TD fixed point solution minimizes Q-approximation error. However, approximations are needed to make this approach tractable. Additionally, this algorithm does address the support mismatch problem.

Another approach is TD Distribution Optimization (TD-DO) [64], which seeks to reweight the sampling distribution such that the TD updates satisfy the contraction mapping condition, thereby ensuring that TD updates converge. Unfortunately, the use of this approach has been limited because it does not scale to modern RL tasks. This is due to the need to run a batch SDP solver to solve the associated distributional optimization task for each batch update.

Chapter 3

Enforcing robust control guarantees within neural network policies

Policies learned using RL techniques are notoriously brittle to small changes in the dynamics at deployment time. This lack of robustness is a key barrier to RL methods being deployed in real-world, safety-critical systems. However, compared to current robust control methods, control policies learned using RL techniques tend to have significantly better performance in the average case.

In this chapter, we introduce a new method to augment RL algorithms to provide robustness guarantees to the resulting policies. While our method slightly reduces average-case performance compared to the vanilla RL methods, our method is robust to worst-case disturbances, while RL methods are not and quickly diverge in the presence of disturbances. Additionally, our methods out-perform other robust control techniques in the average case. This technique could allow traditionally brittle RL techniques to be applied in safety-critical systems.

3.1 Introduction

The field of robust control, dating back many decades, has been able to provide rigorous guarantees on when controllers will succeed or fail in controlling a system of interest. In particular, if the uncertainties in the underlying dynamics can be bounded in specific ways, these techniques can produce controllers that are provably robust even under worst-case conditions. However, as the resulting policies tend to be simple (i.e., often linear), this can limit their performance in typical (rather than worst-case) scenarios. In contrast, recent high-profile advances in deep reinforcement learning have yielded state-of-the-art performance on many control tasks, due to their ability to capture complex, nonlinear policies. However, due to a lack of robustness guarantees, these techniques have still found limited application in safety-critical domains where an incorrect action (either during training or at runtime) can substantially impact the controlled system.

In this chapter, we propose a method that combines the guarantees of robust control with the flexibility of deep reinforcement learning (RL). Specifically, we consider the setting of nonlinear,

time-varying systems with unknown dynamics, but where (as common in robust control) the uncertainty on these dynamics can be bounded in ways amenable to obtaining provable performance guarantees. Building upon specifications provided by traditional robust control methods in these settings, we construct a new class of nonlinear policies that are parameterized by neural networks, but that are nonetheless *provably robust*. In particular, we *project* the outputs of a nominal (deep neural network-based) controller onto a space of stabilizing actions characterized by the robust control specifications. The resulting nonlinear control policies are trainable using standard approaches in deep RL, yet are *guaranteed* to be stable under the same worst-case conditions as the original robust controller.

We describe our proposed deep nonlinear control policy class and derive efficient, differentiable projections for this class under various models of system uncertainty common in robust control. We demonstrate our approach on several different domains, including synthetic linear differential inclusion (LDI) settings, the cart-pole task, a quadrotor domain, and a microgrid domain. Although these domains are simple by modern RL standards, we show that purely RL-based methods often produce unstable policies in the presence of system disturbances, both during and after training. In contrast, we show that our method remains stable even when worst-case disturbances are present, while improving upon the performance of traditional robust control methods.

3.2 Background on LQR and robust control specifications

In this chapter, our aim is to control nonlinear (continuous-time) dynamical systems of the form

$$\dot{x}(t) \in A(t)x(t) + B(t)u(t) + G(t)w(t), \tag{3.1}$$

where $x(t) \in \mathbb{R}^s$ denotes the state at time t; $u(t) \in \mathbb{R}^a$ is the control input; $w(t) \in \mathbb{R}^d$ captures both external (possibly stochastic) disturbances and any modeling discrepancies; $\dot{x}(t)$ denotes the time derivative of the state x at time t; and $A(t) \in \mathbb{R}^{s\times s}$, $B(t) \in \mathbb{R}^{s\times a}$, $G(t) \in \mathbb{R}^{s\times d}$. This class of models is referred to as linear differential inclusions (LDIs); however, we note that despite the name, this class does indeed characterize *nonlinear* systems, as, e.g., $w(t)$ can depend arbitrarily on $x(t)$ and $u(t)$ (though we omit this dependence in the notation for brevity). Within this class of models, it is often possible to construct robust control specifications certifying system stability. Given such specifications, our proposal is to learn nonlinear (deep neural network-based) policies that *provably* satisfy these specifications while optimizing some objective of interest. We start by giving background on the robust control specifications and objectives considered in this work.

3.2.1 Robust control specifications

In the continuous-time, infinite-horizon settings we consider here, the goal of robust control is often to construct a time-invariant control policy $u(t) = \pi(x(t))$, alongside some certification that guarantees that the controlled system will be stable (i.e., that trajectories of the system will converge to an equilibrium state, usually $x = 0$ by convention; see [47] for a more formal definition). For many classes of systems,[1] this certification is typically in the form of a positive

[1]In this work, we consider sub-classes of system equation 3.1 that may indeed be stochastic (e.g., due to a stochastic external disturbance $w(t)$), but that can be bounded so as to be amenable to deterministic stability analysis.

definite Lyapunov function $V : \mathbb{R}^s \to \mathbb{R}$, with $V(0) = 0$ and $V(x) > 0$ for all $x \neq 0$, such that the function is decreasing along trajectories – for instance,

$$\dot{V}(x(t)) \leq -\alpha V(x(t)) \tag{3.2}$$

for some design parameter $\alpha > 0$. (This particular condition implies *exponential stability* with a rate of convergence α.[2]) For certain classes of bounded dynamical systems, time-invariant linear control policies $u(t) = Kx(t)$, and quadratic Lyapunov functions $V(x) = x^T Px$, it is possible to construct such guarantees using semidefinite programming. For instance, consider the class of norm-bounded LDIs (NLDIs)

$$\dot{x} = Ax(t) + Bu(t) + Gw(t), \quad \|w(t)\|_2 \leq \|Cx(t) + Du(t)\|_2, \tag{3.3}$$

where $A \in \mathbb{R}^{s\times s}$, $B \in \mathbb{R}^{s\times a}$, $G \in \mathbb{R}^{s\times d}$, $C \in \mathbb{R}^{k\times s}$, and $D \in \mathbb{R}^{k\times a}$ are time-invariant and known, and the disturbance $w(t)$ is arbitrary (and unknown) but obeys the norm bounds above.[3] For these systems, it is possible to specify a set of stabilizing policies via a set of linear matrix inequalities (LMIs, [21]):

$$\begin{bmatrix} AS + SA^T + \mu GG^T + BY + Y^T B^T + \alpha S & SC^T + Y^T D^T \\ CS + DY & -\mu I \end{bmatrix} \preceq 0, \quad S \succ 0, \quad \mu > 0, \tag{3.4}$$

where $S \in \mathbb{R}^{s\times s}$ and $Y \in \mathbb{R}^{a\times s}$. For matrices S and Y satisfying equation 3.4, $K = YS^{-1}$ and $P = S^{-1}$ are then a stabilizing linear controller gain and Lyapunov matrix, respectively. While the LMI above is specific to NLDI systems, this general paradigm of constructing stability specifications using LMIs applies to many settings commonly considered in robust control (e.g., settings with norm-bounded disturbances or polytopic uncertainty, or H_∞ control settings). More details about these types of formulations are given in, e.g., Boyd et al. [21]; in addition, we provide the relevant LMI constraints for the settings we consider in this work in Section 3.3.

3.2.2 LQR control objectives

In addition to designing for stability, it is often desirable to optimize some objective characterizing controller performance. While our method can optimize performance with respect to any arbitrary cost or reward function, to make comparisons with existing methods easier, for this chapter we consider the well-known infinite-horizon "linear-quadratic regulator" (LQR) cost, defined as

$$\int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t)\right) dt, \tag{3.5}$$

for some $Q \in \mathbb{S}^{s\times s} \succeq 0$ and $R \in \mathbb{S}^{a\times a} \succ 0$. If the control policy is assumed to be time-invariant and linear as described above (i.e., $u(t) = Kx(t)$), minimizing the LQR cost subject to stability

However, other settings may require stochastic stability analysis; please see [10].

[2] See, e.g., [47] for a more rigorous definition of (local and global) exponential stability. Condition equation 3.2 comes from *Lyapunov's Theorem*, which characterizes various notions of stability using Lyapunov functions.

[3] A slightly more complex formulation involves an additional term in the norm bound, i.e., $Cx(t)+Du(t)+Hw(t)$, which creates a quadratic inequality in w. The mechanics of obtaining robustness specifications in this setting are largely the same as presented here, though with some additional terms in the equations. As such, as is often done, we assume that $H = 0$ for simplicity.

constraints can be cast as an SDP (see, e.g., [125]) and solved using off-the-shelf numerical solvers – a fact that we exploit in our work. For example, to obtain an optimal linear time-invariant controller for the NLDI systems described above, we can solve

$$\underset{S,Y}{\text{minimize}} \quad \text{tr}(QS) + \text{tr}(R^{1/2}YS^{-1}Y^TR^{1/2}) \quad \text{s. t. Equation 3.4 holds.} \tag{3.6}$$

3.3 Details on robust control specifications

As described inthe previous section, for many dynamical systems of the form equation 3.1, it is possible to specify a set of linear, time-invariant policies guaranteeing infinite-horizon exponential stability via a set of LMIs. Here, we derive the LMI equation 3.4 provided in the main text for the NLDI system equation 3.3, and additionally describe relevant LMI systems for systems characterized by polytopic linear differential inclusions (PLDIs) and for H_∞ control settings.

3.3.1 Exponential stability in NLDIs

Consider the general NLDI system equation 3.3. We seek to design a time-invariant control policy $u(t) = Kx(t)$ and a quadratic Lyapunov function $V(x) = x^TPx$ with $P \succ 0$ for this system that satisfy the exponential stability criterion $\dot{V}(x) \leq -\alpha V(x),\ \forall t$. We derive an LMI characterizing such a controller and Lyapunov function, closely following and expanding upon the derivation provided in [21].

Specifically, consider the NLDI system equation 3.3, reproduced below:

$$\dot{x} = Ax + Bu + Gw, \quad \|w\|_2 \leq \|Cx + Du\|_2. \tag{3.7}$$

The time derivative of this Lyapunov function along the trajectories of the closed-loop system is

$$\begin{aligned}
\dot{V}(x) &= \dot{x}^TPx + x^TP\dot{x} \\
&= (Ax + Bu + Gw)^TPx + x^TP(Ax + Bu + Gw) \\
&= ((A + BK)x + Gw)^TPx + x^TP((A + BK)x + Gw) \\
&= \begin{bmatrix} x \\ w \end{bmatrix}^T \begin{bmatrix} (A + BK)^TP + P(A + BK) & PG \\ G^TP & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}.
\end{aligned} \tag{3.8}$$

The exponential stability condition $\dot{V}(x) \leq -\alpha V(x)$ is thus implied by inequality

$$\begin{bmatrix} x \\ w \end{bmatrix}^T M_1 \begin{bmatrix} x \\ w \end{bmatrix} := \begin{bmatrix} x \\ w \end{bmatrix}^T \begin{bmatrix} (A + BK)^TP + P(A + BK) + \alpha P & PG \\ G^TP & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \leq 0. \tag{3.9}$$

Additionally, the norm bound on w can be equivalently expressed as

$$\begin{bmatrix} x \\ w \end{bmatrix}^T M_2 \begin{bmatrix} x \\ w \end{bmatrix} := \begin{bmatrix} x \\ w \end{bmatrix}^T \begin{bmatrix} (C + DK)^T(C + DK) & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \geq 0. \tag{3.10}$$

Using the S-procedure, it follows that for some $\lambda \geq 0$, the following matrix inequality is a sufficient condition for exponential stability:

$$M_1 + \lambda M_2 \preceq 0. \tag{3.11}$$

Using Schur Complements, this matrix inequality is equivalent to

$$(A+BK)^T P + P(A+BK) + \alpha P + \lambda (C+DK)^T (C+DK) + \frac{1}{\lambda} PGG^T P \preceq 0. \tag{3.12}$$

Left- and right-multiplying both sides by P^{-1}, and making the change of variables $S = P^{-1}$, $Y = KS$, and $\mu = 1/\lambda$, we obtain

$$SA^T + AS + Y^T B^T + BY + \alpha S + \frac{1}{\mu} \left(SC^T + Y^T D^T\right)(CS + DY) + \mu GG^T \preceq 0. \tag{3.13}$$

Using Schur Complements again on this inequality, we obtain our final system of linear matrix inequalities as

$$\begin{bmatrix} AS + SA^T + \mu GG^T + BY + Y^T B^T + \alpha S & SC^T + Y^T D^T \\ CS + DY & -\mu I \end{bmatrix} \preceq 0, \quad S \succ 0, \quad \mu > 0, \tag{3.14}$$

where then $K = YS^{-1}$ and $P = S^{-1}$. Note that the first matrix inequality is homogeneous; we can therefore assume $\mu = 1$ (and therefore, $\lambda = 1$), without loss of generality.

3.3.2 Exponential stability in PLDIs

Consider the setting of polytopic linear differential inclusions (PLDIs), where the dynamics are of the form

$$\dot{x}(t) = A(t)x(t) + B(t)u(t), \quad (A(t), B(t)) \in \mathrm{Conv}\{(A_1, B_1), \ldots, (A_L, B_L)\}. \tag{3.15}$$

Here, $A(t) \in \mathbb{R}^{s \times s}$ and $B(t) \in \mathbb{R}^{s \times a}$ can vary arbitrarily over time, as long as they lie in the convex hull (denoted Conv) of the set of points above, where $A_i \in \mathbb{R}^{s \times s}$, $B_i \in \mathbb{R}^{s \times a}$ for $i = 1, \ldots, L$.

We seek to design a time-invariant control policy $u(t) = Kx(t)$ and quadratic Lyapunov function $V(x) = x^T P x$ with $P \succ 0$ for this system that satisfy the exponential stability criterion $\dot{V}(x) \leq -\alpha V(x), \ \forall t$. Such a controller and Lyapunov function exist if there exist $S \in \mathbb{R}^{s \times s} \succ 0$ and $Y \in \mathbb{R}^{a \times s}$ such that

$$A_i S + B_i Y + S A_i^T + Y^T B_i^T + \alpha S \preceq 0, \quad \forall i = 1, \ldots, L, \tag{3.16}$$

where then $K = YS^{-1}$ and $P = S^{-1}$. The derivation of this LMI follows similarly to that for exponential stability in NLDIs, and is well-described in [21].

3.3.3 H_∞ control

Consider the following H_∞ control setting with linear time-invariant dynamics

$$\dot{x}(t) = Ax(t) + Bu(t) + Gw(t), \quad w \in \mathcal{L}_2, \tag{3.17}$$

where A, B, and G are time-invariant as for the NLDI case, and where we define $\mathcal{L}_2$ as the set of time-dependent signals with finite $\mathcal{L}_2$ norm.[4]

In cases such as these with larger or more unstructured disturbances, it may not be possible to guarantee asymptotic convergence to an equilibrium. In these cases, our goal is to construct a robust controller with bounds on the extent to which disturbances affect some performance output (e.g., LQR cost), as characterized by the $\mathcal{L}_2$ gain of the disturbance-to-output map. Specifically, we consider the stability requirement that this $\mathcal{L}_2$ gain be bounded by some parameter $\gamma > 0$ when disturbances are present, and that the system be exponentially stable in the disturbance-free case. This requirement can be characterized via the condition that for all t and some $\sigma \geq 0$,

$$\mathcal{E}(x, \dot{x}, u) := \dot{V}(x) + \alpha V(x) + \sigma \left(x^T Q x + u^T R u - \gamma^2 \|w\|_2^2 \right) \leq 0. \tag{3.18}$$

We note that when $\mathcal{E}(x(t), \dot{x}(t), u(t)) \leq 0$ for all t, both of our stability criteria are met. To see this, note that integrating both sides of equation 3.18 from 0 to ∞ and ignoring the non-negative terms on the left hand side after integration yields

$$\int_0^\infty (x(t)^T Q x(t) + u(t)^T R u(t)) dt \leq \gamma^2 \int_0^\infty \|w(t)\|_2^2 dt + (1/\sigma) V(x(0)). \tag{3.19}$$

This is precisely the desired bound on the $\mathcal{L}_2$ gain of the disturbance-to-output map (see [62]). We also note that in the disturbance-free case, substituting $w = 0$ into equation 3.18 yields

$$\dot{V}(x) \leq -\alpha V(x) - \sigma \left(x^T Q x + u^T R u \right) \leq -\alpha V(x), \tag{3.20}$$

where the last inequality follows from the non-negativity of the LQR cost; this is precisely our condition for exponential stability.

We now seek to design a time-invariant control policy $u(t) = Kx(t)$ and quadratic Lyapunov function $V(x) = x^T P x$ with $P \succ 0$ that satisfies the above condition. In particular, we can write

$$\mathcal{E}\left(x(t), (A + BK)x(t) + Gw(t), Kx(t)\right) = \begin{bmatrix} x(t) \\ w(t) \end{bmatrix}^T M_1 \begin{bmatrix} x(t) \\ w(t) \end{bmatrix}, \tag{3.21}$$

where

$$M_1 := \begin{bmatrix} (A + BK)^T P + P(A + BK) + \alpha P + \sigma(Q + K^T R K) & PG \\ G^T P & -\gamma^2 \sigma I \end{bmatrix}. \tag{3.22}$$

[4]The $\mathcal{L}_2$ norm of a time-dependent signal $w(t) \colon [0, \infty) \to \mathbb{R}^d$ is defined as $\sqrt{\int_0^\infty \|w(t)\|_2^2 dt}$.

Therefore, we seek to find a $P \in \mathbb{R}^{s\times s} \succ 0$ and $K \in \mathbb{R}^{s\times a}$ that satisfy $M_1 \preceq 0$, for some design parameters $\alpha > 0$ and $\sigma > 0$. Using Schur complements, the matrix inequality $M_1 \preceq 0$ is equivalent to

$$(A+BK)^T P + P(A+BK) + \alpha P + \sigma(Q + K^T R K) + PGG^T P/(\gamma^2\sigma) \preceq 0. \tag{3.23}$$

As in Section 3.3.1, we left- and right-multiply both sides by P^{-1}, and make the change of variables $S = P^{-1}$, $Y = KS$, and $\mu = 1/\sigma$ to obtain

$$SA^T + AS + Y^T B^T + BY + \alpha S + \frac{1}{\mu}\left((SQ^{1/2})(Q^{1/2}S) + (Y^T R^{1/2})(R^{1/2}Y)\right) + \mu GG^T/\gamma^2 \preceq 0.$$

Using Schur Complements again, we obtain the LMI

$$\begin{bmatrix} SA^T + AS + Y^T B^T + BY + \alpha S + \mu GG^T/\gamma^2 & \begin{bmatrix} SQ^{1/2} & Y^T R^{1/2}\end{bmatrix} \\ \begin{bmatrix} Q^{1/2}S \\ R^{1/2}Y \end{bmatrix} & -\mu I \end{bmatrix} \preceq 0, \quad S \succ 0, \quad \mu > 0, \tag{3.24}$$

where then $K = YS^{-1}$, $P = S^{-1}$, and $\sigma = 1/\mu$.

3.4 Enforcing robust control guarantees within neural networks

We now present the main contribution: A class of *nonlinear* control policies, potentially parameterized by deep neural networks, that is guaranteed to obey the same stability conditions enforced by the robustness specifications described above. The key insight of our approach is as follows: While it is difficult to derive specifications that globally characterize the stability of a generic nonlinear controller, if we are given *known* robustness specifications, we can create a sufficient condition for stability by simply enforcing that our policy satisfies these specifications at all t. For instance, given a known Lyapunov function, we can enforce exponential stability by ensuring that our policy sufficiently decreases this function (e.g., satisfies Equation 3.2) at any given $x(t)$.

In the following sections, we present our nonlinear policy class, as well as our general framework for learning provably robust policies using this policy class. We then derive the instantiation of this framework for various settings of interest. In particular, this involves constructing (custom) differentiable projections that can be used to adjust the output of a nominal neural network to satisfy desired robustness criteria. For simplicity of notation, we will often suppress the t-dependence of x, u, and w, but we note that these are continuous-time quantities as before.

3.4.1 A provably robust nonlinear policy class

Given a dynamical system of the form equation 3.1 and a quadratic Lyapunov function $V(x) = x^T P x$, let

$$\mathcal{C}(x) := \{u \in \mathbb{R}^a \,|\, \dot{V}(x) \leq -\alpha V(x) \quad \forall \dot{x} \in A(t)x + B(t)u + G(t)w\} \tag{3.25}$$

Algorithm 1 Learning provably robust controllers with deep RL

input performance objective ℓ *// e.g., LQR cost*
input stability requirement *// e.g.,* $\dot{V}(x) \leq -\alpha V(x)$
input policy optimizer $\mathcal{A}$ *// e.g., a planning or RL algorithm*
compute P, K satisfying LMI constraints *// e.g., by optimizing equation 3.6*
construct specifications $\mathcal{C}(x)$ using P *// as defined in Equation 3.25*
construct robust policy class π_θ using $\mathcal{C}$ *// as defined in Equation 3.26*
train π_θ via $\mathcal{A}$ to optimize Equation 3.27
return π_θ

denote a set of actions that, for a *fixed* state $x \in \mathbb{R}^s$, are guaranteed to satisfy the exponential stability condition equation 3.2 (even under worst-case realizations of the disturbance w). We note that this "safe" set is non-empty if P satisfies the relevant LMI constraints (e.g., system equation 3.4 for NLDIs) characterizing robust linear time-invariant controllers, as there is then some K corresponding to P such that $Kx \in \mathcal{C}(x)$ for all states x.

Using this set of actions, we then construct a robust nonlinear policy class that *projects* the output of some neural network onto this set. More formally, consider an arbitrary nonlinear (neural network-based) policy class $\hat{\pi}_\theta : \mathbb{R}^s \to \mathbb{R}^a$ parameterized by θ, and let $\mathcal{P}_{(\cdot)}$ denote the projection operator for some set $(\cdot)$. We then define our robust policy class as $\pi_\theta : \mathbb{R}^s \to \mathbb{R}^a$, where

$$\pi_\theta(x) = \mathcal{P}_{\mathcal{C}(x)}(\hat{\pi}_\theta(x)). \tag{3.26}$$

We note that this policy class is differentiable if the projections can be implemented in a differentiable manner (e.g., using convex optimization layers [5], though we construct efficient custom solvers for our purposes). Importantly, as all policies in this class satisfy the stability condition equation 3.2 for all states x and at all times t, these policies are *certifiably* robust under the same conditions as the original (linear) controller for which the Lyapunov function $V(x)$ was constructed.

Given this policy class and some performance objective ℓ (e.g., LQR cost), our goal is to then find parameters θ such that the corresponding policy optimizes this objective – i.e., to solve

$$\underset{\theta}{\text{minimize}} \quad \int_0^\infty \ell\,(\,x,\,\pi_\theta(x)\,)\,dt \quad \text{s.t. } \dot{x} \in A(t)x + B(t)\pi_\theta(x) + G(t)w. \tag{3.27}$$

Since π_θ is differentiable, we can solve this problem via a variety of approaches, e.g., a model-based planning algorithm if the true dynamics are known, or virtually any (deep) RL algorithm if the dynamics are unknown.[5]

This general procedure for constructing stabilizing controllers is summarized in Algorithm 1. While seemingly simple, this formulation presents a powerful paradigm: by simply transforming the output of a neural network, we can employ an expressive policy class to optimize an objective

[5]While this problem is infinite-horizon and continuous in time, in practice, one would optimize it in discrete time over a large finite time horizon.

of interest while *ensuring* the resultant policy will stabilize the system during both training and testing.

We instantiate our framework by constructing "safe" sets $\mathcal{C}(x)$ and their associated (differentiable) projections $\mathcal{P}_{\mathcal{C}(x)}$ for three settings of interest: NLDIs, polytopic linear differential inclusions (PLDIs), and H_∞ control settings.

3.4.2 Constructing the exponential stability set for NLDIs

In order to apply our framework to the NLDI setting equation 3.3, we first compute a quadratic Lyapunov function $V(x) = x^T P x$ by solving the optimization problem equation 3.6 for the given system via semidefinite programming. We then use the resultant Lyapunov function to compute the system-specific "safe" set $\mathcal{C}(x)$, and then create a fast, custom differentiable solver to project onto this set.

Computing sets of stabilizing actions

Given P, we compute $\mathcal{C}_{\text{NLDI}}(x)$ as the set of actions $u \in \mathbb{R}^a$ that, for each state $x \in \mathbb{R}^s$, satisfy the stability condition equation 3.2 at that state *under even a worst-case realization of the dynamics* (i.e., in this case, even under a worst-case disturbance w). The form of the resultant set is given below.

Theorem 1. *Consider the NLDI system equation 3.3, some stability parameter $\alpha > 0$, and a Lyapunov function $V(x) = x^T P x$ with P satisfying Equation 3.4. Assuming P exists, define*

$$\mathcal{C}_{\textit{NLDI}}(x) := \left\{ u \in \mathbb{R}^a \mid \|Cx + Du\|_2 \leq \frac{-x^T P B}{\|G^T P x\|_2} u - \frac{x^T (2PA + \alpha P) x}{2\|G^T P x\|_2} \right\}$$

for all states $x \in \mathbb{R}^s$. For all x, $\mathcal{C}_{\textit{NLDI}}(x)$ is a non-empty set of actions that satisfy the exponential stability condition equation 3.2. Further, $\mathcal{C}_{\textit{NLDI}}(x)$ is a convex set in u.

Proof. We seek to find a set of actions such that the condition equation 3.2 is satisfied along *all* possible trajectories of equation 3.3. A set of actions satisfying this condition at a given x is given by

$$\mathcal{C}_{\text{NLDI}}(x) := \left\{ u \in \mathbb{R}^a \mid \sup_{w : \|w\|_2 \leq \|Cx + Du\|_2} \dot{V}(x) \leq -\alpha V(x) \right\}.$$

Let $\mathcal{S} := \{w : \|w\|_2 \leq \|Cx + Du\|_2\}$. We can then rewrite the left side of the above inequality as

$$\begin{aligned} \sup_{w \in \mathcal{S}} \dot{V}(x) &= \sup_{w \in \mathcal{S}} \dot{x}^T P x + x^T P \dot{x} \;=\; 2x^T P(Ax + Bu) + \sup_{w \in \mathcal{S}} 2x^T P G w \\ &= 2x^T P(Ax + Bu) + 2\|G^T P x\|_2 \|Cx + Du\|_2, \end{aligned}$$

by the definition of the NLDI dynamics and the closed-form minimization of a linear term over an L_2 ball. Rearranging yields an inequality of the desired form. We note that by definition of the specifications equation 3.4, there is some K corresponding to P such that the policy

$u = Kx$ satisfies the exponential stability condition equation 3.2; thus, $Kx \in \mathcal{C}_{\text{NLDI}}$, and $\mathcal{C}_{\text{NLDI}}$ is non-empty. Further, as the above inequality represents a second-order cone constraint in u, this set is convex in u. □

We further consider the special case where $D = 0$, i.e., the norm bound on w does not depend on the control action. This form of NLDI arises in many common settings (e.g., where w characterizes linearization error in a nonlinear system but the dynamics depend only linearly on the action), and is one for which we can compute the relevant projection in closed form (as described shortly).

Corollary 1. *Consider the NLDI system equation 3.3 with $D = 0$, some stability parameter $\alpha > 0$, and Lyapunov function $V(x) = x^T P x$ with P satisfying Equation 3.4. Assuming P exists, define*

$$\mathcal{C}_{\textit{NLDI-0}}(x) := \left\{u \in \mathbb{R}^a \mid 2x^T PBu \leq -x^T(2PA + \alpha P)x - 2\|G^T Px\|_2 \|Cx\|_2\right\}$$

for all states $x \in \mathbb{R}^s$. For all x, $\mathcal{C}_{\textit{NLDI-0}}(x)$ is a non-empty set of actions that satisfy the exponential stability condition equation 3.2. Further, $\mathcal{C}_{\textit{NLDI-0}}(x)$ is a convex set in u.

Proof. The result follows by setting $D = 0$ in Theorem 1 and rearranging terms. As the above inequality represents a linear constraint in u, this set is convex in u. □

Deriving efficient, differentiable projections

For the general NLDI setting equation 3.3, we note that the relevant projection $\mathcal{P}_{\mathcal{C}_{\text{NLDI}}(x)}$ (see Theorem 1) represents a projection onto a second-order cone constraint. As this projection does not necessarily have a closed form, we must implement it using a differentiable optimization solver (e.g., [5]). For computational efficiency purposes, we implement a custom solver that employs an accelerated projected dual gradient method for the forward pass, and employs implicit differentiation through the fixed point equations of this solution method to compute relevant gradients for the backward pass. Derivations and additional details are provided in Section 3.5.

In the case where $D = 0$ (see Corollary 1), we note that the projection operation $\mathcal{P}_{\mathcal{C}_{\text{NLDI-0}}(x)}$ does have a closed form, and can in fact be implemented via a single ReLU operation. Specifically, defining $\eta^T := 2x^T PB$ and $\zeta := -x^T(2PA + \alpha P)x - 2\|G^T Px\|_2 \|Cx\|_2$, we see that

$$\mathcal{P}_{\mathcal{C}_{\text{NLDI-0}}(x)}\left(\hat{\pi}(x)\right) = \begin{cases} \hat{\pi}(x) & \text{if } \eta^T \hat{\pi}(x) \leq \zeta \\ \hat{\pi}(x) - \frac{\eta^T \hat{\pi}(x) - \zeta}{\eta^T \eta}\eta & \text{otherwise} \end{cases} = \hat{\pi}(x) - \text{ReLU}\left(\frac{\eta^T \hat{\pi}(x) - \zeta}{\eta^T \eta}\right)\eta. \tag{3.28}$$

3.4.3 Constructing the exponential stability set for PLDIs

For the general PLDI system equation 3.15, relevant sets of exponentially stabilizing actions $\mathcal{C}_{\text{PLDI}}$ are given by the following theorem.

Theorem 2. *Consider the PLDI system equation 3.15, some stability parameter $\alpha > 0$, and a Lyapunov function $V(x) = x^T P x$ with P satisfying equation 3.16. Assuming P exists, define*

$$\mathcal{C}_{PLDI}(x) := \left\{ u \in \mathbb{R}^a \mid \begin{bmatrix} 2x^T P B_1 \\ 2x^T P B_2 \\ \vdots \\ 2x^T P B_L \end{bmatrix} u \leq - \begin{bmatrix} x^T(\alpha P + 2PA_1)x \\ x^T(\alpha P + 2PA_2)x \\ \vdots \\ x^T(\alpha P + 2PA_L)x \end{bmatrix} \right\}$$

for all states $x \in \mathbb{R}^s$. For all x, $\mathcal{C}_{PLDI}(x)$ is a non-empty set of actions that satisfy the exponential stability condition equation 3.2. Further, $\mathcal{C}_{PLDI}(x)$ is a convex set in u.

Proof. We seek to find a set of actions such that the condition equation 3.2 is satisfied along *all* possible trajectories of equation 3.15, i.e., for *any* allowable instantiation of $(A(t), B(t))$. A set of actions satisfying this condition at a given x is given by

$$\mathcal{C}_{\text{PLDI}}(x) := \{u \in \mathbb{R}^a \mid \dot{V}(x) \leq -\alpha V(x) \;\; \forall (A(t), B(t)) \in \text{Conv}\{(A_1, B_1), \ldots, (A_L, B_L)\}.$$

Expanding the left side of the inequality above, we see that for some coefficients $\gamma_i \in \mathbb{R} \geq 0, i = 1, \ldots, L$ satisfying $\sum_{i=1}^{L} \gamma_i(t) = 1$,

$$\begin{aligned} \dot{V}(x) \;&=\; \dot{x}^T P x + x^T P \dot{x} \;=\; 2x^T P \left(A(t)x + B(t)u\right) \\ &= 2x^T P \left(\sum_{i=1}^{L} \gamma_i(t) A_i x + \gamma_i(t) B_i u \right) \;=\; \sum_{i=1}^{L} \gamma_i \left(2x^T P \left(A_i x + B_i u\right) \right) \end{aligned}$$

by definition of the PLDI dynamics and of the convex hull. Thus, if we can ensure

$$2x^T P \left(A_i x + B_i u\right) \leq -\alpha V(x) = -\alpha x^T P x, \;\; \forall i = 1, \ldots, L,$$

then we can ensure that exponential stability holds. Rearranging this condition and writing it in matrix form yields an inequality of the desired form. We note that by definition of the specifications equation 3.16, there is some K corresponding to P such that the policy $u = Kx$ satisfies all of the above inequalities; thus, $Kx \in \mathcal{C}_{\text{PLDI}}(x)$, and $\mathcal{C}_{\text{PLDI}}(x)$ is non-empty. Further, as the above inequality represents a linear constraint in u, this set is convex in u. □

We note that the relevant projection $\mathcal{P}_{\mathcal{C}_{\text{PLDI}}(x)}$ represents a projection onto an intersection of halfspaces, and can thus be implemented via differentiable quadratic programming [9].

3.4.4 Constructing the stability set for H_∞ control

For the H_∞ control system equation 3.17, relevant sets of actions satisfying the condition equation 3.18 are given by the following theorem.

Theorem 3. *Consider the system equation 3.17, some stability parameter $\alpha > 0$, and a Lyapunov function $V(x) = x^T P x$ with P satisfying Equation 3.24. Assuming P exists, define*

$$\mathcal{C}_{H_\infty}(x) := \left\{ u \in \mathbb{R}^a \mid u^T R u + (2B^T P x)^T u + x^T \left(PA + A^T P + \alpha P + Q + \gamma^{-2} P G G^T P \right) x \leq 0 \right\}$$

for all states $x \in \mathbb{R}^s$. For all x, $\mathcal{C}_{H_\infty}(x)$ is a non-empty set of actions that guarantee condition equation 3.18, i.e., that the $\mathcal{L}_2$ gain of the disturbance-to-output map is bounded by γ and that the system is exponentially stable in the disturbance-free case. Further, $\mathcal{C}_{H_\infty}(x)$ is convex in u.

Proof. We seek to find a set of actions such that the condition $\mathcal{E}(x, \dot{x}, u) \leq 0$ is satisfied along *all* possible trajectories of equation 3.17, where $\mathcal{E}$ is defined as in equation 3.18. A set of actions satisfying this condition at a given x is given by

$$\mathcal{C}_{H_\infty}(x) := \{u \in \mathbb{R}^a \mid \sup_{w \in \mathcal{L}_2} \ \mathcal{E}(x, \dot{x}, u) \leq 0, \ \dot{x} = Ax + Bu + Gw\}.$$

To begin, we note that

$$\begin{aligned}\mathcal{E}(x, Ax + Bu + Gw, u) = x^T P(Ax + Bu + Gw) + (Ax + Bu + Gw)^T Px + \alpha x^T Px \\ + \sigma \left(x^T Qx + u^T Ru - \gamma^2 \|w\|_2^2\right)\end{aligned}$$

We then maximize $\mathcal{E}$ over w:

$$w^\star = \arg\max_w \ \mathcal{E}(x, Ax + Bu + Gw, u) = G^T Px/(\sigma\gamma^2). \tag{3.29}$$

Therefore,

$$\mathcal{C}_{H_\infty}(x) = \{u \,|\, \mathcal{E}(x, Ax + Bu + Gw^\star, u, w^\star) \leq 0\}. \tag{3.30}$$

Expanding and rearranging terms, this becomes

$$\mathcal{C}_{\mathcal{H}_\infty}(x) = \{u \,|\, u^T(\sigma R)u + (2B^T Px)^T u + x^T \left(PA + A^T P + \alpha P + \sigma Q + PGG^T P/(\sigma\gamma^2)\right) x \leq 0\}. \tag{3.31}$$

We note that by definition of the specifications equation 3.24, there is some K corresponding to P such that the policy $u = Kx$ satisifies the conditions above (see equation 3.23); thus, $Kx \in \mathcal{C}_{H_\infty}$, and $\mathcal{C}_{H_\infty}$ is non-empty. We note further that $\mathcal{C}_{\mathcal{H}_\infty}$ is an ellipsoid in the control action space, and is thus convex in u. □

We rewrite the set $\mathcal{C}_{H_\infty}(x)$ such that the projection $\mathcal{P}_{\mathcal{C}_{H_\infty}(x)}$ can be viewed as a second-order cone projection, in order to leverage our fast custom solver (Section 3.5). In particular, defining $\tilde{P} = \sigma R$, $\tilde{q} = B^T Px$, and $\tilde{r} = x^T \left(PA + A^T P + \alpha P + \sigma Q + PGG^T P/(\sigma\gamma^2)\right) x$, we can rewrite the ellipsoid above as

$$\mathcal{C}_{H_\infty}(x) = \{u \,|\, u^\top \tilde{P} u + 2\tilde{q}^\top u + \tilde{r} \leq 0\}. \tag{3.32}$$

We note that as $\tilde{P} \succ 0$ and $\tilde{r} - \tilde{q}^\top \tilde{P}^{-1} \tilde{q} < 0$, this ellipsoid is non-empty (see, e.g., section B.1 in [20]). We can then rewrite the ellipsoid as

$$\mathcal{C}_{H_\infty}(x) = \{u \,|\, \|\tilde{A}u + \tilde{b}\|_2 \leq 1\} \tag{3.33}$$

where $\tilde{A} = \sqrt{\frac{\tilde{P}}{\tilde{q}^\top \tilde{P}^{-1} \tilde{q} - \tilde{r}}}$ and $\tilde{b} = \sqrt{\frac{P}{q^\top P^{-1} q - r}} P^{-1} q$. The constraint $\|\tilde{A}u + \tilde{b}\|_2 \leq 1$ is then a second-order cone constraint in u.

3.5 Fast, differentiable solver for second-order cone projection

In order to construct the robust policy class described in Section 3.4 for the general NLDI system equation 3.3 and the H_∞ setting equation 3.17, we must project a nominal (neural network-based) policy onto the second-order cone constraints describing the stability sets. As this projection operation does not necessarily have a closed form, we implement it via a custom differentiable optimization solver.

More generally, consider a set of the form

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid \|Ax + b\|_2 \leq c^T x + d\} \tag{3.34}$$

for some $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}$. Given some input $y \in \mathbb{R}^n$, we seek to compute the second-order cone projection $\mathcal{P}_\mathcal{C}(y)$ by solving the problem

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\|x - y\|_2^2 \\ \text{subject to} \quad & \|Ax + b\|_2 \leq c^T x + d. \end{aligned} \tag{3.35}$$

Let $\mathcal{F}$ denote the ℓ_2 norm cone, i.e., $\mathcal{F} := \{(w, t) \mid \|w\|_2 \leq t\}$. Introducing the auxiliary variable $z \in \mathbb{R}^{m+1}$, we can then rewrite the above optimization problem equivalently as

$$\begin{aligned} \underset{x \in \mathbb{R}^n,\, z \in \mathbb{R}^{m+1}}{\text{minimize}} \quad & \frac{1}{2}\|x - y\|_2^2 + \mathbf{1}_\mathcal{F}(z) \\ \text{subject to} \quad & z = \begin{bmatrix} Ax + b \\ c^T x + d \end{bmatrix} =: Gx + h, \end{aligned} \tag{3.36}$$

where for brevity we define $G = \begin{bmatrix} A \\ c^T \end{bmatrix}$ and $h = \begin{bmatrix} b \\ d \end{bmatrix}$, and where $\mathbf{1}_\mathcal{F}$ denotes the indicator function for membership in the set $\mathcal{F}$.

We describe our fast solution technique for computing this projection, as well as our method for obtaining gradients through the solution.

3.5.1 Computing the projection

We construct a fast solver for problem equation 3.36 using an accelerated projected dual gradient method. Specifically, define $\mu = \mathbb{R}^{m+1}$ as the dual variable on the equality constraint in Equation 3.36. The Lagrangian for this problem can then be written as

$$\mathscr{L}(x, z, \mu) = \frac{1}{2}\|x - y\|_2^2 + \mathbf{1}_\mathcal{F}(z) + \mu^T(z - Gx - h), \tag{3.37}$$

and the dual problem is given by $\max_\mu \min_{x,z} \mathscr{L}(x, z, \mu)$. To form the dual problem, we minimize the Lagrangian with respect to x and z as

$$\inf_{x,z} \mathscr{L}(x, z, \mu) = \inf_x \frac{1}{2}\left\{\|x - y\|_2^2 - \mu^T G x\right\} + \inf_z\{\mu^T z + \mathbf{1}_\mathcal{F}(z)\} - \mu^T h. \tag{3.38}$$

We note that the first term on the right side is minimized at $x^\star(\mu) = y + G^T\mu$. Thus, we see that

$$\inf_x \ \frac{1}{2}\{\|x-y\|_2^2 - \mu^T G x\} = -\frac{1}{2}\mu^T G G^T \mu - \mu^T G y. \tag{3.39}$$

For the second term, denote $\mu = (\tilde{\mu}, s)$ and $z = (\tilde{z}, t)$. We can then rewrite this term as

$$\inf_z\{\mu^T z + \mathbf{1}_{\mathcal{F}}(z)\}\} = \inf_{t\geq 0} \ \inf_{\tilde{z}} \ \{t\cdot s + \tilde{\mu}^T\tilde{z} \,|\, \|\tilde{z}\|_2 \leq t\}. \tag{3.40}$$

For a fixed $t \geq 0$, the above objective is minimized at $\tilde{z} = -t\tilde{\mu}/\|\tilde{\mu}\|_2$. (The problem is infeasible for $t < 0$.) Substituting this minimizer into equation 3.40 and minimizing the result over $t \geq 0$ yields

$$\inf_z\{\mu^T z + \mathbf{1}_{\mathcal{F}}(z)\} = \inf_{t\geq 0} \ t(s - \|\tilde{\mu}\|_2) = -\mathbf{1}_{\mathcal{F}}(\mu) \tag{3.41}$$

where the last identity follows from definition of the second-order cone $\mathcal{F}$. Hence the negative dual problem becomes

$$\underset{\mu}{\text{minimize}} \ \ \frac{1}{2}\mu^T G G^T \mu + \mu^T(Gy + h) + \mathbf{1}_{\mathcal{F}}(\mu). \tag{3.42}$$

We now solve this problem via Nesterov's accelerated projected dual gradient method [86]. For notational brevity, define $f(\mu) := \frac{1}{2}\mu^T G G^T \mu + \mu^T(Gy + h)$. Then, starting from arbitrary $\mu^{(-1)}, \mu^{(0)} \in \mathbb{R}^{m+1}$ we perform the iterative updates

$$\begin{aligned} \nu^{(k)} &= \mu^{(k)} + \beta^{(k)}(\mu^{(k)} - \mu^{(k-1)}) \\ \mu^{(k+1)} &= \mathcal{P}_{\mathcal{F}}\left(\nu^{(k)} - \frac{1}{L_f}\nabla f(\nu^{(k)})\right), \end{aligned} \tag{3.43}$$

where $L_f = \lambda_{\max}(GG^T)$ is the Lipschitz constant of f, and $\mathcal{P}_{\mathcal{F}}$ is the projection operator onto $\mathcal{F}$ (which has a closed form solution; see [12]). Letting $m_f = \lambda_{\min}(GG^T)$ denote the strong convexity constant of f, the momentum parameter is then scheduled as [86]

$$\beta^k = \begin{cases} \dfrac{k-1}{k+2} & \text{if } m_f = 0 \\ \dfrac{\sqrt{L_f} - \sqrt{m_f}}{\sqrt{L_f} + \sqrt{m_f}} & \text{if } m_f > 0. \end{cases} \tag{3.44}$$

After computing the optimal dual variable $\mu^\star$, i.e., the fixed point of equation 3.43, the optimal primal variable can be recovered via the equation $x^\star = y + G^T\mu^\star$ (as can be observed from the first-order conditions of the Lagrangian equation 3.37).

3.5.2 Obtaining gradients

In order to incorporate the above projection into our neural network, we need to compute the gradients of all problem variables (i.e., G, h, and y) through the solution $x^\star$. In particular, we note

that $x^\star$ has a direct dependence on both G and y, and an indirect dependence on all of G, h, and y through $\mu^\star$.

To compute the relevant gradients through $\mu^\star$, we apply the implicit function theorem to the fixed point of the update equations equation 3.43. Specifically, as these updates imply that $\mu^\star = \nu^\star$, their fixed point can be written as

$$\mu^\star = \mathcal{P}_\mathcal{F}\left(\mu^\star - \frac{1}{L_f}\nabla f(\mu^\star)\right). \tag{3.45}$$

Define $M := \frac{\partial \mathcal{P}_\mathcal{F}(\cdot)}{\partial(\cdot)}\Big|_{(\cdot)=\mu^\star - \frac{1}{L_f}\nabla f(\mu^\star)}$, and note that $\nabla f(\mu^\star) = GG^T\mu^\star + Gy + h$. The differential of the above fixed-point equation is then given by

$$\mathrm{d}\mu^\star = M \times \left(\mathrm{d}\mu^\star - \frac{1}{L_f}\left(\mathrm{d}GG^T\mu^\star + G\mathrm{d}G^T\mu^\star + GG^T\mathrm{d}\mu^\star + \mathrm{d}Gy + G\mathrm{d}y + \mathrm{d}h\right)\right). \tag{3.46}$$

Rearranging terms to separate the differentials of problem outputs from problem variables, we see that

$$\left(I - M + \frac{1}{L_f}MGG^T\right)\mathrm{d}\mu^\star = -\frac{1}{L_f}M\left(\mathrm{d}GG^T\mu^\star + G\mathrm{d}G^T\mu^\star + \mathrm{d}Gy + G\mathrm{d}y + \mathrm{d}h\right), \tag{3.47}$$

where I is the identity matrix of appropriate size.

As described in e.g. [9], we can then use these equations to form the Jacobian of $\mu^\star$ with respect to any of the problem variables by setting the differential of the relevant problem variable to I and of all other problem variables to 0; solving the resulting equation for $\mathrm{d}\mu^\star$ then yields the value of the desired Jacobian. However, as these Jacobians can be large depending on problem size, we rarely want to form them explicitly. Instead, given some backward pass vector $\frac{\partial \ell}{\partial \mu^\star} \in \mathbb{R}^{1\times(m+1)}$ with respect to the optimal dual variable, we want to directly compute the gradient of the loss with respect to the problem variables: e.g., for y, we want to directly form the result of the product $\frac{\partial \ell}{\partial \mu^\star}\frac{\partial \mu^\star}{\partial y} \in \mathbb{R}^{1\times n}$. We do this via a similar method as presented in [9], and refer the reader there for a more in-depth explanation of the method described below.

Define $J := I - M + \frac{1}{L_f}MGG^T$ to represent the coefficient of $\mathrm{d}\mu^\star$ on the left side of Equation 3.47. Given $\frac{\partial \ell}{\partial \mu^\star}$, we then compute the intermediate term

$$\mathrm{d}_\mu := -J^{-T}\left(\frac{\partial \ell}{\partial \mu^\star}\right)^T. \tag{3.48}$$

We can then form the relevant gradient terms directly as

$$\begin{aligned}
\left(\frac{\partial \ell}{\partial \mu^\star}\frac{\partial \mu^\star}{\partial G}\right)^T &= \frac{1}{L_f}M\left(\mathrm{d}_\mu(G^T\mu^\star)^T + \mu^\star(G^T\mathrm{d}_\mu)^T + \mathrm{d}_\mu y^T\right) \\
\left(\frac{\partial \ell}{\partial \mu^\star}\frac{\partial \mu^\star}{\partial h}\right)^T &= \frac{1}{L_f}M\mathrm{d}_\mu \\
\left(\frac{\partial \ell}{\partial \mu^\star}\frac{\partial \mu^\star}{\partial y}\right)^T &= \frac{1}{L_f}G^TM\mathrm{d}_\mu.
\end{aligned} \tag{3.49}$$

In these computations, we note that as our solver returns $x^\star$, the backward pass vector we are given is actually $\frac{\partial \ell}{\partial x^\star} \in \mathbb{R}^{1\times n}$; thus, we compute $\frac{\partial \ell}{\partial \mu^\star} = \frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial \mu^\star} = \frac{\partial \ell}{\partial x^\star}G^T$ for use in Equation 3.48.

Accounting additionally for the direct dependence of some of the problem variables on $x^\star$ (recalling that $x^\star = y + G^T u^\star$), the desired gradients are then given by

$$\begin{aligned}
\left(\frac{\partial \ell}{\partial G}\right)^T &= \left(\frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial G} + \frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial u^\star}\frac{\partial u^\star}{\partial G}\right)^T = \mu^\star \frac{\partial \ell}{\partial x^\star} + \frac{1}{L_f} M\left(\mathrm{d}_\mu (G^T\mu^\star)^T + \mu^\star (G^T \mathrm{d}_\mu)^T + \mathrm{d}_\mu y^T\right) \\
\left(\frac{\partial \ell}{\partial h}\right)^T &= \left(\cancel{\frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial h}}^{0} + \frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial u^\star}\frac{\partial u^\star}{\partial h}\right)^T = \frac{1}{L_f} M \mathrm{d}_\mu \\
\left(\frac{\partial \ell}{\partial y}\right)^T &= \left(\frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial y} + \frac{\partial \ell}{\partial x^\star}\frac{\partial x^\star}{\partial u^\star}\frac{\partial u^\star}{\partial y}\right)^T = \left(\frac{\partial \ell}{\partial x^\star}\right)^T + \frac{1}{L_f} G^T M \mathrm{d}_\mu.
\end{aligned} \tag{3.50}$$

3.6 Experiments

Having instantiated our general framework, we demonstrate the power of our approach on a variety of simulated control domains.[6] In particular, we evaluate performance on the following metrics:

- **Average-case performance:** How well does the method optimize the performance objective (i.e., LQR cost) under average (non-worst case) dynamics?
- **Worst-case stability:** Does the method remain stable even when subjected to adversarial (worst-case) dynamics?

In all cases, we show that our method is able to improve performance over traditional robust controllers under average conditions, while still guaranteeing stability under worst-case conditions.

3.6.1 Description of dynamics settings

We evaluate our approach on five NLDI settings: two synthetic NLDI domains, a synthetic PLDI and H_∞ domain, the cart-pole task, a quadrotor domain, and a microgrid domain. For each setting, we choose a time discretization based on the speed at which the system evolves, and run each episode for 200 steps over this discretization. In all cases except the microgrid setting, we use a randomly generated LQR objective where the matrices $Q^{1/2}$ and $R^{1/2}$ are drawn i.i.d. from a standard normal distribution.

Synthetic NLDI settings. We generate NLDIs of the form equation 3.3 with $s = 5$, $a = 3$, and $d = k = 2$ by generating matrices A, B, G, C and D i.i.d. from normal distributions, and producing the disturbance $w(t)$ using a randomly-initialized neural network (with its output scaled

[6] Code for all experiments is available at `https://github.com/locuslab/robust-nn-control`

to satisfy the norm-bound on the disturbance). We investigate settings both where $D \neq 0$ and where $D = 0$. In both cases, episodes are run for 2 seconds at a discretization of 0.01 seconds.

Synthetic PLDI setting. We generate PLDI instances equation 3.15 with $s = 5$, $a = 3$, and $L = 3$. Specifically, we generate convex hull matrices $(A_1, B_1), \ldots, (A_3, B_3)$ i.i.d. from normal distributions, and generate $(A(t), B(t))$ by using a randomly-initialized neural network with softmax output to weight the convex hull matrices. Episodes were run for 2 seconds at a discretization of 0.01 seconds.

Synthetic H_∞ setting. We generate H_∞ control instances equation 3.17 with $s = 5$, $a = 3$, and $d = 2$ by generating matrices A, B and G i.i.d. from normal distributions. The disturbance $w(t)$ was produced using a randomly-initialized neural network, with its output scaled to satisfy the $\mathcal{L}_2$ bound on the disturbance. Specifically, we scaled the output of the neural network to satisfy an attenuating norm-bound on the disturbance; at time t, the norm-bound was given by $20 \times f(2 \times t/T)$, where T is the time horizon and f is the standard normal PDF function. Episodes were run for $T = 2$ seconds at a discretization of 0.01 seconds.

Cart-pole. In the cart-pole task, our goal is to balance an inverted pendulum resting on top of a cart by exerting horizontal forces on the cart. For our experiments, we linearize this system as an NLDI with $D \neq 0$ (see Section 3.6.4), and add a small additional randomized disturbance satisfying the NLDI bounds. Episodes are run for 10 seconds at a discretization of 0.05 seconds.

Planar quadrotor. In this setting, our goal is to stabilize a quadcopter in the two-dimensional plane by controlling the amount of force provided by the quadcopter's right and left thrusters. We linearize this system as an NLDI with $D = 0$ (see Section 3.6.5), and add a small disturbance as in the cart-pole setting. Episodes are run for 4 seconds at a discretization of 0.02 seconds.

Microgrid. In this final setting, we aim to stabilize a microgrid by controlling a storage device and a solar inverter. We augment the system given in [72] with LQR matrices and NLDI bounds (see Section 3.6.6). Episodes are run for 2 seconds at a discretization of 0.01 seconds.

3.6.2 Experimental setup

We demonstrate our approach by constructing a robust policy class equation 3.26 for each of these settings, and optimizing this policy class via different approaches. Specifically, we construct a nominal nonlinear control policy class as $\hat{\pi}_\theta(x) = Kx + \tilde{\pi}_\theta(x)$, where K is obtained via robust LQR optimization equation 3.6, and where $\tilde{\pi}_\theta(x)$ is a feedforward neural network. To construct the projections $\mathcal{P}_\mathcal{C}$, we employ the value of P obtained when solving for K. For the purposes of demonstration, we then optimize our robust policy class $\pi_\theta(x) = \mathcal{P}_\mathcal{C}(\hat{\pi}_\theta(x))$ using two different methods:

- **Robust MBP** (ours)**:** A model-based planner that assumes the true dynamics are known.
- **Robust PPO** (ours)**:** An RL approach based on PPO [97] that does not assume known dynamics (beyond the bounds used to construct the robust policy class).

Robust MBP is optimized using gradient descent for 1,000 updates, where each update samples 20 roll-outs. Robust PPO is trained for 50,000 updates, where each update samples 8 roll-outs;

we choose the model that performs best on a hold-out set of initial conditions during training. We note that while we use PPO for our demonstration, our approach is agnostic to the particular method of training, and can be deployed with many different (deep) RL paradigms.

We compare our robust neural network-based method against the following baselines:

- **Robust LQR:** Robust (linear) LQR controller obtained via Equation 3.6.
- **Robust MPC:** A robust model-predictive control algorithm [65] based on state-dependent LMIs. (As the relevant LMIs are not always guaranteed to solve, our implementation temporarily reverts to the Robust LQR policy when that occurs.)
- **RARL:** The robust adversarial reinforcement learning algorithm [92], which trains an RL agent in the presence of an adversary. (We note that unlike the other robust methods considered here, this method is not *provably* robust.)
- **LQR:** A standard non-robust (linear) LQR controller.
- **MBP** and **PPO:** The non-robust neural network policy class $\hat{\pi}_\theta(x)$ optimized via a model-based planner and the PPO algorithm, respectively.

In order to evaluate performance, we *train* all methods on the dynamical settings described in Section 3.6.1, and *evaluate* them on two different variations of the dynamics:

- **Original dynamics:** The dynamical settings described above ("average case").
- **Adversarial dynamics:** Modified dynamics with an adversarial test-time disturbance $w(t)$ generated to maximize loss ("worst case"). We generate this disturbance separately for each method described above (see Section 3.6.3 for more details).

Initialization states are randomly generated for all experiments. For the synthetic NLDI and microgrid settings, these are generated from a standard normal distribution. For both cart-pole and quadrotor, because our NLDI bounds model linearization error, we must generate initial points within a region where this linearization holds. In particular, the linearization bounds only hold for a specified L_∞ ball, B_{NLDI}, around the equilibrium. We use a simple heuristic to construct this ball and jointly find a smaller L_∞ ball, B_{init}, such that there exists a level set L of the Robust LQR Lyapunov function with $B_{\text{init}} \subseteq L \subseteq B_{\text{NLDI}}$. Since Robust LQR (and by extension our methods) are guaranteed to decrease the relevant Lyapunov function, this guarantees that these methods will never leave B_{NLDI} when initialized starting from any point inside B_{init} – i.e., that our NLDI bounds will always hold throughout the trajectories produced by these methods.

Initial states. To pick initial states in our experiments, for the synthetic settings, we sample each attribute of the state i.i.d. from a standard Gaussian distribution. For cart-pole and planar quadrotor, we sample uniformly from bounds chosen such that the non-robust LQR algorithm (under the original dynamics) did not go unstable. For cart-pole, these bounds were chosen to be $p_x \in [-1, 1]$, $\varphi \in [-0.1, 0.1]$, $\dot{p}_x = \dot{\varphi} = 0$. For planar quadrotor, these bounds were $p_x, p_z \in [-1, 1]$, $\varphi \in [-0.05, 0.05]$, $\dot{p_x} = \dot{p_z} = \dot{\varphi} = 0$.

Constructing NLDI bounds. Given these initial states, for the cart-pole and quadrotor settings, we needed to construct our NLDI disturbance bounds such that they would hold over the *entire*

trajectory of the robust policy; if not, the robustness specification equation 3.14 would not hold, and our agent might in fact increase the Lyapunov function. To ensure this approximately, we used a simple heuristic: we ran the (non-robust) LQR agent for a full episode with 50 different starting conditions, and constructed an L_∞ ball around all states reached in any of these trajectories. We then used these L_∞ balls on the states to construct the matrices C and D for our disturbance bounds, using the procedure described in Sections 3.6.4 and 3.6.5.

3.6.3 Generating an adversarial disturbance

In the NLDI settings explored in our experiments, we seek to construct an "adversarial" disturbance $w(t)$ that obeys the relevant norm bounds $\|w(t)\|_2 \leq \|Cx(t) + Du(t)\|_2$ while maximizing the loss. To do this, we use a model predictive control method where the actions taken are $w(t)$. Specifically, for each policy π, we model $w(t)$ as a neural network specific to that policy. Every 10 steps of a roll-out, we optimize $w(t)$ through gradient descent to maximize the loss over a horizon of 40 steps, subject to the constraint $\|w(t)\|_2 \leq \|Cx(t) + Du(t)\|_2$.

Trajectory plots. Figure 3.3 shows sample trajectories of different methods in the cart-pole domain under adversarial dynamics. The non-robust LQR and model-based planning approaches both diverge and the non-robust PPO doesn't diverge, but doesn't clearly converge after 10 seconds. The robust methods, on the other hand, all clearly converge after 10 seconds.

Runtime comparison. Tables 3.2 and 3.3 show the evaluation and training time of our methods and the baselines over 50 episodes run in parallel. In the NLDI cases where $D = 0$, i.e., Generic NLDI ($D = 0$) and Quadrotor, our projection adds only a very small computational cost. In the other cases, the additional computational cost is more significant, but our method is still far less expensive than the Robust MPC method.

3.6.4 Writing the cart-pole problem as an NLDI

In the cart-pole task, our goal is to balance an inverted pendulum resting on top of a cart by exerting horizontal forces on the cart. Specifically, the state of this system is defined as $x = \left[p_x, \dot{p_x}, \varphi, \dot{\varphi}\right]^T$, where p_x is the cart position and φ is the angular displacement of the pendulum from its vertical position; we seek to stabilize the system at $x = \vec{0}$ by exerting horizontal forces $u \in \mathbb{R}$ on the cart. For a pendulum of length ℓ and mass m_p, and for a cart of mass m_c, the dynamics of the system are (as described in [112]):

$$\dot{x} = \begin{bmatrix} \dot{p_x} \\ \frac{u + m_p \sin\varphi(\ell\dot{\varphi}^2 - g\cos\varphi)}{m_c + m_p \sin^2\varphi} \\ \dot{\varphi} \\ \frac{(m_c+m_p)g\sin\varphi - u\cos\varphi - m_p\ell\dot{\varphi}^2\cos\varphi\sin\varphi}{l(m_c + m_p\sin^2\varphi)} \end{bmatrix}, \tag{3.51}$$

where $g = 9.81\ \mathrm{m/s^2}$ is the acceleration due to gravity. We rewrite this system as an NLDI by defining $\dot{x} = f(x,u)$ and then linearizing the system about its equilibrium point as

$$\dot{x} = \mathrm{J}_f(0,0)\begin{bmatrix} x \\ u \end{bmatrix} + I_n w, \quad \|w\| \leq \|Cx + Du\|, \tag{3.52}$$

where J_f is the Jacobian of the dynamics, $w = f(x,u) - \mathrm{J}_f(0,0)\begin{bmatrix} x & u \end{bmatrix}^T$ is the linearization error, and I_n is the $n \times n$ identity matrix. We bound this linearization error by numerically obtaining the matrices C and D, assuming that x and u are within a neighborhood of the origin. We describe this process in more detail below. As a note, while we employ an NLDI here to characterize the linearization error, it is also possible to characterize this error via polytopic uncertainty (see Section 3.6.7); we choose to use an NLDI here as it yields a much smaller problem description than a PLDI in this case.

Deriving $\mathrm{J}_f(0,0)$ For $\dot{x} = f(x,u)$, we see that

$$\mathrm{J}_f(x,u) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \partial\ddot{p_x}/\partial\varphi & \partial\ddot{p_x}/\partial\dot{\varphi} & \partial\ddot{p_x}/\partial u \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \partial\ddot{\varphi}/\partial\varphi & \partial\ddot{\varphi}/\partial\dot{\varphi} & \partial\ddot{\varphi}/\partial u, \end{bmatrix}, \tag{3.53}$$

where

$$\frac{\partial\ddot{p_x}}{\partial\varphi} = \frac{m_p\cos\varphi\left(\dot{\varphi}^2 l - g\cos\varphi\right) + g m_p \sin^2\varphi}{m_c + m_p\sin^2\varphi} - \frac{2m_p\sin\varphi\cos\varphi\left(m_p\sin\varphi\left(\dot{\varphi}^2 l - g\cos\varphi\right) + u\right)}{\left(m_c + m_p\sin^2\varphi\right)^2},$$

$$\frac{\partial\ddot{p_x}}{\partial\dot{\varphi}} = \frac{2\dot{\varphi} l m_p \sin\varphi}{m_c + m_p\sin^2\varphi},$$

$$\frac{\partial\ddot{p_x}}{\partial u} = \frac{1}{m_c + m_p\sin^2\varphi},$$

$$\frac{\partial\ddot{\varphi}}{\partial\varphi} = \frac{\begin{matrix} g(m_c+m_p)\cos\varphi + \dot{\varphi}^2 l m_p \sin^2\varphi \\ -\dot{\varphi}^2 l m_p\cos^2\varphi + u\sin\varphi \end{matrix}}{l\left(m_c + m_p\sin^2\varphi\right)} - \frac{\begin{matrix} 2m_p\sin\varphi\cos\varphi(g(m_c+m_p)\sin\varphi \\ -\dot{\varphi}^2 l m_p\sin\varphi\cos\varphi - u\cos\varphi \end{matrix}}{l\left(m_c + m_p\sin^2\varphi\right)^2},$$

$$\frac{\partial\ddot{\varphi}}{\partial\dot{\varphi}} = \frac{-2\dot{\varphi} m_p\sin\varphi\cos\varphi}{m_c + m_p\sin^2\varphi},$$

$$\frac{\partial\ddot{\varphi}}{\partial u} = \frac{-\cos\varphi}{l(m_c + m_p\sin^2\varphi)}.$$

We thus see that

$$\mathrm{J}_f(0,0) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -m_p g/m_c & 0 & 1/m_c \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & g(m_c+m_p)/l m_c & 0 & -1/m_c \end{bmatrix}. \tag{3.54}$$

Obtaining C and D We then seek to construct matrices C and D that bound the linearization error w between the true dynamics $\dot{x}$ and our first-order linear approximation $\mathrm{J}_f(0,0)\begin{bmatrix}x\\u\end{bmatrix}$. To do so, we bound the error of this approximation entry-wise: that is, for each entry $i = 1, \ldots, s$, we want to find F_i such that for all x in some region $\underline{x} \leq x \leq \bar{x}$, and all u in some region $\underline{u} \leq u \leq \bar{u}$,

$$w_i^2 = \left(\nabla f_i(0)\begin{bmatrix}x\\u\end{bmatrix} - \dot{x}_i\right)^2 \leq \begin{bmatrix}x\\u\end{bmatrix}^T F_i \begin{bmatrix}x\\u\end{bmatrix}. \tag{3.55}$$

Then, given the matrix

$$M = \begin{bmatrix}F_1^{T/2} & F_2^{T/2} & F_3^{T/2} & F_4^{T/2} & F_5^{T/2} & F_6^{T/2}\end{bmatrix}^T \tag{3.56}$$

we can then obtain $C = M_{1:s}$ and $D = M_{s:s+m}$ (where the subscripts indicate column-wise indexing).

We solve separately for each F_i to minimize the difference between the right and left sides of Equation 3.55 (while enforcing that the right side is larger than the left side) over a discrete grid of points within $\underline{x} \leq x \leq \bar{x}$ and $\underline{u} \leq u \leq \bar{u}$. By assuming that F_i is symmetric, we are able to cast this as a linear program in the upper triangular entries of F_i.

To obtain the matrices C and D used for the cart-pole experiments in the main paper, we let $\bar{x} = \begin{bmatrix}1.5 & 2 & 0.2 & 1.5\end{bmatrix}^T$, $\bar{u} = 10$, $\underline{x} = -\bar{x}$, and $\underline{u} = -\bar{u}$. As each entry-wise difference in Equation 3.55 contained exactly three variables (i.e., a total of three entries from x and u), we solved each entry-wise linear program over a mesh grid of 50 points per variable.

3.6.5 Writing quadrotor as an NLDI

In the planar quadrotor setting, our goal is to stabilize a quadcopter in the two-dimensional plane by controlling the amount of force provided by the quadcopter's right and left thrusters. Specifically, the state of this system is defined as $x = \begin{bmatrix}p_x & p_z & \varphi & \dot{p_x} & \dot{p_z} & \dot{\varphi}\end{bmatrix}^T$, where (p_x, p_z) is the position of the quadcopter in the vertical plane and φ is its roll (i.e., angle from the horizontal position); we seek to stabilize the system at $x = \vec{0}$ by controlling the amount of force $u = \begin{bmatrix}u_r, u_l\end{bmatrix}^T$ from right and left thrusters. We assume that our action u is additional to a baseline force of $\begin{bmatrix}mg/2 & mg/2\end{bmatrix}^T$ provided by the thrusters by default to prevent the quadcopter from falling. For a quadrotor with mass m, moment-arm ℓ for the thrusters, and moment of inertia J about the roll axis, the dynamics of this system are then given by (as modified from [98]):

$$\dot{x} = \begin{bmatrix}\dot{p_x}\cos\varphi - \dot{p_z}\sin\varphi\\ \dot{p_x}\sin\varphi + \dot{p_z}\cos\varphi\\ \dot{\varphi}\\ \dot{p_z}\dot{\varphi} - g\sin\varphi\\ -\dot{p_x}\dot{\varphi} - g\cos\varphi + g\\ 0\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 1/m & 1/m\\ \ell/J & -\ell/J\end{bmatrix} u, \tag{3.57}$$

where $g = 9.81\ \mathrm{m/s^2}$. We linearize this system via a similar method as for the cart-pole setting, i.e., as in Equation 3.52. We describe this process in more detail below. We note that since the dependence of the dynamics on u is linear, we have that $D = 0$ for our resultant NLDI. As for cart-pole, while we employ an NLDI here to characterize the linearization error, it is also possible to characterize this error via polytopic uncertainty (see Section 3.6.7); we choose to use an NLDI here as it yields a much smaller problem description than a PLDI in this case.

Deriving $\mathrm{J}_f(0,0)$ For $\dot{x} = f(x,u)$, we see that

$$\mathrm{J}_f(x,u) = \begin{bmatrix} 0 & 0 & -\dot{p}_x \sin\varphi - \dot{p}_z \cos\varphi & \cos\varphi & -\sin\varphi & 0 & 0 \\ 0 & 0 & \dot{p}_x \cos\varphi - \dot{p}_z \sin\varphi & \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -g\cos\varphi & 0 & \dot{\varphi} & \dot{p}_z & 0 \\ 0 & 0 & g\sin\varphi & -\dot{\varphi} & 0 & -\dot{p}_x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.58}$$

and thus

$$\mathrm{J}_f(0,0) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.59}$$

Obtaining C **and** D We obtain the matrices C and D via a similar method as described in Section 3.6.4, though in practice we only consider the linearization error with respect to x (i.e., since the dynamics are linear with respect to u, we have $D = 0$). We let $\bar{x} = \begin{bmatrix} 1 & 1 & 0.15 & 0.6 & 0.6 & 1.3 \end{bmatrix}$ and $\underline{x} = -\bar{x}$. As for cart-pole, each entry wise difference in the equivalent of Equation 3.55 contained exactly three variables (i.e., a total of three entries from x and u), and each entry-wise linear program was solved over a mesh grid of 50 points per variable.

3.6.6 Details on the microgrid setting

For our experiments, we build upon the microgrid setting given in [72]. In this system, the state $x \in \mathbb{R}^3$ captures voltage deviations, frequency deviations, and the amount of power generated by a diesel generator connected to the grid; the action $u \in \mathbb{R}^2$ describes the current associated with a storage device and a solar PV inverter; and the disturbance $w \in \mathbb{R}$ describes the difference between the amount of power demanded and the amount of power produced by solar panels on the grid. The authors also define a performance index $y \in \mathbb{R}^2$ which captures voltage and frequency deviations (i.e., two of the entries of the state x).

To construct an NLDI of the form equation 3.3 for this system, we directly use the A, B, and G matrices given in [72]. We generate C i.i.d. from a normal distribution and let $D = 0$, to represent the fact that the disturbance w and the entries of the state x are correlated, but that w is likely not correlated with the actions u. Finally, we let Q and R be diagonal matrices with 1 in the entries

corresponding to quantities represented in the performance index y, and with 0.1 in the rest of the diagonal entries, to emphasize that the variables in y are the most important in describing the performance of the system.

3.6.7 Notes on linearization via PLDIs and NLDIs

While we linearize the cart-pole and quadrotor dynamics via NLDIs in our experiments, we note that these dynamics can also be characterized via PLDIs. More generally, in this section, we show how we can use the framework of PLDIs to model linearization errors arising in the analysis of nonlinear systems.

Consider the nonlinear dynamical system

$$\dot{x} = f(x, u) \text{ with } f(0, 0) = 0. \tag{3.60}$$

for $x \in \mathbb{R}^s$ and $u \in \mathbb{R}^a$. Define $\xi = (x, u)$. We would like to represent the above system as a PLDI in the region $\mathcal{R} := \{\xi \,|\, \underline{\xi} \leq \xi \leq \bar{\xi}\}$ including the origin. The mean value theorem states that for each component of f, we can write

$$f_i(\xi) = f_i(0) + \nabla f_i(z)^T \xi, \tag{3.61}$$

for some $z = t\xi$, where $t \in [0, 1]$. Now, let $p = s + a$. Defining the Jacobian of f as

$$\mathrm{J}_f(z) = \begin{bmatrix} \nabla f_1(z)^T \\ \vdots \\ \nabla f_p(z)^T \end{bmatrix}, \tag{3.62}$$

and recalling that $f(0) = 0$, we can rewrite equation 3.61 as

$$f(\xi) = \mathrm{J}_f(z)\xi. \tag{3.63}$$

Now, suppose we can find component-wise bounds on the matrix $\mathrm{J}_f(z)$ over $\mathcal{R}$, i.e,

$$\underline{M} \leq \mathrm{J}_f(z) \leq \bar{M} \text{ for all } z \in \mathcal{R}. \tag{3.64}$$

We can then write

$$\mathrm{J}_f(z) = \sum_{1 \leq i,j \leq p} m_{ij}(t) E_{ij} \quad \text{with} \quad m_{ij}(t) \in [\underline{m}_{ij}, \bar{m}_{ij}], \tag{3.65}$$

where $E_{ij} = e_i e_j^T$ and e_i is the i-th unit vector in $\mathbb{R}^p$.

We now seek to bound the Jacobian using polytopic bounds. To do this, note that we can write

$$\mathrm{J}_f(z) = \sum_{\kappa=1}^{2^{p^2}} \gamma_\kappa A_\kappa \quad \gamma_\kappa \geq 0, \; \sum_\kappa \gamma_\kappa = 1, \tag{3.66}$$

where A_κ's are the vertices of the polytope in equation 3.65, i.e.,

$$A_\kappa \in \mathcal{V} = \left\{ \sum_{1 \le i,j \le p} m_{ij} E_{ij} \,|\, m_{ij} \in \{\underline{m}_{ij}, \bar{m}_{ij}\} \right\}. \tag{3.67}$$

Together, Equations equation 3.61, equation 3.63, equation 3.66, and equation 3.67 characterize the original nonlinear dynamics as a PLDI.

We note that this PLDI description is potentially very large; in particular, the size of $\mathcal{V}$ is exponential in the square of the number of non-constant entries in the Jacobian $\mathrm{J}_f(z)$, which could be as large as $2^{p^2} = 2^{(s+a)^2}$. This problem size may therefore become intractable for larger control settings.

We note, however, that we can in fact express this PLDI more concisely as an NLDI. More precisely, we would like to find matrices A, B, C parameterizing the form of NLDI below, which is equivalent to that presented in Equation 3.3 (see Chapter 4 of [21]):

$$Df(z) \in \{A + B\Delta C \,|\, \|\Delta\|_2 \le 1\} \quad \text{for all } z \in \mathcal{R}. \tag{3.68}$$

It can shown that the solution to the SDP

$$\begin{aligned} \text{minimize} \;\; & \mathrm{tr}(V + W) \\ \text{subject to} \;\; & W \succ 0 \\ & \begin{bmatrix} V & (A_\kappa - A)^T \\ A_\kappa - A & W \end{bmatrix} \succeq 0, \; \forall A_\kappa \in \mathcal{V} \end{aligned} \tag{3.69}$$

yields the matrices A, B, and C with $V = C^T C$ and $W = BB^T$, which can be used to construct NLDI equation 3.68. While the NLDI here is more concise than the PLDI, the trade-off is that the NLDI norm bounds obtained via this method may be rather loose. As such, for our settings, we obtain NLDI bounds numerically (see Section 3.6.4 and Section 3.6.5), as these are tighter than NLDI specifications obtained via the above method (though they are potentially slightly inexact). An alternative approach would be to examine how to tighten the conversion from PLDIs to NLDIs, which has been explored in other work (e.g. [66]).

3.6.8 Additional experimental details

Computing infrastructure and runtime. All experiments were run on an XPS 13 laptop with an Intel i7 processor. The planar quadrotor and synthetic NLDI experiment with $D = 0$ took about 1 day to run (since the projections were simple half-space projections), while all the other synthetic domains and cart-pole took about 3 days to run. The majority of the run-time was in computing the adversarial disturbances for test-time evaluations.

Hyperparameter selection. For our experiments, we did not perform large parameter searches. The learning rate we chose for our model-based planner, (both robust and non-robust) remained constant for the different domains; we tried learning rates of $1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}$ and

found 1×10^{-3} worked best for the non-robust version and 1×10^{-4} worked best for the robust version. For our PPO hyperparameters, we simply used those used in the original PPO paper.

One parameter we had to tune for each environment was the time step. In particular, we had to pick a time step high enough that we could run episodes for a reasonable total length of time (within which the non-robust agents would go unstable), but low enough to reasonably approximate a continuous-time setting (since, for our robustness guarantees, we assume the agent's actions evolve in continuous time). Our search space was small, however, consisting of 0.05, 0.02, 0.01, and 0.005 seconds.

3.6.9 Results

Table 3.1 shows the performance of the above methods. We report the integral of the quadratic loss over the prescribed time horizon on a test set of states, or indicate cases where the relevant method became unstable (i.e., the loss became orders of magnitude larger than for other approaches).

These results illustrate the basic advantage of our approach. In particular, both our Robust MBP and Robust PPO methods show **improved "average-case" performance over the other provably robust methods** (namely, Robust LQR and Robust MPC). As expected, however, the non-robust LQR, MBP, and PPO methods often perform better within the original nominal dynamics, as they are optimizing for expected performance but do not need to consider robustness under worst-case scenarios. However, when we apply allowable adversarial perturbations (that still respect our disturbance bounds), the non-robust LQR, MBP, and PPO approaches diverge or perform very poorly. Similarly, the RARL agent performs well under the original dynamics, but diverges under adversarial perturbations in the generic NLDI settings. In contrast, both of our provably robust approaches (as well as Robust LQR) **remain stable under even "worst-case" adversarial dynamics**. (We note that the baseline Robust MPC method goes unstable in one instance, though this is due to numerical instability issues, rather than issues with theoretical guarantees.)

Figure 3.1 additionally shows the performance of all neural network-based methods on the test set over training epochs. While the robust and non-robust MBP and PPO approaches both converge quickly to their final performance levels, both non-robust versions become unstable under the adversarial dynamics very early in the process. The RARL method also frequently destabilizes during training. Our Robust MBP and PPO policies, on the other hand, **remain stable throughout the *entire* optimization process**, i.e., do not destabilize during either training *or* testing. Overall, these results show that our method is able to learn policies that are more expressive than traditional robust methods, while *guaranteeing* these policies will be stable under the same conditions as Robust LQR.

3.7 Conclusion

In this chapter, we have presented a class of nonlinear control policies that combines the expressiveness of neural networks with the provable stability guarantees of traditional robust control. This policy class entails projecting the output of a neural network onto a set of stabilizing actions,

Environment		LQR	MBP	PPO	Robust LQR	Robust MPC	RARL	Robust MBP*	Robust PPO*
Generic NLDI ($D = 0$)	O	373	**16**	21	253	253	**27**	69	33
	A	——— *unstable* ———			1009	873	*unstable*	1111	2321
Generic NLDI ($D \neq 0$)	O	278	**15**	82	199	199	147	**69**	80
	A	——— *unstable* ———			1900	1667	*unstable*	1855	1669
Generic PLDI	O	96.3	**3.3**	8.0	19.2	19.2	15.8	18.6	**10.2**
	A	——— *unstable* ———			43.3	44.1	*unstable*	21.9	16.1
Generic H_∞	O	181	**88**	114	165	N/A	**115**	116	125
	A	219	112	143	206	N/A	145	147	158
Cart-pole	O	36.3	**3.6**	7.2	10.2	10.2	**8.3**	9.7	8.4
	A	— *unstable* —		172.1	42.2	47.8	41.2	50.0	16.3
Quadrotor	O	5.4	**2.5**	7.7	13.8	13.8	12.2	11.0	**8.3**
	A	*unstable*	545.7	137.6	64.8	*unstable*†	63.1	25.7	26.5
Microgrid	O	4.59	**0.60**	0.61	0.73	0.73	0.67	**0.61**	**0.61**
	A	——— *unstable* ———			0.99	0.92	2.17	7.68	8.91

Table 3.1: Performance of various approaches, both robust (right) and non-robust (left). We report average quadratic loss over 50 episodes under the original dynamics (O) and under an adversarial disturbance (A). For the original dynamics (O), the best performance for both non-robust methods and robust methods is in bold (lower loss is better). Under the adversarial dynamics (A), we seek to observe whether or not methods remain stable; we use "*unstable*" to indicate cases where the relevant method becomes unstable (and † to denote any instabilities due to numerical, rather than theoretical, issues). Our robust methods (denoted by *) improve performance over Robust LQR and Robust MPC in the average case while remaining stable under adversarial dynamics, whereas the non-robust methods and RARL either go unstable or receive much larger losses.

Environment	LQR	MBP	PPO	Robust LQR	Robust MPC	RARL	Robust MBP*	Robust PPO*
Generic NLDI ($D = 0$)	0.63	0.61	0.84	0.57	718.06	0.71	0.73	0.94
Generic NLDI ($D \neq 0$)	0.64	0.62	0.83	0.58	824.86	0.81	15.13	25.38
Cart-pole	0.55	0.67	0.84	0.53	646.90	0.84	10.12	13.37
Quadrotor	0.95	0.98	1.19	0.88	3348.68	1.14	1.15	1.30
Microgrid	0.58	0.61	0.79	0.57	601.90	0.74	8.14	10.25
Generic PLDI	0.57	0.54	0.76	0.51	819.24	0.73	69.35	64.03
Generic H_∞	0.84	0.80	1.03	0.76	N/A	1.00	47.81	63.67

Table 3.2: Time (in seconds) taken to run each method on the test set of every environment for 50 episodes run in parallel.

Environment	MBP	PPO	RARL	Robust MBP*	Robust PPO*
Generic NLDI ($D = 0$)	26.36	101.77	102.37	30.78	114.60
Generic NLDI ($D \neq 0$)	26.46	100.79	82.53	221.35	1158.28
Cart-pole	25.49	87.04	98.90	146.34	689.93
Quadrotor	41.24	131.48	112.95	46.13	159.06
Microgrid	23.03	112.52	87.71	113.61	436.64

Table 3.3: Time (in minutes) taken to train each method in every environment.

parameterized via robustness specifications from the robust control literature, and can be optimized using a model-based planning algorithm if the dynamics are known or virtually any RL algorithm if the dynamics are unknown. We instantiate our general framework for dynamical systems characterized by several classes of linear differential inclusions that capture many common robust control settings. In particular, this entails deriving efficient, differentiable projections for each setting, via implicit differentiation techniques. We show over a variety of simulated domains that our method improves upon traditional robust LQR techniques while, unlike non-robust LQR and neural network methods, remaining stable even under worst-case allowable perturbations of the underlying dynamics.

We believe that our approach highlights the possible connections between traditional control methods and (deep) RL methods. Specifically, by enforcing more structure in the classes of deep networks we consider, it is possible to produce networks that *provably* satisfy many of the constraints that have typically been thought of as outside the realm of RL. We hope that this work paves the way for future approaches that can combine more structured uncertainty or robustness guarantees with RL, in order to improve performance in settings traditionally dominated by classical robust control.

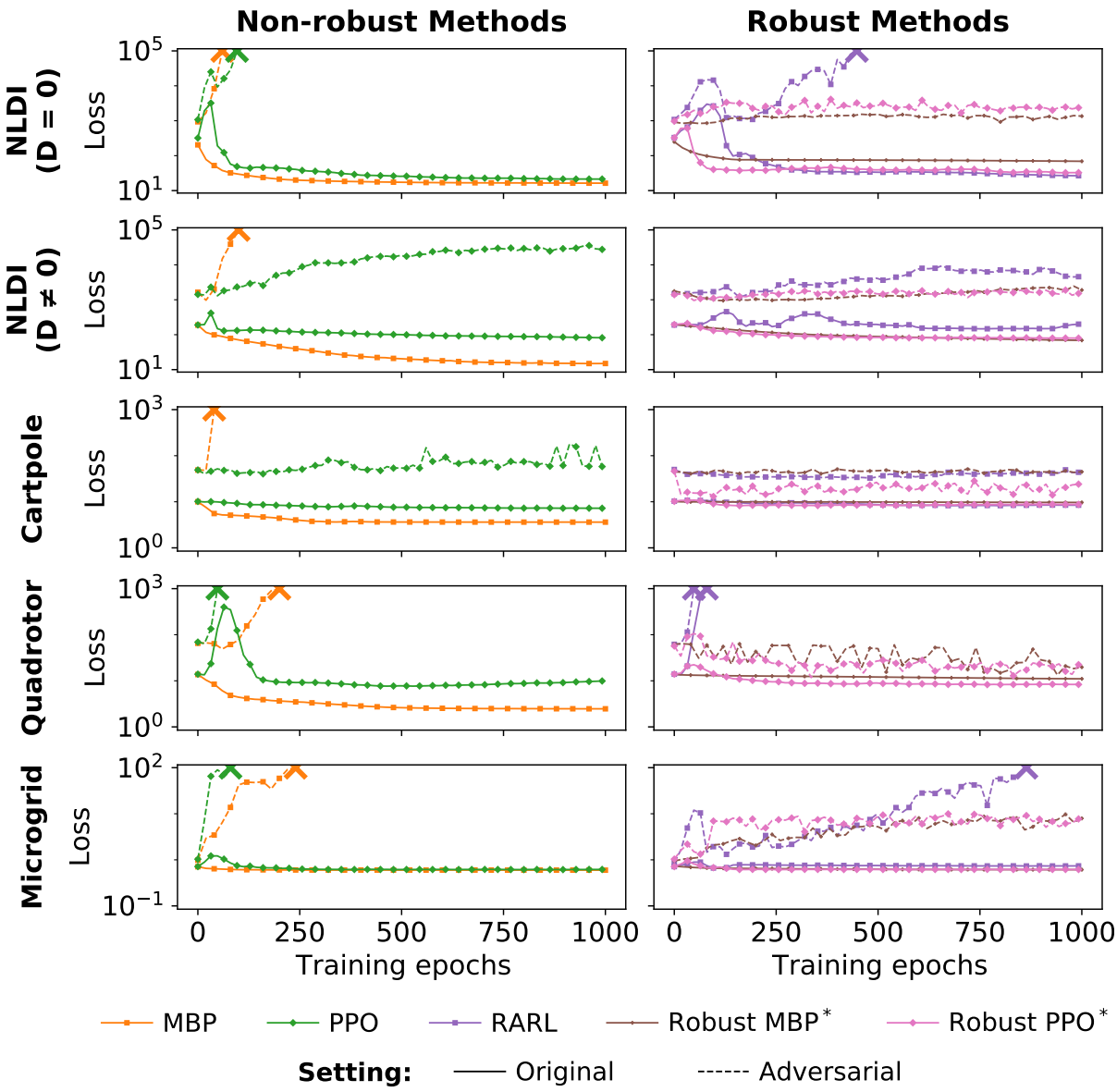


Figure 3.1: Test performance over training epochs for all learning methods employed in our experiments. For each training epoch (10 updates for the MBP model and 18 for PPO), we report average quadratic loss over 50 episodes, and use "X" to indicate cases where the relevant method became unstable. (Lower loss is better.) Our robust methods (denoted by *), unlike the non-robust methods and RARL, remain stable under adversarial dynamics throughout training.

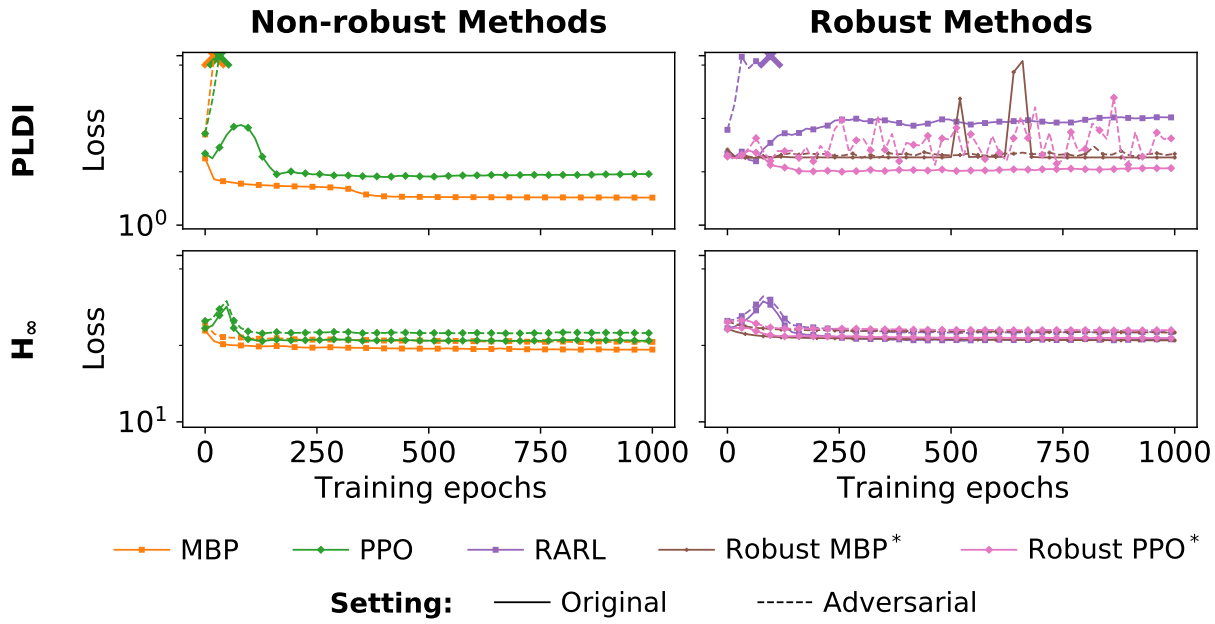
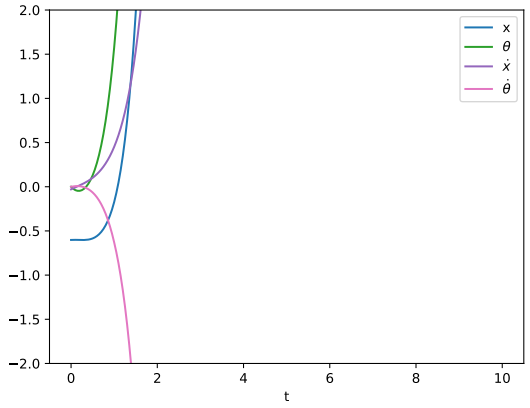
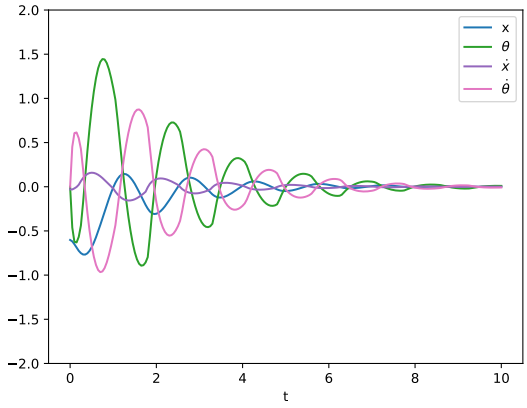


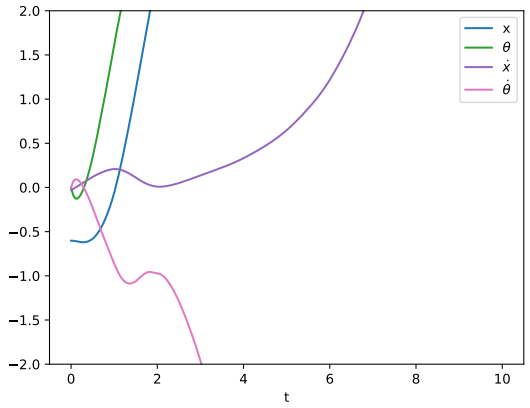
Figure 3.2: Representative results for our experimental settings. For each training epoch (10 updates for the MBP model and 18 for PPO), we report average quadratic loss over 50 episodes, and use "X" to indicate cases where the relevant method became unstable. (Lower loss is better.) Our robust methods (denoted by *) improve performance over Robust LQR in the average case, while (unlike the non-robust methods) remaining stable under adversarial dynamics throughout the training process.



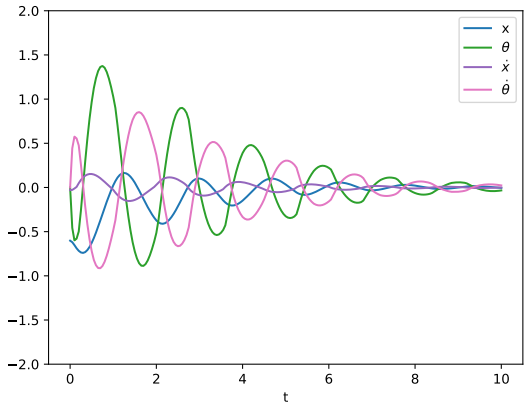
(a) LQR



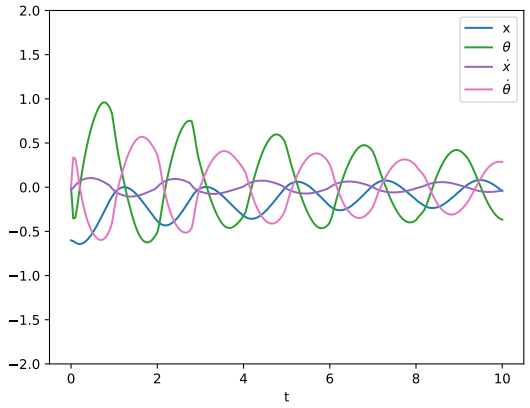
(b) Robust LQR



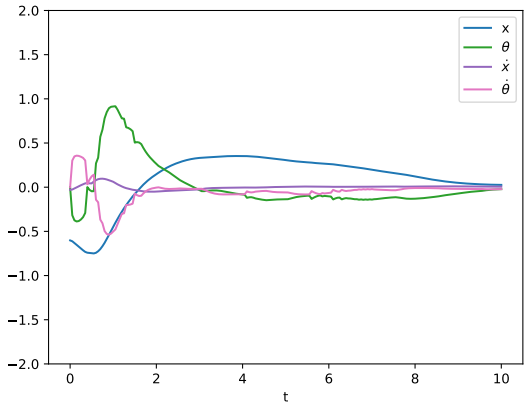
(c) MBP



(d) Robust MBP



(e) PPO



(f) Robust PPO

Figure 3.3: Trajectories of 6 different methods on the cart-pole domain under adversarial dynamics.

Chapter 4

Provably Safe PAC-MDP Exploration Using Analogies

The previous chapter discussed how we can provide robustness guarantees on policies learned through RL. However, this method assumes we have enough prior knowledge to stabalize the control system. In domains where we do not have this prior knowledge, we need to explore the state-action space in order to aquire the requisite knowledge. But how do we do so on safety-critical systems?

In this chapter, we present Analogous Safe-state Exploration (ASE), a new method for performing provably and optimal safe exploration on safety-critical domains with stochastic dynamics. To the best of our knowledge this is the first provably safe and optimal exploration algorithm that is able to be applied on system with stochastic dynamics. While requiring significant additional assumptions, we argue all assumptions are needed to guarantee this strong notion of safety. In our experiments, ASE maintains safety throughout the training process and significantly improves sample efficiency over other safe exploration techniques.

4.1 Introduction

Imagine you are Phillipe Petit in 1974, about to make a tight-rope walk between two thousand-foot-tall buildings. There is no room for error. You would want to be certain that you could successfully walk across without falling. And to do so, you would naturally want to practice walking a tightrope on a similar length of wire, but only a few feet off the ground, where there is no real danger.

This example illustrates one of the key challenges in applying reinforcement learning (RL) to safety-critical domains, such as autonomous driving or healthcare, where a single mistake could cause significant harm or even death. While RL algorithms have been able to significantly improve over human performance on some tasks in the average case, most of these algorithms do not provide any guarantees of safety either during or after training, making them too risky to be used in real-world, safety-critical domains.

True MDP	$M = \langle S, A, T, R, \gamma \rangle$
True safe, optimal goal policy	π^*_{safe}
Empirical transition probability	$\hat{T}$
Empirical $L1$ confidence interval width	$\hat{\epsilon}_T$
Arbitrary MDP	$M^\dagger = \langle S, A, T^\dagger, R^\dagger, \gamma^\dagger \rangle$
Optimal Q-function on the optimistic MDP $\overline{M}_{\text{goal}}$	$\overline{Q}_{\text{goal}}$
Optimal Q-function on the optimistic MDP $\overline{M}_{\text{explore}}$	$\overline{Q}_{\text{explore}}$
Optimal Q-function on the optimistic MDP $\overline{M}_{\text{switch}}$	$\overline{Q}_{\text{switch}}$
Optimistic goal policy (defined by $\overline{Q}_{\text{goal}}$)	$\overline{\pi}_{\text{goal}}$
Optimistic explore policy (defined by $\overline{Q}_{\text{goal}}$)	$\overline{\pi}_{\text{explore}}$
Optimistic switching policy (defined by $\overline{Q}_{\text{switch}}$)	$\overline{\pi}_{\text{switch}}$
Analogy-based empirical probabilities and confidence intervals	$\overset{\Delta}{T}, \overset{\Delta}{\epsilon}_T$
Optimal value function for some MDP $M^\dagger$	$V^*_{M^\dagger}$

Table 4.1: Here we provide a useful reference for some of the notation used throughout the proofs.

Taking inspiration from the tight-rope example, we propose a new approach to safe exploration in reinforcement learning. Our approach, Analogous Safe-state Exploration (ASE), seeks to explore state-action pairs that are analogous to those along the path to the goal, but are guaranteed to be safe. Our work fits broadly into the context of a great deal of recent work in safe reinforcement learning, but compared with past work, our approach is novel in that 1) it guarantees safety during exploration in a stochastic, unknown environment (with high probability), 2) it finds a near-optimal policy in a PAC-MDP (Probably Approximately Correct-MDP) sense, and 3) it guides exploration to focus only on state-action pairs that provide necessary information for learning the optimal policy. Specifically, in our setting we assume our agent has access to a set of initial state-action pairs that are guaranteed to be safe and a function that indicates the similarity between state-action pairs. Our agent constructs an optimistic policy, following this policy only when it can establish that this policy won't lead to a dangerous state-action pair. Otherwise, the agent explores state-action pairs that inform the safety of the optimistic path.

In conjunction with proposing this new approach, we make two main contributions. First, we prove that ASE guarantees PAC-MDP optimality, and also safety of the *entire* training trajectory, with high probability. To the best of our knowledge, this is the first algorithm with this two-fold guarantee in stochastic environments. Second, we evaluate ASE on two illustrative MDPs, and show empirically that our proposed approach substantially improves upon existing PAC-MDP methods, either in safety or sample efficiency, as well as existing methods modified to guarantee safety.

4.2 Problem Setup

We model the environment as a Markov Decision Process (MDP), a 5-tuple $\langle S, A, R, T, \gamma \rangle$ with discrete, finite sets of states S and actions A, a *known, deterministic* reward function $R : S \times A \to \mathbb{R}$, an *unknown, stochastic* dynamics function $T : S \times A \to \mathcal{P}_S$ which maps a

state-action pair to a probability distribution over next states, and discount factor $\gamma \in (0, 1)$. We assume the environment has a fixed initial state and denote it by s_{init}. We also assume that the rewards are known a-priori and bounded between -1 and 1; the rewards that are negative denote dangerous state-actions.[1] This of course means that the agent knows a priori what state-actions are "immediately" dangerous; but we must emphasize that the agent is still faced with the non-trivial challenge of learning about other a priori unknown state-actions that can be dangerous in the long-term – we elaborate on this in the "Safety" section below. Also note that while most RL literature does not assume the reward function is known, we think this is a reasonable assumption for many real-world problems where reward functions are constructed by engineers. Moreover, this assumption is not uncommon in RL theory [73, 107].

Analogies. As highlighted in Turchetta et al. [115], some prior knowledge about the environment is required for ensuring the agent never reaches a catastrophic state. In prior work, this knowledge is often provided as some notion of similarity between state-action pairs; for example, kernel functions used in previous work employing GPs to model dynamics or Lipschitz continuity assumptions placed on the dynamics. Intuitively, such notions of similarity can be exploited to learn about unknown (and potentially dangerous) state-action pairs, by exploring a "proxy" state-action pair that is sufficiently similar and known to be safe. Below, we define a notion of similarity, but the key difference between this and previous formulations is that ours also applies to stochastic environments. Specifically, we introduce the notation of *analogies* between state-action pairs. More concretely, the agent is given *an analogous state function* $\alpha : (S \times A \times S) \times (S \times A) \to S$ and a *pairwise state-action distance mapping* $\Delta : (S \times A) \times (S \times A) \to [0, 2]$ such that, for any $(s, a, \tilde{s}, \tilde{a}) \in (S \times A) \times (S \times A)$

$$\sum_{s' \in S} |T(s, a, s') - T(\tilde{s}, \tilde{a}, \alpha(\cdots, s'))| \leq \Delta((s, a), (\tilde{s}, \tilde{a}))$$

where $\alpha(\cdots, s') = \alpha(s, a, s', \tilde{s}, \tilde{a})$ represents, intuitively, the next state that is "equivalent" to s' for $(\tilde{s}, \tilde{a})$.[2] In other words, for any two state-action pairs, we are given a bound on the L_1 distance between their dynamics: one that is based on a mapping between analogous next states. The hope is that α can provide a much more useful analogy than a naive identity mapping between the respective next states.

To provide some intuition, recall the tight-rope walker example mentioned in the introduction: The tight-rope walker agent must cross a dangerous tight-rope, but wants to guarantee it can do so safely. In this situation, the agent has a "practice" tight-rope (that's only a few feet off the ground) and a "real" tight-rope. In this example, if the agent is in a particular position and takes a particular action on either tight-rope, the change in its position will be the same regardless of what rope it was on.[3] This analogy between the two ropes can be mathematically captured as follows. Consider representing the agent's state as a tuple of the form (prac, x) or (real, x) where the first element

[1] Note that our work can be easily extended to have a separate safety and reward function, but we have combined them in this work for convenience.

[2] Note that although α is defined for every $(s, a, s', \tilde{s}, \tilde{a})$ tuple, not all such pairs of state-actions need be analogous to each other. In such cases, we can imagine that $\alpha(s, a, s', \tilde{s}, \tilde{a})$ maps to a dummy state and $\Delta((s, a), (\tilde{s}, \tilde{a})) = 1$.

[3] For this example, we assume the dynamics of the agent on the practice and real tight-ropes are identical, but small differences could be captured by making the $\Delta(\cdot, \cdot, \cdot, \cdot)$ function non-zero.

denotes which of the two ropes the agent is on, and the second element denotes the position within that rope. Then for any action a, we can say that $\alpha((\text{prac}, x), a, (\text{prac}, x'), (\text{real}, x), a) = (\text{real}, x')$ and $\Delta(((\text{prac}, x), a), ((\text{real}, x), a) = 0$. This would thus imply that by learning a model of the dynamics in the practice setting, and the corresponding optimal policy, an agent would still be able to learn a policy that guaranteed safety even in the real setting.

State-action sets. For simplicity, for any set of state-action pairs $Z \subset S \times A$, we say that $s \in Z$ if there exists any $a \in A$ such that $(s, a) \in Z$. Also, we say that (s, a) is an **edge** of Z if $(s, a) \notin Z$ but $s \in Z$. We use $\mathrm{P}[\cdot|\pi]$ to denote the probability of an event occurring while following a policy π.

Definition 1. *We say that $Z \subset S \times A$ is **closed** if for every $(s, a) \in Z$ and for every next s' for which $T(s, a, s') > 0$, there exists a' such that $(s', a') \in Z$.*

Intuitively, if a set Z is closed, then we know that if the agent starts at a state in Z and follows a policy π such that for all $s \in Z$, $(s, \pi(s)) \in Z$, then we can guarantee that the agent never exits Z (see Fact 1). We will use $\pi \in \Pi(Z)$ to denote that, for all $s \in Z$, $(s, \pi(s)) \in Z$.

Definition 2. *A subset of state-action pairs, $Z \subset S \times A$ is said to be **communicating** if Z is closed and for any $s' \in Z$, there exists a policy $\pi_s \in \Pi(Z)$ such that $\forall s,\ \mathrm{P}[\exists t,\ s_t = s' \,|\pi_{s'}, s_0 = s] = 1$.*

In other words, every two states in Z must be reachable through a policy that never exits the subset Z. Note that this definition is equivalent to the standard definition of communicating when Z is the set of all state-action pairs in the MDP (see Fact 2).

Safe-PAC-MDP. One of the main objectives of this work is to design an agent that, with high probability (over all possible trajectories the agent takes), learns an optimal policy (in the PAC-MDP sense) while also never taking dangerous actions (i.e. actions with negative rewards) at any point along its arbitrarily long, trajectory. This is a very strong notion of safety, but critical for assuring safety for long trajectories. Such a strong notion is necessary in many real-world applications such as health and self-driving cars where a dangerous action spells complete catastrophe.

We formally state this notion of Safe-PAC-MDP below. The main difference between this definition and that of standard PAC-MDP is (a) the safety requirement on all timesteps and (b) instead of competing against an optimal policy (which could potentially be unsafe), the agent now competes with a "safe-optimal policy" (that we will define later). To state this formally, as in Strehl et al. [100], let the trajectory of the agent until time t be denoted by p_t and let the value of the algorithm $\mathcal{A}$ be denoted by $V^{\mathcal{A}}(p_t)$ – this equals the cumulative sum of rewards in expectation over all future trajectories (see Def 5).

Definition 3. *We say that an algorithm $\mathcal{A}$ is **Safe-PAC-MDP** if, for any $\epsilon, \delta \in (0, 1]$, with probability at least $1 - \delta$, $R(s_t, a_t) \geq 0$ for all timesteps t and additionally, the sample complexity of exploration i.e., the number of timesteps t for which $V^{\mathcal{A}}(p_t) > V^{\pi^*_{safe}}(p_t) - \epsilon$, is bounded by a polynomial in the relevant quantities, $(|S|, |A|, 1/\epsilon, 1/\delta, 1/(1-\gamma), 1/\tau, H_{com})$. Here, π^*_{safe} is the safe-optimal policy defined in Def. 8, τ is the minimum non-zero transition probability (see Assumption 4) and H_{com} is "communication time" (see Assumption 3).*

We must emphasize that this notion of safety must *not* be confused with the weaker notion where one simply guarantees safety with high probability *at every step* of the learning process. In such a case, for sufficiently long training trajectories, the agent is guaranteed to take a dangerous action i.e., with probability 1, the trajectory taken by the agent will lead it to a dangerous action as $t \to \infty$.

Safety. Since our agent is provided the reward mapping, the agent knows a priori which state-action pairs are "immediately" dangerous (namely, those with negative rewards). However, the agent is still faced with the challenge of determining which actions may be dangerous *in the long run*: an action may momentarily yield a non-negative reward, but by taking that action, the agent may be doomed to a next state (or a future state) where all possible actions have negative rewards. For example, at the instant when a tight-rope walker loses balance, they may experience a zero reward, only to eventually fall down and receive a negative reward. In order to avoid such "delayed danger", below we define a natural notion of a safe set: a closed set of non-negative reward state-action pairs; as long as the agent takes actions within such a safe set, it will never find itself in a position where its only option is to take a dangerous action. Our agent will then aim to learn such a safe set; note that accomplishing this is non-trivial despite knowing the rewards, because of the unknown stochastic dynamics.

Definition 4. *We say that $Z \subset S \times A$ is a **safe set** if Z is* closed *and for all $(s,a) \in Z$, $R(s,a) \geq 0$. Informally, we also call every $(s,a) \in Z$ as a safe state-action pair.*

Policy of an algorithm. In PAC-MDP models, an algorithm is considered to be a non-stationary policy $\mathcal{A}$ which, at any instant t, takes as input the path taken so far $p_t := s_0, a_0, \ldots, a_{t-1}, s_t$ and outputs an action. More formally, $\mathcal{A} : \{S \times A\}^* \times S \to A$. Note that since the algorithm is already given the true reward function, we do not provide rewards as input to this non-stationary policy. Then the value of the policy is formally defined as given below.

Definition 5. *For any p_t, we define the value of the non-stationary policy $\mathcal{A}$ of our algorithm on the MDP M as:*

$$V_M^{\mathcal{A}}(p_t) = \mathbb{E}\left[\sum_{t'=t}^{\infty} R(s_{t'}, a_{t'}) \middle| p_t, \mathcal{A}\right].$$

For any $H > 0$, we denote the truncated value function of $\mathcal{A}$ as:

$$V_M^{\mathcal{A}}(p_t, H) = \mathbb{E}\left[\sum_{t'=t}^{t+H} R(s_{t'}, a_{t'}) \middle| p_t, \mathcal{A}\right].$$

Following $\pi \in \Pi(Z)$. Below we state the fact that for a closed set Z, by following $\pi \in \Pi(Z)$, the agent always remains in Z with probability 1.

Fact 1. *For any closed set of state-action pairs Z, and for any policy $\pi \in \Pi(Z)$ and for any initial state $s_0 \in Z$:*

$$\mathrm{P}[\forall t, s_t \in Z \mid \pi, s_0 \in Z] = 1$$

Proof. For any t, if $s_t \in Z$ (and this is true for $t = 0$), we have that $(s_t, \pi(s_t)) \in Z$ since $\pi \in \Pi(Z)$. Since Z is closed, this means that $s_{t+1} \in Z$. Hence, by induction the above claim is true. □

Communicatingness. We now discuss the standard notion of communicating and argue that it is equivalent to the notion defined in the paper (Definition 2). Recall that the standard notion of communicatingness of an MDP [94] is that, for any pair of states in the MDP, there exists a stationary policy that takes the agent from one to the other with positive probability in finite steps. This can be easily generalized to a subset of closed states as follows:

Definition 6. *A closed subset of state-action pairs, $Z \subset S \times A$ is said to be **communicating** if for any two states, $s, s' \in Z$, there exists a stationary policy $\pi_{s \to s'} \in \Pi(Z)$ such that for some $n \geq 1$*

$$\mathrm{P}[s_n = s' \,|\pi_{s \to s'}, s_0 = s] > 0$$

There are two key differences between this definition and Definition 2. (1) Recall that in Definition 2, we defined communicating to mean that for a particular destination state, there exists a single stationary policy that can take the agent from *any* state inside Z to that destination state, whereas in Definition 6 there is a specific policy for every pair of states. (2) Definition 6 only requires the probability of reaching s' to be positive as opposed to 1. Below, we note why these definitions are equivalent:

Fact 2. *Definition 2 and Definition 6 are equivalent.*

Proof. Informally, we need to show that "for any pair of states in Z, there exists a policy that takes the agent from one to the other with positive probability" if and only if "there exists a single policy that reaches a destination state from anywhere in Z with probability 1".

The sufficient direction is clearly true: if such a $\pi_{s'}$ and t from Definition 2 exist, then we know that for any $s \in S$, we can set $\pi_{s \to s'}$ and n in Definition 6 to be $\pi_{s'}$ and t to prove communicatingness (since if it holds with probability 1, it also holds with positive probability).

For the necessary direction, we will show that for a communicating $Z = S \times A$, for any $s' \in Z$, the optimal policy of another MDP satisfies the requirements of $\pi_{s'}$. For a communicating Z with Definition 6, we know that for any $s, s' \in S$, there exists a policy $\pi_{s \to s'}$ and $n \geq 1$ such that $\mathrm{P}[s_n = s' | \pi_{s \to s'}, s_0 = s] > 0$.

Now we construct the MDP. For a given s', define an MDP $M_{s'} = \langle S, A, T, R_{s'}, \gamma_{s'} \rangle$ where $\gamma_{s'} = 1$, $R(s) = \mathbf{1}\{s = s'\}$, and s' is terminal. Note that, since $R(s') = 1$ and is zero everywhere else and s' is terminal, for any policy π, the state value function $V^{\pi}_{M_{s'}}(s) = \mathrm{P}[\exists\, t,\ s.t.\ s_t = s' | \pi, s_0 = s]$. Now define $\pi_{s'}$ to be the optimal policy of this MDP, i.e. the policy that maximizes this value

function. This implies that for any s, $V_{M_{s'}}^{\pi_{s'}}(s) \geq V_{M_{s'}}^{\pi_{s,s'}}(s)$. Then,

$$\begin{aligned}
\mathrm{P}[\exists\, t,\ s.t.\ s_t = s' | \pi_{s'}, s_0 = s] &= V_{M_{s'}}^{\pi_{s'}}(s) \\
&\geq V_{M_{s'}}^{\pi_{s,s'}}(s) \\
&= \mathrm{P}[\exists\, t,\ s.t.\ s_t = s' | \pi_{s,s'}, s_0 = s] \\
&\geq \mathrm{P}[s_n = s' | \pi_{s,s'}, s_0 = s] \\
&> 0.
\end{aligned}$$

Thus, $\mathrm{P}[\exists\ t,\ s.t.\ s_t = s' | \pi_{s'}, s_0 = s] > 0$. Since this condition is satisfied for all starting states $s \in S$ and since we know that Z is closed, we can use Lemma 5 to show that in fact $\mathrm{P}[\exists\, t,\ s.t.\ s_t = s' | \pi_{s'}, s_0 = s] = 1$, proving the existence of $\pi_{s'}$ as claimed.

□

Z_{safe} is safe. Below, we prove that the set Z_{safe} defined in Definition 7 is indeed a safe set.

Fact 3. *Z_{safe} is a safe set.*

Proof. For any $(s, a) \in Z_{\text{safe}}$, we have by definition that $R(s, a) \geq 0$. Now, if for some s' for which $T(s, a, s') > 0$, consider $a' = \pi_{\text{return}}(s')$. Then, by definition, if the agent were to start at (s', a') and continue following π_{return}, with probability 1, it would reach Z_0 while experiencing only positive rewards. Hence, $(s', a') \in Z_{\text{safe}}$. Thus, Z_{safe} is closed. □

4.2.1 Assumptions

We will dedicate a fairly large part of our discussion below detailing the assumptions we make. Some of these are strong and we will explain why they are in fact required to guarantee PAC-MDP optimality in conjunction with the strong form of safety that we care about (being safe on all actions taken in an infinitely long trajectory) in an environment with unknown stochastic dynamics.

First, in order to gain any knowledge of the world safely, the agent must be provided some prior knowledge about the safety of the environment. Without any such knowledge (either in the form of a safe set, prior knowledge of the dynamics, etc) it is impossible to make safety guarantees about the first and subsequent steps of learning. We provide this to the agent in the form of an initial safe set of state-action pairs, Z_0, that is also communicating. We note that this kind of assumption is common in safe RL literature [14, 16]. Additionally, we chose to make this set communicating so that the agent has the freedom to roam and try out actions inside Z_0 without getting stuck.

Assumption 1 (Initial safe set)**.** *The agent is initially given a safe, communicating set $Z_0 \subset S \times A$ such that $s_{init} \in Z_0$.*

In the PAC-MDP setting, we care about how well the agent's policy compares to an optimal policy over the whole MDP. However, in our setting, that would be an unfair benchmark since such an

optimal policy might potentially travel through unsafe state-actions. To this end, we will first suitably characterize a safe set Z_{safe} and then set our benchmark to be the optimal policy confined to Z_{safe}.

We begin by defining Z_{safe} to be the set of state-action pairs from which there exists some (non-negative-reward) return path to Z_0. Indeed, *returnability* is a key aspect in safe reinforcement learning – it has been similarly assumed in previous work [82, 115] and is also very similar to the notion of stability used to define safety in other works [6, 14]. Defining Z_{safe} in terms of returnability ensures that Z_{safe} does not contain any "safe islands" i.e., are safe regions that the agent can venture into, but without a safe way to exit. At a high-level, this criterion helps prevent the agent from getting stuck in a safe island and acting sub-optimally forever. The reasoning for why we need this assumption and traditional PAC-MDP algorithm do not is a bit nuanced and we discuss this in detail in Section 4.3.4.

Definition 7. *We define Z_{safe} to be the set of state-action pairs (s, a) such that $\exists \pi_{return}$ for which:*

$$\mathrm{P}\left[\begin{array}{c} \exists t \geq 0 \; s.t. (s_t, a_t) \in Z_0 \\ \& \; \forall t, R(s_t, a_t) \geq 0 \end{array} | \pi_{return}, (s_0, a_0) = (s, a) \right] = 1$$

Note that it follows that Z_{safe} is a safe set (see Fact 3). Additionally, we will assume that Z_{safe} is communicating; note that given that all actions in Z_{safe} satisfy returnability to Z_0, this assumption is equivalent to assuming that all actions in Z_{safe} are also *reachable* from Z_0. This is reasonable since we care only about the space of trajectories beginning from the initial state, which lies in Z_0. Having characterized Z_{safe} this way, we then define the safe optimal policy using Z_{safe}.

Assumption 2 (Communicatingness of safe set)**.** *We assume $Z_{safe} \subset S \times A$ is communicating.*

Definition 8. *π^*_{safe} is a **safe-optimal** policy in that*

$$\pi^*_{safe} \in \arg \max_{\pi \in \Pi(Z_{safe})} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | \pi, s_0 = s_{init} \right].$$

Besides returnability, another important aspect in the safe PAC-MDP setting turns out to be the time it takes to travel between states. In normal (unsafe) reinforcement learning settings, the agent can gather information from any state-action by experiencing it directly. In this setting, on the other hand, not all state-actions can be experienced safely so the agent must indirectly gather information on a state-action pair of interest by experiencing an analogous state-action. Thus, the agent must be able to visit the informative state-action and return to that state-action pair of interest in polynomial time.

To formalize this, we will assume that within any communicating subset of state-action pairs, we can ensure polynomial-time reachability between states, with non-negligible probability. While this assumption is not made in the normal (unsafe) PAC-MDP setting, we emphasize that this assumption applies to a wide variety of real-world problems and is only violated in contrived examples, such as in a random-walk setting. Specifically, to violate this assumption, there must be two state-action pairs in the safe set where the expected number of steps to move from one

to the next is exponential in the state-action size. This can only happen in random-walk-like scenarios where moving backwards has an equal (or higher) probability than moving forward, which occur very rarely in the real-world. To make this last statement concrete, imagine a 1D grid where any action the agent takes leads it to either of the adjacent states with equal probability of $1/2$. Then, to reach a state that is n steps away with probability $1/2$, it would take the agent, in expectation, an exponential number of steps in n, thus not satisfying Assumption 3. However, if the probabilities of moving forward was $3/4$ for one action and moving backward was $1/4$ for the other action, then the agent could reach a state that is n steps away with probability $1/2$ in less than $2n$ steps, satisfying Assumption 3.

Assumption 3. *(Poly-time communicating) There exists $H_{com} = poly(|S|, |A|)$ such that, for any communicating set $Z \subset S \times A$, and $\forall s' \in Z$, there exists a policy $\pi_{s'} \in \Pi(Z)$ for which*

$$\forall s,\ \mathrm{P}_M[\exists t \leq H_{com}, s_t = s' | \pi_{s'}, s_0 = s] \geq 1/2.$$

Our next assumption is about the transition dynamics: we assume that we know a constant such that any transition either has zero probability or is larger than that known constant. This assumption is *necessary* to perform learning under our strict safety constraints in *unknown, stochastic* dynamics. Specifically, given this assumption, we can use finitely many samples to determine the support of a particular state-action pair's next state distribution. This is critical since the agent cannot take an action unless it knows every possible next state that it could land in. Note that, in a finite MDP, such a τ always exists, we simply assume we have a lower-bound on it.

Assumption 4 (Minimum transition probability)**.** *There exists a* known $\tau > 0$ *such that* $\forall s, s' \in S$ *and* $a \in A$, $T(s, a, s') \in [0] \cup [\tau, 1]$.

Finally, we make an assumption that will help the agent expand its current estimate of the safe-set along its edges. Specifically, note that to establish safety of an edge state-action pair, it is necessary to establish a return path from it to the current safe-set (see Section 4.3.4 for a more in-depth reasoning behind this). Motivated by this we assume the following. Consider any safe subset of state-actions Z and a state-action (s, a) at the edge of Z that also belongs to Z_{safe}. Then, we assume that for every state-action pair that is on the path that starts from this edge and returns to Z, there exists an element in Z that is sufficiently similar to that pair. In other words, for *any* safe subset Z of Z_{safe}, this assumption provides us hope that Z can be expanded by exploring suitable state-actions inside Z.

The fact that this allows expansion of any safe subset Z might seem like a stringent assumption. But we must emphasize this is required to show PAC-MDP optimality: to learn a policy is near-optimal with respect to π^*_{safe}, intuitively, our algorithm must *necessarily* establish the safety of the set of state-actions that π^*_{safe} could visit. Now, depending on what the (unknown) π^*_{safe} looks like, this set could be as large as (the unknown set) Z_{safe} itself. To take this into account, our assumption essentially allows the agent to use analogies to expand its safe set to a set as large as Z_{safe} if the need arises. Without this assumption, the agent might at some point not be able to expand the safe set and end up acting sub-optimally forever. We note earlier works [6, 14, 115] do not make this assumption because they crucially didn't need to establish PAC-MDP optimality.

Algorithm 2 Analogous Safe-state Exploration $(\alpha, \Delta, m, \delta_T, R, \gamma, \gamma_{\text{explore}}, \gamma_{\text{switch}}, \tau)$

Initialize: $\hat{Z}_{\text{safe}} \leftarrow Z_0$; $n(s,a), n(s,a,s') \leftarrow 0$; $\overline{Z}_{\text{goal}} \leftarrow S \times A$; $s_0 \leftarrow s_{\text{init}}$.
Initialize: $\hat{Z}_{\text{unsafe}} \leftarrow \{(s,a) \in S \times A : R(s,a) < 0\}$.
Compute confidence intervals, $\hat{\overset{\Delta}{T}}$ and $\hat{\overset{\Delta}{\epsilon}}_T$, using Alg 7 with
    state-action-state counts n, parameter δ_T, and analogy function α, Δ.
Compute $\overline{\pi}_{\text{goal}}, \overline{Z}_{\text{goal}}, Z_{\text{explore}}, \hat{Z}_{\text{unsafe}}$ using Alg 4 and 5 with parameters γ, τ
    and reward function R.
Compute $\overline{\pi}_{\text{explore}}, \overline{\pi}_{\text{switch}}$ using value iteration (Section 4.3.3) with parameters $\gamma_{\text{explore}}, \gamma_{\text{switch}}$.
for $t = 1, 2, 3, \ldots$ **do**

$$a_t \leftarrow \begin{cases} \overline{\pi}_{\text{goal}}(s_t) & \textbf{if } s_t \in \overline{Z}_{\text{goal}} \;\&\; \overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}} \\ \overline{\pi}_{\text{explore}}(s_t) & \textbf{if } \overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}} \\ \overline{\pi}_{\text{switch}}(s_t) & \textbf{otherwise}. \end{cases}$$

    Take action a_t and observe next state s_{t+1}.
    if $n(s_t, a_t) < m$ **then**
        $n(s_t, a_t) \mathrel{+}= 1$, $n(s_t, a_t, s_{t+1}) \mathrel{+}= 1$.
        Recompute confidence intervals, then expand $\hat{Z}_{\text{safe}}$ using Alg 3 with parameter τ.
        Recompute $\overline{\pi}_{\text{goal}}, \overline{Z}_{\text{goal}}, Z_{\text{explore}}, \hat{Z}_{\text{unsafe}}, \overline{\pi}_{\text{explore}}, \overline{\pi}_{\text{switch}}$ as above.

Assumption 5 (Similarity of return paths)**.** *For any safe set Z such that $Z_0 \subset Z$ and for any $(\tilde{s}, \tilde{a}) \in Z_{safe}$ such that $\tilde{s} \in Z$ and $(\tilde{s}, \tilde{a}) \notin Z$, we know from Assumption 2 that the agent can return to Z_0 (and by extension, to Z) from $(\tilde{s}, \tilde{a})$ through at least one $\pi_{return} \in \Pi(Z_{safe})$. Let $\tilde{Z}$ denote the set of state-action pairs (s,a) visited by π_{return} before reaching Z, in that,*

$$\mathrm{P}_M\left[\exists t \geq 0 \; \begin{smallmatrix}(s_t,a_t)=(s,a)\\ \forall t'<t \;\; s_{t'} \notin Z\end{smallmatrix} \;\middle|\; (s_0,a_0) = (\tilde{s},\tilde{a}), \pi_{return}\right] > 0.$$

We assume that for all $(s,a) \in \tilde{Z} \setminus Z$, there exists $(s',a') \in Z$ such that $\Delta((s,a),(s',a')) \leq \tau/4$.

To provide some intuition behind this assumption, consider again the tight-rope walker example. For this assumption to be satisfied, the agent must be able to safely learn how to return from any point along the real tight-rope. Since the agent has access to a similar practice tight-rope, the agent can learn to safely cross the practice tight-rope, turn around, and return safely. Once the agent has learned this policy, it has established a safe return path from any point along the real tight-rope. Thus, this assumption is satisfied.

4.3 Analogous Safe-state Exploration

Given these assumptions, we now detail the main algorithmic contribution of the chapter, the Analogous Safe-state Exploration (ASE) algorithm, which we later prove is Safe-PAC-MDP. In addition to safety and optimality, we also do not want to exhaustively explore the state-action space, like R-Max [22], as that can be prohibitively expensive in large domains. We want to guide our exploration, like MBIE [99], to explore only the state-action pairs that are needed to find the safe-optimal policy. MBIE does this by maintaining confidence intervals of the dynamics

Algorithm 3 Compute Safe Set (R, τ)

Require: Estimated safe set $\hat{Z}_{\text{safe}}$ and confidence intervals $\overset{\Delta}{T}$, $\overset{\Delta}{\epsilon}_T$.
$Z_{\text{candidate}} \leftarrow \{(s,a) \in (S \times A) \setminus \hat{Z}_{\text{safe}}\ s.t.\ \overset{\Delta}{\epsilon}_T\ (s,a) < \tau/2, R(s,a) \geq 0\}$.
while $Z_{\text{candidate}} \neq Z_{\text{closed}}$ in the last iteration **do**
  $Z_{\text{reachable}} \leftarrow \{(s,a) \in Z_{\text{candidate}} : s \in \hat{Z}_{\text{safe}}\}$.
  while $Z_{\text{reachable}}$ changed in the last iteration **do**
    for $(s,a) \in Z_{\text{reachable}} \cup \hat{Z}_{\text{safe}}$ **do**
      Add $\{(s',a') \in Z_{\text{candidate}}\ s.t.\ \overset{\Delta}{T}\ (s,a,s') > 0\}$ to $Z_{\text{reachable}}$.
  $Z_{\text{returnable}} \leftarrow \emptyset$.
  while $Z_{\text{returnable}}$ changed in the last iteration **do**
    for $(s,a) \in Z_{\text{reachable}}$ **do**
      if $\exists\ (s',a') \in Z_{\text{returnable}} \cup \hat{Z}_{\text{safe}}\ s.t.\ \overset{\Delta}{T}\ (s,a,s') > 0$ **then**
        Add (s,a) to $Z_{\text{returnable}}$.
  $Z_{\text{closed}} \leftarrow Z_{\text{returnable}}$.
  while Z_{closed} changed in the last iteration **do**
    for $(s,a) \in Z_{\text{closed}}$ **do**
      if $\exists\ s' \in S\ s.t.\ \overset{\Delta}{T}\ (s,a,s') > 0$ and $\forall\ a' \in A,\ (s',a') \notin Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$ **then**
        Remove (s,a) from Z_{closed}.
  $Z_{\text{candidate}} \leftarrow Z_{\text{closed}}$.
$\hat{Z}_{\text{safe}} \leftarrow Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$.

of the MDP, and then by following an "optimistic policy" computed using the most optimistic model of the MDP that falls within the computed confidence intervals. We build on this standard MBIE approach and equip it with a significant amount of machinery to meet our three objectives simultaneously: safety, guided exploration, and optimality in the PAC-MDP sense.

Policies maintained by ASE. ASE maintains and updates three different policies: (a) an optimistic policy $\overline{\pi}_{\text{goal}}$ that seeks to maximize reward – this is the same as the optimistic policy as in standard MBIE computed on M (except some minor differences), (b) an exploration policy $\overline{\pi}_{\text{explore}}$ that guides the agent towards states in a set called Z_{explore} (described shortly) and finally (c) a "switching" policy $\overline{\pi}_{\text{switch}}$ that can be thought of as a policy that aids the agent in switching from $\overline{\pi}_{\text{explore}}$ to $\overline{\pi}_{\text{goal}}$ (by carrying it from Z_{explore} to $\overline{Z}_{\text{goal}}$ as explained shortly) See Section 4.3.3 for details on these policies.

Sets maintained by ASE. ASE also maintains and updates three major subsets of state-action pairs. 1) a safe set $\hat{Z}_{\text{safe}}$ which is initialized to Z_0 and gradually expanded over time (using Alg 3). 2) an "optimistic trajectory" set $\overline{Z}_{\text{goal}}$ (computed in Alg 4), which contains all state-action pairs that we expect the agent would visit if it were to follow $\overline{\pi}_{\text{goal}}$ from s_{init} under *optimistic* transitions. 3) an "exploration set" Z_{explore} (computed using Algs 4 and 5) which contains state-action pairs that, when explored, will provide information critical to expand the safe set. Besides these state-action sets, the algorithm also maintains a set of L_1 confidence intervals as detailed in Algorithm 7 and Section 4.3.1. The key detail here is that the interval for a given state-action pair is not only

Algorithm 4 Compute $\overline{\pi}_{\text{goal}}$, $\overline{Z}_{\text{goal}}$ and Z_{explore} (α, Δ, τ)

Require: Estimated safe and unsafe sets $\hat{Z}_{\text{safe}}, \hat{Z}_{\text{unsafe}}$ and confidence intervals $\overset{\Delta}{T}$, $\overset{\Delta}{\epsilon}_T$.
  Initialize: $Z_{\text{explore}} \leftarrow \emptyset$.
  for $i = 1, 2, \ldots$ **do**
    Compute $\overline{\pi}_{\text{goal}}$ using Eq 4.5.
    Compute $\overline{Z}_{\text{goal}}$ using Alg 6.
    if $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ **then**
      Break.
    $Z_{\text{edge}} \leftarrow \{(s,a) \in \overline{Z}_{\text{goal}} \setminus \hat{Z}_{\text{safe}} \mid s \in \hat{Z}_{\text{safe}}\}$.
    Compute Z_{explore} using Alg 5.
    if $Z_{\text{explore}} = \emptyset$ **then**
      Add Z_{edge} to $\hat{Z}_{\text{unsafe}}$.
    else
      Break.

Algorithm 5 Compute Z_{explore} (α, Δ, τ)

Require: Sets of state-action pairs on the edge of the safe set, Z_{edge}, and on the goal path, $\overline{Z}_{\text{goal}}$, along with $\hat{Z}_{\text{safe}}, \hat{Z}_{\text{unsafe}}$ and $\overset{\Delta}{T}$, $\overset{\Delta}{\epsilon}_T$.
  Initialize: $Z_{\text{explore}}, Z_{\text{return}} \leftarrow \emptyset$, $Z_{\text{candidate}} \leftarrow \{(s,a) \in \overline{Z}_{\text{goal}} : s \in \hat{Z}_{\text{safe}}\}$, $L \leftarrow 0$, $Z^0_{\text{next}} \leftarrow Z_{\text{edge}}$.
  while $Z_{\text{explore}} = \emptyset$ and $Z^L_{\text{next}} \neq \emptyset$ **do**
    $Z^{L+1}_{\text{next}} \leftarrow \emptyset$.
    for $(s,a) \in Z^L_{\text{next}}$ **do**
      Add (s,a) to Z_{return}
      if $\overset{\Delta}{\epsilon}_T\,(s,a) > \tau/2$ **then**
        Add $\{\tilde{s}, \tilde{a} \in \hat{Z}_{\text{safe}} : \Delta((s,a),(\tilde{s},\tilde{a})) < \tau/4\}$ to Z_{explore}.
      else
        Add $\{s', a' \in S \times A : \overset{\Delta}{T}\,(s,a,s') > 0\} \setminus (Z_{\text{return}} \cup \hat{Z}_{\text{safe}} \cup \hat{Z}_{\text{unsafe}})$ to Z^{L+1}_{next}.
    $L \leftarrow L + 1$.

updated using its samples, but also by exploiting the samples seen at any other well-explored state-action pair that is sufficiently similar to the given pair, according to the given analogies.

How ASE schedules the policies. We discuss how, at any timestep, the agent chooses between one of the above three policies to take an action. First, the agent follows $\overline{\pi}_{\text{goal}}$ whenever it can establish that doing so would be safe. Specifically, whenever (a) $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ and (b) the current state s_t belongs to $\overline{Z}_{\text{goal}}$, it is easy to argue that following $\overline{\pi}_{\text{goal}}$ is safe (see proof of safety in Theorem 4). On the other hand, when (a) does not hold, the agent follows $\overline{\pi}_{\text{explore}}$. In doing so, the hope is that, it can explore Z_{explore} well and use analogies to expand $\hat{Z}_{\text{safe}}$ until it is large enough to subsume $\overline{Z}_{\text{goal}}$ (which means (a) would hold then). As a final case, assume (a) holds, but (b) does not i.e., $s_t \notin \overline{Z}_{\text{goal}}$. This could happen if the agent has just explored a state-action pair far away from $\overline{Z}_{\text{goal}}$, which subsequently helped establish (a) i.e., $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$. Here, we use $\overline{\pi}_{\text{switch}}$ to carry the agent back to $\overline{Z}_{\text{goal}}$. Once carried there, both (a) and (b) hold, so it can switch to

Algorithm 6 Compute $\overline{Z}_{\text{goal}}\ (\alpha, \Delta, \tau)$

Require: Confidence intervals $\overset{\Delta}{T}$, $\overset{\Delta}{\epsilon}_T$.
For convenience, denote $\rho^{\overline{M}_{\text{goal}}}_{\overline{\pi}_{\text{goal}}, s_{\text{init}}}$ as $\overline{\rho}_{\text{goal}}$.
Initialize: $\overline{Z}_{\text{goal}} \leftarrow \emptyset$, $\overline{\rho}_{\text{goal}}(\cdot, \cdot, 0)$ as in Eq 4.3.
Using dynamic programming, compute $\overline{\rho}_{\text{goal}}(\cdot, \cdot, t)$ as in Eq 4.4 for $t = 1, 2, \ldots, |S|$.
Add all $s, a \in S \times A$ where $\overline{\rho}_{\text{goal}}(s, a, |S|) > 0$ to $\overline{Z}_{\text{goal}}$

following $\overline{\pi}_{\text{goal}}$.

Guided exploration. We would like the agent to explore only relevant states by using the optimistic policy as a guide, like in MBIE. This is automatically the case whenever the agent explores using the optimistic goal policy $\overline{\pi}_{\text{goal}}$. As a key addition to this, we use the optimistic policy to also guide the exploration policy $\overline{\pi}_{\text{explore}}$. As explained below, we do this by only conservatively populating Z_{explore} based on the optimistic trajectory set $\overline{Z}_{\text{goal}}$. Recall that the hope from exploring Z_{explore} is that it can help $\hat{Z}_{\text{safe}}$ expand in a way that it is large enough to satisfy $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$. Keeping this in mind, the naive way to set Z_{explore} would be to add all of $\hat{Z}_{\text{safe}}$ to it; this will force us to do a brute-force exploration of the safe set, and consequently aggressively expand the safe set in all directions. Instead of doing so, roughly speaking, we compute Z_{explore} in a way that it can help establish the safety of only those actions that (a) are on the edge of $\hat{Z}_{\text{safe}}$ and (b) also belong to $\overline{Z}_{\text{goal}}$. This will help us conservatively expand the safe set in "the direction of the optimistic goal policy". (Note that all this entails non-trivial algorithmic challenges since we operate in an unknown stochastic environment. Due to lack of space we discuss these challenges in Section 4.5.2)

Algorithm 3 details how the agent computes the current estimate of the safe set while ensuring reachability, returnability, and closedness. The correctness and efficiency of this algorithm is proven in Section 4.6.1. Algorithms 4, 5, and 6 together provide an overview of how ASE computes the goal and explore policies.

4.3.1 Confidence intervals

We let $\hat{T}$ denote the empirical transition probabilities. We then let $\hat{\epsilon}_T(s, a)$ denote the width of the $L1$ confidence interval for the empirical transition probability $\hat{T}(s, a)$. As shown in Strehl and Littman [99], by using the Hoeffding bound, we can ensure that if

$$\hat{\epsilon}_T(s, a) = \sqrt{\frac{2[\ln(2^{|S|} - 2) - \ln(\delta_T)]}{n(s, a)}} \tag{4.1}$$

where $n(s, a)$ is the number of times we have experienced state-action (s, a), our $L1$ confidence interval hold with probability δ_T. Using the given analogies, we can then derive tighter confidence intervals of width $\overset{\Delta}{\epsilon}_T$ (centered around an estimated $\overset{\Delta}{T}$), especially for state-action pairs we have not experienced, as in Algorithm 7. The algorithm essentially transfers the confidence interval

Algorithm 7 Compute Analogy-based Confidence Intervals (α, Δ)

Require: State-action and state-action-state counts $n(\cdot,\cdot)$, $n(\cdot,\cdot,\cdot)$.
  Construct $\hat{T}$ (the empirical transition probabilities) using $n(\cdot,\cdot)$, $n(\cdot,\cdot,\cdot)$.
  Compute $\hat{\epsilon}_T(s,a)$ with Eq 4.1 using δ_T.
  for $(s,a) \in S \times A$ **do**
    $(\tilde{s},\tilde{a}) \leftarrow \arg\min\{\hat{\epsilon}_T(s,a),\ \min_{\tilde{s},\tilde{a}} \epsilon_T(\tilde{s},\tilde{a}) + \Delta((s,a),(\tilde{s},\tilde{a}))\}$.
    $\overset{\Delta}{\epsilon}_T(s,a) := \hat{\epsilon}_T(\tilde{s},\tilde{a})$.
    for $s' \in S$ **do**
      $\tilde{s}' \leftarrow \alpha((s,a,s'),(\tilde{s},\tilde{a}))$.
      $\overset{\Delta}{T}(s,a,s') := \hat{T}(\tilde{s},\tilde{a},\tilde{s}')$.

from a sufficiently similar, well-explored state-action pair to the under-explored state-action pair, using analogies.

Given these analogy-based $L1$ confidence intervals, we now define a slightly narrower space of candidate transition probabilities than the space defined by these confidence intervals in order to fully establish the support of certain transitions. Specifically, we take into account Assumption 4, to rule out candidates which do not have sufficiently large transition probabilities. We also make sure that a transition probability is a candidate only if Z_0 is closed under it, as assumed in Assumption 1.

Definition 9. *Given the transition probabilities $\overset{\Delta}{T}$ and confidence interval widths $\overset{\Delta}{\epsilon}_T$: $S \times A \to \mathbb{R}$, we say that $T^\dagger$ is a **candidate** transition if it satisfies the following for all $(s,a) \in S \times A$:*

1. $\|T^\dagger(s,a) - \overset{\Delta}{T}(s,a)\|_1 \leq \overset{\Delta}{\epsilon}_T(s,a)$.
2. *if for some s', $\overset{\Delta}{T}(s,a,s') = 0$ and $\overset{\Delta}{\epsilon}_T(s,a) < \tau$, then $T^\dagger(s,a,s') = 0$.*
3. *if $(s,a) \in Z_0$, then $\forall s' \notin Z_0$, $T^\dagger(s,a,s') = 0$*

Furthermore, we let $CI(\overset{\Delta}{T})$ denote the space of all candidate transition probabilities.

4.3.2 Discounted future state distribution.

Below we define the notion of a discounted future state distribution (originally defined in Sutton et al. [104]), and then describe how we compute it in practice. We will need this notion in order to compute $\overline{Z}_{\text{goal}}$ (discussed in the section titled "Goal MDP").

Given an MDP $M^\dagger$, policy π, and state s, the discounted future state distribution is defined as follows:

$$\rho^{M^\dagger}_{\pi,s}(s',a') = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathrm{P}\left(s_t = s', a_t = a' | \pi, s_0 = s\right) \tag{4.2}$$

In words, for any state action pair (s',a'), $\rho_{\pi,s}(s',a')$ denotes the sum of discounted probabilities that (s',a') is taken at any $t \geq 0$ following policy π from s in $M^\dagger$.

Computing discounted future state distributions. We use a dynamic programming approach to approximate the discounted future state distribution. Note that we are assuming that the policy π is deterministic.

First, for all $\tilde{s} \in S$ and $\tilde{a} \in A$, we set:

$$\rho^{M^\dagger}_{\pi,s}(\tilde{s}, \tilde{a}, 0) = \begin{cases} 1-\gamma & \tilde{s} = s \text{ and } \tilde{a} = \pi(s) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Then, at each step, $t+1$, we will set:

$$\rho^{M^\dagger}_{\pi,s}(\tilde{s}, \tilde{a}, t+1) = \rho^{M^\dagger}_{\pi,s}(\tilde{s}, \tilde{a}, t) + \gamma \sum_{\tilde{s}' \in S} T(\tilde{s}', \pi(\tilde{s}'), \tilde{s}) \rho^{M^\dagger}_{\pi,s}(\tilde{s}', \pi(\tilde{s}'), t) \tag{4.4}$$

4.3.3 Computing optimistic policies

$\overline{\pi}_{\text{goal}}$, $\overline{\pi}_{\text{explore}}$, and $\overline{\pi}_{\text{switch}}$ are the optimisitc policies for three different MDPs, M_{goal}, M_{explore}, and M_{switch} (described below). For the theory, we assume that these policies are the true optimistic policy, but in practice this is computed using finite-horizon optimistic form of value iteration introduced in Strehl and Littman [99]. Here we describe this optimistic value iteration procedure.

Optimistic Value Iteration Let $M^\dagger$ be an MDP that is the same as M but with an arbitrary reward function $R^\dagger$ and discount factor $\gamma^\dagger$. Then, the optimistic state-action value function is computed as follows.

$$\begin{aligned} \overline{Q}^\dagger(s,a,0) &= 0 \\ \overline{Q}^\dagger(s,a,1) &= R^\dagger(s,a) \\ \overline{Q}^\dagger(s,a,t) &= R^\dagger(s,a) + \gamma^\dagger \max_{T^\dagger \in CI(\hat{T})} \sum_{s' \in S} T^\dagger(s,a,s') \max_{a' \in A} \overline{Q}^\dagger(s',a',t-1), \ \forall t > 0. \end{aligned} \tag{4.5}$$

As $t \to \infty$, $\overline{Q}^\dagger(s,a,t)$ converges to a value $\overline{Q}^\dagger(s,a)$ since the above mapping is a contraction mapping. For our theoretical discussion, we assume that we compute these values for an infinite horizon i.e., we compute $\overline{Q}^\dagger(s,a)$.

We then let $\overline{T}^\dagger$ denote the transition probability from $CI(\hat{T})$ that corresponds to the optimistic transitions that maximize $\overline{Q}^\dagger$ in Equation 4.5. Also, we let $\overline{M}^\dagger$ denote the 'optimistic' MDP, $\langle S, A, \overline{T}^\dagger, R^\dagger, \gamma^\dagger \rangle$.

Goal MDP. We define M_{goal} to be an MDP that is the same as M, but without the state-action pairs from $\hat{Z}_{\text{unsafe}}$ (which is a set of state-action pairs that we will mark as unsafe). More concretely, $M_{\text{goal}} = \langle S, A, T, R_{\text{goal}}, \gamma_{\text{goal}} \rangle$, where:

$$R_{\text{goal}}(s,a) = \begin{cases} -\infty & (s,a) \in \hat{Z}_{\text{unsafe}} \\ R(s,a) & \text{otherwise.} \end{cases}$$

We then define $\overline{Q}_{\text{goal}}$ to be the finite-horizon optimistic Q-value computed on M_{goal}, and $\overline{\pi}_{\text{goal}}$ the policy dictated by the estimate of $\overline{Q}_{\text{goal}}$. Also, let $\overline{T}_{\text{goal}}$ denote the optimistic transition probability and $\overline{M}_{\text{goal}}$ the optimistic MDP.

Using the above quantities, we now describe **how to compute $\overline{Z}_{\text{goal}}$** (which we also summarize in Alg 6). Recall that we want $\overline{Z}_{\text{goal}}$ to be the set of all state-actions that would be visited with some non-zero probability by following $\overline{\pi}_{\text{goal}}$ under the optimistic MDP $\overline{M}_{\text{goal}}$. More concretely, for convenience, first define $\overline{\rho}_{\text{goal}} := \rho^{\overline{M}_{\text{goal}}}_{\overline{\pi}_{\text{goal}}, s_{\text{init}}}$, where $\rho^{\overline{M}_{\text{goal}}}_{\overline{\pi}_{\text{goal}}, s_{\text{init}}}$ is as defined in Equation 4.2. Then, we would like $\overline{Z}_{\text{goal}}$ to be the set $\{(s,a) \in S \times A : \overline{\rho}_{\text{goal}}(s,a) > 0\}$.

However, directly computing this infinite-horizon estimate in practice is impractical. Instead, here we make use of Lemma 11 and Corollary 4, which allow us to exactly compute $\overline{Z}_{\text{goal}}$ through a finite-horizon estimate of $\overline{\rho}_{\text{goal}}$. Specifically, consider the finite-horizon estimate of $\overline{\rho}_{\text{goal}}$ (i.e., the finite horizon estimate of $\rho^{\overline{M}_{\text{goal}}}_{\overline{\pi}_{\text{goal}}, s_{\text{init}}}$ as defined in Equation 4.3 and Equation 4.4) which can be computed using dynamic programming. In Lemma 11, we show that if we set the horizon $H \geq |S|$, then $\overline{\rho}_{\text{goal}}(s,a,H) > 0$ if and only if $\overline{\rho}_{\text{goal}}(s,a) > 0$. Hence, we fix H to be any value greater than or equal to $|S|$ and then compute

$$\overline{Z}_{\text{goal}} := \{(s,a) \in S \times A : \overline{\rho}_{\text{goal}}(s,a,H) > 0.\} \tag{4.6}$$

As stated in Corollary 4, this will guarantee what we need, namely that $\overline{Z}_{\text{goal}} = \{(s,a) \in S \times A : \overline{\rho}_{\text{goal}}(s,a) > 0\}$. We summarize this algorithm in Algorithm 6

Explore MDP. We define $M_{\text{explore}} = \langle S, A, T, R_{\text{explore}}, \gamma_{\text{explore}} \rangle$ to be an MDP with the same states, actions, and transition function as M, but with a different reward function, R_{explore} (computed in Algorithm 5), and discount factor, γ_{explore}. R_{explore} is defined as follows:

$$R_{\text{explore}}(s,a) = \begin{cases} 1 & (s,a) \in Z_{\text{explore}} \\ 0 & (s,a) \in \hat{Z}_{\text{safe}} \setminus Z_{\text{explore}} \\ -\infty & \text{otherwise.} \end{cases}$$

Switch MDP. We define $M_{\text{switch}} = \langle S, A, T, R_{\text{switch}}, \gamma_{\text{switch}} \rangle$ to be an MDP with the same states, actions, and transition function as M, but with a different reward function, R_{switch}, and discount factor, γ_{switch}. More specifically, R_{switch} is defined as follows:

$$R_{\text{switch}}(s,a) = \begin{cases} 1 & (s,a) \in \overline{Z}_{\text{goal}} \\ 0 & (s,a) \in \hat{Z}_{\text{safe}} \setminus \overline{Z}_{\text{goal}} \\ -\infty & \text{otherwise.} \end{cases}$$

4.3.4 Regarding Safe Islands

Here, we elaborate on the motivation behind involving the notion of returnability (a) in Assumption 5 and (b) in the definition of Z_{safe} in Definition 7.

Returnability in Assumption 5. Recall that a key motivation behind Assumption 5 was that in order to add a safe subset of state-actions to the current safe set, it is necessary for the agent to establish that subset's returnability i.e., establish a return path from the to-be-added subset to the current safe set. Here we explain why this is necessary. Consider a hypothetical agent that tries to expand its safe set without ensuring that whatever it adds to the safe set is returnable. Such an agent might venture into a safe island: although the agent knows that the subset of state-action pairs it has entered into is safe, the agent does not know of any safe path from that subset back to the original safe set. There are two distinct kinds of such safe islands. The first is where there is truly no safe return path; the second is where there does, in fact, exist some safe return path, but the agent has not yet established that this path is indeed safe. We will refer to these islands as True Safe Islands and False Safe Islands.

Although entering into a True Safe Island is not a problem for ensuring optimality in the PAC-MDP sense, entering into a False Safe Island creates trouble. More concretely, in a True Safe Island, since there is no safe way to leave such an island, even the safe-optimal policy *must* remain on this True Safe Island. Thus, the agent that has ventured into a True Safe Island, can potentially find the ϵ-optimal policy, even though it may be forever stuck in this island. However, in a False Safe Island, since there is indeed a safe path to leave this island, it can be the case that the safe-optimal policy from this island will leave the island (and then achieve far higher future reward, than a policy confined to the False Safe Island). Hence, for the agent to be PAC-MDP optimal, it must first establish safety of this path. However, for an agent stuck inside this island, there may be no means to establish safety of that path simply by exploring that island – unless the island is rich enough with analogous states like Z_0 is (which may not be the case if this happens to be a tiny island). Thus, the agent could be forever stuck in the False Safe Island and even worse, it might act ϵ-sub-optimally forever (by choosing to remain instead of exiting). Hence, it's necessary for the agent to establish returnability of any state-action pair before adding it (and Assumption 5 enables us to do this).

Returnability in Definition 7. Next, we explain the motivation behind defining Z_{safe} in Definition 7 to be a "returnable" set. Specifically, recall that Z_{safe} is a safe subset of state-actions, and we would like to compete against the optimal policy on this subset; more importantly, we defined this in a way that any state-action pair in this set is returnable, meaning that it has a return path to Z_0.

Consider the hypothetical scenario where Z_{safe} is defined to allow non-returnable state-actions. Here, we argue that the agent will have to navigate some impractical complications. To begin with, this alternative definition of Z_{safe} could mean that the safe-optimal policy may lead one into True Safe Islands i.e., safe subsets of state-actions from which there is no safe path back to Z_0. This in turn could potentially require the agent to enter into a True Safe Island in order to be PAC-MDP-optimal. Therefore, when the agent expands its safe set, it is necessary for it to find True Safe Islands and add them to the safe set; while doing so, crucially, as discussed earlier, the agent must also avoid adding False Safe Islands to the safe set. Then, in order to meet these

two objectives, the agent should consider every possible safe island and consider all its possible return paths, and establish their safety. If it can be established that no safe return paths exist for a particular safe island, the agent can label the island as a True Safe Island and add it to its safe set.

Thus, in theory, the above fairly exhaustive algorithm can address the more liberal definition of Z_{safe}; however, in many practical settings, it may be expensive to fully determine the safety of every state-action pair. Hence, we choose to ignore this situation by enforcing that Z_{safe} is returnable. With this framework, our algorithm can grow the safe set by establishing return paths from the edges of the safe set (as against having to also look for safe islands and establish safety of all their possible return paths).

4.4 Theoretical Results

Below we state our main theoretical result, that ASE is Safe-PAC-MDP. We must however emphasize that this result does not intend to provide a tight sample complexity bound for ASE; nor does it intend to compete with existing sample complexity results of other (unsafe) PAC-MDP algorithms. In fact, our sample complexity result does not capture the benefits of our guided exploration techniques – instead, we use practical experiments to demonstrate that these benefits are significant (see Section 4.7). The goal of this theorem is to establish that ASE is indeed PAC-MDP and safe, which in itself is highly non-trivial as ASE has much more machinery than existing PAC-MDP algorithms like MBIE. In the interest of space we will give a brief overview of the proof & algorithm. A more detailed proof outline can be found in Section 4.5. The full (lengthy) proof is detailed in Section 4.6.

Theorem 4. *For any constant* $c \in (0, 1/4]$, $\epsilon, \delta \in (0, 1]$, *MDP* $M = \langle S, A, T, \mathcal{R}, \gamma \rangle$, *for* $\delta_T = \delta/(2|S||A|m)$, $\gamma_{explore} = \gamma_{switch} = c^{1/H}$, *and* $m = O\left((|S|/\tilde{\epsilon}^2) + (1/\tilde{\epsilon}^2)\ln\left(|S||A|/\tilde{\epsilon}\right)\right)$ *where* $\tilde{\epsilon} = \min\left(\tau, \epsilon(1-\gamma)^2, 1/H^2\right)$ *and* $H = O\left(\max\left\{H_{com}\log H_{com}, (1/(1-\gamma))\ln(1/\epsilon(1-\gamma))\right\}\right)$ *ASE is Safe-PAC-MDP with a sample complexity bounded by* $O\left(Hm|S||A|(1/\epsilon(1-\gamma))\ln(1/\delta)\right)$.

To prove safety, we show in Lemma 3 that Alg 3, which computes $\hat{Z}_{\text{safe}}$, always ensures that $\hat{Z}_{\text{safe}}$ is a safe set. Then, in the main proof of Theorem 4, we argue that the agent always picks state-actions inside $\hat{Z}_{\text{safe}}$. So, it follows that the agent always experiences only positive rewards. Next, in order to prove PAC-MDP-ness, while we build on the core ideas from the proof for PAC-MDP-ness of MBIE [99], our proof is a lot more involved. This is because we need to show that all the added machinery in ASE work in a way that (a) the agent never gets "stuck" and (b) whenever the agent takes a series of sub-optimal actions (e.g., while following $\overline{\pi}_{\text{explore}}$ or $\overline{\pi}_{\text{switch}}$), it can "make progress" in some form. As an example of (a), Lemma 4 shows that $\hat{Z}_{\text{safe}}$ is always a communicating set, so the agent can always freely move between the states in $\hat{Z}_{\text{safe}}$. This is critical to show that when the agent follows $\overline{\pi}_{\text{explore}}$ (or $\overline{\pi}_{\text{switch}}$), it can reach Z_{explore} (or $\overline{Z}_{\text{goal}}$) without being stuck anywhere (see Lemma 12). As an example of (b), Lemma 9 and Lemma 10 together show that only informative state-action pairs are added to Z_{explore} i.e., when explored, they *will* help us expand $\hat{Z}_{\text{safe}}$.

4.5 Proof Outline

The following subsections describe the overall techniques and intuition, and serve as a rough sketch of the proof of Theorem 4.

4.5.1 Establishing Safety

We now highlight the key algorithmic aspects which ensure provably safe learning, in other words, that (w.h.p) the agent always experiences only non-negative rewards. Recall that our agent maintains a safe set $\hat{Z}_{\text{safe}}$, and in order to add new state-action pairs to $\hat{Z}_{\text{safe}}$ while ensuring that $\hat{Z}_{\text{safe}}$ is closed, we must be able to determine a "safe return policy" to $\hat{Z}_{\text{safe}}$. However, doing this in a setting with unknown stochastic dynamics poses a significant challenge: we must be able to find a return policy where, for every state-action pair in the return path, we know the exact support of its next state; furthermore, all these state-action pairs should return to $\hat{Z}_{\text{safe}}$ with probability 1. Below, we lay out the key aspects of our approach to tackling this.

"Transfer" of confidence intervals. As a first step, we start by establishing confidence intervals on the transition distributions of all state-action pairs as described below. Let $\hat{T}$ denote the empirical transition probabilities. Just as in Strehl and Littman [99], we can compute $L1$ confidence intervals of these estimates using the Hoeffding bound (details in Section 4.3.1). Let $\hat{\epsilon}_T(s,a)$ denote the $L1$ confidence interval for the empirical transition probability $\hat{T}(s,a)$. Using the provided distance and analogy function Δ and α, and using simple triangle inequalities, we can then derive tighter confidence intervals $\overset{\Delta}{\epsilon}_T$ (centered around an estimated $\overset{\Delta}{T}$) as in Alg 7. The idea here is to "transfer" the confidence interval from a sufficiently similar, well-explored state-action pair to an under-explored state-action pair, using analogies.

Learning the next-state support. Crucially, we can use these transferred confidence intervals to infer the support of state-action pairs we have not experienced. More concretely, in Lemma 2 we show that, when a confidence interval is sufficiently tight, specifically when $\overset{\Delta}{\epsilon}(s,a) \leq \tau/2$ for some (s,a) (where τ is the smallest non-zero transition probability defined in Assumption 4), we can exactly recover the support of the next state distribution of (s,a). This fact is then exploited by Algorithm 3 to expand the safe set whenever the confidence intervals are updated.

Correctness of $\hat{Z}_{\text{safe}}$. To expand $\hat{Z}_{\text{safe}}$ while ensuring that it is safe and communicating, Algorithm 3 first creates a candidate set, $Z_{\text{candidate}}$, of all state-action pairs (s,a) with sufficiently tight confidence intervals and non-negative rewards (and so, we know their next state supports). The algorithm then executes three (inner) loops each of which prunes this candidate set. To ensure communicatingness, the first loop eliminates candidates that have no probability of reachability from $\hat{Z}_{\text{safe}}$, and the second loop eliminates those from which there is no probability of return to $\hat{Z}_{\text{safe}}$. In order to ensure closedness, the third loop eliminates those that potentially lead us outside of $\hat{Z}_{\text{safe}}$ or the remaining candidates. We repeat these three loops until convergence. We prove in Lemmas 4 and 3 that Algorithm 3 correctly maintains the safety and communicatingness of $\hat{Z}_{\text{safe}}$ and in Lemma 1 that the algorithm terminates in polynomial time. Note that in Theorem 4, we prove that (w.h.p.) our agent always picks actions only from $\hat{Z}_{\text{safe}}$.

Completeness of $\hat{Z}_{\text{safe}}$. While the above aspects ensure correctness of Algorithm 3, these would

be satisfied even by a trivial algorithm that always only returns Z_0. Hence, it is important to establish that for given set of confidence intervals, $\hat{Z}_{\text{safe}}$ is "as large as it can be". More concretely, consider any state on the edge of $\hat{Z}_{\text{safe}}$ for which there exists a return policy to $\hat{Z}_{\text{safe}}$ which passes only through (non-negative reward) state-action pairs with confidence intervals at most $\tau/2$; this means that we know all possible trajectories in this policy, and all of these lead to $\hat{Z}_{\text{safe}}$. In such a case, we show in Lemma 6 that Algorithm 3 does indeed add this edge action *and all of the actions in every possible return trajectory* to $\hat{Z}_{\text{safe}}$.

4.5.2 Guided Exploration

To be able to add a state-action pair to our conservative estimate of the safe-set, $\hat{Z}_{\text{safe}}$, we not only need to tighten the confidence intervals of that state-action pair but also that of every state-action in all its return trajectories to $\hat{Z}_{\text{safe}}$. Observe that this can be accomplished by exploring state-action pairs inside $\hat{Z}_{\text{safe}}$ that are similar to this return path, and using analogies to transfer their confidence intervals. However, this raises two main algorithmic challenges.

Selecting unexplored actions for establishing safety. First, for which unexplored state-action pairs outside $\hat{Z}_{\text{safe}}$ do we want to establish safety? Instead of expanding $\hat{Z}_{\text{safe}}$ arbitrarily, we will keep in mind the objective mentioned in our outline of ASE: we want to expand $\hat{Z}_{\text{safe}}$ so that we can get to a stage where every possible trajectory when following the optimistic goal policy, $\overline{\pi}_{\text{goal}}$, is guaranteed to be safe, allowing the agent to safely follow $\overline{\pi}_{\text{goal}}$. By letting $\overline{Z}_{\text{goal}}$ denote the set of all state-action pairs on any path following $\overline{\pi}_{\text{goal}}$ from the initial state s_{init}, this condition can be equivalently stated as $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$.

So, to carefully select such unexplored state-action pairs, ASE calls Algorithm 4, which is an iterative procedure: in each iteration, it first (re)computes the optimistic goal policy $\overline{\pi}_{\text{goal}}$ and the set $\overline{Z}_{\text{goal}}$. Using this, it then creates a set Z_{edge}, which is the intersection of $\overline{Z}_{\text{goal}}$ and the set of all edge state-action pairs of $\hat{Z}_{\text{safe}}$. We then hope to establish safety of Z_{edge}, so that, intuitively, we can *expand the frontier of our safe set only along the direction of the optimistic path*. To this end, Algorithm 4 calls Algorithm 5 to compute a corresponding $Z_{\text{explore}} \subset \hat{Z}_{\text{safe}}$ to explore (we will describe Algorithm 5 shortly).

Now, in the case Z_{explore} is non-empty, Algorithm 4 returns control back to ASE, for it to pursue $\overline{\pi}_{\text{explore}}$ – and Lemma 12 shows that $\overline{\pi}_{\text{explore}}$ indeed explores Z_{explore} in poly-time. But if Z_{explore} is empty, Algorithm 4 adds all of Z_{edge} to $\hat{Z}_{\text{unsafe}}$; in the next iteration, $\overline{\pi}_{\text{goal}}$ is updated to ignore $\hat{Z}_{\text{unsafe}}$. In Lemma 9, we use Assumption 5, to prove that the elements added to $\hat{Z}_{\text{unsafe}}$ are indeed elements that do not belong to Z_{safe} (and so we can confidently ignore $\hat{Z}_{\text{unsafe}}$ while computing $\overline{\pi}_{\text{goal}}$). In Lemma 10, we show that this iterative approach terminates in poly-time and either returns a non-empty Z_{explore} that can be explored by $\overline{\pi}_{\text{explore}}$, or updates $\overline{\pi}_{\text{goal}}$ in a way that $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$. In the case that $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$, using Lemma 12, we show that the agent first takes $\overline{\pi}_{\text{switch}}$ to enter into $\overline{Z}_{\text{goal}}$ in finite time, so that the agent can pursue $\overline{\pi}_{\text{goal}}$.

Selecting safe actions for exploration. To establish safety of an unexplored $(s, a) \in Z_{\text{edge}}$, we must explore state-action pairs from $\hat{Z}_{\text{safe}}$ that are similar to state-actions along an *unknown* return policy from (s, a) in order to learn that *unknown* policy. While such a policy does exist

if $(s, a) \in Z_{\text{safe}}$ (according to Assumption 5), the challenge is to resolve this circularity, without exploring $\hat{Z}_{\text{safe}}$ exhaustively.

Instead, Algorithm 5 uses a breadth-first-search (BFS) from (s, a) which essentially enumerates a superset of trajectories that contains the true return trajectories. Specifically, it first enumerates a list of state-action pairs that are a 1-hop distance away and if any of them have a loose confidence interval, it adds to Z_{explore} a corresponding similar state-action pair from $\hat{Z}_{\text{safe}}$ (if any exist). If Z_{explore} is empty at this point, Algorithm 5 repeats this process for 2-hop distance, 3-hop distance and so on, until either Z_{explore} is non-empty or the BFS tree cannot be grown any further. Lemma 9 argues that this procedure does populate Z_{explore} with all the state-action pairs necessary to establish the required return paths; Lemma 7 demonstrates its polynomial run-time. Although we cannot guarantee that this method does not explore all of $\hat{Z}_{\text{safe}}$, we do see this empirically, as we show in our experiments (see Section 4.7).

4.6 Proof of Theorem 4

This section details our proof of Theorem 4, that ASE is guaranteed to be safe with high probability and is optimal in the PAC-MDP sense. We start by restating Theorem 4 and proving it. We then examine proofs for the correctness and polynomial computation time of Algorithm 3. Specifically, we show that the computed $\hat{Z}_{\text{safe}}$ is closed and communicating. Next we show that Algorithms 5 and 4 correctly compute the desired Z_{explore} and an estimate of $\overline{Z}_{\text{goal}}$ in polynomial time. The following section shows that the computed estimate of $\overline{Z}_{\text{goal}}$ is in fact correct, under certain conditions. Subsection 4.6.4 provides the key lemmas for proving PAC-MDP, namely that that our agent, following $\overline{\pi}_{\text{goal}}$, $\overline{\pi}_{\text{explore}}$, or $\overline{\pi}_{\text{switch}}$ either performs the desired behavior (acting optimally or reaching certain state-action pairs) or learns something new about the transition function. By bounding the number of times our agent learns something new, we can show that the agent follows the optimal after a polynomial number of steps. The final subsection provides and proves additional supporting lemmas.

Proof.

Proof of admissibility. We first establish the probability with which our confidence intervals remain admissible throughout the entire execution of the algorithm. Note that we only calculate each confidence interval m times for every state-action pair. Thus, by the union bound and our choice of δ_T, the confidence intervals defined by $\hat{T}$ and $\hat{\epsilon}_T$ hold with probability $1 - \delta/2$. Then, by the triangle inequality, even the tighter confidence intervals computed by Algorithm 7 – defined by $\overset{\Delta}{T}$ and $\overset{\Delta}{\epsilon}_T$ – are admissible.

Proof of safety. Next we will show that, given that the confidence intervals are admissible, the algorithm never takes a state-action pair outside of Z_{safe}. Corollary 3, Lemma 3 and 4 together show that $\hat{Z}_{\text{safe}}$ is a safe (which also implies, closed), communicating subset of Z_{safe}. Using this, we will inductively show that the agent is always safe under our algorithm. Specifically, assume that at any time instant, starting from s_0, the agent has so far only taken actions from $\hat{Z}_{\text{safe}}$. Since

$\hat{Z}_{\text{safe}}$ is closed and safe, this means that the agent has so far been safe, and is currently at $s_t \in \hat{Z}_{\text{safe}}$. We must establish that even now the agent takes an action a_t such that $(s_t, a_t) \in \hat{Z}_{\text{safe}}$.

Now, at each step, recall that according to Algorithm 2, the agent follows either $\overline{\pi}_{\text{explore}}$, $\overline{\pi}_{\text{switch}}$, or $\overline{\pi}_{\text{goal}}$. Consider the case when the agent follows either $\overline{\pi}_{\text{explore}}$ or $\overline{\pi}_{\text{switch}}$.

In this case, for all $(s, a) \notin \hat{Z}_{\text{safe}}$ the rewards are set to be $-\infty$, and for all $(s, a) \in \hat{Z}_{\text{safe}}$ the rewards are set to be non-negative. To address both $\overline{\pi}_{\text{explore}}$ and $\overline{\pi}_{\text{switch}}$ together, let the assigned rewards be $R^\dagger$.

Now, for any $t \geq 0$, recall that $\overline{Q}^\dagger(s, a, t)$ denotes the estimate of $\overline{Q}^\dagger(s, a)$ after t iterations of dynamic programming (not to be confused with the finite-horizon value of the optimistic policy). We first claim that, for all $(s, a) \in \hat{Z}_{\text{safe}}$ and for all iterations $t \geq 1$, the resulting optimistic Q-values are such that $\overline{Q}^\dagger(s, a, t) \geq 0$ if $(s, a) \in \hat{Z}_{\text{safe}}$ and $\overline{Q}^\dagger(s, a, t) = -\infty$ otherwise.

We prove this claim by induction on t, assuming that $\overline{Q}^\dagger(s, a, t)$ is initialized to some non-negative value for $t = 0$. For $t = 1$, our claim is satisfied because the Q-values equal to the sum of the reward function and some positive quantity.

Consider any $t > 1$ and $(s, a) \in \hat{Z}_{\text{safe}}$. We know from Equation 4.5 that the Q-value for this horizon can be decomposed into a sum of the reward and the maximum Q-value of the next states (with a positive, multiplicative discount factor). For $(s, a) \in \hat{Z}_{\text{safe}}$, in Equation 4.5, we will have that the first term, which is the reward function, is non-negative. The second term is an expectation over the maximum Q-values (for a horizon of $t - 1$), where the expectation corresponds to the probability distribution of $\overline{T}^\dagger(s, a, \cdot)$ over the next states. Since $(s, a) \in \hat{Z}_{\text{safe}}$ and since $\overline{T}^\dagger$ is a candidate transition function, by Corollary 2, all the next states according to this transition function, belong to $\hat{Z}_{\text{safe}}$. Now, for any $s' \in \hat{Z}_{\text{safe}}$, there exists a' such that $(s', a') \in \hat{Z}_{\text{safe}}$. By induction, we have $\overline{Q}^\dagger(s', a', t-1) \geq 0$, and hence $\max_{a''} \overline{Q}^\dagger(s', a'', t-1) \geq 0$. Hence, even the second term in the expansion of $\overline{Q}^\dagger(s, a, t)$ is non-negative, implying that $\overline{Q}^\dagger(s, a, t) \geq 0$. Now, for any $(s, a) \notin \hat{Z}_{\text{safe}}$, it follows trivially from Equation 4.5 that $\overline{Q}^\dagger(s, a, t) = -\infty$ since the reward is set to be $-\infty$.

Thus, for any state $s \in \hat{Z}_{\text{safe}}$, we have established that there exists a such that $(s, a) \in \hat{Z}_{\text{safe}}$ and $\overline{Q}^\dagger(s, a) \geq 0$. Furthermore, for any action a' such that $(s, a') \notin \hat{Z}_{\text{safe}}$, $\overline{Q}^\dagger(s, a') = -\infty$. Therefore, $\overline{\pi}^\dagger(s)$ must be an action a such that $(s, a) \in \hat{Z}_{\text{safe}}$. This proves that $\overline{\pi}^\dagger \in \Pi(\hat{Z}_{\text{safe}})$. In other words, this means that at $s \in \hat{Z}_{\text{safe}}$ the agent takes an action a such that $(s, a) \in \hat{Z}_{\text{safe}}$. This completes our argument for $\overline{\pi}_{\text{explore}}$ and $\overline{\pi}_{\text{switch}}$.

As the final case, consider a time instant when the agent follows π_{goal}. By design of Algorithm 2, we know that this happens only if $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ and $s \in \overline{Z}_{\text{goal}}$. Now, by definition of $\overline{Z}_{\text{goal}}$, since $s \in \overline{Z}_{\text{goal}}$, we know that $(s, \overline{\pi}_{\text{goal}}(s)) \in \overline{Z}_{\text{goal}}$. Then, since $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$, $(s, \overline{\pi}_{\text{goal}}(s)) \in \hat{Z}_{\text{safe}}$, implying that the algorithm picks only safe actions, even when it follows $\overline{\pi}_{\text{goal}}$.

Proof of PAC-MDP. Now we will show that ASE is PAC-MDP. To do this, we will show that at any step of the algorithm, assuming our confidence intervals are admissible, the agent will either

act ϵ-optimally or reach a state outside of the known set $K = \{(s,a) \in S \times A : n(s,a) \geq m\}$ in some polynomial number of steps with some positive polynomial probability. To prove this, recall the agent follows either $\overline{\pi}_{\text{explore}}$, $\overline{\pi}_{\text{switch}}$, or $\overline{\pi}_{\text{goal}}$ in three mutually exclusive cases; let us examine each of these three cases.

Case 1: $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$. If $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$, then the agent follows $\overline{\pi}_{\text{explore}}$. In this case, we will show that the agent will experience a state-action pair from K^c (the complement of K) in the first H_1 steps, where $H_1 = O(H^2/c)$.

First note that the condition $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$ can only change if $\overline{Z}_{\text{goal}}$ or $\hat{Z}_{\text{safe}}$ are modified, which can only happen if the agent experiences a state action pair outside K; so, if before the agent takes its H_1th step, this condition changes, we know that the agent has experienced a state-action pair outside K, and hence, we are done.

Consider the case when the agent does not experience any element of K^c in the first $H_1 - 1$ steps; hence the agent follows a fixed $\overline{\pi}_{\text{explore}}$ for these steps. By Lemma 10, since $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$, there must exist some element in Z_{explore} (where $Z_{\text{explore}} \subset \hat{Z}_{\text{safe}}$ by design of Algorithm 5). Now, recall that $\overline{\pi}_{\text{explore}}$ is computed using rewards R_{explore}, which are set to 1 on Z_{explore}, and either 0 or $-\infty$ otherwise, depending on whether the state-action pair is in $\hat{Z}_{\text{safe}}$ or not. If we define $M_{\text{explore}} = \langle S, A, T, R_{\text{explore}}, \gamma_{\text{explore}} \rangle$, then we can invoke Lemma 12 for M_{explore} to establish that the agent reaches Z_{explore} or K^c.

To do this, we must establish that all requirements of Lemma 12 hold. In particular, we have $Z_{\text{explore}} \subset \hat{Z}_{\text{safe}}$ by design of Algorithm 5. We also have $\hat{Z}_{\text{safe}}$ is communicating (Lemma 4) and closed (Lemma 3) and that $\overline{\pi}_{\text{explore}} \in \Pi(\hat{Z}_{\text{safe}})$ (from our proof for safety of Algorithm 2). Finally, since m is sufficiently large, by Lemma 12, the agent will reach Z_{explore} or K^c in $H_1 = O(H^2/c)$ steps with probability at least $1/2$ as long as $H \geq H_{\text{com}} \log \frac{16 H_{\text{com}}}{c} / \log \frac{1}{c}$. Note that although Lemma 12 guarantees this for the behavior of the agent on M_{explore}, the same would apply for M as well, since both these MDPs share the same transition. Finally, note that since $Z_{\text{explore}} \notin K$ (by Lemma 8), this means that the agent escapes K in H_1 steps.

Case 2: $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$, $s_t \notin \overline{Z}_{\text{goal}}$. Now consider the next mutually exclusive case where $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ but the current state $s \notin \overline{Z}_{\text{goal}}$. In this case our agent will attempt to return to $\overline{Z}_{\text{goal}}$ by following $\overline{\pi}_{\text{switch}}$. In this case, we argue that, in the next $H_2 = O(H^2/c)$ steps, the agent either does reach $\overline{Z}_{\text{goal}}$ or experiences a state-action pair in K^c. To see why, note that the current condition can only change if $\overline{Z}_{\text{goal}}$ or $\hat{Z}_{\text{safe}}$ change or if the agent reaches a state $s \in \overline{Z}_{\text{goal}}$. As noted before, $\overline{Z}_{\text{goal}}$ or $\hat{Z}_{\text{safe}}$ are modified only if the agent experiences a state action pair outside K; so, if before the agent takes its H_2th step, this condition changes, we know that the agent has either experienced a state-action pair outside K or has reached $\overline{Z}_{\text{goal}}$, and hence, we are done.

Consider the case when the agent does not experience any element of K^c in the first $H_2 - 1$ steps, and hence follows a fixed $\overline{\pi}_{\text{switch}}$ for these steps. Since $\overline{Z}_{\text{goal}} \neq \emptyset$ (which is trivially true since $s_0 \in \overline{Z}_{\text{goal}}$ always), using the same reasoning as the previous case, we can again use Lemma 12 to show that the agent will reach a state-action pair in $\overline{Z}_{\text{goal}}$ or outside of K in $H_2 = O(H^2/c)$ steps with probability at least $1/2$, since m is sufficiently large and $H \geq H_{\text{com}} \log \frac{16 H_{\text{com}}}{c} / \log \frac{1}{c}$.

Case 3: $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$, $s_t \in \overline{Z}_{\text{goal}}$. Finally, we consider the last case where $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ and the current state $s \in \overline{Z}_{\text{goal}}$. In this case, we argue that the agent either takes an action that is near-optimal, or in the next H steps, it reaches K^c with sufficiently large probability.

Let $\mathrm{P}(A_M)$ be the probability that starting at this step, the Algorithm 2 leads the agent out of K in H steps, conditioned on the history p_t. Now, if $\mathrm{P}(A_M) \geq \epsilon(1-\gamma)/4$, the agent will escape K in H steps with sufficient probability. Hence, consider the case when $\mathrm{P}(A_M) \leq \epsilon(1-\gamma)/4$.

Then, assuming $H \geq O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}\right)$ and sufficiently large m, we can use the above probability bounds and Lemma 14 to show that in this case the state s_t that the agent currently is in satisfies:

$$V_M^{\mathcal{A}}(p_t) \geq V^*_{M_{\text{goal}}}(s_t) - \epsilon. \tag{4.7}$$

To complete our discussion of this case, we need to lower bound the right hand side in terms of the value of the safe-optimal policy π^*_{safe} on the true MDP M. Recall that π^*_{safe} is a policy that maximizes $V_M^{\pi^\star_{\text{safe}}}$ subject to the constraint that $\pi^*_{\text{safe}} \in \Pi(Z_{\text{safe}})$. Now, consider an MDP M^*_{goal} with the same transitions as M. However it has rewards R^*_{goal} such that for all $(s,a) \notin Z_{\text{safe}}$, $R^*_{\text{goal}}(s,a) = -\infty$ and everywhere else $R^*_{\text{goal}}(s,a) = R(s,a)$. Now, since $\pi^*_{\text{safe}} \in \Pi(Z_{\text{safe}})$, from Fact 1, we have that following π^*_{safe} from any $s \in Z_{\text{safe}}$, the agent would never exit Z_{safe}. Hence, for any $s \in Z_{\text{safe}}$, $V^{\pi^*_{\text{safe}}}_{M^*_{\text{goal}}}(s) = V^{\pi^*_{\text{safe}}}_M(s)$. Note that this equality applies to the current state s_t since it is inside $\overline{Z}_{\text{goal}}$, and we know that $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ and $\hat{Z}_{\text{safe}} \subset Z_{\text{safe}}$ (by Corollary 3).

Now, let us compare M^*_{goal} and M_{goal}. Recall that M_{goal} has its rewards set to $-\infty$ only on $\hat{Z}_{\text{unsafe}}$ (and everywhere else, it equals $R(s,a)$). By Lemma 9, we know that $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$, and therefore $\hat{Z}_{\text{unsafe}} \subset Z^c_{\text{safe}}$. Thus, both M_{goal} and $M^\star_{\text{goal}}$ have the same rewards as M, except M_{goal} has the rewards set to $-\infty$ on a set $\hat{Z}_{\text{unsafe}}$, while $M^\star_{\text{goal}}$ has rewards set to $-\infty$ on a superset of $\hat{Z}_{\text{unsafe}}$, Z^c_{safe}. In other words, the rewards of M_{goal} are greater than or equal to the rewards of M^*_{goal}. Thus, the value of the optimal policy on M_{goal} cannot be less than that of M^*_{goal}. Formally, for all s, $V^*_{M_{\text{goal}}}(s) \geq V^{\pi^*_{\text{safe}}}_{M^*_{\text{goal}}}(s)$. Since we also have $V^{\pi^*_{\text{safe}}}_{M^*_{\text{goal}}}(s_t) = V^{\pi^*_{\text{safe}}}_M(s_t)$, we get:

$$V^*_{M_{\text{goal}}}(s_t) \geq V^{\pi^*_{\text{safe}}}_M(s_t). \tag{4.8}$$

Thus, from Equations 4.7 and 4.8, we get that at this state, $V^{\mathcal{A}}_{M_{\text{goal}}}(p_t) \geq V^{\pi^*_{\text{safe}}}_M(s_t) - \epsilon$.

In summary, the agent does at least one of the following at any timestep:

1. reach K^c in H_1 steps (starting from Case 1) with probability at least $1/2$
2. reach K^c in H_2 steps (starting from Case 2), with probability at least $1/4$
3. reach K^c in H steps (starting from Case 3) with probability at least $\Omega(\epsilon(1-\gamma))$.
4. reach $\overline{Z}_{\text{goal}}$ in H_2 steps (starting from Case 2) with probability at least $1/4$.

5. take a nearly-optimal action (in Case 3).

Note that in the above list, we have slightly modified the guarantee from Case 2. In particular, Case 2 guaranteed that with a probability of $1/2$ the agent would reach either K^c or $\overline{Z}_{\text{goal}}$; from this we have concluded that at least one of these events would have a probability of at least $1/4$.

We first upper bound the number of sub-optimal steps corresponding to the first three events. Observe that the agent can only experience a state-action pair outside of K a total of $m|S||A|$ times. Now, by the Hoeffding bound, we have that it takes at most $O\left(m|S||A|\frac{1}{\epsilon(1-\gamma)}\ln\frac{1}{\delta}\right)$ independent trials to see $m|S||A|$ heads in a coin that has a probability of at least $\Omega(\min(1/4, \epsilon(1-\gamma))) = \Omega(\epsilon(1-\gamma))$ as turning out to be heads. Hence, these trajectories would correspond to at most $O\left(\max(H_1, H_2, H)\cdot m|S||A|\cdot\frac{1}{\epsilon(1-\gamma)}\ln\frac{1}{\delta}\right)$ many sub-optimal steps.

Next, we upper bound the number of sub-optimal steps corresponding to the fourth event. Consider two successful occurrences of this event. That is, in both instances, the agent did indeed reach $\overline{Z}_{\text{goal}}$. In such a case, there must be at least one time instant between these two occurrences when the agent experienced K^c; if not, after the first occurrence, by design of Algorithm 2, the agent would have remained in $\overline{Z}_{\text{goal}}$, thereby precluding the second occurrence of the event from happening. Thus, there can be at most as many successful occurrences of this event as $m|S||A|+1$. Again, by a Hoeffding bound, this corresponds to $O\left(H_2\times m|S||A|\frac{1}{\epsilon(1-\gamma)}\ln\frac{1}{\delta}\right)$ many sub-optimal steps, with probability $1-\delta/2$.

Combining the above two bounds and plugging in the bound on m gives our final bound on the number of sub-optimal steps.

□

4.6.1 Proofs about Algorithm 3

Recall that Algorithm 3 expands the set of safe state-action pairs $\hat{Z}_{\text{safe}}$ (by making use of an updated set of confidence intervals) by first creating a candidate set and then iteratively pruning the set until it stops changing.

First, in Lemma 1, we establish this procedure terminates in polynomial time. In the lemmas that follow after that, we prove soundness and completeness. In particular, in Lemma 3, we establish that the updated $\hat{Z}_{\text{safe}}$ is indeed safe. In Lemma 4, we establish that $\hat{Z}_{\text{safe}}$ is communicating and as a result of which in Corollary 3, we establish that $\hat{Z}_{\text{safe}} \subset Z_{\text{safe}}$. Finally, in Lemma 6 we prove completeness in that, if there exists a state-action pair on the edge of $\hat{Z}_{\text{safe}}$, and if there exists a return path from that edge that only takes state-action pairs whose confidence intervals are sufficiently small, then that state-action pair and all of that return path is added to $\hat{Z}_{\text{safe}}$.

Lemma 1. *Algorithm 3 terminates in $poly(|S|, |A|)$ time.*

Proof. The algorithm begins with a set $Z_{\text{candidate}}$ with $|S|\cdot|A|$ many elements. Now in each outer iteration, $Z_{\text{candidate}}$ either ends up losing some elements, or remains the same. If it does remain the

same, then we break out of the outer loop. Thus, there can be at most $\mathcal{O}(|S| \cdot |A|)$ many iterations of the outer loop.

Next, consider the first inner while block. In each iteration of the while loop, we either add an element to $Z_{\text{reachable}}$ or break out of the loop if $Z_{\text{reachable}}$ does not change. Thus, there can be at most $\mathcal{O}(|S| \cdot |A|)$ many iterations of this while loop. As for the time complexity of the inner while loops, since $\hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$ is a finite set, it is easy to see that these iterations terminate in polynomial time. A similar argument holds for the next while block too. For the third while block, we must apply a slightly different version of this argument where we make use of the fact that in each run of the while loop, we either remove a state-action pair from Z_{closed} or break out of the loop. From the above arguments, it follows that the algorithm terminates in polynomial time. □

Note: In the following discussions, unless otherwise specified, Z_{closed} denotes the set as it is in the last step of Algorithm 3.

Before we prove our other lemmas about Algorithm 3, we first establish a result about Algorithm 7 that computes the tighter confidence intervals. Specifically we show that these confidence intervals are computed in a way that if a particular interval is sufficiently tight, then every candidate transition probability in that interval has the same support of next state-action pairs as the true support.

Lemma 2. *Assume the confidence intervals are admissible. Then, for any* (s, a) *such that* $\overset{\Delta}{\epsilon}_T(s, a) < \tau/2$*, for all* $s' \in S$*,* $\overset{\Delta}{T}(s, a, s') > 0$ *if and only if* $T(s, a, s') > 0$*.*

Proof. First consider the case when $T(s, a, s') = 0$; we will show that $\overset{\Delta}{T}(s, a, s') = 0$. To see why, consider the $(\tilde{s}, \tilde{a})$ that contributed to this confidence interval as computed in the step of Algorithm 7. Since $\overset{\Delta}{\epsilon}_T(s, a) < \tau/2$, from Algorithm 7, we have that $\Delta((s, a), (\tilde{s}, \tilde{a})) \leq \tau/2$. This implies that if $\tilde{s}' := \alpha((s, a, s'), (\tilde{s}, \tilde{a}))$, then $|T(\tilde{s}, \tilde{a}, \tilde{s}') - T(s, a, s')| \leq \tau/2$. Since we assumed $T(s, a, s') = 0$, we have that $|T(\tilde{s}, \tilde{a}, \tilde{s}')| \leq \tau/2$. However, by Assumption 4, we have that the range of T lies in $[0] \cup [\tau, 1]$, therefore, to satisfy the above inequality, we must have that $T(\tilde{s}, \tilde{a}, \tilde{s}') = 0$. This would imply that the empirical probability $\hat{T}(\tilde{s}, \tilde{a}, \tilde{s}')$ too is zero, which then is assigned to $\overset{\Delta}{T}(s, a, s')$ in Algorithm 7. Thus, $\overset{\Delta}{T}(s, a, s') = 0$.

Now consider the case when $T(s, a, s') > 0$. We will show that $\overset{\Delta}{T}(s, a, s') > 0$. First, since $T(s, a, s') > 0$, by Assumption 4, it means that $T(s, a, s') \geq \tau$. As argued in the previous case, since $|T(\tilde{s}, \tilde{a}, \tilde{s}') - T(s, a, s')| \leq \tau/2$, we will also have $T(\tilde{s}, \tilde{a}, \tilde{s}') > \tau/2$. Again, by Assumption 4, this would imply $T(\tilde{s}, \tilde{a}, \tilde{s}') \geq \tau$. Note that, from Algorithm 7, we have that, since $\overset{\Delta}{\epsilon}_T(s, a) < \tau/2$, $\hat{\epsilon}_T(\tilde{s}, \tilde{a}) < \tau/2$. And since confidence intervals are admissible, this means that $|T(\tilde{s}, \tilde{a}, \tilde{s}') - \hat{T}(\tilde{s}, \tilde{a}, \tilde{s}')| \leq \tau/2$. This, together with the fact that $T(\tilde{s}, \tilde{a}, \tilde{s}') > \tau/2$, means that $\hat{T}(\tilde{s}, \tilde{a}, \tilde{s}') > \tau/2$. In Algorithm 7, we would assign this value to $\overset{\Delta}{T}(\tilde{s}, \tilde{a}, \tilde{s}')$, thus, resulting in $\overset{\Delta}{T}(s, a, s') > 0$. □

This lemma allows us to compute the support of state-action pairs we have never experienced. Algorithm 3 uses this idea to expand $\hat{Z}_{\text{safe}}$ in a way that retains closedness, thus, safety.

Lemma 3. *Assume our confidence intervals are admissible. Whenever Algorithm 2 calls Algorithm 3, in the final step of Algorithm 3, $\hat{Z}_{safe} \cup Z_{closed}$ is a safe set.*

Proof. We will prove this statement using induction. That is we assume that, before every call to Algorithm 3, $\hat{Z}_{\text{safe}}$ is a safe set. As the base case, this is satisfied in the first call because then, $\hat{Z}_{\text{safe}} = Z_0$ and we have assumed Z_0 to be a safe set in Assumption 1.

Since we populate $Z_{\text{candidate}}$ with only those state-action pairs with non-negative rewards, it follows directly from the run of the algorithm that all state-action pairs that are eventually found in Z_{closed} have non-negative rewards. Thus, to show that $Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$ is safe, we only need to show that $Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$ is a closed set. That is, we need to show that for any $(s, a) \in Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$, every possible next state has an action in $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ i.e., for all $s' \in \{s' \in S : T(s, a, s') > 0\}$, there exists an $a' \in A$ where $(s', a') \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. From the induction assumption (that $\hat{Z}_{\text{safe}}$ is closed), this trivially holds for all $(s, a) \in \hat{Z}_{\text{safe}}$.

Hence, consider any $(s, a) \in Z_{\text{closed}}$. From our choice of $Z_{\text{candidate}}$ in the first step of the algorithm, we know that $\overset{\Delta}{\epsilon}_T (s, a) < \tau/2$. Then from Lemma 2, we know that the set $\overset{\Delta}{S}' = \{s' \in S : \overset{\Delta}{T} (s, a, s') > 0\}$ is identical to the true support of the next state-action pairs of (s, a). But from the third inner while loop of our algorithm, we have that for all $s' \in \overset{\Delta}{S}'$, we ensure that there exists an $a' \in A$ where $(s', a') \in Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$, implying that all possible next states of (s, a) have a corresponding action in $Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$. Thus, $Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$ is closed, and also, safe. □

As a corollary of the above result, we can also show that $\hat{Z}_{\text{safe}}$ is closed even if we replaced the true transitions by some candidate transition.

Corollary 2. *Assume the confidence intervals are admissible. During the run of Algorithm 2, we always have that, for any $T^\dagger \in CI(\overset{\Delta}{T})$ and for any $(s, a) \in \hat{Z}_{safe}$, if there exists $s' \in S$ such that $T^\dagger(s, a, s') > 0$, then $s' \in \hat{Z}_{safe}$.*

Proof. By design of Algorithm 3, we know that for any $(s, a) \in \hat{Z}_{\text{safe}}$, either $(s, a) \in Z_0$ or $\overset{\Delta}{\epsilon} (s, a) < \tau/2$.

Consider the case where $(s, a) \in Z_0$. Since the confidence intervals are admissible, by the third requirement in the definition of the candidate transition set (Definition 9), we have that if $T^\dagger(s, a, s') > 0$, then $s' \in Z_0$. Since $Z_0 \subset \hat{Z}_{\text{safe}}$, $s' \in \hat{Z}_{\text{safe}}$.

Consider the case where $\overset{\Delta}{\epsilon} (s, a) < \tau/2$. Since the confidence intervals are admissible, we have from Lemma 2 that if $T^\dagger(s, a, s') > 0$, then $T(s, a, s') > 0$. Since we have established in Lemma 3 that $\hat{Z}_{\text{safe}}$ is closed, this means that $s' \in \hat{Z}_{\text{safe}}$. □

In order to make sure that our agent can continue exploring without ever getting stuck, we must ensure that whenever Algorithm 3 expands $\hat{Z}_{\text{safe}}$, it remains communicating.

Lemma 4. *Assume our confidence intervals are admissible. Whenever Algorithm 2 calls Algorithm 3, in the final step of Algorithm 3, $\hat{Z}_{safe} \cup Z_{closed}$ is communicating.*

Proof. We will prove this statement using induction. We first assume that, before every call to Algorithm 3, $\hat{Z}_{\text{safe}}$ is an communicating set, and using this, prove that the updated safe set, namely $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ computed at the end of Algorithm 3, is also communicating. As the base case, this is satisfied in the first call because $\hat{Z}_{\text{safe}} = Z_0$ and we have assumed Z_0 to be an communicating set in Assumption 1.

Informally, to show that $\hat{Z}_{\text{safe}}$ is communicating, we will first show that for every state in $\hat{Z}_{\text{safe}}$, the agent has a return policy which ensures that from anywhere in $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$, it can reach that state with non-zero probability. As a second step, we will show that for every state in Z_{closed}, the agent has a 'reach' policy which ensures that from anywhere in $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ it can reach that state with non-zero probability. Finally, we will put these together to establish communicatingness.

Proof that Z_{closed} is returnable. As the first part of the proof, we will show that,

$$\forall \tilde{s} \in \hat{Z}_{\text{safe}}\ \exists \pi_{\text{return}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}),\ \ s.t.\ \forall s \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}\ \mathrm{P}\left[\exists t,\ s_t = \tilde{s} \middle|\, \pi_{\text{return}}, s_0 = s\right] > 0. \tag{4.9}$$

Fix an $\tilde{s} \in \hat{Z}_{\text{safe}}$. We will prove the existence of a suitable π_{return} by induction. Consider the last outer iteration of our algorithm during which we know that $Z_{\text{returnable}}$ at the end of the second inner while block is identical to Z_{closed} at the end of the third inner while block. In this round, consider some (s, a) that is about to be added to $Z_{\text{returnable}}$. For the induction hypothesis, we will consider a hypothesis that is stronger than the one above. In particular, assume that there exists π_{return} such that for every initial state s' currently in $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$, there is non-zero probability of returning to $\tilde{s}$, *while visiting only those states in $\hat{Z}_{safe} \cup Z_{returnable}$ on the way to $\tilde{s}$.* Formally, assume that $\exists \pi_{\text{return}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$ such that:

$$\forall s' \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}},\ \mathrm{P}\left[\exists t,\ \substack{s_t = \tilde{s}, \\ \forall t' < t\ s_{t'} \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}} \middle|\, \pi_{\text{return}}, s_0 = s'\right] > 0. \tag{4.10}$$

This assumption is of course true initially when $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}} = \hat{Z}_{\text{safe}}$, by communicatingness of $\hat{Z}_{\text{safe}}$.

Now, by the manner in which the second while block works, we know that there exists an s' such that $\overset{\Delta}{T}(s, a, s') > 0$ and there exists a' such that $(s', a') \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$. Note that since $\overset{\Delta}{\epsilon}_T(s, a) < \tau/2$ (by our initial choice of $Z_{\text{candidate}}$) and since $\overset{\Delta}{T}(s, a, s') > 0$, it follows from Lemma 2 that $T(s, a, s') > 0$.

Next, consider π'_{return} that is identical to π everywhere, except $\pi'_{\text{return}}(s) = a$. Note that since $(s, a) \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$, $\pi'_{\text{return}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$.

First, we have that the induction assumption still holds for every $s' \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$. That is, for every $s' \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$, with non-zero probability, π'_{return} starts from s' to return to $\tilde{s}$, without visiting any state outside $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$. This is because, for a given random seed, if the agent were to follow π_{return} from s to return to s'', without visiting any states outside of $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$, the agent would do the same under π'_{return} since the two policies would agree on all the visited states.

Next, we have the following when starting from s:

$$\begin{aligned}
\mathrm{P}\left[\exists t,\ \forall t'<t\ s_{t'}\in\hat{Z}_{\text{safe}}\cup Z_{\text{returnable}}^{\,s_t=\tilde{s}}\,\middle|\,\pi'_{\text{return}}, s_0=s\right] &\geq T(s,a,s')\mathrm{P}\left[\exists t,\ \forall t'<t\ s_{t'}\in\hat{Z}_{\text{safe}}\cup Z_{\text{returnable}}^{\,s_t=\tilde{s}}\,\middle|\,\pi'_{\text{return}}, s_0=s'\right] \\
&\geq T(s,a,s')\mathrm{P}\left[\exists t,\ \forall t'<t\ s_{t'}\in\hat{Z}_{\text{safe}}\cup Z_{\text{returnable}}^{\,s_t=\tilde{s}}\,\middle|\,\pi_{\text{return}}, s_0=s'\right] \\
&> 0.
\end{aligned}$$

Here, the first inequality simply follows from the fact that one possible way to reach $\tilde{s}$ from s, is by first taking a step to s'. In the second step, we were able to replace π'_{return} with π_{return} by a similar logic as before. Specifically, for a given random seed, if an agent starts from s' to reach $\tilde{s}$ following π_{return} without ever visiting s, it should do the same under π'_{return} too.

Finally, we have that both the above terms are strictly positive. Hence, we establish that π'_{return}, with non-zero probability, allows the agent to return to $\tilde{s}$, while never visiting any state outside $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}} \cup \{(s,a)\}$.

Proof for Z_{closed} is reachable. For the second part of the proof, we will show that $\forall s \in Z_{\text{closed}}$,

$$\exists \pi_{\text{reach}} \in \Pi(Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}),\ s.t. \forall \tilde{s} \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}\ \mathrm{P}[\exists t, s_t = s | s_0 = \tilde{s}, \pi_{\text{reach}}] > 0. \tag{4.11}$$

We will prove this by induction. Consider the last outer iteration of the algorithm and consider the first inner while loop where we populate $Z_{\text{reachable}}$. Note that since this is the last outer iteration, at the end of this while block, $Z_{\text{reachable}}$ is exactly equal to Z_{closed} that is output at the end of the algorithm. (Thus, during the run of this while loop, we always have that $Z_{\text{reachable}} \subset Z_{\text{closed}}$.) In this while loop, consider the instant at which some (s', a') is about to be added to $Z_{\text{reachable}}$. We will assume by induction that for all $\tilde{s}$ that are currently in $\hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$, Equation 4.11 holds i.e., $\exists \pi_{\text{reach}}$ such that from anywhere in $\hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$, we can reach s with non-zero probability.

As the base case, because of how we have initialized $Z_{\text{reachable}}$, we have that for all $s \in \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$, $s \in \hat{Z}_{\text{safe}}$. Thus, for this case, the induction assumption holds from the fact that we have proven returnability to $\hat{Z}_{\text{safe}}$.

Now, let us turn to the point when the algorithm is about to add (s', a') to $Z_{\text{reachable}}$. Then, note that this means that there exists $(s,a) \in \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$ such that $\hat{\overset{\Delta}{T}}(s,a,s') > 0$. Since $\hat{\overset{\Delta}{\epsilon}}_T(s,a) < \tau/2$ (by our initial choice of $Z_{\text{candidate}}$) and since $\hat{\overset{\Delta}{T}}(s,a,s') > 0$, by Lemma 2, we have that $T(s,a,s') > 0$.

Next, consider the policy π_{reach} guaranteed by our induction assumption, to reach s from anywhere in $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. Then, define a policy π'_{reach} which is identical to π_{reach} on all states, except that $\pi'_{\text{reach}}(s) = a$. Note that since $(s,a) \in \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$, and since $\pi_{\text{reach}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$, we have $\pi'_{\text{reach}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$.

Now, we can show that π'_{reach} reaches s' from any $\tilde{s} \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ because of the following:

$$\begin{aligned}
\mathrm{P}\left[\exists t, s_t = s' \middle|\, \pi'_{\text{reach}}, s_0 = \tilde{s}\right] &\geq T(s,a,s') Pr\left[\exists t, s_{t-1} = s \middle|\, \pi'_{\text{reach}}, s_0 = \tilde{s}\right] \\
&\geq T(s,a,s') Pr\left[\exists t, s_{t-1} = s \middle|\, \pi_{\text{reach}}, s_0 = \tilde{s}\right] \\
&> 0.
\end{aligned}$$

Here, the first inequality comes from the fact that one way to reach s' is by traveling to s and then taking the action a. In the next inequality, we make use of the fact that, for a given random seed, if the agent follows π_{reach} from $\tilde{s}$ to visit s for the first time, it would follow the same steps to reach s even under π'_{reach}, since the policies would agree until then. Finally, we have from our induction assumption that the probability term in the penultimate line is strictly positive; since the transition probability is strictly positive too, the last inequality holds.

This proves that $\pi'_{\text{reach}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$ reaches s' from anywhere in $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ with non-zero probability.

Proof for communicatingness. Finally, we will wrap the above results to establish communicatingness. From the above results, we have that $\forall s' \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$:

$$\exists \pi_{\text{visit}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}),\ s.t. \forall s \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}\ \ \mathrm{P}\left[\exists t,\ s_t = s' \middle|\, \pi_{\text{visit}}, s_0 = s\right] > 0.$$

To establish communicatingness, we need to show that this probability is in fact 1. To do this, we will first note that $\pi_{\text{visit}} \in \Pi(\hat{Z}_{\text{safe}} \cup Z_{\text{closed}})$ and since $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ is closed (as proven in Lemma 3), then use Lemma 5 to establish communicatingness.

□

Corollary 3. *Assume our confidence intervals are admissible. Whenever Algorithm 2 calls Algorithm 3, in the final step of Algorithm 3,* $\hat{Z}_{safe} \cup Z_{closed} \subset Z_{safe}$.

Proof. Note that $Z_0 \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ by construction. Then, since we have established communicatingness of $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ in Lemma 4, for every $(s, a) \in \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$, there should exist a policy π_{return} that travels from (s, a) and goes to any states in Z_0 with probability 1. Furthermore since we established safety in Lemma 3, this also means that every state-action pair visited in this path has non-negative reward. Thus, by definition of Z_{safe}, the claim follows. □

Although our definition of communicating, Def 2, seems strict since its guarantee must hold with probability 1, we show here that this is no stronger than having the guarantee simply hold with positive probability. This lemma helps us prove that our definition of communicating is equivalent to that of the standard definition (see Fact 2) as well as help prove that the safe set we construct is indeed communicating (see Lemma 4).

Lemma 5. *If there exists a closed set of state-action pairs* Z *such that* $\forall s' \in Z$*:*

$$\exists\, \pi_{visit} \in \Pi(Z),\ s.t. \forall s \in Z\ \ \mathrm{P}\left[\exists t,\ s_t = s' \middle|\, \pi_{visit}, s_0 = s\right] > 0$$

then it must be the case that this probability is, in fact, equal to 1*; specifically,* $\forall s' \in Z$*:*

$$\exists\, \pi_{visit} \in \Pi(Z),\ s.t. \forall s \in Z\ \ \mathrm{P}\left[\exists t,\ s_t = s' \middle|\, \pi_{visit}, s_0 = s\right] = 1$$

Proof. From the above, we have that for a given $s' \in Z$, there must exists a constant $H \geq 0$ and a constant $p > 0$ such that:

$$\forall s \in Z \;\; \mathrm{P}\left[\exists t \leq H,\; s_t = s' \middle|\, \pi_{\text{visit}}, s_0 = s\right] \geq p. \tag{4.12}$$

We will start by equating the probability of never visiting s' by decomposing the trajectory into a prefix of H steps and the rest, and then applying the Markov property, as follows:

$$\mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s] = \sum_{s'' \in S} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s'']\mathrm{P}\left[\begin{smallmatrix}\forall t \leq H, s_t \neq s' \\ s_H = s''\end{smallmatrix} \,\middle|\, \pi_{\text{visit}}, s_0 = s\right].$$

Note that since $\pi_{\text{visit}} \in \Pi(Z)$ and since Z is closed, we have from Fact 1 that s'', which is the Hth state in the trajectory, satisfies $s'' \in Z$. Therefore, we can restrict the summation to Z (the remaining terms would zero out). Hence,

$$\begin{aligned}
&\mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s] \\
&= \sum_{s'' \in Z} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s'']\mathrm{P}\left[\begin{smallmatrix}\forall t \leq H, s_t \neq s' \\ s_H = s''\end{smallmatrix} \,\middle|\, \pi_{\text{visit}}, s_0 = s\right] \\
&\leq \left(\max_{s'' \in Z} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s'']\right)\left(\sum_{s'' \in Z} \mathrm{P}\left[\begin{smallmatrix}\forall t \leq H, s_t \neq s' \\ s_H = s''\end{smallmatrix} \,\middle|\, \pi_{\text{visit}}, s_0 = s\right]\right) \\
&= \left(\max_{s'' \in Z} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s'']\right)\left(\mathrm{P}\left[\forall t \leq H, s_t \neq s' \,|\, \pi_{\text{visit}}, s_0 = s\right]\right) \\
&\leq \left(\max_{s'' \in Z} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s'']\right)(1 - p)
\end{aligned}$$

Since the above inequality holds for any $s \in Z$, we can apply a $\max_{s \in Z}$ on the left hand side and rearrange to get:

$$\max_{s'' \in Z} \mathrm{P}[\forall t, s_t \neq s' \,|\pi_{\text{visit}}, s_0 = s''] \cdot p \leq 0$$

Since $p > 0$ (see Equation 4.12), this means that the first term here is equal to zero. In other words, with probability 1, $\exists t$ such that $s_t = s'$ when the agent starts from any s and follows π_{visit}, as claimed. □

Finally, we want to show that Algorithm 3 computes the largest possible $\hat{Z}_{\text{safe}}$ that still retains safety and communicatingness. This is necessary since, if this were not true, we could get into a situation where we know enough to ensure we can perform the optimal policy, but our agent remains trapped in $\hat{Z}_{\text{safe}}$ forever.

Lemma 6. *Assume that the confidence intervals are admissible. Consider some call of Algorithm 3 while executing Algorithm 2. Consider $(\tilde{s}, \tilde{a}) \notin \hat{Z}_{safe}$ such that $\tilde{s} \in \hat{Z}_{safe}$. Let $\exists \pi_{return}$ such that starting at $(\tilde{s}, \tilde{a})$, π_{return} reaches a state in $\hat{Z}_{safe}$ with probability 1. Let $\tilde{Z}$ be the set of state-action pairs visited by the agent starting $(\tilde{s}, \tilde{a})$ following π_{return} before reaching $\hat{Z}_{safe}$. Formally, let:*

$$\tilde{Z} = \{(s, a) \in S \times A : \mathrm{P}\left[\exists t \geq 0 \; \begin{smallmatrix}(s_t, a_t) = (s, a) \\ \forall t' < t \;\; s_{t'} \notin \hat{Z}_{safe}\end{smallmatrix} \,\middle|\, (s_0, a_0) = (\tilde{s}, \tilde{a}), \pi_{return}\right] > 0\}.$$

In the final step of Algorithm 3, if $\forall (s,a) \in \tilde{Z} \setminus \hat{Z}_{safe}$, $\hat{\epsilon}_T(s,a) < \tau/2$, *then* $\tilde{Z} \subset \hat{Z}_{safe} \cup Z_{closed}$.

Proof. Informally, we will show that in each iteration of the algorithm, after the first inner while block, we have $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$; and using this, we will show that after the second inner while block, we have $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$; and finally, because of this, after the third inner while block, $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. Below, we prove these three statements, and finally wrap them up to prove the main claim.

Proof for $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$. Assume there exists $(\tilde{s}, \tilde{a})$ as specified in the lemma statement. Then, formally, we will show that if $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{candidate}}$ before the beginning of the first inner while loop, then $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$ at the end of the loop.

We will prove this by contradiction. Assume for the sake of contradiction that there exists a non-empty $\tilde{Z}_{\text{bad}} \subset \tilde{Z}$ such that $\tilde{Z}_{\text{bad}}$ has no intersection with $\hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$. Let $\tilde{Z}_{\text{good}} = \tilde{Z} \setminus \tilde{Z}_{\text{bad}}$. First we note why $\tilde{Z}_{\text{good}}$ is non-empty. Since $\tilde{Z} \setminus \hat{Z}_{\text{safe}} \subset Z_{\text{candidate}}$ (from our initial assumption), since $(\tilde{s}, \tilde{a}) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$, and since $\tilde{s} \in \hat{Z}_{\text{safe}}$ (from lemma statement), when initializing $Z_{\text{reachable}}$ with $\{(s,a) \in Z_{\text{candidate}} : s \in \hat{Z}_{\text{safe}}\}$, we would add $(\tilde{s}, \tilde{a})$ to $Z_{\text{reachable}}$. Thus, $\tilde{Z}_{\text{good}}$ must contain at least $(\tilde{s}, \tilde{a})$.

Next, we argue that there must exist some $(s', a') \in \tilde{Z}_{\text{bad}}$ such that there exists an $(s,a) \in \tilde{Z}_{\text{good}}$ for which $T(s,a,s') > 0$ and $s \notin \hat{Z}_{\text{safe}}$. If not, then starting from $(\tilde{s}, \tilde{a}) \in \tilde{Z}_{\text{good}}$, the agent can never hope to reach $\tilde{Z}_{\text{bad}}$ before visiting a state $\hat{Z}_{\text{safe}}$, contradicting the fact that $\tilde{Z}_{\text{bad}}$ is a subset of state-action pairs that it visits before reaching $\hat{Z}_{\text{safe}}$.

Now, consider such an (s', a') (which was not added to $Z_{\text{reachable}}$) and its predecessor (s,a), which belongs to $\hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$ because it belongs to $\tilde{Z}_{\text{good}}$. In some iteration of this while loop, we must have examined $(s,a) \in \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$. Since $s \notin \hat{Z}_{\text{safe}}$, we also have $(s,a) \notin \hat{Z}_{\text{safe}}$. Since $(s,a) \in \tilde{Z}_{\text{good}}$, and $\tilde{Z}_{\text{good}} \subset \tilde{Z}$, this further means that $(s,a) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$. From our lemma statement, we then have that the confidence interval of (s,a) is less than $\tau/2$. Hence $\hat{\epsilon}_T(s,a) < \tau/2$. Then, from Lemma 2, since $T(s,a,s') > 0$, we have that $\hat{T}(s,a,s') > 0$. Then, since $(s', a') \in Z_{\text{candidate}}$, we would have added (s', a') to $Z_{\text{reachable}}$ in this iteration, contradicting the fact that we never added it to $Z_{\text{reachable}}$ in the first place. Thus, $\tilde{Z}_{\text{bad}}$ must be empty, implying that $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$.

Proof for $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$. Formally, we will show that if $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{reachable}}$ before the beginning of the second while block, then $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$ at the end of the block.

At the end of the second block, let us define $\tilde{Z}_{\text{bad}} := \tilde{Z} \setminus (\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}})$. We want to show that $\tilde{Z}_{\text{bad}}$ is empty, but assume on the contrary it is not. First we argue that there must exist $(s,a) \in \tilde{Z}_{\text{bad}}$ such that one of its next states belongs to $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$. If this was not the case, then whenever the agent enters $\tilde{Z}_{\text{bad}}$, it will never be able to return to $\hat{Z}_{\text{safe}}$. This is however in contradiction to the definition of $\tilde{Z}$.

For the rest of the discussion, consider such an $(s,a) \in \tilde{Z}_{\text{bad}}$ such that $\exists (s', a') \in \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$ for which $T(s,a,s') > 0$. Since $(s,a) \in \tilde{Z}_{\text{bad}}$ and $Z_{\text{bad}} \subset \tilde{Z} \setminus \hat{Z}_{\text{safe}}$, it means that $(s,a) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$.

Then, from our lemma statement we have $\hat{\overset{\Delta}{\epsilon}}_T(s,a) \leq \tau/2$. Now, from Lemma 2, we have that since the confidence intervals are admissible and since $T(s,a,s') > 0$, it must be the case that $\hat{\overset{\Delta}{T}}(s,a,s') > 0$. Then, in the iteration of the while loop during which (s',a') is present in $\hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$, (s,a) would in fact be added to $Z_{\text{returnable}}$. This, however, contradicts our assumption that $(s,a) \in \tilde{Z}_{\text{bad}}$. Therefore, $\tilde{Z}_{\text{bad}}$ should in fact be empty. Thus, $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$ at the end of this while block.

Proof for $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. Formally, we will show that if $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{returnable}}$ before the beginning of the third while block, then $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ at the end of the block.

Assume on the contrary that there exists $(s,a) \in \tilde{Z}$ such that $(s,a) \notin \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ at the end of the third block. Since Z_{closed} is initialized with all of $\tilde{Z}$ contained in it, consider the *first* such (s,a) that is removed from Z_{closed} during the course of this second inner iteration. Now, just before the moment at which (s,a) is removed, by design of the algorithm, we would have that there exists s' such that $\hat{\overset{\Delta}{T}}(s,a,s') > 0$ and $s' \notin \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. Note that at this point, we also still have $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$. Therefore, this means that $s' \notin Z$. Now, we have that $(s,a) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$, which means, by the lemma statement, $\hat{\overset{\Delta}{\epsilon}}_T(s,a) \leq \tau/2$. Then, since $\hat{\overset{\Delta}{T}}(s,a,s') > 0$, from Lemma 2, we have $T(s,a,s') > 0$. However, this contradicts the fact, if $(s,a) \in \tilde{Z}$, then every next state of (s,a) must lie in $\tilde{Z}$. Thus, no (s,a) belonging to $\tilde{Z}$ must have been removed from $\hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ during this while block, implying that $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ at the end of this block.

Proof of main claim. From the above arguments, we have that whenever the outer iteration begins with $\tilde{Z} \subset Z_{\text{candidate}} \cup \hat{Z}_{\text{safe}}$ it ends with $\tilde{Z} \subset Z_{\text{candidate}} \cup Z_{\text{closed}}$. Now, at the beginning of Algorithm 3, we must have $\tilde{Z} \setminus \hat{Z}_{\text{safe}} \subset Z_{\text{candidate}}$ due to the fact that all elements of $\tilde{Z}$ have confidence intervals at most $\tau/2$. In other words, $\tilde{Z} \subset Z_{\text{candidate}} \cup \hat{Z}_{\text{safe}}$. Then, from the above arguments, we have that $\tilde{Z} \subset \hat{Z}_{\text{safe}} \cup Z_{\text{closed}}$ at the end of the first iteration. Since $Z_{\text{candidate}}$ at the beginning of the second outer iteration is equal to Z_{closed} from the end of the previous outer iteration, we again have $\tilde{Z} \subset Z_{\text{candidate}} \cup \hat{Z}_{\text{safe}}$ at the beginning of the second iteration. Thus, by a similar argument we have that the algorithm preserves the condition that $\tilde{Z} \subset Z_{\text{closed}} \cup \hat{Z}_{\text{safe}}$ at the end of every outer iteration, proving our main claim.

□

4.6.2 Proofs about Algorithm 5 and Algorithm 4

In the next few lemmas, we prove results about Algorithm 5 and Algorithm 4. Recall that Algorithm 5 takes as input a set of edge state-action pairs (which are state-action pairs that do not belong to $\hat{Z}_{\text{safe}}$ but whose state belongs to $\hat{Z}_{\text{safe}}$) and outputs a set of elements from $\hat{Z}_{\text{safe}}$ which need to be explored in order to learn the return paths from the edges. Also recall that the idea of Algorithm 4 is to return an updated $\overline{Z}_{\text{goal}}$ (the set of states in the optimistic goal path) and Z_{explore} (by making a call to Algorithm 5).

In Lemma 7 we demonstrate that Algorithm 5 terminates in polynomial time.

Lemma 8 helps establish that running Algorithm 5 allows the agent progress without getting stuck. More concretely, in Lemma 8 we argue that the elements of Z_{explore} do not belong to K. That is, the elements of Z_{explore} are those that have not already been explored (as otherwise, the agent may be stuck perpetually in exploring what has already been explored).

In Lemma 9, we establish that Algorithm 5 works correctly (and hence, so does Algorithm 4). In particular, we show that Algorithm 5 does not terminate with an empty Z_{explore} when some of the edge state-action pairs of $\hat{Z}_{\text{safe}}$ are indeed safe. If we did not have this guarantee, then it's possible that even though there are some edge state-action pairs are safe, our agent may be stuck without exploring any state-action pair within $\hat{Z}_{\text{safe}}$. On the other hand, with this guarantee, we can be confident that our agent will explore to learn the return path of such edge states, and then establish their safety, using which it can then expand $\hat{Z}_{\text{safe}}$ in the future. As a corollary of this guarantee, we show that Algorithm 4 always ensures that $\hat{Z}_{\text{unsafe}}$ contains no element that actually belongs to Z_{safe}.

Finally, in Lemma 10, we prove that Algorithm 4 terminates in polynomial time, guaranteeing that either Z_{explore} is non-empty (which means the agent can explore $\hat{Z}_{\text{safe}}$ to learn a return path and expand $\hat{Z}_{\text{safe}}$) or that $\overline{Z}_{\text{goal}}$ has been updated in a way that $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ (which means that the agent can stop exploring, and instead, start exploiting).

Below, we prove our result about the run-time complexity of Algorithm 5.

Lemma 7. *Algorithm 5 terminates in poly*$(|S|, |A|)$ *time.*

Proof. Recall that Algorithm 5 executes an iteration of a while loop whenever $Z^L_{\text{next}} \neq \emptyset$ and $Z_{\text{explore}} = \emptyset$. Also recall that inside the while loop, the algorithm executes a for loop that iterates over all elements of Z^L_{next}. Hence, during the while loop, since Z^L_{next} is non-empty, we must also execute at least one iteration of the inner for loop. Now, note that, by design of the algorithm, Z^L_{next} is populated only with elements that do not belong to Z_{return}. Since the for loop adds all these elements to Z_{return}, every call to the for loop corresponds to increasing the cardinality of Z_{return} by at least one. Thus, there can be at most as many executions of the for loop as there are state-action pairs, $|S| \times |A|$. By extension, the while loop can be executed at most $|S| \times |A|$ times, after which it should terminate. □

Next, we show that the state-action pairs marked for exploration by Algorithm 5 have not already been explored well before.

Lemma 8. *Algorithm 5 returns* $Z_{explore}$ *such that for every* $(\tilde{s}, \tilde{a}) \in Z_{explore}$, $(\tilde{s}, \tilde{a}) \in K^c$ *where* $K = \{(s, a) \in S \times A : n(s, a) \geq m\}$ *when* $m \geq O\left(\frac{|S|}{\tau^2} + \frac{1}{\tau^2} \ln \frac{|S||A|}{\tau^2 \delta}\right)$.

Proof. Assume on the contrary that there exists $(\tilde{s}, \tilde{a}) \in Z_{\text{explore}}$ such that $(\tilde{s}, \tilde{a}) \in K$. By design of Algorithm 5, we have that there exists $(s, a) \notin \hat{Z}_{\text{safe}}$ which resulted in the addition of $(\tilde{s}, \tilde{a})$ to Z_{explore}. In particular, we would have that $\overset{\Delta}{\epsilon}_T (s, a) > \tau/2$ and $\Delta((s, a), (\tilde{s}, \tilde{a})) < \tau/2$. Since $(\tilde{s}, \tilde{a}) \in K$, we also have $\hat{\epsilon}_T(\tilde{s}, \tilde{a}) \leq \tau/4$ (and this follows from Lemma 16). Then, we would have $\overset{\Delta}{\epsilon}_T (s, a) \leq \hat{\epsilon}_T(\tilde{s}, \tilde{a}) + \Delta((s, a), (\tilde{s}, \tilde{a}))$, implying $\overset{\Delta}{\epsilon}_T (s, a) \leq \tau/4$ which is a contradiction. Thus, the above claim is correct. □

Below, we show that as long as there is a edge to $\hat{Z}_{\text{safe}}$ with a safe return path to $\hat{Z}_{\text{safe}}$, Algorithm 5 will return with a non-empty Z_{explore}.

Lemma 9. *Assume our confidence intervals are admissible. During a run of Algorithm 2, at the end of every call to Algorithm 5, we will have $Z_{explore} = \emptyset$ only if $\forall\ (\tilde{s}, \tilde{a}) \in Z_{edge}$, $(\tilde{s}, \tilde{a}) \notin Z_{safe}$. As a corollary of this, we always have that $\hat{Z}_{unsafe} \cap Z_{safe} = \emptyset$.*

Proof. We will prove this by induction. Specifically, since Algorithm 4 is the only block where we call Algorithm 5 and modify $\hat{Z}_{\text{unsafe}}$, we will consider a particular iteration of the while loop in Algorithm 4. Then, we will assume that in all the previous iterations of this while loop, when Algorithm 5 was called, it satisfied the above guarantee. Additionally, we will assume that $\hat{Z}_{\text{unsafe}}$ satisfies $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$ in the beginning of this loop. Then, we will show that the guarantee about Algorithm 5 is satisfied even when we call it in this loop, and that, by the end of this loop, $\hat{Z}_{\text{unsafe}}$ continues to satisfy $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$.

As the base case, in the first iteration, since we have never called Algorithm 5 before, the induction hypothesis about Algorithm 5 is trivially satisfied. Furthermore, since $\hat{Z}_{\text{unsafe}}$ is initialized to be empty set, we again trivially have $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$ in the beginning of this loop.

Now, consider any arbitrary iteration of the while loop of Algorithm 4. Through the next few paragraphs below, we will argue why the call to Algorithm 5 in this loop, satisfies the above specified guarantee. In the final paragraph, we provide a simple argument showing why $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$ at the end of the loop.

Proof for claim about Algorithm 5. Assume that during this particular call to Algorithm 5, there exists $(\tilde{s}, \tilde{a}) \in Z_{\text{edge}}$ such that $(\tilde{s}, \tilde{a}) \in Z_{\text{safe}}$. To prove the above claim, we only need to show that in this case Algorithm 5 will result in a non-empty Z_{explore}. So, for the sake of contradiction, we will assume that Z_{explore} is empty after the execution of Algorithm 5.

The outline of our idea is to make use of the fact that by Assumption 5 we are guaranteed a return path from $(\tilde{s}, \tilde{a})$ that is sufficiently similar to state-action pairs in $\hat{Z}_{\text{safe}}$. We will then show that we can pick a particular trajectory from this return path which visits a 'bad' state-action pair – a state-action pair whose counterpart in $\hat{Z}_{\text{safe}}$ has not been explored sufficiently. Then, under the assumption that Z_{explore} remains empty, we will argue that Algorithm 5 will visit all the state-action pairs in this trajectory, and finally, when it encounters the bad state-action pair, the algorithm will add the counterpart of this bad pair to Z_{explore}, reaching a contradiction.

First, let us apply Assumption 5 to $(\tilde{s}, \tilde{a})$ which guarantees a return path from $(\tilde{s}, \tilde{a})$ because it is an edge state-action i.e., $\tilde{s} \in \hat{Z}_{\text{safe}}$ and $(\tilde{s}, \tilde{a}) \notin \hat{Z}_{\text{safe}}$. But to apply this assumption, we must establish that $Z_0 \subset \hat{Z}_{\text{safe}}$. This is indeed true as it follows from how Algorithm 2 initializes $\hat{Z}_{\text{safe}}$ with Z_0 and every call to Algorithm 3 only adds elements to $\hat{Z}_{\text{safe}}$.

Now, consider the π_{return} guaranteed by Assumption 5. Starting from (s, a), and following π_{return}, the agent returns to Z_0 with probability 1. Furthermore, if we define the set of state-action pairs visited by π_{return} on its way to $\hat{Z}_{\text{safe}}$ as:

$$\tilde{Z} = \{(s, a) \in S \times A : \mathrm{P}\left[\exists t \geq 0 \,{}^{(s_t, a_t) = (s, a)}_{\forall t' < t \;\; s_{t'} \notin \hat{Z}_{\text{safe}}} \;\middle|\; (s_0, a_0) = (\tilde{s}, \tilde{a}), \pi_{\text{return}}\right] > 0\},$$

then we are given that every element of $\tilde{Z} \setminus \hat{Z}_{\text{safe}}$ corresponds to an element $(s', a') \in \hat{Z}_{\text{safe}}$ such that $\Delta((s, a), (s', a')) \leq \tau/4$.

To make our discussion easier, let us partition the elements of $\tilde{Z} \setminus \hat{Z}_{\text{safe}}$ into two sets $\tilde{Z}_{\text{good}}$ and $\tilde{Z}_{\text{bad}}$ as follows, depending on whether or not the corresponding (s', a') has been explored sufficiently well or not:

$$\tilde{Z}_{\text{good}} = \{(s, a) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}} \mid \exists (s', a') \in S \times A \text{ s.t. } \Delta((s, a), (s', a')) + \hat{\epsilon}_T(s', a') \leq \tau/2\}$$

and

$$\tilde{Z}_{\text{bad}} = \{(s, a) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}} \mid \forall (s', a') \in S \times A \text{ s.t. } \Delta((s, a), (s', a')) + \hat{\epsilon}_T(s', a') > \tau/2\}.$$

Note that every element of $\tilde{Z} \setminus \hat{Z}_{\text{safe}}$ belongs to exactly one of $\tilde{Z}_{\text{good}}$ and $\tilde{Z}_{\text{bad}}$. Also note that for all $(s, a) \in \tilde{Z}_{\text{good}}$, $\hat{\epsilon}_T(s, a) \leq \tau/2$ since $\Delta((s, a), (s', a')) + \hat{\epsilon}_T(s', a') \leq \tau/2$. However, for all $(s, a) \in \tilde{Z}_{\text{bad}}$, since there exists no sufficiently explored (s', a') that is also sufficiently smaller, $\hat{\epsilon}_T(s, a) > \tau/2$.

Next, we argue that there must exist at least one element in $\tilde{Z}_{\text{bad}}$. If this was not the case, then we would have that all elements in $\tilde{Z} \setminus \hat{Z}_{\text{safe}}$ belong to $\tilde{Z}_{\text{good}}$ and therefore have a confidence interval at most $\tau/2$. Then, from Lemma 6, we will have that when Algorithm 2 executed Algorithm 3 just before calling Algorithm 4, $\hat{Z}_{\text{safe}}$ is updated in a way that $\tilde{Z} \subset \hat{Z}_{\text{safe}}$, which would imply that $(\tilde{s}, \tilde{a}) \in \hat{Z}_{\text{safe}}$. However, this contradicts our assumption in the beginning that that $(\tilde{s}, \tilde{a})$ is an edge state-action pair that does not belong to $\hat{Z}_{\text{safe}}$.

Since $\tilde{Z}_{\text{bad}}$ is non-empty, and since $\tilde{Z}_{\text{bad}} \subset \tilde{Z}$, there should exist a trajectory of π_{return} starting from $(\tilde{s}, \tilde{a})$ that passes through an element of $\tilde{Z}_{\text{bad}}$ before visiting $\hat{Z}_{\text{safe}}$. Let $(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)$ be one such trajectory, where $(s_0, a_0) = (\tilde{s}, \tilde{a})$. Let (s_n, a_n) be the first element in this trajectory that belongs to $\tilde{Z}_{\text{bad}}$. Since all the elements in this trajectory preceding (s_n, a_n) belong to $\tilde{Z}$ but not $\hat{Z}_{\text{safe}}$ or $\tilde{Z}_{\text{bad}}$, all these elements must belong to $\tilde{Z}_{\text{good}}$.

We will now use the fact that, under our initial assumption, Algorithm 5 returns with an empty Z_{explore} to argue by induction that, during that run of Algorithm 5, all state-action pairs in this trajectory up until and including (s_n, a_n) are added to Z_{return}. (After this, we will reach a contradiction).

For the base case, consider (s, a). When the while loop condition is executed the first time, $Z_{\text{explore}} = \emptyset$ by initialization, and $Z^L_{\text{next}} \neq \emptyset$ because it is equal to Z_{edge} which has at least one state-action pair, namely (s, a). Thus, the while loop will be executed, and every element in Z^L_{next} will be added to Z_{return}. Since $Z^L_{\text{next}} = Z_{\text{edge}}$ at this point, this implies that (s, a) will be added to Z_{return}.

Next, for some $i \in [1, n]$, assume by induction that all state-action pairs preceding (s_k, a_k), where $k < i$, have been added to Z_{return}; we must prove the same happens to (s_i, a_i). Consider the loop when (s_{i-1}, a_{i-1}) is examined and added to Z_{return}. Since $(s_{i-1}, a_{i-1}) \in \tilde{Z}_{\text{good}}$, its confidence interval is at most as large as $\tau/2$; thus, in this loop, we would execute the else branch of the if-condition. As a result of this, we can argue that (s_i, a_i) is added to Z^{L+1}_{next} from the following four observations.

First, since the considered trajectory has non-zero probability, we have $T(s_{i-1}, a_{i-1}, s_i) > 0$. Furthermore, since $\overset{\Delta}{\epsilon}_T(s_{i-1}, a_{i-1}) \leq \tau/2$, and since the confidence intervals are admissible, by Lemma 2, we have $\overset{\Delta}{T}(s_{i-1}, a_{i-1}, s_i) > 0$. Second, if (s_i, a_i) was part of Z_{return} at this point, we are already done; so let us consider the case that currently $(s_i, a_i) \notin Z_{\text{return}}$. Thirdly, since $(s_i, a_i) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$, $(s_i, a_i) \notin \hat{Z}_{\text{safe}}$. Finally, from our induction assumption, we have that $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$; and since π_{return} is a safe policy, it follows that $(s_i, a_i) \notin \hat{Z}_{\text{unsafe}}$. As a result of these four observations, in this else branch, we would add (s_i, a_i) to Z^L_{next}.

Now, consider the instant when the algorithm evaluates the while condition after exiting the for loop that examined (s_{i-1}, a_{i-1}). At this point, by our initial assumption, Z_{explore} is still empty while Z^L_{next} is not as it contains (s_i, a_i). Thus the algorithm would proceed with executing this while loop (as against exiting from it then). Now since $(s_i, a_i) \in Z^L_{\text{next}}$, inside the inner for loop, there must be an iteration when (s_i, a_i) is examined and added to Z_{return}, proving our induction statement.

Thus, consider the for loop iteration when (s_n, a_n) is added to Z_{return}. Since $(s_n, a_n) \in \tilde{Z}_{\text{bad}}$, $\overset{\Delta}{\epsilon}_T(s_n, a_n) > \tau/2$. Hence, we would enter the if-branch of the if-else condition. Again, since $(s_n, a_n) \in \tilde{Z} \setminus \hat{Z}_{\text{safe}}$, from Assumption 5, we know there must exist $(s, a) \in \hat{Z}_{\text{safe}}$ such that $\Delta((s, a), (s_n, a_n)) \leq \tau/4$. As a result, (s, a) will be added to Z_{explore} contradicting the fact that Algorithm 5 exited the while loop without adding any element to Z_{explore}. Thus our initial assumption must be wrong, proving the main claim about Algorithm 5.

Proof for $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$. From the induction assumption, we have that $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$ in the beginning of this while loop. During this loop, $\hat{Z}_{\text{unsafe}}$ is modified by Algorithm 4 only when the call to Algorithm 5 returns with an empty Z_{explore}. In such a case, $\hat{Z}_{\text{unsafe}}$ is modified by adding Z_{edge} to it. Fortunately, from the above discussion, we know that when Z_{explore} is empty, Z_{edge} contains no element from Z_{safe}. Thus, $\hat{Z}_{\text{unsafe}} \cap Z_{\text{safe}} = \emptyset$ even at the end of the while loop.

□

Finally, we show that Algorithm 4 terminates in finite time ensuring that $\overline{Z}_{\text{goal}}$ has been updated in a way that all of it has been established to be safe, or Z_{explore} is non-empty.

Lemma 10. *Algorithm 4 terminates in poly$(|S|, |A|)$ time, after which either $\overline{Z}_{goal} \subset \hat{Z}_{safe}$ or $Z_{explore} \neq \emptyset$.*

Proof. First, we show that, in Algorithm 4, whenever the condition $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ fails, the subsequently computed Z_{edge} is non-empty. Assume for the sake of contradiction that even though $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$, Z_{edge} is empty. Now, recall that $\overline{Z}_{\text{goal}}$ is the set of all state-action pairs visited starting from s_0 following $\overline{\pi}_{\text{goal}}$ in the MDP $\overline{M}_{\text{goal}}$, with transition probabilities $\overline{T}_{\text{goal}}$.

Then, consider any trajectory $(s_0, a_0), (s_1, a_1), \ldots$ of non-zero probability under this corresponding policy and transition function. Since $s_0 \in \hat{Z}_{\text{safe}}$ and Z_{edge} is empty, $(s_0, a_0) \in \hat{Z}_{\text{safe}}$. Then, since $\overline{T}_{\text{goal}} \in CI(\overset{\Delta}{T})$, and since the confidence intervals are admissible, we have from Corollary 2 that $s_1 \in \hat{Z}_{\text{safe}}$. Since Z_{edge} is empty, by a similar argument, we can establish that $(s_1, a_1) \in \hat{Z}_{\text{safe}}$, and

so on for all (s_t, a_t). Since this holds for any trajectory under this policy and transition function, it would mean that $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$, which is a contradiction. Thus, Z_{edge} is indeed non-empty.

Next, we show that Algorithm 4 terminates in polynomial time. Whenever Algorithm 4 does not break out of a loop, then by the design of the algorithm, $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$ and $Z_{\text{explore}} = \emptyset$. In addition to this, Z_{edge} must have been added to $\hat{Z}_{\text{unsafe}}$. Furthermore, from the above argument, since $\overline{Z}_{\text{goal}} \not\subset \hat{Z}_{\text{safe}}$, Z_{edge} must be non-empty. In other words, in each loop that does not break, we take a non-empty subset of $\overline{Z}_{\text{goal}}$, namely Z_{edge} and add it to $\hat{Z}_{\text{unsafe}}$. Note that since we computed $\overline{Z}_{\text{goal}}$ in a way that it does not include any of $\hat{Z}_{\text{unsafe}}$, this also means that $Z_{\text{edge}} \cap \hat{Z}_{\text{unsafe}} \neq \emptyset$. Thus, by the end of this loop, we increase the cardinality of $\hat{Z}_{\text{unsafe}}$. Since $\hat{Z}_{\text{unsafe}}$ cannot be any larger than the finite quantity $|S| \times |A|$, we are guaranteed that no more than $O(|S| \times |A|)$ for loops are run when Algorithm 4 is executed. (In fact, we can say something stronger: no more than $O(|S| \times |A|)$ for loops are run, across multiple calls to Algorithm 4 during the whole run of Algorithm 2).

Finally, observe that, by design of the algorithm, whenever the algorithm terminates, it must have broken out of the for loop. This is possible only if either $\overline{Z}_{\text{goal}} \subset \hat{Z}_{\text{safe}}$ or $Z_{\text{explore}} \neq \emptyset$, thus proving all of our claim. □

4.6.3 Proofs about computing the set of state-actions along the goal path

Recall that $\overline{Z}_{\text{goal}}$ is intended to be the set of state-action pairs visited by the optimistic goal policy under optimistic transitions. To compute this set, we need to know which states have a positive probability of being reached from s_{init} following $\overline{\pi}_{\text{goal}}$, or equivalently the state-actions (s, a) for which $\overline{\rho}_{\text{goal}}(s, a) > 0$. Formally, we must enumerate the set

$$\{(s, a) \in S \times A : \overline{\rho}_{\text{goal}}(s, a) > 0\}.$$

However, computing this as defined is not feasible in finite time (as we must enumerate infinite length trajectories). Instead, recall from Equation 4.6 that we can approximate the above set by only computing the finite-horizon estimate $\overline{\rho}_{\text{goal}}(\cdot, \cdot, H)$ for some horizon H. Fortunately, computing $\overline{Z}_{\text{goal}}$ does not require a good estimate of $\overline{\rho}_{\text{goal}}(\cdot, \cdot)$, but only requires knowing when the $\overline{\rho}_{\text{goal}}(\cdot, \cdot)$ is positive or 0. Lemma 11 shows that as long as $H \geq |S|$, $\overline{\rho}_{\text{goal}}(\cdot, \cdot, H) > 0$ if and only if $\overline{\rho}_{\text{goal}}(s, a) > 0$; as a corollary of which we have that $\overline{Z}_{\text{goal}}$ is exactly what we intend it to be.

Lemma 11. *For any policy π, starting state $s \in S$, and state-action pair $(s', a') \in S \times A$, $\rho^M_{\pi,s}(s', a', H) > 0$ if and only if $\rho^M_{\pi,s}(s', a') > 0$ as long as $H \geq |S|$.*

Proof. First we establish sufficiency. That is, if the finite-horizon estimate is positive, then, so is

the infinite horizon estimate. Consider the following inequality relating these two quantities:

$$\begin{aligned}\rho^M_{\pi,s}(s', a', H) &= (1-\gamma)\sum_{t=0}^{H}\gamma^t \mathrm{P}\left(s_t = s', a_t = a' | \pi, s_0 = s\right) \\ &\leq \lim_{H\to\infty}(1-\gamma)\sum_{t=0}^{H}\gamma^t \mathrm{P}\left(s_t = s', a_t = a' | \pi, s_0 = s\right) \\ &= \rho^M_{\pi,s}(s', a')\end{aligned}$$

The second step uses the fact that $\gamma > 0$ and $\mathrm{P}\left(s_t = s', a_t = a' | \pi, s_0 = s\right) \geq 0$ for all t. Thus, if $\rho^M_{\pi,s}(s', a', H) > 0$, then $\rho^M_{\pi,s}(s', a') > 0$, establishing sufficiency.

Now we will establish necessity i.e., if the infinite horizon estimate was positive, then the same must hold for the finite-horizon estimate. If $\rho^M_{\pi,s}(s', a') > 0$, then there exists at least one sequence of states $(s_0, s_1, \ldots s_n)$ where $s_0 = s$, $s_n = s'$, and $T(s_i, \pi(s_i), s_{i+1}) > 0$ for all $0 \leq i < n$. Without loss of generality, consider the shortest such sequence. Now, for the sake of contradiction, assume that $n > |S|$. By the pigeonhole principle, there exists at least one state that is repeated at least twice. That is, for two indices j, k ($j < k$), we have $s_j = s_k$. Since $s_j = s_k$ and $T(s_k, \pi(s_k), s_{k+1}) > 0$, then $T(s_j, \pi(s_j), s_{k+1}) > 0$. Thus, we can construct a shorter sequence by removing all indices i such that $j < i \leq k$ and this sequence still satisfies the fact that every transition observed has non-zero probability. This contradicts our assertion that this is the shortest such sequence. Thus, $n \leq |S|$. Given $H \geq |S| \geq n$, we have $\rho^M_{\pi,s}(s', a', H) > 0$, establishing necessity. □

As a straightforward corollary of the above, we have:

Corollary 4. *$\overline{Z}_{goal}$ as computed in Equation 4.6 satisfies:*

$$\overline{Z}_{goal} = \{(s, a) \in S \times A : \overline{\rho}_{goal}(s, a) > 0\}.$$

when $H \geq |S|$.

4.6.4 Proofs about goal, explore, and switching policies

This subsection details the key lemmas for proving that ASE is PAC-MDP. The main idea is to show that, under these policies, our agent either will perform the desired behavior, i.e. act ϵ-optimally or reach a desired state-action set, or reach an insufficiently explored state-action pair (i.e., not in K). Since a state-action pair that is experienced m times is added to K, we can bound the number of times we reach a state-action pair outside of K. With this bound and the following lemmas, we can bound the number of times the agent performs undesired behaviors, e.g. acting sub-optimally. We start by proving this claim for $\overline{\pi}_{\text{explore}}$ and $\overline{\pi}_{\text{switch}}$ (Lemma 12), then for $\overline{\pi}_{\text{goal}}$ (Lemma 13 and Lemma 14).

Recall that the $\overline{\pi}_{\text{explore}}$ is based on a reward system where the rewards are non-zero only on state-action pairs that are in $\hat{Z}_{\text{safe}}$ and are not sufficiently explored (i.e., not in K). We first show that if we were to follow the $\overline{\pi}_{\text{explore}}$ policy, in polynomially many steps, we are guaranteed to obtain a

non-zero reward i.e., we are guaranteed to reach a state-action pair not in K. In other words, if we were to follow $\overline{\pi}_{\text{explore}}$, we will definitively obtain a useful sample and learn something new. Similarly, if we were to follow $\overline{\pi}_{\text{switch}}$, we will definitively return to $\overline{Z}_{\text{goal}}$ or learn something new.

Lemma 12. *Assume that the confidence intervals are admissible. Consider an MDP $M^\dagger = \langle S, A, T, R^\dagger, \gamma^\dagger \rangle$, which is the same as the true MDP but with different rewards and discount factor. Let Z be a closed, communicating subset of $S \times A$ and $Z^\dagger$ a non-empty subset of Z. Let $R^\dagger$ be defined such that:*

$$R^\dagger(s,a) = \begin{cases} 1 & (s,a) \in Z^\dagger \\ 0 & (s,a) \in Z \setminus Z^\dagger \\ -\infty & (s,a) \notin Z \end{cases}$$

Let $H = H_{com} \log \frac{16 H_{com}}{c} / \log \frac{1}{c}$ and let $\gamma^\dagger = c^{1/H}$ where $c \in (0, 1/4]$ is a constant. Let $\tilde{H} = \max\left(H \frac{1}{\sqrt{8c}}, \frac{1}{\sqrt{\tau}}\right)$. Let $\overline{Q}^\dagger$ denote the optimistic value function of this MDP, and $\overline{\pi}^\dagger$ be the optimistic policy i.e., $\overline{\pi}^\dagger(s,a) = \arg\max_{a \in A} \overline{Q}^\dagger(s,a)$.

Then, for any $\delta > 0$ and $\epsilon \in (0, c/8]$, starting from any state in Z and following $\overline{\pi}^\dagger$, the agent will reach a state-action pair either in $Z^\dagger$ or outside of $K = \{(s,a) \in S \times A : n(s,a) \geq m\}$, where $m \geq O\left(\tilde{H}^4 |S| + \tilde{H}^4 \ln \frac{|S||A|\tilde{H}^2}{\delta}\right)$, in at most $O(\frac{H^2}{c})$ time steps, with probability at least $1/2$, provided $\overline{\pi}^\dagger \in \Pi(Z)$.

Proof. Consider any $s \in Z$. We will first upper bound the optimistic value $\overline{V}^\dagger(s)$ and then derive a lower bound on it, and then relate these two bounds together to prove our claim. Note that if we let $\overline{M}^\dagger$ be the same MDP as $M^\dagger$, but with the optimistic transitions (the transitions $\overline{T}^\dagger \in CI(\overset{\triangle}{T})$ which maximize the optimistic Q-values), then $\overline{V}^\dagger(\cdot) = \overline{V}^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(\cdot)$.

We will begin by upper bounding the finite-horizon value of $\overline{\pi}^\dagger$ on $M^\dagger$ (and then relate it to its value on $\overline{M}^\dagger$). Let $s_0, s_1, s_2, \ldots,$ denote the random sequence of states visited by the agent by following $\overline{\pi}^\dagger$ from $s_0 = s$ on $M^\dagger$. Then, we have:

$$\begin{aligned} V^{\overline{\pi}^\dagger}_{M^\dagger}(s, H) &= \mathbb{E}\left[\sum_{i=0}^{H} (\gamma^\dagger)^i R^\dagger(s_i, \overline{\pi}^\dagger(s_i))\right] \\ &\leq \mathbb{E}\left[\sum_{i=0}^{H} R^\dagger(s_i, \overline{\pi}^\dagger(s_i))\right] \\ &\leq \mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger) \cdot H \end{aligned}$$

The first inequality follows from the fact that $\gamma^\dagger < 1$. The second inequality follows from the fact that every trajectory of $\overline{\pi}^\dagger$ that experiences a positive cumulative reward, must experience some state-action pair in $Z^\dagger$; and such a trajectory can at best experience a reward of 1 at each timestep.

In the next step we will upper bound the finite-horizon optmistic value of following $\overline{\pi}^\dagger$ in $\overline{M}^\dagger$. To do this, define M' to be an MDP that is identical to $M^\dagger$ on $(s,a) \in K$, and identical to $\overline{M}^\dagger$

everywhere else. Then,

$$\begin{aligned}
V^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(s,H) &\leq V^{\overline{\pi}^\dagger}_{M'}(s,H) + \frac{c}{8} \\
&\leq V^{\overline{\pi}^\dagger}_{M^\dagger}(s,H) + H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \notin K) + \frac{c}{8} \\
&\leq H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger) + H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \notin K) + \frac{c}{8} \\
&\leq 2H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger \cup K^c) + \frac{c}{8}.
\end{aligned}$$

Here, the first inequality follows from Lemma 15 and 16. Specifically, from Lemma 16, we have when $m \geq O\left(\tilde{H}^4|S| + \tilde{H}^4 \ln \frac{|S||A|\tilde{H}^2}{\delta}\right)$, the width of the confidence interval of $(s,a) \in K$ is at most $1/\tilde{H}^2 \leq c/(8H^2)$. Then, Lemma 15 can be used to bound the difference in their value functions by $c/8$.

Note that to apply Lemma 15 we must also ensure that for all $(s,a) \in K$, the support of the next state distribution is the same under M' and $\overline{M}^\dagger$. To see why this is true, observe that for all $(s,a) \in K$, the confidence interval is at most $1/\tilde{H}^2 \leq \tau/2$. Then, since the transition probabilities for $(s,a) \in K$ in $\overline{M}^\dagger$ and M' correspond to $\overline{T}^\dagger \in CI(\hat{\overline{T}})$ and T respectively, Lemma 2 implies that the support of these state-action pairs are indeed the same for these two transition functions. Also note that Lemma 15 implies that these value functions are either close to each other or both equal to $-\infty$; even in the latter case, the above inequalities would hold (although, we will show in the remaining part of the proof that these quantities are lower bounded by some positive value).

The second inequality follows from Lemma 17. Note that in order to apply Lemma 17, we must establish that, with probability 1, the agent experiences only non-negative rewards, when it starts from s and follows $\overline{\pi}^\dagger$ for H steps. This is indeed true because we know that Z is closed and $\overline{\pi}^\dagger \in \Pi(Z)$. Then, we know from Fact 1 that the agent always remains in Z, which means it experiences only non-negative rewards.

The third inequality above uses the upper bound on $V^\pi_{M^\dagger}(s,H)$ that we derived previously.

Having established the above inequalities, we are now ready to upper bound the optimistic value using Lemma 18 as follows:

$$\begin{aligned}
\overline{V}^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(s) &\leq \overline{V}^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(s,H) + \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger} \\
&\leq 2H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger \cup K^c) + \frac{c}{8} + \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger} \qquad (4.13)
\end{aligned}$$

Next, as the second part of our proof, we will derive a lower bound on the optimistic value using Assumption 3. Choose some $(s',a') \in Z^\dagger$ (which is given to be non-empty). Since Z is given to be communicating, by Assumption 3, we know that there exists a policy π_{com} which has a probability of at least $\frac{1}{2}$ of reaching s' from s in H_{com} steps, while visiting only state-action pairs in Z. Without loss of generality, let us assume that $\pi_{\text{com}}(s') = a'$ (since, regardless of what the action at s' is, it guarantees reachability of s' from everywhere else.).

Let us lower bound the value of this policy. Let $s_0, s_1, s_2, \ldots,$ denote the random sequence of states visited by the agent by following π_{com} from $s_0 = s$ on $M^\dagger$. Then:

$$\begin{aligned}
V^{\pi_{\text{com}}}_{M^\dagger}(s) &\geq V^{\pi_{\text{com}}}_{M^\dagger}(s, H_{\text{com}}) \\
&= \mathbb{E}\left[\sum_{i=0}^{H_{\text{com}}} (\gamma^\dagger)^i R^\dagger(s_i, \pi_{\text{com}}(s_i))\right] \\
&\geq (\gamma^\dagger)^{H_{\text{com}}} \mathbb{E}\left[\sum_{i=0}^{H_{\text{com}}} R^\dagger(s_i, \pi_{\text{com}}(s_i))\right] \\
&\geq (\gamma^\dagger)^{H_{\text{com}}} \mathrm{P}(\exists\, i : (s_i, \pi_{\text{com}}(s_i)) \in Z^\dagger) \\
&\geq (\gamma^\dagger)^{H_{\text{com}}} \frac{1}{2}.
\end{aligned}$$

Here, the first step follows from Fact 1 which says that, since $\pi_{\text{com}} \in \Pi(Z)$ and Z is closed, π_{com} only visits state-action pairs in Z, all of which have non-negative $R^\dagger$ reward; as a result, truncating the value function to H_{com} steps only maintains/decreases the value.

The third step follows from the fact that $\gamma^\dagger < 1$. The fourth step comes from the fact that, since π_{com} takes only state-action pairs in Z, $R^\dagger(s_i, \pi_{\text{com}}(s_i)) \in \{0, 1\}$; then, every trajectory with a total non-zero reward has a reward of at least 1. The last step follows from the guarantee of Assumption 3.

Now, as the final step in our proof we note that if our confidence intervals are admissible, by Lemma 19, we know that $V^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(s) \geq V^*_{M^\dagger}(s)$. Furthermore, $V^*_{M^\dagger}(s)$ must be lower bounded by the value of π_{com} on $M^\dagger$, which we just lower bounded. Now, equating this with the upper bound from Equation 4.13, we get:

$$\begin{aligned}
2H\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger \cup K^c) + \frac{c}{8} + \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger} &\geq (\gamma^\dagger)^{H_{\text{com}}} \frac{1}{2} \\
\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger \cup K^c) &\geq \frac{1}{2H}\left((\gamma^\dagger)^{H_{\text{com}}} \frac{1}{2} - \frac{c}{8} - \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger}\right) \\
&\geq \frac{1}{2H}\left(\frac{3c}{8} - \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger}\right) \qquad (4.14)
\end{aligned}$$

Here, in the last step, we make use of the fact that $\gamma^\dagger = c^{1/H_{\text{com}}}$. Next, we will upper bound the last term by making use of the inequality: if $c < 1$, then $\forall x > 0, c^x \leq 1 - x(1-c)$. Then, we get:

$$\frac{1}{1-\gamma^\dagger} = \frac{1}{1-c^{1/H_{\text{com}}}} \leq \frac{H_{\text{com}}}{1-c}$$

Furthermore, since $\log(16H_{\text{com}}/c) = (H/H_{\text{com}})\log(1/c)$, by applying $\exp(\cdot)$ on both sides, we have:

$$\frac{c}{16H_{\text{com}}} = c^{H/H_{\text{com}}} = (\gamma^\dagger)^H$$

From the above two inequalities, we have:

$$\frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger} \leq \frac{(\gamma^\dagger)^H}{1-\gamma^\dagger} \leq \frac{c}{16(1-c)} \leq \frac{c}{8}$$

In the first step above, we make use of $\gamma^\dagger < 1$ and in the second step, $c < 1/2$. Plugging this back in Equation 4.14, we get:

$$\mathrm{P}(\exists\, i \leq H : (s_i, \overline{\pi}^\dagger(s_i)) \in Z^\dagger \cup K^c) \geq \frac{c}{8H}$$

In other words, following policy $\overline{\pi}^\dagger$ for H steps in $M^\dagger$, the agent will reach a state-action pair either in $Z^\dagger$ or outside K with probability at least $c/(8H)$. Then, by the Hoeffding bound applied to multiple subsequent trajectories each of H timesteps, we would have that with probability at least $1/2$, following policy $\overline{\pi}_{\text{goal}}$ for $O(H \cdot \frac{H}{c})$ timesteps, the agent will reach an element either in Z or outside K.

□

In Lemma 13, we show that once our algorithm begins following the optimistic goal policy $\overline{\pi}_{\text{goal}}$, it will continue to do so until it learns something new. (Since we can bound the number of times the agent learns something new, observe that this also means that, eventually, the agent will follow $\overline{\pi}_{\text{goal}}$ for all time.)

Lemma 13. *Assume the confidence intervals are admissible. Then, during the run of Algorithm 2, if the agent is currently following $\overline{\pi}_{goal}$, then it will continue to do so until it experiences a state-action pair outside $K = \{(s,a) \in S \times A : n(s,a) \geq m\}$ when $m \geq O\left(\frac{|S|}{\tau^2} + \frac{1}{\tau^2} \ln \frac{|S||A|}{\tau^2 \delta}\right)$.*

Proof. Assume that during a run of Algorithm 2, the agent is currently at s and takes the action $\overline{\pi}_{\text{goal}}(s)$. By design of Algorithm 2, we have $s \in \overline{Z}_{\text{goal}}$. Recall that $\overline{Z}_{\text{goal}}$ is the set of all state-action pairs that can be visited by the agent if it were to following $\overline{\pi}_{\text{goal}}$ starting from s_0 under the optimistic transitions $\overline{T}_{\text{goal}}$. More formally, we have from Corollary 4 that $\overline{Z}_{\text{goal}} = \{(s,a) \in S \times A : \overline{\rho}_{\text{goal}}(s,a) > 0\}$.

Now, to prove our claim, we only need to argue that if $(s, \overline{\pi}_{\text{goal}}(s)) \in K$, then the next state s' belongs to $\overline{Z}_{\text{goal}}$. Then, by design of Algorithm 2, the agent will take $\overline{\pi}_{\text{goal}}$ even in the next state, proving our claim. To argue this, observe that since $(s, \overline{\pi}_{\text{goal}}(s)) \in K$ and $m \geq O\left(\frac{|S|}{\tau^2} + \frac{1}{\tau^2} \ln \frac{|S||A|}{\tau^2 \delta}\right)$, by Lemma 16, the confidence interval of $(s, \overline{\pi}_{\text{goal}}(s))$ has width at most

$\tau/2$. Then, from Lemma 2, since $T(s, \overline{\pi}_{\text{goal}}(s), s') > 0$ and since $\overline{T}_{\text{goal}} \in CI(\hat{\overline{T}})$, we have $\overline{T}_{\text{goal}}(s, \overline{\pi}_{\text{goal}}(s), s') > 0$. Thus, since we know that $s \in \overline{Z}_{\text{goal}}$, by definition of $\overline{Z}_{\text{goal}}$, s' should also belong to $\overline{Z}_{\text{goal}}$. □

In the following lemma, we show that in the MDP M_{goal} (which is the same as the original MDP but with the unsafe state-action pairs set to $-\infty$ rewards), when we follow the optimistic goal policy $\overline{\pi}_{\text{goal}}$, we either take a near-optimal action (with respect to M_{goal}) or we experience an action outside of K with sufficient probability in the next H steps.

Lemma 14. *Assume the confidence intervals are admissible. Consider any instant when the agent has taken a trajectory p_t and is at state s_t, and the Algorithm 2 instructs the agent to follow $\overline{\pi}_{goal}$. Let $\mathrm{P}(A_M)$ be the probability that starting at this step, the Algorithm 2 leads the agent out of $K = \{(s,a) \in S \times A : n(s,a) \geq m\}$ in H steps, conditioned on p_t. Then, for any $\epsilon, \delta \in (0,1)$, and for $H = O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}\right)$ and $m \geq O\left(\frac{1}{\epsilon^2(1-\gamma)^4}\left(\frac{|S|}{\tilde{\epsilon}} + \frac{1}{\tilde{\epsilon}^2} \ln \frac{|S||A|}{\tilde{\epsilon}^2 \delta}\right)\right)$, where $\tilde{\epsilon} = O\left(\min\left(\frac{\tau}{2}, \frac{\epsilon(1-\gamma)^2}{3}\right)\right)$, we have:*

$$V_M^{\mathcal{A}}(p_t) \geq V_{M_{goal}}^*(s_t) - \frac{\epsilon}{2} - 2\frac{\mathrm{P}(A_M)}{1-\gamma}.$$

Proof. While we follow the general outline of the proof of Theorem 1 from Strehl and Littman [99], we note that there are crucial differences for incorporating safety (such as dealing with rewards of $-\infty$).

At the outset, we establish two useful inequalities. First, since $H = O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}\right)$, by Lemma 18, we have that:

$$V_{\overline{M}_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s_t, H) \geq V_{\overline{M}_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s_t) - \frac{\epsilon}{3} \tag{4.15}$$

Secondly, let M'_{goal} be an MDP that is equivalent to M_{goal} for all state-action pairs in K and equal to $\overline{M}_{\text{goal}}$ otherwise. (Note that all these MDPs have the same reward function, namely R_{goal}.) We claim that for all a:

$$Q_{M'_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s_t, a, H) \geq Q_{\overline{M}_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s_t, a, H) - \frac{\epsilon}{3} \tag{4.16}$$

Let us see why this inequality holds. From Lemma 16, we have when $m \geq O\left(\frac{1}{\epsilon^2(1-\gamma)^4}\left(\frac{|S|}{\tilde{\epsilon}} + \frac{1}{\tilde{\epsilon}^2} \ln \frac{|S||A|}{\tilde{\epsilon}^2 \delta}\right)\right)$, the width of the confidence interval of $(s,a) \in K$ is at most $O(\tilde{\epsilon}) = \frac{\epsilon}{3}\frac{(1-\gamma)^2}{\gamma}$. Then, Lemma 15 can be used to bound the difference in the value functions by $\epsilon/3$ as above.

Note that to apply Lemma 15, we must also ensure that for all $(s,a) \in K$, the support of the next state distribution is the same under M'_{goal} and $\overline{M}_{\text{goal}}$. To see why this is true, observe that for all $(s,a) \in K$, the width of the confidence interval is at most $O(\tilde{\epsilon}) = \tau/2$. Then, since the transition probability for $(s,a) \in K$ in $\overline{M}^{\dagger}$ and M' correspond to $\overline{T}^{\dagger} \in CI(\hat{\overline{T}})$ and T respectively, Lemma 2 implies that the support of these state-action pairs are indeed the same for these two transition functions. Also note that Lemma 15 implies that these value functions are either ϵ_1-close to each other or both equal to $-\infty$; even in the latter case, the above inequality would hold (although,

to be precise, this latter case does not really matter since Theorem 4 eventually shows that these quantities are lower bounded by a positive quantity).

Having established the above inequalities, we now begin lower bounding the value of the algorithm. First, since the rewards $R(\cdot, \cdot)$ are bounded below by -1, and since $H = O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}\right)$, by Lemma 18, we can lower bound the infinite-horizon value of the algorithm by its finite-horizon value as

$$V_M^{\mathcal{A}}(p_t) \geq V_M^{\mathcal{A}}(p_t, H) - \frac{\epsilon}{3}. \tag{4.17}$$

Next, we claim to bound the value of following the algorithm for the next H steps as follows:

$$V_M^{\mathcal{A}}(p_t, H) \geq V_{M'_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s, H) - 2\frac{\mathrm{P}\left(A_M\right)}{1-\gamma}. \tag{4.18}$$

In other words, we have lower bounded the value of following the algorithm on M, in terms of following $\overline{\pi}_{\text{goal}}$ on M'_{goal}. Let us see why this is true. For the sake of convenience, let us define two cases corresponding to the above inequality: Case A, where the agent follows Algorithm 2 to take actions for H steps starting from s_t in MDP M and Case B, where the agent follows the fixed policy $\overline{\pi}_{\text{goal}}$ to take actions for H steps starting from s_t in MDP M'_{goal}.

Now, recall that we are considering a state s_t where Algorithm 2 currently follows $\overline{\pi}_{\text{goal}}$, and will continue to follow it until it takes an action in K^c. Then, consider a particular random seed for which, in Case A, the agent does not reach K^c.

We argue that for this random seed, the agent will see the same cumulative discounted reward over H steps in both Case A and Case B. To see why, recall that M'_{goal} and M share the same transition functions on K. Next, since in Case A, the agent does not escape K, by Lemma 13, we know that the agent follows only $\overline{\pi}_{\text{goal}}$ for H steps. Thus in both these cases, the agent experiences the same sequence of state-action pairs for H steps. It only remains to argue that these state-action pairs have the same rewards in both cases. To see why, recall that by design of Algorithm 2, since the agent does not escape K, all these state-action pairs would belong to $\overline{Z}_{\text{goal}}$ which in turn is a subset of $\hat{Z}_{\text{safe}}$. Now the MDP M and M'_{goal} share the same rewards on $\hat{Z}_{\text{safe}}$; this is because, their rewards differ only in $\hat{Z}_{\text{unsafe}}$, and we know $\hat{Z}_{\text{unsafe}} \cup Z_{\text{safe}} = \emptyset$ (Lemma 9). Thus, in both cases, the agent experiences the same sequence of rewards.

As a result, the value functions in Case A and B differ only due to trajectories where the algorithm either leads the agent to K^c in H steps or leads the agent to negative rewards any time in the future. Hence, we can upper bound the difference $V_{M'_{\text{goal}}}^{\overline{\pi}_{\text{goal}}}(s_t, H) - V_M^{\mathcal{A}}(p_t, H)$ in terms of $\mathrm{P}(A_M)$ multiplied by the maximum difference between the respective cumulative rewards. We know that the cumulative reward in Case A is at least $-1/(1-\gamma)$, because the rewards $R(\cdot, \cdot)$ are bounded below by -1. On the other hand, in Case B, the cumulative reward experienced is at most $1/(1-\gamma)$, assuming the agent receives a reward of 1 for each step. From this, we establish Equation 4.18.

Subsequently, we further lower bound $V^{\overline{\pi}_{\text{goal}}}_{M'_{\text{goal}}}(s_t, H)$ as follows:

$$\begin{aligned} V^{\overline{\pi}_{\text{goal}}}_{M'_{\text{goal}}}(s_t, H) &\geq V^{\overline{\pi}_{\text{goal}}}_{\overline{M}_{\text{goal}}}(s_t, H) - \frac{\epsilon}{3} \\ &\geq V^{\overline{\pi}_{\text{goal}}}_{\overline{M}_{\text{goal}}}(s_t) - 2\frac{\epsilon}{3} \\ &\geq V^{*}_{M_{\text{goal}}}(s_t) - \epsilon \end{aligned}$$

Here, the first inequality comes from Equation 4.16. The second inequality comes from Equation 4.15. The last step comes from Lemma 19 (given admissibility). Then, by combining the above inequality with Equations 4.18 and 4.17, we get our final result. □

4.6.5 Supporting lemmas for showing PAC-MDP

The following lemmas are necessary for proving our algorithm is PAC-MDP. Note that most of these lemmas are similar to lemmas from Strehl and Littman [99]. However, because we construct MDPs with infinitely negative rewards, additional care must be taken to ensure that these properties still hold.

Here we provide a quick description of the lemmas detailed in this section. We start with Lemma 15, which shows that if two MDPs have sufficiently similar transition functions, the optimal Q-values on these two MDPs must also be similar. This, together with Lemma 16, allows us to show that, if we have sufficiently explored the state-space, we can accurately estimate the optimal policy on any MDP. Next we show, in Lemma 17, that the difference between the value functions of the true and an estimated MDP for a given policy is proportional to the probability of reaching an under-explored state-action pair, i.e. a state-action pair outside of K. This allows us to claim that either the probability of reaching an element outside of K is sufficiently large, or our estimated value function is sufficiently accurate. Lemma 18 bounds the difference between the finite horizon and infinite horizon value functions, allowing us to consider only finite length trajectories. Lemma 19 simply shows that our optimistic value function always over-estimates the true value function (given that our confidence intervals are admissible).

Lemma 15. *Let $M_1 = \langle S, A, T_1, R^\dagger, \gamma^\dagger \rangle$ and $M_2 = \langle S, A, T_2, R^\dagger, \gamma^\dagger \rangle$ be two MDPs with identical rewards that either belong to $[0, 1]$ or equal $-\infty$ and $\gamma^\dagger < 1$. Let K be a subset of state-action pairs such that*

1. *for all $(s, a) \notin K$, $T_1(s, a, \cdot) = T_2(s, a, \cdot)$,*
2. *for all $(s, a) \in K$, $\|T_1(s, a, \cdot) - T_2(s, a, \cdot)\|_1 \leq \beta$ and*
3. *for all $(s, a) \in K$, the next state distribution (s, a) has identical support under both T_1 and T_2.*

Then, for any (stationary, deterministic) policy π, and for any (s, a) and any $H \geq 0$, we have that, either:

$$\left|Q^\pi_{M_1}(s, a, H) - Q^\pi_{M_2}(s, a, H)\right| \leq \min\left(\frac{\gamma^\dagger \beta}{(1 - \gamma^\dagger)^2}, \beta H^2\right).$$

or

$$Q^{\pi}_{M_1}(s,a,H) = Q^{\pi}_{M_2}(s,a,H) = -\infty.$$

Proof. First, we note that for any (s,a) such that $R^{\dagger}(s,a) = -\infty$, $Q^{\pi}_{M_1}(s,a,H) = Q^{\pi}_{M_2}(s,a,H) = -\infty$ for all H. Hence, for the rest of the discussion, we will consider (s,a) such that $R^{\dagger}(s,a) \neq -\infty$.

We prove our claim by induction on H. For $H = 1$, for all (s,a), $Q^{\pi}_{M_1}(s,a,H) = Q^{\pi}_{M_2}(s,a,H) = R^{\dagger}(s,a)$.

Consider any arbitrary H. First, we show that, if there exists a next state s' in the support of $T_1(s,a,\cdot)$ such that $Q^{\pi}_{M_1}(s',\pi(s'),H-1) = -\infty$, then $Q^{\pi}_{M_1}(s,a,H) = Q^{\pi}_{M_2}(s,a,H) = -\infty$. Note that, by conditions 1 and 3 of the Lemma statement, we have that for all (s,a), regardless of whether in K or not, the support of the next state distribution is identical between T_1 and T_2. Hence, if there exists a next state s' in the support of T_1 such that $Q^{\pi}_{M_1}(s',\pi(s'),H-1) = -\infty$, then s' would belong even to the support of T_2 and, thus, we would have that $Q^{\pi}_{M_2}(s',\pi(s'),H-1) = -\infty$. Hence, by definition of Q-values, we would have $Q^{\pi}_{M_1}(s,a,H) = Q^{\pi}_{M_2}(s,a,H) = -\infty$.

Now consider a case where none of the next states s' in the support of T_1 (and T_2, without loss of generality) have Q-value $Q^{\pi}_{M_1}(s',\pi(s'),H-1) = -\infty$. We will prove by induction that in this case,

$$\left|Q^{\pi}_{M_1}(s,a,H) - Q^{\pi}_{M_2}(s,a,H)\right| \leq \frac{\gamma^{\dagger}\beta}{1-\gamma^{\dagger}} \cdot \frac{1-\gamma^{\dagger H}}{1-\gamma^{\dagger}}.$$

Note that when $H = 0$, the right hand side above resolves to zero, which is indeed true.

Consider any $H > 0$. Now, observe that for all (s,a), regardless of whether in K or not, we have $\|T_1(s,a,\cdot) - T_2(s,a,\cdot)\|_1 \leq \beta$. Besides, since $R^{\dagger}(s,a) \neq -\infty$, we can upper bound the difference in the Q values as follows:

$$\begin{aligned}
&|Q^{\pi}_{M_1}(s,a,H) - Q^{\pi}_{M_2}(s,a,H)| \\
&= \gamma^{\dagger}\left|\sum_{s'} T_1(s,\pi(s'),s')Q^{\pi}_{M_1}(s',\pi(s'),H-1) - \sum_{s'} T_2(s,\pi(s'),s')Q^{\pi}_{M_2}(s',\pi(s'),H-1)\right| \\
&\leq \gamma^{\dagger}\left|\sum_{s'} T_1(s,\pi(s'),s')\left(Q^{\pi}_{M_1}(s',\pi(s'),H-1) - Q^{\pi}_{M_2}(s',\pi(s'),H-1)\right)\right| \\
&\quad + \gamma^{\dagger}\left|\sum_{s'} \left(T_1(s,\pi(s'),s') - T_2(s,\pi(s'),s')\right) Q^{\pi}_{M_2}(s',\pi(s'),H-1)\right| \\
&\leq \gamma^{\dagger}\left(\frac{\gamma^{\dagger}\beta}{1-\gamma^{\dagger}} \cdot \frac{1-\gamma^{\dagger H-1}}{1-\gamma^{\dagger}}\right) + \gamma^{\dagger}\frac{\beta}{1-\gamma^{\dagger}} \\
&= \frac{\gamma^{\dagger}\beta}{1-\gamma^{\dagger}}\left(\frac{\gamma^{\dagger}(1-\gamma^{\dagger H-1})}{1-\gamma^{\dagger}} + 1\right) \\
&= \frac{\gamma^{\dagger}\beta}{1-\gamma^{\dagger}} \cdot \frac{1-\gamma^{\dagger H}}{1-\gamma^{\dagger}}
\end{aligned}$$

Here, the second step follows by a simple algebraic rearrangement that decomposes the difference in the Q-values, in terms of the difference in the transitions and the difference in the next-state Q-values. In the third step, for the first term, we make use of the fact that in this case, the next state s' has a Q-value that is not $-\infty$; this means that $(s', \pi(s'))$ does not have any next states with Q-value $-\infty$ and therefore the induction assumption holds. By applying the induction assumption for $H-1$, we get the first term. For the second term, we make use of the fact that the total sum of transition probabilities equals 1. Furthermore, we also make use of the fact the maximum magnitude of the Q-value is at most $\frac{1}{1-\gamma^\dagger}$ if it is not $-\infty$; this is because, if the Q-value is not $-\infty$, it is a discounted summation of expected rewards that lie between 0 and 1.

Hence, our induction hypothesis is true. Our main upper bound can then be established by noting that $1 - \gamma^{\dagger H} < 1$.

To prove our other upper bound, we consider the induction hypothesis:

$$\left|Q^\pi_{M_1}(s, a, H) - Q^\pi_{M_2}(s, a, H)\right| \leq \beta H^2.$$

Then, in the third step above, we would instead have:

$$\begin{aligned} |Q^\pi_{M_1}(s, a, H) - Q^\pi_{M_2}(s, a, H)| &\leq \gamma^\dagger \left(\beta(H-1)^2\right) + \gamma^\dagger \beta H \\ &\leq \beta(H-1)^2 + \beta H \\ &\leq \beta H^2. \end{aligned}$$

To get the first term on the right hand side, we again make use of the induction assumption. For the second term, we simply upper bound the sum of the maximum discounted rewards to be H. Finally, we make use of the fact that $\gamma^\dagger < 1$ and $H^2 - (H-1)^2 \geq 2H - 1 \geq H$ when $H > 0$. □

Lemma 16. *Suppose that as input to Algorithm 2, we set $\delta_T = \delta/(2|S||A|m)$ and all confidence intervals computed by our algorithm are admissible. Then, for any $\beta > 0$, there exists an $m = O\left(\frac{|S|}{\beta^2} + \frac{1}{\beta^2} \ln \frac{|S||A|}{\beta\delta}\right)$ such that $\|\hat{T}(s, a, \cdot) = T(s, a, \cdot)\|_1 \leq \beta$ holds for all state-action pairs (s, a) that have been experienced at least m times.*

Proof. See Lemma 5 from Strehl and Littman [99]. □

Lemma 17. *Let $M^\dagger = \langle S, A, R^\dagger, T, \gamma^\dagger \rangle$ be an MDP that is the same as the true MDP M, but with arbitrary rewards $R^\dagger$ (bounded above by 1) and discount factor $\gamma^\dagger < 1$. Let K be some set of state-action pairs. Let $M' = \langle S, A, R^\dagger, T', \gamma^\dagger \rangle$ be an MDP such that T' is identical to T on all elements inside K. Consider a policy π. Let $A_{M^\dagger}$ be the event that a state-action pair not in K is encountered in a trial generated by starting from state s_1 and following π for H steps in $M^\dagger$ (where H is a positive constant). If, with probability 1, the agent starting at s_1 and following π in $M^\dagger$ will receive only non-negative rewards then,*

$$V^\pi_M(s_1, H) \geq V^\pi_{M'}(s_1, H) - \min\left(\frac{1}{(1-\gamma^\dagger)}, H\right) \mathrm{P}(A_M).$$

Proof. This proof is similar to that of Lemma 3 from Strehl and Littman [99]. The only aspect we need to be careful about is the magnitude of the rewards.

The two MDPs $M^\dagger$ and M' differ only in their transition functions, and moreover, only outside the set K. Then, observe that, for a fixed random seed, if the agent were to follow π starting from s_1, it would receive the same cumulative reward for H steps in both $M^\dagger$ and M', if it remained in K for all those H steps. In other words, the value of π in these two MDPs differs only because of those random seeds which led the agent out of K in $M^\dagger$. Thus, the difference $V^\pi_{M'}(s_1, H) - V^\pi_{M^\dagger}(s_1, H)$ cannot be any larger than the respective cumulative rewards. Our claim then follows by lower bounding the cumulative reward in $M^\dagger$ and upper bounding the cumulative reward in M'. More concretely, note that cumulative reward in $M^\dagger$ can not be any lower than zero, since we assume that the agent receives rewards bounded in $[0, 1]$. On the other hand, in M', the agent can receive a reward of 1 in all H steps, so the value function can be upper bounded by $\min\left(\frac{1}{(1-\gamma^\dagger)}, H\right)$.

□

Lemma 18. *Consider an MDP $M^\dagger = \langle S, A, T^\dagger, R^\dagger, \gamma^\dagger\rangle$ with rewards bounded above by 1, and a stationary or non-stationary policy π and state s. Then, for any $H \geq 0$, we have:*

$$V^\pi_{M^\dagger}(s, H) \geq V^\pi_{M^\dagger}(s) - \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger}.$$

As a corollary of this, for $H \geq \frac{1}{1-\gamma^\dagger}\ln\frac{1}{\epsilon(1-\gamma^\dagger)}$, we have:

$$V^\pi_{M^\dagger}(s, H) \geq V^\pi_{M^\dagger}(s) - \epsilon.$$

By the same argument, in the case that the rewards are bounded below by -1,

$$V^\pi_{M^\dagger}(s, H) \leq V^\pi_{M^\dagger}(s) + \epsilon.$$

Proof. This proof follows the proof of Lemma 2 from Kearns and Singh [61].

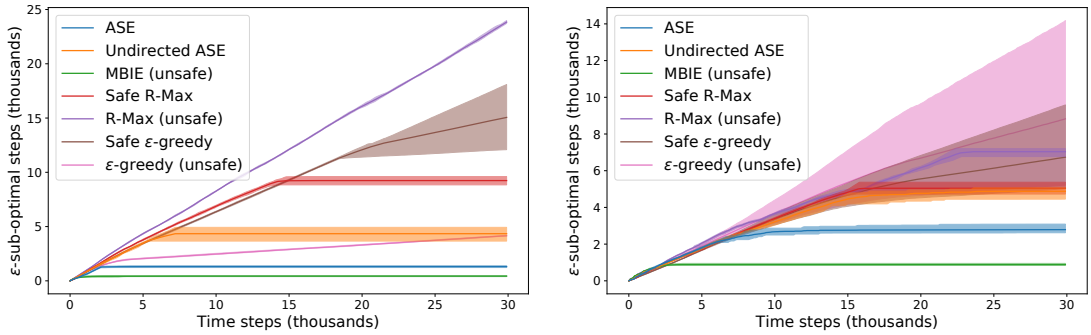
Observe that by truncating any trajectory to H steps, the cumulative discounted reward for this trajectory can drop by a value of at most:

$$\sum_{t=H+1}^{\infty}(\gamma^\dagger)^t = \frac{(\gamma^\dagger)^{H+1}}{1-\gamma^\dagger},$$

which happens when it receives a reward of 1 at every time step after H. Thus for any H such that the above quantity is lesser than or equal to ϵ, we will have $V^\pi(s, H) \geq V^\pi(s) - \epsilon$. This is indeed true for $H \geq \frac{1}{1-\gamma^\dagger}\ln\frac{1}{\epsilon(1-\gamma^\dagger)}$. □

Lemma 19. *Suppose that all confidence intervals are admissible. Let $M^\dagger = \langle S, A, T, R^\dagger, \gamma^\dagger\rangle$ be the same MDP as M except with arbitrary rewards that are upper bounded by 1 and discount factor $\gamma^\dagger < 1$. Let $\overline{\pi}^\dagger$ denote the optimal policy i.e., $\forall s$, $\overline{\pi}^\dagger(s) = \arg\max_{a\in A}\overline{Q}^\dagger(s, a)$. Let $\overline{M}^\dagger$ denote the optimal MDP. Then, for all $H \geq 0$ and for all (s, a), we have:*

$$V^{\overline{\pi}^\dagger}_{\overline{M}^\dagger}(s) \geq V^*_{M^\dagger}(s)$$



(a) Unsafe Grid World

(b) Discrete Platformer

Figure 4.1: Number of ϵ-sub-optimal steps taken by each agent throughout training. Lines denote averages over five trials and shaded regions mark the max and min.

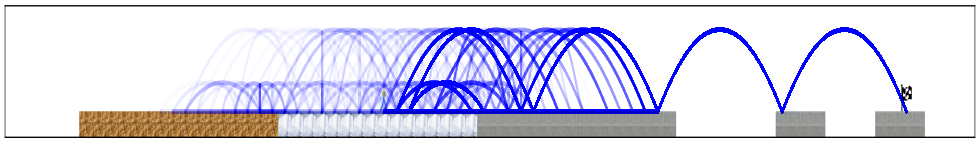
Proof. See Lemma 6 from Strehl and Littman [99]. Note that this proof also applies to MDPs with negative rewards.

□

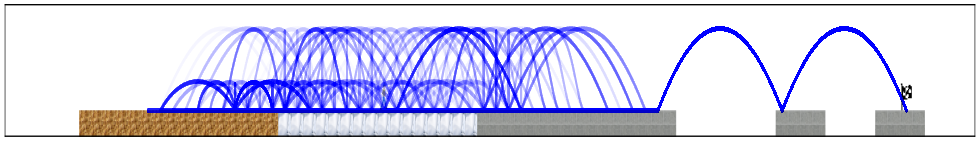
4.7 Experiments

Through experiments, we aim to show that (a) ASE effectively guides exploration, requiring significantly less exploration than exhaustive exploration methods, and (b) the agent indeed never reaches a dangerous state under realistic settings of the parameters (namely m and δ_T in Alg 2). Below, we outline our experiments. For our experiments, we consider two environments. The first is a stochastic grid world containing five islands of grid cells surrounded by "dangerous states" (i.e., states where all actions result in a negative reward). The agent can take actions that allow it to jump over dangerous states to transition between islands and reach the goal state. The second is a stochastic platformer game where certain actions can doom the agent to eventually reaching a dangerous state by jumping off the edge of a platform. In the grid world environment, two state-actions are analogous if the actions are equivalent and the states are near each other (L_∞ distance), and in the stochastic platformer game, if the actions and all attributes but the horizontal position in the state are equivalent and the two states are on the same "surface type".

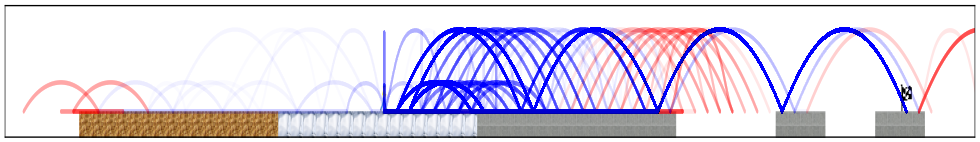
We compare the behavior of our algorithm against both "unsafe" and "safe" approaches to learning reward-based policies. For the unsafe baselines, we consider the original (unsafe) MBIE algorithm [99], R-Max [22], and ϵ-greedy, all adapted to use the analogy function (without which, exploring would take prohibitively long). For safe baselines, unfortunately, there is no existing algorithm because no prior work has simultaneously addressed the two objectives of provably safe exploration and learning a reward-based policy in environments with unknown stochastic dynamics. To this end, we create safe versions of R-Max and ϵ-greedy (by restricting the allowable set of actions the agent can take to $\hat{Z}_{\text{safe}}$, and using analogies to expand $\hat{Z}_{\text{safe}}$), and also consider an "Undirected ASE," which is a naïver version of ASE that expands $\hat{Z}_{\text{safe}}$ in all directions (not



(a) ASE



(b) Safe R-Max



(c) MBIE (unsafe)

Figure 4.2: All trajectories of different agents on the Discrete Platformer domain. Unsafe trajectories are drawn in red. The brown, white, and grey squares correspond to the different surface types: sand, ice, and concrete, respectively. The agent starts in the center of the leftmost island. The flag represents the goal state.

just along the goal policy).

Source code for the experiments is available at `https://github.com/locuslab/ase`.

4.7.1 Unsafe Grid World

The first domain we consider is a grid world domain with dangerous states, where the agent receives a reward of -1 for any action and the episode terminates. The agent starts on a 7×7 island of safe states and is surrounded by four 5×5 islands of safe states in all four directions, separated from the center island by a one-state-thick line of dangerous states (see Figure 4.3). The goal is placed on one of the surrounding islands. The agent can take actions up, down, left, or right to move in those directions one step, or can take actions jump up, jump down, jump left, or jump right to move two steps, allowing the agent to jump over dangerous states. There is a slipping probability of 60%, which causes the agent to fall left or right of the intended target (30% for either side).

The initial safe set provided to the agent is the whole center island (except for the corners) and all actions that with probability 1 will keep the agent on the center island. The distance function Δ provided to the agent is $\Delta((s,a),(\tilde{s},\tilde{a})) = 0$ if $a = \tilde{a}$ and s and $\tilde{s}$ are within 5 steps from each other (in L_∞ norm) and $\Delta((s,a),(\tilde{s},\tilde{a})) = 1$ otherwise. The analogous state function α is simply $\alpha((s,\cdot,s'),(\tilde{s},\cdot)) = (x_{s'} + (x_{\tilde{s}} - x_s), y_{s'} + (y_{\tilde{s}} - y_s))$, where the subscripts denote the state to which the attribute belongs.

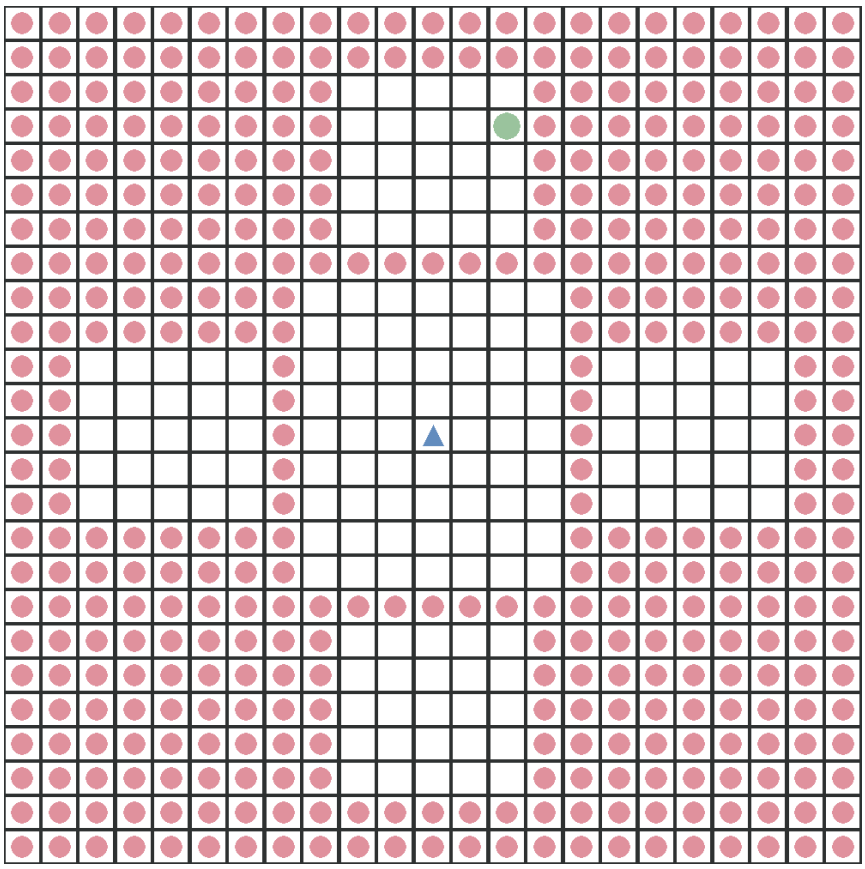


Figure 4.3: Full map of the Unsafe Grid World environment. The green circle marks the goal, the blue triangle marks the initial location of the agent s_{init}, and red circles correspond to dangerous states.

4.7.2 Discrete Platformer

We also consider a more complicated discrete Platformer domain. The states space consists of tuples $(x, y, \dot{x}, \dot{y})$ where x, y are the coordinates of the agent and $\dot{x}, \dot{y}$ are the directional velocities of the agent. The actions provided to the agent are the tuple $(\dot{x}_{\text{desired}}, j)$ where $\dot{x}_{\text{desired}}$ is the desired $\dot{x}$ and ranges from -2 to 2, and j is a boolean indicating whether or not the agent should jump. While on the ground, at every step $\dot{x}$ changes by at most 1 in the direction of $\dot{x}_{\text{desired}}$ and $\dot{y} \in \{1, 2\}$ if $j = 1$ (otherwise $\dot{y}$ remains unchanged). While in the air, however, the agent's actions have no effect and gravity decreases $\dot{y}$ by one at every step. When the agent returns to the ground, $\dot{y}$ is set to 0.

There are three types of surfaces in the environment: 1) concrete, 2) ice, and 3) sand. These surfaces change how high the agent can jump. On concrete, when the agent jumps, $\dot{y} = 2$ with probability 1; on ice $\dot{y} = 2$ with probability 0.5 and $\dot{y} = 1$ with probability 0.5; and on sand $\dot{y} = 1$ with probability 1.

The environment is arranged into three islands. The first island has all three surface materials from left to right: sand, ice, then concrete. The next two islands are just concrete, with the last one containing the goal state (where the reward is 1). The regions surrounding these island are unsafe, meaning they produce rewards of -1 and are terminal. The islands are spaced apart such that the agent must be on concrete to make the full jump to the next islands (and visa versa).

The initial safe set provided to the agent is the whole first island and all actions that with probability 1 will keep the agent on the center island. The distance function Δ provided to the agent is $\Delta((s, a), (\tilde{s}, \tilde{a})) = 0$ if $a = \tilde{a}$ and s and $\tilde{s}$ are either both in the air or both on the same type of surface and $\Delta((s, a), (\tilde{s}, \tilde{a})) = 1$ otherwise. The analogous state function α is simply $\alpha((s, \cdot, s'), (\tilde{s}, \cdot)) = \tilde{s'}$ where $\tilde{s'}$ has the same y, $\dot{x}$, and $\dot{y}$ values as s' with the x value shifted by the x difference between s and $\tilde{s}$.

4.7.3 Baselines

Here we describe the details for the baselines we compare against. We note that all these baselines make use of the distance metric and analogous state function to transfer information between different states, just like our algorithm. For all of our "unsafe" algorithms, we set all negative rewards to be very large to ensure that they converged to the safe-optimal policy. To improve the runtime of the experiments, the value functions and safe sets are only re-computed every 100 time steps.

MBIE MBIE [99] is a guided exploration algorithm that always follows a policy that maximizes an optimistic estimate of the optimal value function. As noted above, one of the motivations of our method was to construct a safe version of MBIE.

R-Max and Safe R-Max The next algorithm we compare against is R-Max [22]. This algorithm sets the value function for all state-action pairs that have been seen fewer than m times (for some integer m) to be equal to $V_{\max}$, the maximum value the agent can obtain. In order to ensure

that all states are sufficiently explored and still make use of the analogous state function, we set the value of any state-action pair, (s, a), to $V_{\max} = 1$ (since all goal states are terminal) only if there is a state-action pair similar to (s, a) with a transferred confidence interval length greater than some $\epsilon' > 0$. In mathematical terms, all state-action pairs in $\{(s, a) \in S \times A : \exists(\tilde{s}, \tilde{a}) \in S \times A$ where $\overset{\Delta}{\epsilon}_T (s, a) < \epsilon'$ and $\Delta((s, a), (\tilde{s}, \tilde{a})) < \tau/2\}$ are set to $V_{\max}$. Clearly, this requires at most every state to be explored m times, but in most cases decreases the number of times each state-action pair needs to be explored. In our experiments we set $\epsilon' = \tau/2$ to give R-Max the most generous comparison against ASE, since ASE requires that a state-action only have a confidence interval of $\tau/2$ before it can be marked as safe. However, note that in many problems $\tau/2$ may be much larger than the desired confidence interval.

Our safe modification of this algorithm, "Safe R-Max," simply restricts the allowable set of actions the agent can take to $\hat{Z}_{\text{safe}}$.

ϵ-greedy and Safe ϵ-greedy Another classic algorithm we compare against is ϵ-greedy. This algorithm acts according to the optimal policy over its internal model at every time step with probability $1 - \epsilon$ and with probability ϵ the agent takes a random action. For our experiments we anneal ϵ between 1 and 0.1 for the first N number of steps ($N = 5{,}000$ for the unsafe grid world and $N = 20{,}000$ for the discrete platformer game). Our safe modification of this algorithm, "Safe ϵ-greedy," simply restricts the allowable set of actions the agent can take to $\hat{Z}_{\text{safe}}$.

Undirected ASE We also compare against a modified version of our algorithm "Undirected ASE." This modification changes Algorithm 4 such that $Z_{\text{edge}} \leftarrow \{(s, a) \in \hat{Z}^c_{\text{safe}} \mid s \in \hat{Z}_{\text{safe}}\}$, removing the use of $\overline{Z}_{\text{goal}}$. With this change, "Undirected ASE" simply tries to expand the safe set in all directions, instead of only along the direction of the optimistic goal policy. This baseline is to illustrate the efficacy of using our directed exploration method.

4.7.4 Results

To measure efficiency of exploration, we count the number of ϵ-sub-optimal steps taken by each agent. To calculate this, we first compute the true safe-optimal Q-function, $Q_M^{\pi^*_{\text{safe}}}$. We then count the number of ϵ-sub-optimal actions taken by the agent, namely the number of times the agent is at a state s_t and takes an action a_t such that $Q_M^{\pi^*_{\text{safe}}}(s_t, a_t) < \max_{a \in A} Q_M^{\pi^*_{\text{safe}}}(s_t, a) - \epsilon$, where $\epsilon = 0.01$. Figure 4.1 shows our algorithm takes far fewer ϵ-sub-optimal actions before it converges compared to all other *safe* algorithms. As for safety, during our experiments, we observe that, in both domains, the safe algorithms do not reach any unsafe states. In the unsafe grid world domain, the MBIE, R-Max, and ϵ-greedy algorithms encounter an average of 85, 5,016, and 915 unsafe states, respectively, and in the discrete platformer game encounter 83, 542, and 768 unsafe states.

In the platformer domain, as we can see from Figure 4.2, our method explores only the necessary parts of the initial safe-set, the right side, unlike the Safe R-Max algorithm. Although standard MBIE also directs exploration, it has many trajectories that end in unsafe states, which ASE avoids.

4.8 Conclusion

We introduced Analogous Safe Exploration (ASE), an algorithm for safe and guided exploration in unknown, stochastic environments using analogies. We proved that, with high probability, our algorithm never reaches an unsafe state and converges to the optimal policy, in a PAC-MDP sense. To the best of our knowledge, this is the first provably safe and optimal learning algorithm for stochastic, unknown environments (specifically, safe during exploration). Finally, we illustrated empirically that ASE explores more efficiently than other non-guided methods. Future directions for the this line of work include extensions to continuous state-action spaces, combining the handling of stochasticity we present here with common strategies in these domains such as kernel-based nonlinear dynamics.

Chapter 5

Projected Off-Policy Q-Learning (POP-QL) for Stabilizing Offline Reinforcement Learning

Safe exploration is a compelling method to safely learn control policies. However, while able to provably provide safety, these methods all need significant prior knowledge of the state-action space, which is often prohibitive in practice. An alternative method for safely learning a control policy is to use offline RL. Many real world control system domains, including autonomous manufacturing and autonomous driving, often have an enormous corpus of control data collected either using human controllers or sub-optimal linear controllers. Could this data be used to learn controllers that improve performance over the currently active controllers? Offline RL is a subfield of RL concerned with learning the best possible policy given a fixed dataset. However, as discussed in the Chapter 2, offline RL presents new challenges over traditional online RL.

Because Model-Based methods can learn completely independently of the data-collection policy, they seem like the obvious choice for offline RL. However, because of the issue of compounding errors, TD methods, such as Conservative Q-Learning (CQL) [69], tend to significantly outperform model-based methods on standard offline RL benchmarks, especially in long-horizon tasks. Thus, TD methods remain the state-of-the-art methods for high-performance in offline RL.

When applied offline, however, TD methods using function approximation are inherently unstable. This is due to the *deadly triad* phenomenon, discussed in Chapter 2. To get around this issue, most methods perform some sort of behavior regularization to keep the learned policy close to the data-collection policy. Unfortunately, this regularization comes at the cost of performance, especially when the data-collection policy is severely sub-optimal.

In this chapter, we propose our method Projected Off-Policy Q-Learning (POP-QL), that modifies the sampling distribution and minimally regularizes the policy in order to stabilize off-policy updates. This work allows us to apply safely apply RL techniques to real-world control problems with severely suboptimal datasets.

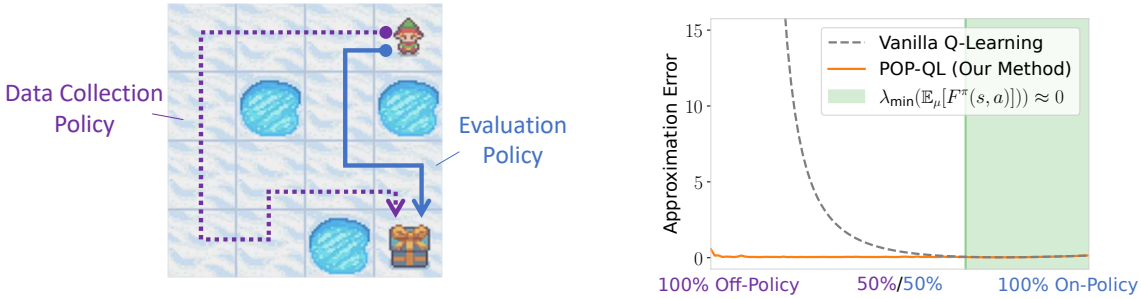


Figure 5.1: Off-policy evaluation on a simple grid environment, "Frozen Lake". The goal of this task is to evaluate a policy (—) from a suboptimal data policy (···) that is ϵ-dithered for sufficient coverage ($\epsilon = 0.2$). The right plot shows approximation error from a linear Q-function trained using Vanilla Q-Learning and POP-QL (our method) with a dataset interpolated between off-policy and on-policy. Unlike Vanilla Q-Learning, POP-QL avoids divergence by projecting the sampling distribution μ onto the convex constraint $\mathbb{E}_{(s,a)\sim\mu}[F^\pi(s,a)] \succeq 0$, which enforced that the contraction mapping condition holds. The shaded region (▬), indicates when the dataset approximately satisfies this condition (without reweighting); specifically, where $\lambda_{\min}(\mathbb{E}_{(s,a)\sim\mu}[F^\pi(s,a)]) > -0.005$.

5.1 Introduction

Temporal difference (TD) learning is one of the most common techniques for reinforcement learning (RL). Compared to policy gradient methods, TD methods tend to be significantly more data-efficient. One of the primary reasons for this data-efficiency is the ability to perform updates off-policy, where policy updates are performed using a dataset that was collected using a different policy. Unfortunately, due to the differences in distribution between the off-policy and on-policy datasets, TD methods that employ function approximation may diverge or result in arbitrarily poor value approximation when applied in the off-policy setting [102, p. 260]. This issue is exacerbated in the fully offline RL setting, where the training dataset is fixed and the agent must act off-policy in order to achieve high performance. Figure 5.1 illustrates this *distribution shift* issue on the Frozen Lake toy problem. In this example, we perform Q-Learning with a linear function approximate on a fixed dataset collected with one policy, the "data-collection" policy, to evaluate another policy, the "evaluation" policy. We can see that vanilla Q-learning diverges as the dataset shifts further off-policy.

Nearly all methods for addressing off-policy distribution shift fall into two categories: importance sampling methods that reweight samples from the dataset to approximate the on-policy distribution, or regularization methods that minimize the distribution shift directly or through penalizing the value function in low-support areas. The former may lead to extremely high variance gradient updates, and the latter only works well when the data-collection policy is close to optimal, which is not the case in most real-world datasets.

An alternative approach has been suggested by Kolter [64], who provide a contraction mapping condition that, when satisfied, guarantees convergence of TD-learning to a unique fixed point. They propose to project the sampling distribution onto this convex condition and thereby significantly

reduce approximation error. Figure 5.1 shows that when this contraction mapping condition is (approximately) satisfied, Q-learning converges with low approximation error. However, this approach does not scale to modern RL tasks, because it invokes a batch semi-definite programming (SDP) solver to solve the projection for each batch update.

Contribution In this chapter, we build on Kolter [64]'s theoretical contribution and propose Projected Off-Policy Q-Learning (POP-QL), which makes two significant improvements over previous work. First, we consider a new sampling projection that allows for a more computationally efficient algorithm, since a closed-form solution exists for the inner optimization of the dual problem. This computational improvement allows us to extend the technique to high-dimensional deep RL problems. Secondly, we extend this projection to the MDP setting and propose a new Q-Learning algorithm that jointly projects the policy and sampling distribution. Our proposed algorithm can plug into any Q-learning algorithm (such as Soft Actor Critic [46]), significantly reducing the approximation error of the Q-function and improving policy performance. We evaluate our method on a variety of offline RL tasks and compare its performance to other offline RL approaches. Although, in its current iteration, our method struggles to reach state-of-the-art performance on tasks with near-expert data-collection policies, POP-QL is able to outperform many other methods when the data-collection policies are far from optimal (such as the "random" D4RL tasks). Most importantly, our results illustrate the power of the contraction mapping condition and the potential of this new class of offline RL techniques.

5.2 Preliminaries and Problem Setting

In this work, we consider learning a policy that maximizes the cumulative discounted reward on a Markov Decision Process (MDP) defined as the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $p(\cdot|s,a)$ and $r(s,a)$ represent the transition dynamics and reward functions, and $\gamma \in [0,1)$ is the discount factor. We approximate the action-value function as $Q(s,a) \approx w^\top \phi(s,a)$, where $\phi : \mathcal{S} \times \mathcal{A} \to \{x \in \mathbb{R}^k : \|x\|_2 = 1\}$ is a normalized basis function and $w \in \mathbb{R}^k$ are the parameters of the final, linear layer.

In the off-policy setting, we assume the agent cannot directly interact with the environment and instead only has access to samples of the form $(s, a, r(s,a), s')$, where $s' \sim p(\cdot|s,a)$ and $(s,a) \sim \mu$ for some arbitrary sampling distribution μ. Because we can assume a fixed policy for much of our derivations and theory, we can simplify the math significantly by focusing on the finite Markov Reward Process setting instead.

5.2.1 Simplified Setting – Finite Markov Reward Process (MRP)

Consider the finite n-state Markov Reward Process (MRP) $(\mathcal{S}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $p : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_+$ and $r : \mathcal{S} \to \mathbb{R}$ are the transition and reward functions, and $\gamma \in (0,1)$ is the discount factor.[1] Because the state-space is finite, it can be indexed as $\mathcal{S} = \{1, \ldots, n\}$, which allows us to use matrix rather than operator notation. In operator notation, we use matrices P

[1]Note that, given a fixed policy, an MDP reduces to an MRP.

and R, to represent the functions p and r, where each row corresponds to a state. The value function associated with the MRP is the expected γ-discounted future reward of being in each state $V(s) \coloneqq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^\top r(s_t) \middle| s_0 = s\right]$. The value function is consistent with Bellman's equation in matrix form,

$$V = R + \gamma P V. \tag{5.1}$$

We approximate the value function as $V(s) \approx w^\top \phi(s)$, where $\phi : \mathcal{S} \to \{x \in \mathbb{R}^k : \|x\|_2 = 1\}$ is a fixed normalized basis function and we estimate parameters $w \in \mathbb{R}^k$. In matrix notation, we write this as $V \approx \Phi w$. In the off-policy setting, the sampling distribution μ differs from the stationary distribution ν. In this setting, the temporal difference (TD) solution is the fixed point of the projected Bellman equation:

$$\Phi w^\star = \Pi_\mu (R + \gamma P \Phi w^\star), \tag{5.2}$$

where $\Pi_\mu = \Phi(\Phi^\top D_\mu \Phi)^{-1} \Phi^\top D_\mu$ is the projection onto the column space of Φ weighted by the data distribution μ through the matrix $D_\mu = \mathrm{diag}(\mu)$. This projection may be arbitrarily far from the true solution so that the error may be correspondingly large. In practice, $w^\star$ is often computed using TD-learning, a process that starts from some point $w_0 \in \mathbb{R}^k$ and iteratively applies Bellman updates,

$$w_{t+1} = w_t - \lambda \mathbb{E}_\mu \left[\left(\phi(s)^\top w_t - r - \phi\left(s'\right)^\top w_t \right) \phi(s) \right]. \tag{5.3}$$

Unfortunately, in the off-policy setting, TD-learning is not guaranteed to converge.

5.2.2 Contraction Mapping Condition

A γ-contraction mapping[2] is any function, $f : \mathbb{R}^n \to \mathbb{R}^n$, such that for some distribution μ and any $x_1, x_2 \in \mathbb{R}^n$:

$$\|f(x_1) - f(x_2)\|_\mu \leq \gamma \|x_1 - x_2\|_\mu, \tag{5.4}$$

where $\gamma \in [0, 1)$ and $\|\cdot\|_\mu$ is the weighted 2-norm. A key property of contraction mappings is that iteratively applying this function to any starting point $x_0 \in \mathbb{R}^n$ converges to a unique fixed point $x^* = f(x^*)$. This principle is used to prove convergence of on-policy TD-learning.

Under on-policy sampling, $\mu = \nu$, the projected Bellman operator, $\Pi_\mu \mathcal{B}(x) = \Pi_\mu(R + \gamma P x)$, is a contraction mapping.

$$\|\Pi_\mu \mathcal{B}(\Phi w_1) - \Pi_\mu \mathcal{B}(\Phi w_2)\|_\mu \leq \gamma \|\Phi w_1 - \Phi w_2\|_\mu \quad \forall\, w_1, w_2 \in \mathbb{R}^k. \tag{5.5}$$

Tsitsiklis and Van Roy [114] use this property to both prove that on-policy TD Q-learning learning converges to a unique point and bound the approximation error of the resulting fixed point [114, Lemma 6]. However, in the off-policy setting with $\mu \neq \nu$ this property does not always hold. In fact, this condition can be violated even in MRPs with very small state spaces (see Figure 5.3 for an example. Thus, the TD updates are not guaranteed to converge and can diverge under some off-policy sampling distributions.

[2]A contraction mapping can be defined for any metric space, but here we focus on the metric space defined by the Euclidean space and weighted Euclidean metric.

To get around this challenge, Kolter [64] proposed a new approach. First, they transformed the contraction mapping condition into a linear matrix inequality (LMI) through algebraic manipulation:

$$\mathbb{E}_{s\sim\mu}[F(s)] \succeq 0, \text{ where } F(s) = \mathbb{E}_{s'\sim p(\cdot|s)}\left[\begin{bmatrix}\phi(s)\phi(s)^\top & \phi(s)\phi(s')^\top \\ \phi(s')\phi(s)^\top & \phi(s)\phi(s)^\top\end{bmatrix}\right]. \tag{5.6}$$

We provide a derivation of this LMI in Section 5.2.3. Using this formulation, they present an algorithm to find a new sampling distribution that satisfies this contraction mapping condition and proves a bound on the approximation error of their approach. Unfortunately, this method scales poorly because it requires solving an SDP problem alongside each batch update. Thus, the method remains impractical for the deep RL tasks, and has seen virtually no practical usage in the years since.

5.2.3 Derivation of Contraction Mapping LMI

For a finite MRP, the projected Bellman equation can be written in matrix notation as:

$$\mathcal{B}(\Phi w) = \Pi_\mu(R + \gamma P\Phi w), \tag{5.7}$$

By the definition of a contraction mapping, the projected Bellman equation is a γ-contraction mapping if and only if for all $w_1, w_2 \in \mathbb{R}^k$:

$$\|\mathcal{B}(\Phi w_1) - \mathcal{B}(\Phi w_2)\|_\mu \leq \gamma\|\Phi w_1 - \Phi w_2\|_\mu. \tag{5.8}$$

Now, consider the left side of this equation. We can rewrite this as follows:

$$\begin{aligned}\|\mathcal{B}(\Phi w_1) - \mathcal{B}(\Phi w_2)\|_\mu &= \|\Pi_\mu\mathcal{B}(\Phi w_1) - \Pi_\mu\mathcal{B}(\Phi w_2)\|_\mu \\ &= \|\Pi_\mu(R + \gamma P\Phi w_1) - \Pi_\mu(R + \gamma P\Phi w_2)\|_\mu \\ &= \gamma\|\Pi_\mu P\Phi w_1 - \Pi_\mu P\Phi w_2\|_\mu \\ &= \gamma\|\Pi_\mu P\Phi\tilde{w}\|_\mu\end{aligned}$$

where $\tilde{w} = w_1 - w_2$. Thus, the projected Bellman equation is a contraction mapping if and only if for all $w \in \mathbb{R}^k$,

$$\|\Pi_\mu P\Phi w\|_\mu \leq \|\Phi w\|_\mu. \tag{5.9}$$

Plugging in the closed form solution to the projection, we get,

$$\begin{aligned}& w^T\Phi^T P^T D_\mu\Phi\left(\Phi^T D_\mu\Phi\right)^{-1}\Phi^T D_\mu\Phi\left(\Phi^T D_\mu\Phi\right)^{-1}\Phi D_\mu P\Phi^T w \leq w^T\Phi^T D_\mu\Phi w \\ \Leftrightarrow\quad & w^T\left(\Phi^T P^T D_\mu\Phi\left(\Phi^T D_\mu\Phi\right)^{-1}\Phi D_\mu P\Phi^T - \Phi^T D_\mu\Phi\right)w \leq 0 \\ \Leftrightarrow\quad & \Phi^T P^T D_\mu\Phi\left(\Phi^T D_\mu\Phi\right)^{-1}\Phi D_\mu P\Phi^T - \Phi^T D_\mu\Phi \preceq 0\end{aligned}$$

Finally, using Schur Complements, we can convert this to an LMI,

$$F_\mu \equiv \begin{bmatrix}\Phi^T D_\mu\Phi & \Phi^T D_\mu P\Phi \\ \Phi^T P^T D_\mu\Phi & \Phi^T D_\mu\Phi\end{bmatrix} \succeq 0. \tag{5.10}$$

Thus, as long as $F_\mu \succeq 0$, the projected Bellman equation is a contraction mapping.

5.3 Projected Off-Policy Q-Learning (POP-QL)

Our method, Projected Off-Policy Q-Learning (POP-QL), is also centered on the contraction mapping condition (Equation (5.6)). However, unlike previous work, we propose a new method that significantly improves the computational cost of the projection, allowing POP-QL to scale to large-scale domains. Additionally, we introduce a new policy optimization algorithm that simultaneously projects the policy and sampling distribution in order to satisfy the contraction mapping condition. This policy optimization algorithm allows POP-QL to address both the *support mismatch* and *projected-TD instability* challenges of off-policy Q-learning.

We start by deriving the POP-QL reweighting procedure in the finite MRP setting under a fixed policy and later extend our method to the MDP setting together with policy regularization.

5.3.1 POP-QL on Markov Reward Processes

In the MRP setting (or the fixed-policy setting), the goal of POP-QL is to compute a new sampling distribution that satisfies the contraction mapping condition in Equation (5.6) and thus stabilizes off-policy training. However, if the target distribution differs significantly from the source distribution μ, this can result in large reweighting factors, which can decrease the stability of the training process. Thus, we are looking for the "closest" distribution that satisfies Equation (5.6). Unlike Kolter [64], we propose to use the I-projection instead of the M-projection, which allows us to find an analytical solution to the inner part of the Lagrangian dual and thereby significantly simplifies the problem.

The information (I-) projection and moment (M-) projection are defined as follows:

$$\text{I-Projection: } \min_q D_{\mathrm{KL}}(q\|\mu) \qquad \text{M-Projection: } \min_q D_{\mathrm{KL}}(\mu\|q) \tag{5.11}$$

Since the KL-divergence is an asymmetric measure, these projections are usually not equivalent. A key difference between the I- and M-projections is that the I-projection tends to under-estimate the support of the fixed distribution μ, resulting in more density around the modes of μ, while the M-projection tends to over-estimate the support of μ, resulting in a higher variance solution.

In the context of off-policy Q-learning, sampling states and actions with very low or zero support under the sampling distribution, μ, can result in over-estimating the Q-function, which in-turn results in a poor performing policy. For this reason, we argue the more conservative I-projection is a better fit for off-policy Q-learning.

We first formulate the problem as minimizing the KL divergence between the data distribution μ and a reweighted distribution q such that TD update is stable under q,

$$\underset{q}{\text{minimize}}\ D_{\mathrm{KL}}(q \,\|\, \mu) \quad \text{s.t.} \quad \mathbb{E}_{s\sim q}[F(s)] \succeq 0. \tag{5.12}$$

The corresponding unconstrained dual problem based on a Lagrange variable $Z \in \mathbb{R}^{2k}$ is given by

$$\underset{Z\succeq 0}{\text{maximize}}\ \underset{q}{\text{minimize}}\ \mathrm{KL}(q\|\mu) - \operatorname{tr} Z^\top \mathbb{E}_q[F(s)]. \tag{5.13}$$

The solution to this dual problem is equal to the primal problem under strong duality, which holds in practice due to the fact that this corresponds to a convex optimization problem. Now, consider the inner optimization problem over q in Equation (5.13). This optimization problem can be rewritten as $\text{minimize}_q \; -H(q) - \mathbb{E}_q \left[\log \mu(s) + \operatorname{tr} Z^\top F(s)\right]$, which has a simple analytical solution:

$$q^\star(s) \propto \exp\left(\log \mu(s) + \operatorname{tr} Z^\top F(s)\right) = \mu(s) \exp\left(\operatorname{tr} Z^\top F(s)\right). \tag{5.14}$$

Notice that our target distribution $q^\star$ is simply a reweighting of the source distribution μ with weights $\exp\left(\operatorname{tr} Z^\top F(s)\right)$. To compute the weights, we need to solve for the Lagrange variable Z. Plugging the analytical solution for $q^\star$ back into Equation (5.13) yields

$$\underset{Z \succeq 0}{\text{minimize}} \; \mathbb{E}_\mu \left[\exp\left(\operatorname{tr} Z^\top F(s)\right)\right]. \tag{5.15}$$

In practice, we minimize over the set $Z \succeq 0$ by re-parametrizing Z as

$$Z = \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}^\top \tag{5.16}$$

where $A, B \in \mathbb{R}^{k \times k}$. This formulation ensures that Z is positive semi-definite, $Z \succeq 0$, for any A and B. Thus, we can directly optimize over A and B and ignore the positive semi-definite condition. With this formulation for Z and plugging the definition of F (Equation (5.6)) we can rewrite the dual optimization problem (Equation (5.15)) as:

$$\underset{A,B}{\text{minimize}} \; \mathbb{E}_\mu \left[\exp\left(\|A^\top \phi(s)\|_2^2 + \|B^\top \phi(s)\|_2^2 + 2\mathbb{E}_{s' \sim p(\cdot \mid s)} \left[\langle B^\top \phi(s), A^\top \phi(s')\rangle\right]\right)\right] \tag{5.17}$$

Solving for matrices A, B yields the I-projected sampling distribution q^* according to Equation (5.14).

5.3.2 Extension to Markov Decision Processes

The theory presented in the previous section can be extended to the MDP setting through a simple reduction to an MRP. In a MDP, we also have to consider the action space, $\mathcal{A}$, and the policy, π. In this setting, our contraction mapping LMI becomes

$$\mathbb{E}_{(s,a) \sim q}[F^\pi(s,a)] \succeq 0, \tag{5.18}$$

$$\text{where } F^\pi(s,a) = \mathbb{E}_{s' \sim p(\cdot \mid s,a), a' \sim \pi(s')} \left[\begin{bmatrix} \phi(s,a)\phi(s,a)^\top & \phi(s,a)\phi(s',a')^\top \\ \phi(s',a')\phi(s,a)^\top & \phi(s,a)\phi(s,a)^\top \end{bmatrix}\right]. \tag{5.19}$$

Using the idea that, given a fixed policy, any MDP reduces to an MRP, we extend Theorem 2 from Kolter [64] to show that the TD-updates converge to a unique fixed point with bounded approximation error for any finite MDP where π and μ satisfy this condition.

Lemma 20. *Let $w^\star$ be the least-squares solution to the Bellman equation for a fixed policy π:*

$$w^* = \arg\min_w \mathbb{E}_{(s,a) \sim \mu} \left[(\phi(s,a)^\top w - r(s,a) - \gamma \mathbb{E}_{s' \sim p(\cdot \mid s,a), a' \sim \pi(s')} \phi(s',a')^\top w)^2\right] \tag{5.20}$$

and let μ be some distribution satisfying the MDP contraction mapping condition (Equation (5.18)*). Then*

$$\mathbb{E}_\mu\left[(\phi(s,a)^\top w^\star - V(s,a))^2\right] \leq \frac{1+\gamma\sqrt{\delta(\nu,\mu)}}{1-\gamma}\min_w \mathbb{E}_\mu\left[(\phi(s,a)^\top w - V(s,a))^2\right], \quad (5.21)$$

where ν is the stationary distribution, $\delta(\nu,\mu) = \max_{s,a,\tilde{s},\tilde{a}} \frac{\nu(s,a)}{\mu(s.a)} \cdot \frac{\mu(\tilde{s},\tilde{a})}{\nu(\tilde{s},\tilde{a})}$

Proof. To prove this, we will use a simple reduction to a Markov chain.

For a fixed policy, π, our MDP, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ reduces to a MRP. We can write this new MRP as $\mathcal{M}^\pi = (\mathcal{X}, P^\pi, R, \gamma)$ where $\mathcal{X} = \mathcal{S} \times \mathcal{A}$ is the Cartesian product of the state and action spaces of the MDP and $P^\pi : \mathcal{X} \times \mathcal{X} \mapsto \mathcal{R}_+$ is defined as follows:

$$P^\pi((s,a),(s',a')) \coloneqq p(s'|s,a)\pi(a'|s') \qquad \forall\,(s,a),(s',a') \in \mathcal{X}$$

We can see clearly that for all $(s,a) \in \mathcal{X}$, $\sum_{(s',a')\in\mathcal{X}} P^\pi((s,a),(s',a')) = 1$.

Since our MDP, $\mathcal{M}$, is finite, we can define the feature matrix, $\Phi \in \mathcal{R}^{n,k}$, using the MDP feature function, $\Phi_i = \phi(s_i, \pi(s_i)$ for each $s_i \in \mathcal{S}$.

Now, applying Theorem 20 to our MRP, $\mathcal{M}_\pi$, we have that:

$$\|\Phi w^\star - V^\pi\|_\mu \leq \frac{1+\gamma\sqrt{\delta(\nu,\mu)}}{1-\gamma}\|\Pi_\mu V^\pi - V^\pi\|_\mu \quad (5.22)$$

where $w^\star$ is the unique fixed point of Equation (5.2), V^π is the unique fixed point of $V^\pi = R + \gamma P^\pi V^\pi$, ν is the stationary distribution, and $\delta(\nu,\mu) = \max_{x,\tilde{x}} \frac{\nu(x)}{\mu(x)} \cdot \frac{\mu(\tilde{x})}{\nu(\tilde{x})}$.

Mapping this bound back onto the MDP, $\mathcal{M}$, yields the stated bound. □

As before, we are looking to project our sampling distribution to satisfy this condition. With this reduction, we rewrite the analytical solution for the projected sampling distribution from Equation (5.14) as

$$q^\star(s,a) \propto \mu(s,a)\exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\mathbb{E}_{s'\sim p(\cdot\,|\,s),a'\sim\pi(\cdot\,|\,s'}\left[\langle y_B, y'_A\rangle\right]\right) \quad (5.23)$$

where $y_A = A^\top\phi(s,a)$, $y_B = B^\top\phi(s,a)$, and $y'_A = A^\top\phi(s',a')$ and A and B are the solutions to the following optimization problem,

$$\underset{A,B}{\text{minimize}}\ \mathbb{E}_\mu\left[\exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\mathbb{E}_{s'\sim p(\cdot\,|\,s)}\left[\langle y_B, y'_A\rangle\right]\right)\right] \quad (5.24)$$

Now, by Lemma 20 and assuming strong duality holds, the fixed point of the projected Bellman equation under the sampling $q^\star$ has bounded approximation error.

Algorithm 8 Projected Off-Policy Q-Learning (POP-QL)

Initialize: feature function ϕ_{θ^ϕ}, Q-function parameters w, policy π_{θ^π},
g-function g_{θ^g}, and Lagrange matrices A and B.
for step t in $1, \ldots, N$ **do**
$(s, a, r, s')_{1,\ldots,m} \sim \mu$ ▷ Sample minibatch from dataset

$\tilde{a} \sim \pi_{\theta^\pi}(s)\ \tilde{a}' \sim \pi_{\theta^\pi}(s')$ ▷ Sample new actions from policy
$q_{\text{target}} \coloneqq r + \gamma w^\top \phi_{\theta^\phi}(s', \tilde{a}')$ ▷ Compute Q-function target value
$y_A, y_B, y'_A \coloneqq A^\top \phi_{\theta^\phi}(s, a), B^\top \phi_{\theta^\phi}(s, a), A^\top \phi_{\theta^\phi}(s', a')$ ▷ Compute dual values
$u \coloneqq \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2g_{\theta^g}(s, a)\right) / \bar{u}$ ▷ Compute minibatch-normalized weight

$A, B \leftarrow [A, B] - \lambda_{A,B} u \nabla_{A,B} \left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\langle y_B, y'_A\rangle\right)$ ▷ Update Lagrange matrices
$\theta^g \leftarrow \theta^g - \lambda_g \nabla_{\theta^g} \left(g_{\theta^g}(s, a) - \langle y_B, y'_A\rangle\right)^2$ ▷ Update g-function parameters
$[\theta^Q, w] \leftarrow \theta^Q - \lambda_Q u \nabla_{\theta^Q, w} \left(w^\top \phi_{\theta^\phi}(s, a) - q_{\text{target}}\right)^2$ ▷ Update Q-function parameters
$\theta^\pi \leftarrow \theta^\pi - \lambda_\pi \nabla_{\theta^\pi} \left(\mathcal{L}_Q + \alpha \mathcal{L}_{\text{entropy}} - \beta u \langle y_B, y'_A\rangle\right)$ ▷ Augment SAC policy loss

5.3.3 Practical Implementation

Two-Time-Scale Optimization Because there is an expectation inside an exponential, we must perform a two time-scale optimization to be able to use sample-based gradient descent. We introduce a new function approximator g_θ to approximate the inner expectation:

$$g_\theta(s, a) \approx \mathbb{E}_{s' \sim p(s' \mid s), a' \sim \pi(\cdot \mid s')} \left[\langle y_B, y'_A\rangle\right], \tag{5.25}$$

which can be optimized using gradient descent.

If we assume $g_\theta(s)$ has sufficient expressive power and has converged, we can estimate the gradient of our objective with respect to A and B, $\nabla_{A,B} \mathbb{E}_\mu \left[\exp\left(\operatorname{tr} Z^\top F(s)\right)\right]$, using samples from our sampling distribution, μ:

$$\mathbb{E}_{s,a \sim \mu, s' \sim p(s' \mid s), a' \sim \pi(\cdot \mid s')} \left[u(s, a) \cdot \nabla_{A,B} \left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\langle y_B, y_A\rangle\right)\right] \tag{5.26}$$

where $u(s, a)$ are the sample reweighting terms defined as:

$$u(s, a) = \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2g_\theta(s)\right), \quad q^*(s, a) = u(s, a)\mu(s, a). \tag{5.27}$$

With this approximation, we can perform two-time-scale gradient descent. The gradient updates for g_θ and the Lagrange matrices A and B become

$$\begin{aligned} \theta &\leftarrow \theta - \lambda_\theta \nabla_\theta \left(g_\theta(s, a) - \mathbb{E}_{s' \sim p(s' \mid s, a), a' \sim \pi(s')} \left[\langle y_B, y'_A\rangle\right]\right)^2, \\ A, B &\leftarrow [A, B] - \lambda_{A,B} \mathbb{E}_{\mu,p} \left[u(s, a) \cdot \nabla_{A,B} \left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\langle y_B, y'_A\rangle\right)\right], \end{aligned} \tag{5.28}$$

g-function Normalization In practice, we found normalizing the g function by the spectral norm of the A and B matrices improved learning stability. Specifically, we train the g network to

approximate the following quantity:

$$g_\theta(s,a) \approx \frac{1}{\|A\|_2\|B\|_2}\mathbb{E}_{s'\sim p(s'\,|\,s,a),a'\sim\pi(s')}\left[\langle B^\top\phi(s,a), A^\top\phi(s',a')\rangle\right] \tag{5.29}$$

where $\|A\|_2$ represents the spectral norm of A. Note that, by definition of the spectral norm and since $\|\phi(s,a)\|_2 = 1$ for all s, this bounds the range of the g function, $g_\theta(s,a) \in [-1,1]$.

Low-Rank Approximation Empirically, we found the solution to our Lagrange dual optimization problem is typically a low-rank matrix with rank $r \leq 4$; this is not surprising in hindsight: under the on-policy distribution the matrix $\mathbb{E}_{(s,a)\sim q}[F^\pi(s,a)]$ is *already* positive definite (a consequence of the fact that TD will converge on-policy), and so it is intuitive that this bound would only need to be enforced on a low-dimensional subspace, corresponding to a low-rank dual solution. Thus, we can substantially reduce the computational cost of the method by using Lagrange matrices $A, B \in \mathbb{R}^{k\times r}$, where $r = 4$. This is essentially akin to low-rank semi-definite programming [24], which has proven to be an extremely competitive and scalable approach for certain forms of semi-definite programs. Furthermore, while not a primary motivation for the method, we found this low-rank optimization improves convergence of the dual matrices. In total, this leads to a set of updates that are fully linear in the dimension of the final-layer features, and which ultimately presents a relatively modest increase in computational cost over standard Q-Learning.

Target Networks Just as in other actor-critic methods, we use a target network for the features and Q-function weight w to stabilize the Q-learning updates.

We also tested using these target features for training Lagrange matrices A and B, but found this reduced performance. So instead, we do not use any target networks in training the Lagrange matrices.

5.3.4 Policy Optimization

So far, we have assumed a fixed policy in order to compute a sampling distribution and use that sampling distribution to compute a Q-function with low approximation error. Next, we need to find a policy that maximizes this Q-function. However, we want to avoid policies that result in very large reweighting terms for a couple reasons: 1) state action pairs with large reweighting terms correspond to low-support regions of the state-space, and 2) large reweighting terms increase the variance of the gradient updates of our Lagrange matrices. To keep these reweighting terms small, we jointly project π and the sampling distribution μ using a balancing term $\beta \in \mathbb{R}_+$:

$$\underset{\pi,q}{\text{maximize}}\ \mathbb{E}_\mu[Q^\pi(s,a)] - \beta D_{\text{KL}}(q\,\|\,\mu) \quad \text{s.t.} \quad \mathbb{E}_{(s,a)\sim q}[F^\pi(s,a)] \succeq 0 \tag{5.30}$$

We can solve this optimization using the technique from the previous section. To start, we can rewrite Equation (5.30) as:

$$\underset{\pi,q}{\text{maximize}}\ \frac{1}{\beta}\mathbb{E}_\mu[Q^\pi(s,a)] - D_{\text{KL}}(q\,\|\,\mu) \quad \text{s.t.} \quad \mathbb{E}_{(s,a)\sim q}[F^\pi(s,a)] \succeq 0 \tag{5.31}$$

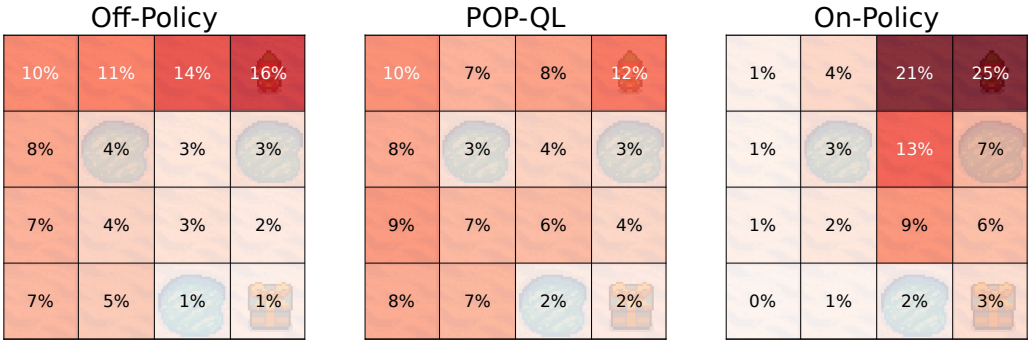


Figure 5.2: Heat-maps of three state distributions for the "Frozen Lake" environment. On the left is the off-policy sampling distribution, on the right is the on-policy sampling distribution, and, in the middle, is the projection of off-policy sampling distribution onto the contraction mapping set (Equation (5.6)) computed by POP-QL. Note that only a minor change to the off-policy sampling distribution is needed to satisfy the contraction mapping condition and, thus, guarantee convergence of TD-learning.

Next, we introduce Lagrange variables to convert this into an unconstrained optimization problem:

$$\begin{aligned}
&\underset{q,\pi}{\text{maximize}}\ \underset{Z\succeq 0}{\text{minimize}}\ \frac{1}{\beta}\mathbb{E}_\mu[Q^\pi(s,a)] - \mathrm{KL}(q\|\mu) + \operatorname{tr} Z^\top \mathbb{E}_q[F^\pi(s,a)] \\
&= \underset{\pi}{\text{maximize}}\left(\frac{1}{\beta}\mathbb{E}_\mu[Q^\pi(s,a)] + \underset{q}{\text{maximize}}\ \underset{Z\succeq 0}{\text{minimize}}\ -\mathrm{KL}(q\|\mu) + \operatorname{tr} Z^\top \mathbb{E}_q[F^\pi(s,a)]\right)
\end{aligned}$$

Now, we focus on the inner optimization problem over q and Z. As in the MRP version, we assume that strong duality holds in this inner optimization problem. Under this assumption, the inner optimization problem can be equivalently written as:

$$\underset{Z\succeq 0}{\text{minimize}}\ \underset{q}{\text{maximize}}\ -\mathrm{KL}(q\|\mu) + \operatorname{tr} Z^\top \mathbb{E}_q[F^\pi(s,a)] \tag{5.32}$$

This problem can be solved as before. First, we solve for the analytical solution of q,

$$q^\star(s,a) \propto \mu(s,a)\exp\left(\operatorname{tr} Z^{\pi\top} F(s,a)\right). \tag{5.33}$$

Next, we plug this solution back into our inner optimization problem:

$$\underset{Z\succeq 0}{\text{minimize}}\ \log\left(\mathbb{E}_\mu\left[\exp\left(\operatorname{tr} Z^\top F^\pi(s,a)\right)\right]\right) \tag{5.34}$$

Now, using the reparameterization of Z in Equation (5.16), the full optimization problem (Equation (5.31) becomes:

$$\underset{\pi}{\text{maximize}}\left(\frac{1}{\beta}\mathbb{E}_\mu[Q^\pi(s,a)] + \underset{A,B}{\text{minimize}}\ \log\left(\mathbb{E}_\mu\left[\exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2\mathbb{E}_{s'\sim p(\cdot\,|\,s,a),a'\sim\pi(\cdot\,|\,s')}\langle y_B, y'_A\rangle\right)\right]\right)\right) \tag{5.35}$$

where $y_A = A^\top \phi_{\theta^\phi}(s,a)$, $y_B = B^\top \phi_{\theta^\phi}(s,a)$, and $y'_A = A^\top \phi_{\theta^\phi}(s',a')$.

Again, we need to perform a 2-timescale gradient descent for A, B since we are approximating an expectation inside of a exponential. Thus, we also learn a parameterized function g_θ to approximate the following:

$$g_\theta(s,a) \approx \mathbb{E}_{s' \sim p(s' \mid s,a), a' \sim \pi(s')} \left[\langle y_B, y'_A \rangle \right] \tag{5.36}$$

Using this approximation, the gradient of A and B can be expressed as:

$$\mathbb{E}_{s,a \sim \mu, s' \sim p(s' \mid s,a), a' \sim \pi(\cdot \mid s')} \begin{bmatrix} \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 g_\theta(s,a) \right) \\ \cdot \nabla_{A,B} \left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 \langle y_B, y'_A \rangle \right) \end{bmatrix} \tag{5.37}$$

Next, we derive the policy updates using both Lagrange variables.

$$\begin{aligned}
&\nabla_\pi \left(\mathbb{E}_\mu [Q^\pi(s,a)] + \beta \log \left(\mathbb{E}_{\mu,\pi} \left[\exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 \langle y_B, y'_A \rangle \right) \right] \right) \right) \\
&\approx \nabla_\pi \mathbb{E}_\mu [Q^\pi(s,a)] + \beta \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 g_\theta(s,a) \right) \nabla_\pi \mathbb{E}_\pi \left[\|y_A\|_2^2 + \|y_B\|_2^2 + 2 \langle y_B, y'_A \rangle \right] \\
&= \nabla_\pi \mathbb{E}_\mu [Q^\pi(s,a)] + 2\beta \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 g_\theta(s,a) \right) \nabla_\pi \mathbb{E}_\pi \left[\langle y_B, y'_A \rangle \right]
\end{aligned}$$

Finally, the Q-learning reweighting terms are simply:

$$u(s,a) \approx \exp\left(\|y_A\|_2^2 + \|y_B\|_2^2 + 2 g_\theta(s,a) \right) \tag{5.38}$$

See Algorithm 8 for the pseudocode of our algorithm.

5.4 Experiments and Discussion

Toy Example – Three-State MRP Here we revisit the Three-State MRP discussed in Section 2.3.2. Again, the value function is given by $V = [1,\ 1,\ 1.05]^\top$, with discount factor $\gamma = 0.99$, reward function $R = (I - \gamma P)V$, and basis Φ where

$$\Phi = \begin{bmatrix} 1 & 0 \\ 0 & -1. \\ 1/2(1.05 + \epsilon) & -1/2(1.05 + \epsilon) \end{bmatrix} \tag{5.39}$$

The basis includes the representation error term $\epsilon = 10^{-4}$. For illustration purposes, we select the family of distributions $\mu = (p/2, p/2, 1-h)$ parameterized by $p \in [0,1]$. This characterizes the possible distributions of data that we will present to POP-QL and naive TD in this experiment. The on-policy distribution corresponds to $p = 0.5$. The contraction mapping condition is satisfied for the left subset of sampling distributions where $p \lesssim 0.51$ and not satisfied for the right subset where $p \lesssim 0.55$. This is immediately apparent in Figure 5.3, where we plot the error at convergence from running naive- and POP-QL above, and the effective distribution of TD updates after reweighing. In the left subset, where the NEC holds, POP-QL does not reweight TD updates at all. Therefore,

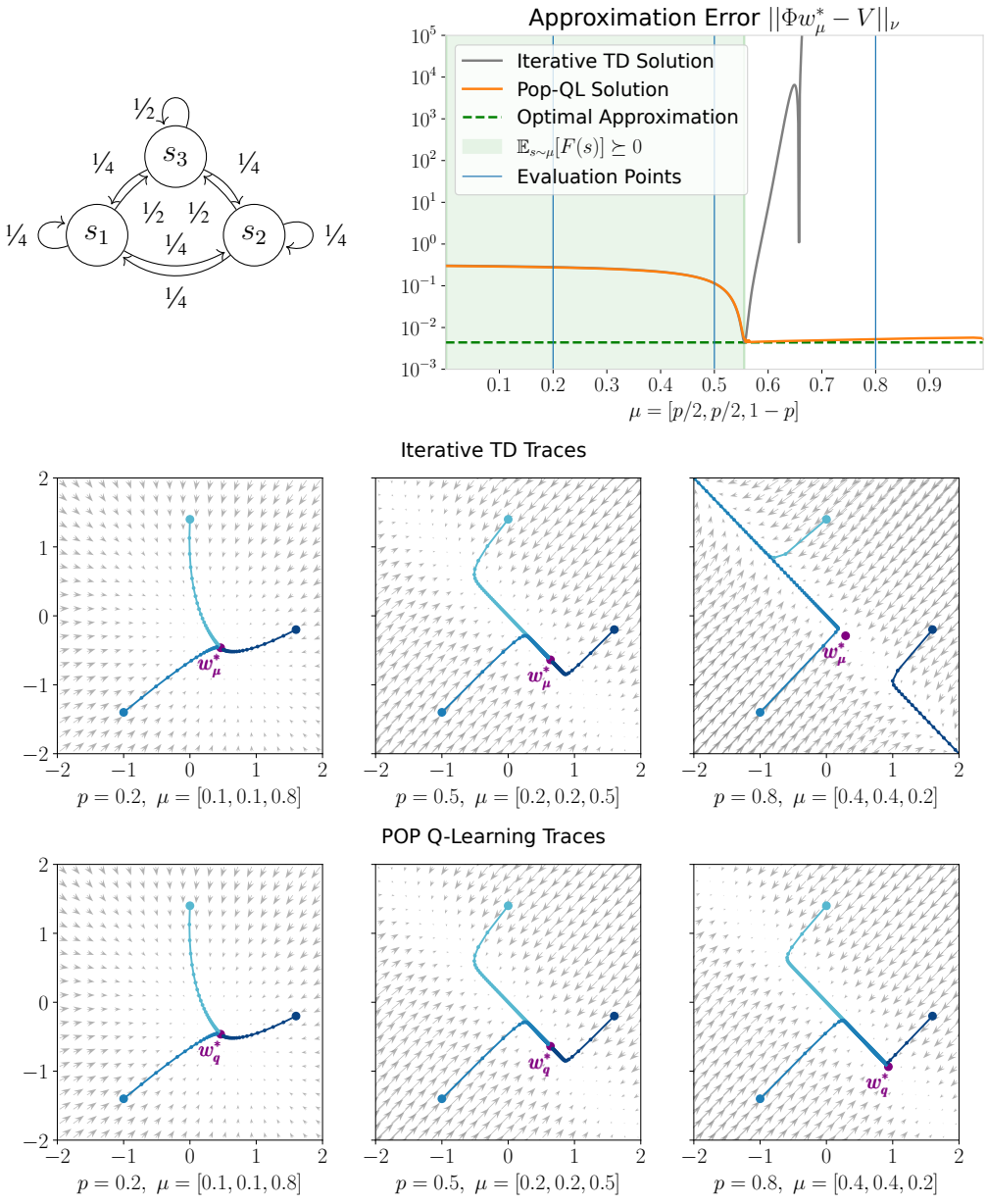


Figure 5.3: The three-state Markov process by Manek and Kolter [80] (top-left), a plot of Q-function approximation error over different sampling distributions using Iterative TD and POP-QL (top-right), and TD traces for at three different evaluation sampling distributions. We can see that when the contraction mapping condition is satisfied ($p \lesssim 0.55$), the Iterative TD and POP-QL solutions are identical. However, when this condition is violated ($p > 0.55$), Iterative TD diverges, whereas POP-QL converges and retains a low approximation error.

the error of POP-TD tracks that of naive TD, and the effective distribution of TD updates in POP-TD and naive TD are the same as the data distribution.

Figure 5.3 also plots TD-Learning traces for three different sampling distributions using both Iterivate TD and POP-QL. We can see when $p = 0.8$ (right), the contraction mapping condition is violated Itertive TD diverges. However, POP-QL reweights the sampling distribution, yielding a new sampling distribution that satisfies the contraction mapping condition. Thus, POP-QL still converges in this toy problem.

Small Scale – Frozen Lake Frozen Lake is a small grid navigation task where the objective is to reach the goal state while avoiding the holes in the ice, which are terminal states. Figure 5.1 shows a visualization of the Frozen Lake environment. For tabular Q-learning, this is a very simple task. However, using function approximation with a linear function approximator can cause offline Q-Learning to quickly diverge.

We illustrate this divergence first with a policy evaluation task. In this task, the goal is to approximate the Q-function for an "evaluation" policy with data collected from a separate "data-collection" policy. We use a random featurization for the state-action space with dimension $k = 60$ (the true state-action space has a cardinality of 64). This featurization was sampled from a uniform distribution $\mathcal{U}([0, 1]^k)$ for each states, then normalized to have a unit norm. We train a linear function approximator with both vanilla Q-learning and POP-QL as we linearly interpolate the dataset between 100% offline (meaning collected entirely from the "data collection" policy) and 100% (meaning collected entirely from the "evaluation" policy). Figure 5.1 shows a graph of the results. When trained offline, vanilla Q-learning quickly diverges, whereas POP-QL remains stable for all the datasets. We also note that the datasets for which vanilla Q-learning converges with low approximation error correspond to those that roughly satisfy our contraction mapping condition, exactly as our theory would predict. Figure 5.2 illustrates how the projection made by POP-QL changes the sampling distribution only slightly compared to exactly projecting onto the on-policy distribution (importance sampling).

We also perform a policy optimization task with the same datasets. In this task, the goal is to compute the policy with the highest return using offline data. We compare our method against online SAC, offline SAC, and CQL. Section 5.4.1 discusses the hyper-parameter tuning procedures for these baseline methods. Figure 5.4 shows the expected normalized returns of the policies computed using various methods.

D4RL Tasks D4RL [35] is a standardized collection of offline RL tasks. Each task consists of an environment and dataset. The datasets for each task are collected by using rollouts of a single policy or a mixture of policies. The goal of each task is to learn a policy exclusively from these offline datasets that maximizes reward on each environment.

We compare our method against vanilla SAC [46] run on offline data, Behavior Cloning, and CQL [69] (with the JaxCQL codebase [42]. For each of these methods, we run for 2.5 million gradient steps. Using the JaxCQL codebase, we were not able to replicate the results of CQL with the hyper-parameters presented in Kumar et al. [69]. Instead we performed our own small

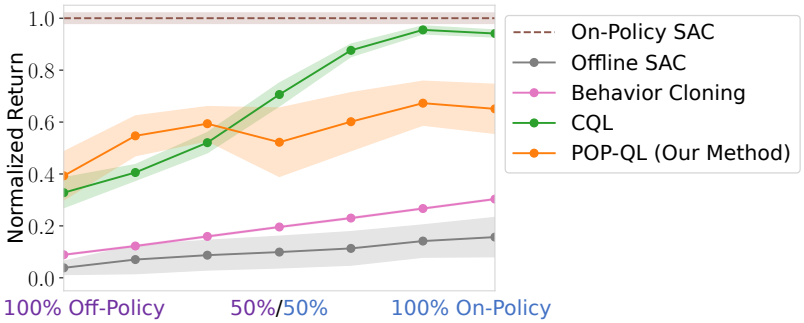


Figure 5.4: Offline policy optimization performance on the Frozen Lake domain (Figure 5.1) averaged over 5 random seeds (shaded area is standard error). As before, we varied the datasets by interpolating between the dataset collected by the "data-collection policy" (···) and the dataset collected by the "evaluation policy" (—), both with ϵ-dithering for sufficient coverage ($\epsilon = 0.2$). Both our POP-QL (our method) and CQL are able to find a policy that outperforms behavior cloning without diverging.

hyper-parameter sweep (details in Section 5.4.1). For further comparison, we also include results for bootstrapping error reduction (BEAR) [67], batch-constrained Q-learning (BCQ) [37], and AlgaeDICE [85] reported in Fu et al. [35]. We looked at 2 categories of environments: 1) The OpenAI Gym [23] environments Hopper, Half Cheetah, and Walker2D, and 3) the Franka Kitchen environments [45]. For each method, we used fixed hyper-parameters for environment category.

Tables 5.1 and 5.2 show the results on these tasks. We can see that our method is competitive or outperforms all other methods on the "random" and "medium" datasets, but falls behind on the "medium-expert" environments. This is because our method, unlike most other offline methods, does not perform any regularization towards the data collection policy.[3]

5.4.1 Hyper-Parameter Search

Q-Function and Policy Learning Rates We looked at using $\lambda_Q, \lambda_\pi =$, 3e-4, 1e-4, 3e-5, and 1e-5. The lower learning rates for the policy seemed to significantly improve asymptotic performance of our method. However, when setting $\lambda_\pi =$ 1e-5, we found it took too long to learn a decent policy. Thus, for our experiments, we chose the middle-ground of $\lambda_\pi =$ 3e-5. We found $\lambda_Q =$ 1e-4 worked the best for POP-QL.

Lagrange Matrices Learning Rate Since we want to learn the Lagrange matrices assuming a fixed policy, we used an increased learning rate for the Lagrange matrices compared to the policy. We tried

g-Function Learning Rate Once we normalized the g-function, we found the performance of the algorithm seemed to be quite robust to the choice of g-function learning rate. We tried 1, 10,

[3]The data collection policy is often called the *behavior policy*.

Table 5.1: Results on the D4RL MuJoCo offline RL tasks [35]. We ran offline SAC (SAC-off), CQL, and POP-QL for 2.5M gradient steps. †Results reported from Fu et al. [35]. We can see our method, POP-QL, outperforms other methods on the very suboptimal datasets ("random"), but falls behind on the others.

	SAC-off	BEAR†	BCQ†	aDICE†	CQL	POP-QL
hopper-random	11.88	9.50	10.60	0.90	0.92	**20.43**
halfcheetah-random	27.97	25.50	2.20	-0.30	-1.12	**29.93**
walker2d-random	4.55	**6.70**	4.90	0.50	-0.02	-0.31
hopper-medium	2.32	47.60	**54.50**	1.20	44.71	24.97
halfcheetah-medium	57.28	38.60	40.70	-2.20	**62.37**	43.03
walker2d-medium	0.94	33.20	**53.10**	0.30	7.36	13.49
hopper-medium-replay	18.54	**96.30**	33.10	1.10	2.05	19.89
halfcheetah-medium-replay	43.34	38.60	38.20	-0.80	**50.03**	34.99
walker2d-medium-replay	5.05	19.20	15.00	0.60	**63.97**	11.61
hopper-medium-expert	2.31	4.00	**110.90**	1.10	45.72	10.09
halfcheetah-medium-expert	6.41	51.70	64.70	-0.80	**82.76**	51.21
walker2d-medium-expert	0.08	10.80	**57.50**	0.40	10.12	36.01

Table 5.2: Results on the D4RL kitchen offline RL tasks [35]. We ran offline SAC (SAC-off), CQL, and POP-QL for 2.5M gradient steps. †Results reported by Fu et al. [35]. Our method, POP-QL, out-performs offline SAC and CQL, but falls behind BEAR and BCQ.

	SAC-off	BEAR†	BCQ†	aDICE†	CQL	POP-QL
kitchen-complete	0.12	0.00	**8.10**	0.00	0.00	0.00
kitchen-partial	0.00	13.10	**18.90**	0.00	6.12	6.38
kitchen-mixed	0.00	**47.20**	8.10	0.00	0.62	1.56

and 20 times the learning rate of the Lagrange Matrices and all performed roughly equally. We used $\lambda_g = 10\lambda_{[A,B]}$ for our experiments.

KL-Q-value weighting parameter β This β term weights how much POP-QL's weights policy performance versus the KL-divergence between the new sampling distribution and the reweighted distribution. The larger β is, the more the policy is projected and the less significant the reweighting terms become.

In the D4RL problems, we need a large β term to make sure the policy does not drive the agent too far outside the data distribution. We tested $\beta = 100$, 30, 10, 3, 1, and 0.3. Since our choice of β depends on the estimated Q-values, we chose a different β beta for each class of domains. For both the Mujoco and Franka Kitchen domains, we chose $\beta = 100$.

5.4.2 Target Networks

Just as in other actor-critic methods, we use a target network for the features and Q-function weight w to stabilize the Q-learning updates.

We also tested using these target features for training Lagrange matrices A and B, but found this reduced performance. So instead, we do not use any target networks in training the Lagrange matrices.

5.5 Conclusion and Future Directions

In this chapter, we present Projected Off-Policy Q-Learning (POP-QL), a new method for reducing approximation errors in off-policy and offline Q-learning. POP-QL performs an approximate projection of both the policy and sampling distribution onto a convex set, which guarantees convergence of TD updates and bounds the approximation error.

Unlike most other offline RL methods, POP-QL does not rely on pushing the learned policy towards the data-collection policy. Instead POP-QL finds the smallest adjustment to the policy and sampling distribution that ensures convergence. This property is exemplified in our experiments, especially when the data-collection policies are significantly sub-optimal. In small-scale experiments, we show that our method significantly reduces approximation error of the Q-function. We also evaluate our method on standardized Deep RL benchmarks. POP-QL outperforms other methods when the datasets are far from the optimal policy distribution, specifically the "random" datasets, and is competitive but falls behind the other methods when the dataset distribution gets closer to the on-policy distribution.

This chapter illustrates the power of the contraction mapping condition first introduced by Kolter [64] for offline RL and introduces a new class of offline RL techniques. While, in its current iteration, this method does not outperform the state-of-the-art methods on every domain, our results suggest the exciting potential of this new technique. We think this reduced performance on some the D4RL tasks is primarily due to training instabilities introduced by the min-max optimization of the policy and Lagrange matrices. As with many other RL algorithms, finding implementation tricks, such as target Q-networks [81] and double Q-networks [36, 53], is critical to stabilizing learning. In future work, we hope to address the instability of the Lagrange matrix optimization, thus providing a method that consistently out-performs competing methods.

Chapter 6

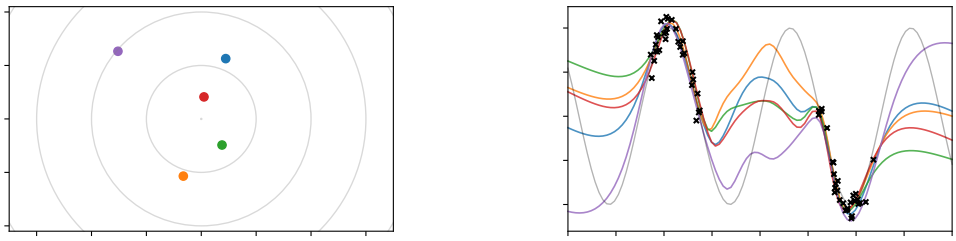
Generative Posterior Networks for Approximately Bayesian Epistemic Uncertainty Estimation

The previous chapter discussed a new method for improving the performance of offline TD methods. While TD methods tend to outperform Model-Based methods on long-horizon domains with a diverse set of data, on shorter horizon domains with less data, Model-Based methods tend to outperform TD methods. However, the challenge of support mismatch remains when performing Model-Based RL offline. To mitigate the problems caused by support mismatch, accurately predicting the epistemic uncertainty, uncertainty deriving from lack of data samples, is critical. With this information, Model-Based planning algorithms can mitigate over-estimating the value of a given policy by, for example, avoiding areas of high uncertainty or acting conservatively in the face of uncertainty. However, estimating epistemic uncertainty remains a challenge for deep learning models.

In this chapter, we introduce Generative Posterior Networks (GPNs), which use unlabeled data to estimate epistemic uncertainty by approximating the Bayesian posterior. Many real-world problems have a limited set of training data, but an abundance of unlabeled data. The use of GPNs in such a domain could significantly improve performance of offline Model-Based RL algorithms.

6.1 Introduction

In supervised learning tasks, the distribution of labeled training data often does not match exactly with the distribution of data the model will see at deployment. This distributional shift can cause significant problems in safety-critical environments where mistakes can be catastrophic. Ideally, we want our learned models to be able to estimate their epistemic uncertainty. Unfortunately, deep learning models struggle to estimate this type of uncertainty. While there are many proposed methods for addressing this problem, epistemic uncertainty estimation in deep learning remains an open problem due to out of distribution (OOD) performance and scalability.



(a) 2D embedding samples

(b) Corresponding posterior samples

Figure 6.1: Samples from a GPN using a 2D embedding trained on a simple sine-function. On the top are samples from the embedding with corresponding posterior samples underneath. Black 'x's represent observed data points.

Concurrently, many recent works have shown incredible performance using unlabeled data [27, 95]. While labeled training data is expensive to collect, in many real-world problems, such as image classification, there is an abundance of unlabeled data. Moreover, this unlabeled data is often much more representative of the deployment distribution. In this work, we propose a new method for estimating epistemic uncertainty that leverages this unlabeled data.

In many real-world tasks, labeled data is limited and expensive to obtain, while unlabeled data is plentiful. Moreover, distribution of labeled data often does not match the desired test distribution. This challenge has led to a growing literature in Domain Adaptation [29, 39, 89, 126] and Test-Time Adaptation [119]. Both of these problem settings assume access to a set of unlabeled data from the test distribution that is used to improve test performance. However, if the test distribution is too far from the training distribution, DA and TTA approaches will still have high errors on the test distribution. For such cases, understanding the model's epistemic uncertainty is vital.

Our work builds off of the extensive ensembling literature. Neural network ensembling is a method that predicts epistemic uncertainty by looking at the disagreement between the ensemble members. Additionally, under the right regularization, ensembles approximate samples from the Bayesian posterior He et al. [54], Pearce et al. [90]. However, these methods have one main drawback, namely that each new sample from the posterior requires training a new network from scratch.

To address this challenge, we introduce Generative Posterior Networks (GPNs), a generative neural network model that directly approximates the posterior distribution by regularizing the output of the network towards samples from the prior distribution. By learning a low-dimensional latent representation of the posterior, our method can quickly sample from the posterior and construct confidence intervals (CIs). We prove that our method approximates the Bayesian posterior over functions and show empirically that our method can improve uncertainty estimation over competing methods.

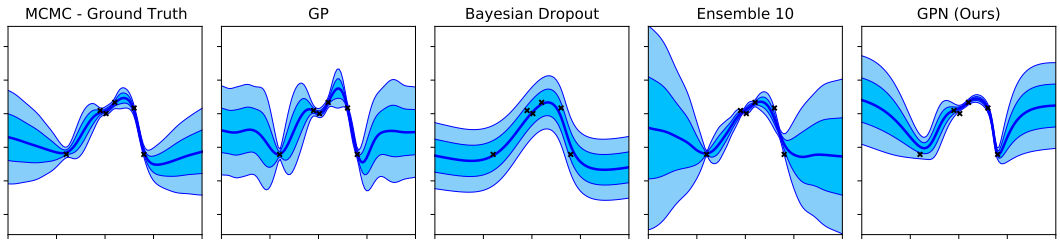


Figure 6.2: Predicted posterior distributions of different methods using the same observed data.

6.2 Generative Posterior Networks

Randomized MAP Sampling We consider a slight variation on the usual Bayesian Inference problem: instead of finding the posterior of the *parameters* of a function, we will instead find the posterior of the *outputs* of that function. Specifically, consider the transformed random variable $\hat{\mathbf{Y}} = f(\mathbf{x}_{\text{sample}}; \boldsymbol{\theta})$ for some function f parameterized by $\boldsymbol{\theta}$ and some set of unlabeled sample points $\mathbf{x}_{\text{sample}}$. We assume a known prior over parameters $P(\boldsymbol{\theta}) \sim \mathcal{N}(\mu_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$ as well as access to a noisy dataset $(\mathbf{x}_{\text{obs}}, \mathbf{y}_{\text{obs}}) = \{(x^1_{\text{obs}}, y^1_{\text{obs}}) \dots, (x^N_{\text{obs}}, y^N_{\text{obs}})\}$, where the observations $y^i_{\text{obs}} = f(x^i_{\text{obs}}; \boldsymbol{\theta}) + \epsilon$ depend on an unknown parameter $\boldsymbol{\theta}$ and are corrupted by Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$. The goal of this Bayesian Inference problem is to approximate samples from the posterior,

$$P(\hat{\mathbf{Y}} \,|\, \mathbf{y}_{\text{obs}}) \propto P(\mathbf{y}_{\text{obs}} \,|\, \hat{\mathbf{Y}}) P(\hat{\mathbf{Y}}). \tag{6.1}$$

Pearce et al. [90] showed how we can use a method called Randomized MAP Sampling (RMS) to approximately sample from the posterior distribution. The idea behind RMS is to use the fact that the parameters that minimize the regularized MSE loss are equal to the maximum a posteriori (MAP) solution for the posterior. Thus, using optimization techniques, like gradient descent, we can easily find MAP solutions. However, the challenge of sampling from the posterior remains. Instead, RMS randomly samples shifted prior *distributions* such that the resulting MAP solutions for each of these shifted distributions form samples from the desired posterior.

Unlike Pearce et al. [90], we consider using the RMS technique on our modified Bayesian Inference problem, to estimate the posterior of the *outputs* $\hat{\mathbf{Y}}$ as opposed to the *parameters* $\boldsymbol{\theta}$. Specifically, RMS in this setting samples "anchor points" $\hat{\mathbf{Y}}_{\text{anc}}$ from some distribution $\hat{\mathbf{Y}}_{\text{anc}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}})$. Each anchor point corresponds to the mean of a shifted prior distribution $P_{\text{anc}}(\hat{\mathbf{Y}}) = \mathcal{N}(\hat{\mathbf{Y}}_{\text{anc}}, \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}})$. Now we define the MAP function which finds the MAP solution using this shifted prior:

$$Y_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}) = \arg\max_{\hat{\mathbf{Y}}} P(\mathbf{y}_{\text{obs}} \,|\, \hat{\mathbf{Y}}) P_{\text{anc}}(\hat{\mathbf{Y}}). \tag{6.2}$$

By sampling different anchor points, we can then construct a probability distribution over MAP solutions $P(Y_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}))$. If we assume that the prior over $\hat{\mathbf{Y}}$ is Gaussian, this distribution is equivalent to the posterior distribution $P(\hat{\mathbf{Y}} \,|\, \mathbf{y}_{\text{obs}})$ for a specific anchor distribution. While this Gaussian prior assumption does not hold exactly in general, it will hold exactly when f is linear or has an infinitely wide final layer.

Theorem 5. *Assume that the prior distribution of* $\hat{\mathbf{Y}}$ *is Gaussian,* $P(\hat{\mathbf{Y}}) = \mathcal{N}(\boldsymbol{\mu}_{\hat{\mathbf{Y}}}, \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}})$*. If we choose the distribution over* $\hat{\mathbf{Y}}_{anc}$ *to be* $P(\hat{\mathbf{Y}}_{anc}) = \mathcal{N}(\boldsymbol{\mu}_{anc}, \boldsymbol{\Sigma}_{anc})$*, where* $\boldsymbol{\mu}_{anc} = \boldsymbol{\mu}_{\hat{\mathbf{Y}}}$ *and* $\boldsymbol{\Sigma}_{anc} = \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}} + \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}\boldsymbol{\Sigma}_{like}^{-1}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}$*, then* $P(\hat{Y}_{MAP}(\hat{\mathbf{Y}}_{anc})) = P(\hat{\mathbf{Y}} \mid \mathbf{y}_{obs})$*.*

Our proof follows a similar technique to Pearce et al. [90] and is provided in Section 6.3.1. Just as in Pearce et al. [90], we use the approximation $\boldsymbol{\Sigma}_{\text{anc}} \approx \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}$ to make it tractable to sample from the anchor distribution.[1] With this assumption, the anchor point sampling distribution $\hat{\mathbf{Y}}_{\text{anc}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}})$ becomes approximately equal to the output prior distribution $P(\hat{\mathbf{Y}})$.

Generative Model While RMS can be used to sample from the posterior, every new sample requires solving a new optimization problem. Instead, we propose to learn a generative model to approximate the MAP function itself. Let g be a neural network parameterized by some vector, ϕ, that takes as input a sample from the anchor distribution, $\hat{\mathbf{Y}}_{\text{anc}}$, and outputs the MAP solution $\hat{\mathbf{Y}}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})$. Since we are using the prior distribution as our anchor distribution, we can sample anchor points using $\hat{\mathbf{Y}}_{\text{anc}} = f(\mathbf{x}_{\text{sample}}; \boldsymbol{\theta}_{\text{anc}})$, where $\boldsymbol{\theta}_{\text{anc}} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$. If we assume that g has enough expressive power to represent the MAP function, then $g(\hat{\mathbf{Y}}_{\text{anc}}; \boldsymbol{\phi}) = \hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})$, if $\boldsymbol{\phi}$ is equal to:

$$\arg\min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta}_{\text{anc}}} \sum_{i=1}^{N} \left(\|y^i_{\text{obs}} - g(x^i_{\text{obs}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})\|_2^2 + \sigma_\epsilon^2 \boldsymbol{\delta}^T \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\delta} \right) \tag{6.3}$$

where $\delta_j = g(x^j_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi}) - f(x^j_{\text{sample}}; \boldsymbol{\theta}_{\text{anc}})$. See Section 6.3.2 for a step-by-step derivation.

Low-Dimensional Embedding In practice, of course, it is expensive to compute the regularization term, $\sigma_\epsilon^2 \boldsymbol{\delta}^T \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\delta}$, for a large set of sample points $\mathbf{x}_{\text{sample}}$. Instead, we approximate this term with $\beta \|\delta\|_2^2$ where β is a hyper-parameter. This assumes independence and uniform variance of the prior, but we found it to work well in practice. Finally, optimizing the generative function g over the the space of all anchor parameters $\boldsymbol{\theta}_{\text{anc}}$ is impractical, as this space is too high dimensional. Instead, we construct a low-dimensional embedding for $\boldsymbol{\theta}_{\text{anc}}$.

There are two key reasons why we would expect to be able to decrease the representational power of the anchor distribution and maintain the same of fidelity in the posterior estimation. (1) Because of the nature of neural networks, there are many settings of parameters $\boldsymbol{\theta}$ that would result in the same output vector $\hat{\mathbf{Y}}$. Because we are focusing on the posterior of $\hat{\mathbf{Y}}$, we need less representational power to model $\hat{\mathbf{Y}}$. (2) And, maybe more importantly, because the posterior parameters are highly correlated, we would expect to need less expressive power to represent the posterior distribution than the prior distribution.

Thus, we would like to construct a simpler estimate of the prior space using a low-dimensional embedding vector, $\mathbf{z}$. That is, we want our generative model to take as input the embedding vector $\mathbf{z} \sim \mathcal{N}(0, I)$, instead of $\boldsymbol{\theta}_{\text{anc}}$, in order to estimate the MAP function. To do this, we need to learn a mapping from anchor parameters $\boldsymbol{\theta}_{\text{anc}}$ to embedding vectors $\mathbf{z}$. For our experiments, we used a 1-1 embedding scheme where we sample k parameters from the true prior $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k \sim P_{\text{prior}}(\boldsymbol{\theta})$

[1] Pearce et al. [90] argue that this approximation, in general, causes RMS to only over-estimate the posterior variance.

and k independent samples from our embedding $\mathbf{z}_1, \ldots, \mathbf{z}_k$. We then jointly optimize over $\boldsymbol{\phi}$ and $\mathbf{z}_1, \ldots, \mathbf{z}_k$ as follows:

$$\underset{\boldsymbol{\phi}, \mathbf{z}_1, \ldots, \mathbf{z}_k}{\arg\min} \mathbb{E}_{\mathbf{x}_{\text{sample}}} \sum_{j=1}^{k} \sum_{i=1}^{N} \|y^i_{\text{obs}} - g(x^i_{\text{obs}}, \mathbf{z}_j + \boldsymbol{\epsilon}; \boldsymbol{\phi})\|_2^2 + \beta \|\delta\|_2^2 + \mathcal{L}_{\text{reg}}(\mathbf{z}_1, \ldots, \mathbf{z}_k), \tag{6.4}$$

where $\delta_j = g(x^j_{\text{sample}}, \mathbf{z}_j + \boldsymbol{\epsilon}; \boldsymbol{\phi}) - f(x^j_{\text{sample}}; \boldsymbol{\theta}_j)$, $x^j_{\text{sample}} \sim P_{\text{sample}}(x)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$ is a noise injection vector that improves the smoothness of interpolations between embedding vectors, and $\mathcal{L}_{\text{reg}}(\mathbf{z}_1, \ldots, \mathbf{z}_k)$ is a regularizer to keep $\mathbf{z}_1, \ldots, \mathbf{z}_k$ roughly normally distributed, which allows us to easily sample from the embedding space. For our experiments we use the KL divergence between $\mathbf{z}$ and the normal distribution, $\mathcal{L}_{\text{reg}}(\mathbf{z}_1, \ldots, \mathbf{z}_k) = D_{\text{KL}}(\mathcal{N}(\bar{\mathbf{z}}, s_{\mathbf{z}}), \mathcal{N}(0, 1))$. Figure 6.1 illustrates how we can sample from this embedding space to construct posterior functions.

Classification Our method uses the assumption that the prior over $\hat{\mathbf{Y}}$ is approximately normally distributed. Of course, this approximation is only exact when either the network is linear or the final layer is infinitely wide and does not have an activation function (assuming the prior parameters are normally distributed). Thus, in classification tasks, where it is beneficial to a soft-max to the final output, this assumption is violated. To get around this, we add the anchor loss to the pre-softmax outputs for classification tasks.

6.3 Proofs

6.3.1 Proof of Theorem 5

Proof. We will start by showing that $P(\mathbf{y}_{\text{obs}}|\hat{\mathbf{Y}})$ is normally distributed.

Let $\hat{Y}_i{}' = f(x^i_{\text{obs}}; \boldsymbol{\theta})$ be a transformation of the random variable $\boldsymbol{\theta}$ corresponding to the outputs of the function f parameterized by $\boldsymbol{\theta}$ evaluated at observation points $\mathbf{x}_{\text{obs}}$. Note that by definition of $\mathbf{y}_{\text{obs}}$, for each $i \in 1, \ldots, N$:

$$y^i_{\text{obs}} = f(x^i_{\text{obs}}; \boldsymbol{\theta}) + \epsilon_i = \hat{Y}_i{}' + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon)$.

By assumption: $[\hat{\mathbf{Y}}, \hat{\mathbf{Y}}']^T$ is normally distributed. Since $\boldsymbol{\epsilon}$ is normally distributed and independent of $[\hat{\mathbf{Y}}, \hat{\mathbf{Y}}']^T$, then the joint distribution $[\hat{\mathbf{Y}}, \hat{\mathbf{Y}}', \boldsymbol{\epsilon}]^T$ is also normally distributed.

Now, by applying a linear transformation, we obtain the joint variable $\begin{bmatrix} \hat{\mathbf{Y}} \\ \hat{\mathbf{Y}}' \\ \mathbf{y}_{\text{obs}} \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & I & I \end{bmatrix} \begin{bmatrix} \hat{\mathbf{Y}} \\ \hat{\mathbf{Y}}' \\ \boldsymbol{\epsilon} \end{bmatrix}$, which is also normally distributed (as it is a linear transformation of a Gaussian random variable).

Now, since the conditional and marginal distributions of a Gaussian distribution are also Gaussian, $P(\mathbf{y}_{\text{obs}}, \hat{\mathbf{Y}}'|\hat{\mathbf{Y}})$ and $P(\mathbf{y}_{\text{obs}}|\hat{\mathbf{Y}})$ are also Gaussian.

Now, using Bayes' rule, we can show $P(\hat{\mathbf{Y}}|\mathbf{y}_{\text{obs}})$ is also normally distributed.

$$\begin{aligned} P(\hat{\mathbf{Y}} \mid \mathbf{y}_{\text{obs}}) &\propto P(\mathbf{y}_{\text{obs}} \mid \hat{\mathbf{Y}}) P(\hat{\mathbf{Y}}) \\ &= \mathcal{N}(\hat{\mathbf{Y}}|\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}) \mathcal{N}(\hat{\mathbf{Y}}|\boldsymbol{\mu}_{\hat{\mathbf{Y}}}, \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}) \end{aligned}$$

for some mean and covariance $\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}$. Since the product of two multivariate Gaussians is a Gaussian, we know that:

$$P(\hat{\mathbf{Y}} \mid \mathbf{y}_{\text{obs}}) = \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}) \tag{6.5}$$

where

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\boldsymbol{\Sigma}_{\text{like}}^{-1} + \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \right)^{-1}, \quad \boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\mu}_{\text{like}} + \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \mu_{\hat{\mathbf{Y}}}. \tag{6.6}$$

Now we consider at the distribution $P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}))$ where $\hat{\mathbf{Y}}_{\text{anc}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}})$. We will show that, when we set

$$\boldsymbol{\mu}_{\text{anc}} = \boldsymbol{\mu}_{\hat{\mathbf{Y}}}, \quad \boldsymbol{\Sigma}_{\text{anc}} = \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}} + \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}$$

the distribution $P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}))$ becomes equal to the posterior distribution $\mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}})$. There are three steps needed to show equality between these distributions:

1. Show that $P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}))$ is normally distributed for some mean and variance, $\boldsymbol{\mu}_{\text{post}}^{\text{RMS}}, \boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}}$.
2. Show that $\boldsymbol{\mu}_{\text{post}}^{\text{RMS}} = \boldsymbol{\mu}_{\text{post}}$.
3. Show that $\boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}} = \boldsymbol{\Sigma}_{\text{post}}$.

Using the same reasoning as above, in the limit as $M \to \infty$,

$$\begin{aligned} P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})) &= \arg\max_{\hat{\mathbf{Y}}} P(\mathbf{y}_{\text{obs}}|\hat{\mathbf{Y}}) P_{\text{anc}}(\hat{\mathbf{Y}}) \\ &= \arg\max_{\hat{\mathbf{Y}}} \mathcal{N}(\hat{\mathbf{Y}}|\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}) \mathcal{N}(\hat{\mathbf{Y}}|\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}}) \end{aligned}$$

Since the max of a Gaussian is the mean, then

$$F_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}}) = A\hat{\mathbf{Y}}_{\text{anc}} + b \tag{6.7}$$

where we define:

$$A = \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \tag{6.8}$$

$$b = \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\mu}_{\text{like}} \tag{6.9}$$

We will now show that $\mathbb{E}[\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})] = \boldsymbol{\mu}_{\text{post}}$. Because we set $\boldsymbol{\mu}_{\text{anc}} = \boldsymbol{\mu}_{\hat{\mathbf{Y}}}$:

$$\begin{aligned} \mathbb{E}[\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})] &= \mathbb{E}[A\hat{\mathbf{Y}}_{\text{anc}} + b] \\ &= A\mathbb{E}[\hat{\mathbf{Y}}_{\text{anc}}] + b \\ &= A\boldsymbol{\mu}_{\hat{\mathbf{Y}}} + b \\ &= \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\mu}_{\hat{\mathbf{Y}}} + \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\mu}_{\text{like}} \\ &= \boldsymbol{\mu}_{\text{post}} \end{aligned}$$

Finally, we will show that $\mathbb{V}\text{ar}[\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})] = \boldsymbol{\Sigma}_{\text{post}}$.

$$\begin{aligned}
\mathbb{V}\text{ar}[\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})] &= \mathbb{V}\text{ar}[A\hat{\mathbf{Y}}_{\text{anc}} + b] \\
&= A\mathbb{V}\text{ar}[\hat{\mathbf{Y}}_{\text{anc}}]A^T \\
&= (\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1})(\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}} + \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}})(\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1})^T \\
&= (\boldsymbol{\Sigma}_{\text{post}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}})(\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1}\boldsymbol{\Sigma}_{\text{post}}) \\
&= \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1}\boldsymbol{\Sigma}_{\text{post}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\text{post}} \\
&= \boldsymbol{\Sigma}_{\text{post}}(\boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} + \boldsymbol{\Sigma}_{\text{like}}^{-1})\boldsymbol{\Sigma}_{\text{post}} \\
&= \boldsymbol{\Sigma}_{\text{post}}
\end{aligned}$$

So, $P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})) \sim \mathcal{N}(\boldsymbol{\mu}_{\text{post}}^{\text{RMS}}, \boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}})$ where $\boldsymbol{\mu}_{\text{post}}^{\text{RMS}} = \boldsymbol{\mu}_{\text{post}}$ and $\boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}} = \boldsymbol{\Sigma}_{\text{post}}$.

Thus, in the limit as $M \to \infty$, $P(\hat{Y}_{\text{MAP}}(\hat{\mathbf{Y}}_{\text{anc}})) = P(\hat{\mathbf{Y}}|\mathbf{y}_{\text{obs}})$.

□

6.3.2 Derivation of loss function

We start with

$$\boldsymbol{\phi} = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta}_{\text{anc}} \sim P_{\text{prior}}(\boldsymbol{\theta}_{\text{anc}})} \log P_{\text{like}}(\mathbf{y}_{\text{obs}} \mid g(\mathbf{x}_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})) + \log P_{\text{anc}}(g(\mathbf{x}_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi}))$$

If we choose $\boldsymbol{\mu}_{\text{anc}} = \boldsymbol{\mu}_{\hat{\mathbf{Y}}}$ and $\boldsymbol{\Sigma}_{\text{anc}} = \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}$, we can simplify the logarithm of the likelihood and anchor distributions as follows:

$$\begin{aligned}
\log P_{\text{like}}(\mathbf{y}_{\text{obs}} \mid g(\mathbf{x}_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})) &= \sum_i \log \mathcal{N}(y_{\text{obs}}^i | g(x_{\text{sample}}^i, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi}), \sigma_\epsilon) \\
&= -\frac{1}{2\sigma_\epsilon^2} \sum_i \|y_{\text{obs}}^i - g(x_{\text{sample}}^i, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})\|_2^2
\end{aligned}$$

$$\begin{aligned}
\log P_{\text{anc}}(g(\mathbf{x}_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})) &= \log \mathcal{N}(g(\mathbf{x}_{\text{sample}}, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi}) | \boldsymbol{\mu}_{\hat{\mathbf{Y}}}, \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}) \\
&= -\frac{1}{2}\boldsymbol{\delta}^T \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\delta}
\end{aligned}$$

where $\delta_j = g(x_{\text{sample}}^j, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi}) - f(x_{\text{sample}}^j; \boldsymbol{\theta}_{\text{anc}})$.

Putting these together, we get:

$$\begin{aligned}
\boldsymbol{\phi} &= \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta}_{\text{anc}} \sim P_{\text{prior}}(\boldsymbol{\theta}_{\text{anc}})} -\frac{1}{2\sigma_\epsilon^2} \sum_i \|y_{\text{obs}}^i - g(x_{\text{sample}}^i, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})\|_2^2 - \frac{1}{2}\boldsymbol{\delta}^T \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\delta} \\
&= \arg\min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta}_{\text{anc}} \sim P_{\text{prior}}(\boldsymbol{\theta}_{\text{anc}})} \frac{1}{N} \sum_i \|y_{\text{obs}}^i - g(x_{\text{sample}}^i, \boldsymbol{\theta}_{\text{anc}}; \boldsymbol{\phi})\|_2^2 + \frac{1}{N}\sigma_\epsilon^2 \boldsymbol{\delta}^T \boldsymbol{\Sigma}_{\hat{\mathbf{Y}}}^{-1} \boldsymbol{\delta}
\end{aligned}$$

Table 6.1: Superconductor regression [50] results.

Methods	In Dist. Loss	In Dist. CI-width	OOD CI-correct	OOD-detect. AUC
Dropout	**0.10**	0.32	17.7%	0.56
DKL GP	0.11	0.74	37.9%	0.52
SNGP	**0.10**	1.45	87.8%	0.82
Ensemble - PR	0.12	**0.23**	83.8%	0.93
Ensemble - OR	0.11	**0.23**	75.4%	0.95
Epistemic NN	0.12	1.07	**95.3%**	0.92
GPN (Ours)	0.11	0.31	86.2%	**0.97**

Table 6.2: CIFAR-10 results with Out of Distribution (OOD) evaluation on SVHN.

Methods	In Dist. Accuracy	In Dist. Entropy	OOD Entropy	OOD-detect. AUC
Dropout	80.7%	0.579	1.214	0.76
DKL GP	79.3%	**0.027**	0.372	0.76
SNGP	77.2%	0.386	0.794	0.77
Ensemble - PR	**80.9%**	0.882	1.344	0.68
Ensemble - OR	80.7%	0.706	2.018	0.98
Epistemic NN	77.9%	0.630	1.145	0.76
GPN (Ours)	79.4%	0.621	**2.256**	**1.00**

6.4 Experiments

The goal of our experiments is to illustrate the ability of our method to accurately model epistemic uncertainty for OOD data while retaining high performance on in distribution data. To do so, similar to related work [76, 116], we consider two separate datasets, "In Distribution" and "Out of Distribution" (OOD). At training time, we provide labeled data from the In Distribution dataset and unlabeled data from both datasets. We evaluate each model on a hold-out set of In Distribution and OOD data.

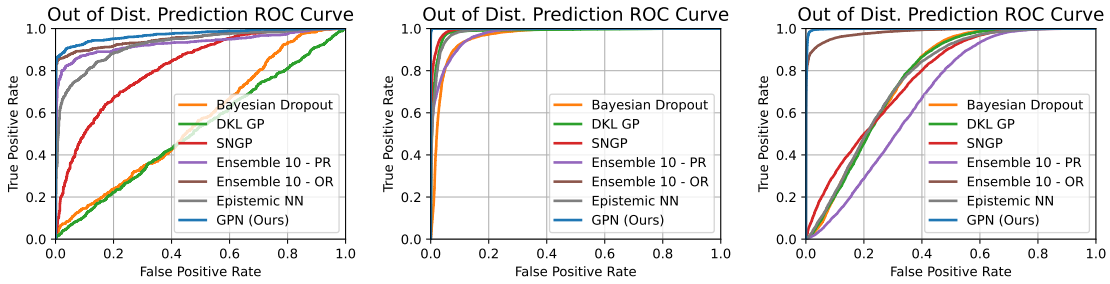
6.4.1 Tasks

Small Scale Regression Our first experiment is a small 1-dimensional regression problem where we can easily compute the ground truth. We provide each method with the same 6 observations. We compute 100 samples from the approximate posterior for each method. For ground truth, we use the Metropolis-Hastings MCMC algorithm.

High-Dimensional Regression For a high-dimensional regression task, we used a Superconductivity prediction dataset [50], the goal of which is to predict the critical temperature of different superconductive materials based on 81 features. For the In Distribution and OOD datasets, we split the full dataset based on the target values. Specifically, the In Distribution and OOD datasets

Table 6.3: MNIST results with Out of Distribution (OOD) evaluation on Fashion MNIST.

Methods	In Dist. Accuracy	In Dist. Entropy	OOD Entropy	OOD-detect. AUC
Dropout	99.0%	0.018	0.581	0.96
DKL GP	98.9%	0.091	1.223	0.99
SNGP	98.5%	2.300	2.300	0.99
Ensemble - PR	98.1%	0.062	0.952	0.97
Ensemble - OR	99.1%	0.085	2.066	**1.00**
Epistemic NN	98.8%	0.024	0.833	0.99
GPN (Ours)	99.1%	0.272	2.076	**1.00**



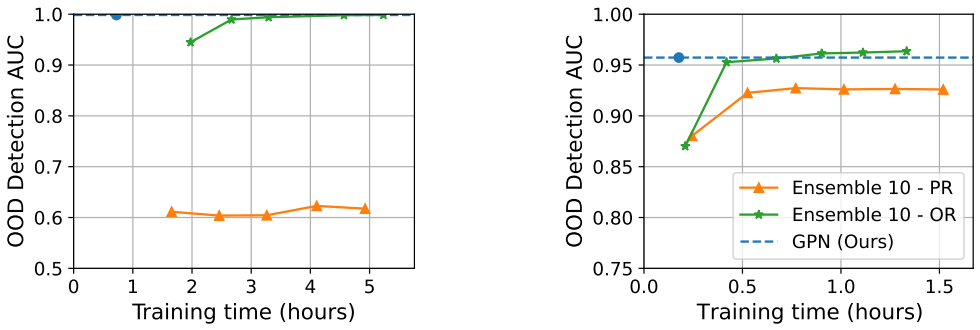
(a) Regression - Superconductor (b) Classification - MNIST (c) Classification - CIFAR-10

Figure 6.3: ROC curves for out of distribution prediction based on sample variance from 100 samples.

contained data with a target values in the intervals $[13.9, +\infty)$ and $[0, 13.9)$, which is roughly 58% of the data in distribution and 42% out of distribution. The unlabeled dataset we provide our method and the Output-Regularized ensemble method is the full training set. Regularization hyper-parameters were chosen by running each method with 5 different settings, then choosing the model with both a low validation loss and high CI-correct on a validation set.

Classification We run two different classification experiments: for one, we use the MNIST as our In Distribution dataset and Fashion MNIST as the OOD dataset, and for the other, we use the CIFAR-10 dataset as our In Distribution dataset and SVHN as the OOD dataset. The unlabeled dataset we provide our method and the Output-Regularized ensemble method consists of unlabeled data 50% from the In Distribution and 50% from the OOD datasets. As in the regression experiments, we ran each method with 5 different regularization parameters, then chose the model with both a high validation accuracy and OOD prediction performance on a validation set.

Table 6.4 details the labeled and unlabeled data provided to the agent at training time and the In Distribution and OOD data used for evaluation at test time.



(a) Classification - CIFAR-10

(b) Regression - Superconductor

Figure 6.4: OOD detection AUC vs. training time on the Superconductor and CIFAR-10 datasets for parameter and output-regularized ensembles and our method (GPN).

6.4.2 Baselines

We compare our method against 4 competing Bayesian methods: Bayesian Dropout [38], GPs, and anchor-regularized neural-network ensembles [90] with 10 ensemble members. For the high-dimensional problems, we need to use approximate GPs. Specifically, we use a grid-interpolated GP with Deep Kernel Learning (DKL) [121] implemented with the GPyTorch library [40] and the Spectral-normalized Neural Gaussian Process (SNGP) method [76]. For the ensembles, since every member requires a unique set of parameters (and anchor points) the number of ensemble members was limited by the memory capacity of our GPUs.

As described in Equation 6.3, our method a method to sample unlabeled data points. In low-dimensional problems, this can simply be a uniform sample from the interval of interest, for example $[-2, 2]$ in Figure 6.2. However, in high-dimensional problems, this requires access to an additional dataset of unlabeled training data. From a practical perspective, this is a reasonable setup as many real-world datasets have far more unlabeled data than labeled data.

This additional data obviously favors our method over the baseline methods as our method is the only method that is able to take advantage of this unlabeled data. To address this, we added an additional ensemble regularized using the same regularization as Equation 6.3; in other words, we regularize the outputs of the ensemble members towards outputs of sampled prior networks. We denote this new method as an "Output-regularized (OR)" Ensemble to distinguish it from the "Parameter-regularized" (PR) Ensemble. We show that despite having the access to the same additional unlabeled data, our method outperforms and scales better than this OR Ensemble method.

Each model uses the same 4-layer (Superconductor), 5-layer (MNIST), or 7-layer (CIFAR) architecture, except the ensembles, which use slightly smaller networks for each ensemble member (to fit on a single GPU).

6.4.3 Metrics

Because we are interested in epistemic uncertainty estimation, we cannot use common aleatoric uncertainty evaluation metrics, such as Expected Calibration Error (ECE) [44]. Instead, we evaluate each methods ability to construct useful posterior estimates using the following metrics.

OOD Detection One desired property of an epistemic uncertainty estimators is in predicting when a data point is outside the training distribution and, thus, any prediction made on such a data point will likely be incorrect. For this reason, OOD detection is a common metric used in epistemic uncertainty literature [76, 116]. For an ideal posterior distribution, we expect high sample variance on OOD data and low sample variance from In Distribution data. Thus, in our experiments, we use posterior sample variance to detect OOD data. Specifically, for each model, we use posterior variance over 100 samples from each model as a score to construct ROC curves and measure the measure the area under the curve (AUC).

Confidence Intervals / Entropy Another desired property of an epistemic uncertainty estimator for a regression task is in constructing confidence intervals. We would expect such confidence intervals (CIs) to contain the true class with high probability. We also expect that, on in distribution examples, confidence intervals are narrow. To construct these intervals, we take 100 samples from each model on every data point in the test data and compute the range of the middle 95% of the samples. *CI-correct* refers to the percentage of true labels that fall in this interval. Note that, unlike the other methods tested, ensembles are not a generative model. Thus, when trying to sample from the ensemble method, we can only sample as many functions as there are members of the ensemble; for our experiments this number is 10.

For classification, CIs are not as informative. Instead, we look at the posterior sample entropy. For an ideal posterior distribution, we expect entropy to be low for data inside the training set (data the model should have confidence on) and high for data outside the training set (which we expect the model to be uncertain on). To measure this sample entropy, we take 100 samples from each model on every data point and take the average of the post-softmax outputs. We then average this sample entropy for both the In Distribution and OOD datasets.

Scalability One key benefit of a generative model over an ensemble is efficiency: every sample from an ensemble method requires learning an entire new network from scratch. Generative models, on the other hand, are able to produce new samples with a single pass through the network. To illustrate the benefit of this efficiency in practice, we ran an experiment to measure OOD detection AUC vs. computation time for both the Superconductor and CIFAR-10 tasks. We trained 2 (Superconductor) and 5 (CIFAR) ensemble members at a time and computed the OOD-detection performance of the cumulative combined ensembles (both parameter-regularized and output-regularized). Each method is trained using an Nvidia 1080TI. Figure 6.4 shows our method is able to achieve high out of distribution prediction performance in significantly less computation time.

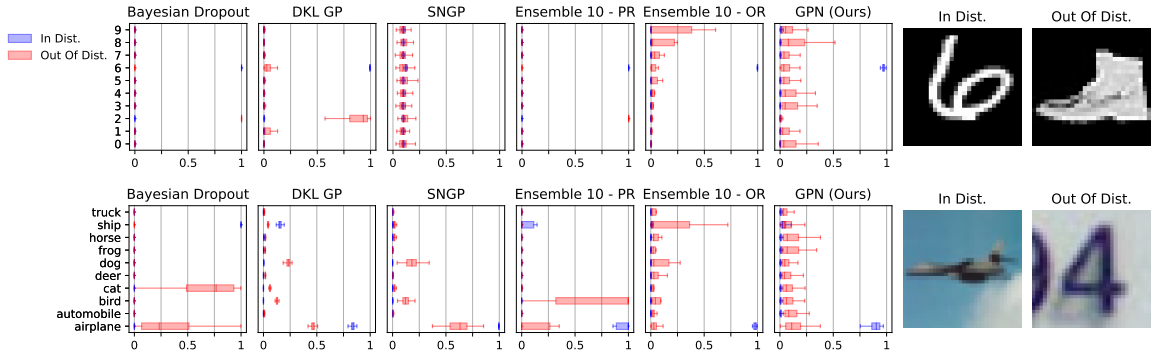


Figure 6.5: Boxplots of 100 posterior samples from every method for 2 test images, one from the In Distribution dataset and one from the OOD dataset.

Table 6.4: Training and test datasets used for each experiment task.

Task	Training		Testing	
	Labeled	Unlabeled	In Dist.	OOD
Small Scale	6 points	$\mathcal{U}[-2, 2]$	N/A	$\mathcal{U}[-2, 2]$
Superconductor	train set where $y_i \in [13.9, +\infty)$	full train set	test set where $y_i \in [13.9, +\infty)$	test set where $y_i \in [0, 13.9)$
MNIST	MNIST train	50% MNIST train 50% F-MNIST train	MNIST test	F-MNIST test
CIFAR-10	CIFAR-10 train	50% CIFAR-10 train 50% SVHN train	CIFAR-10 test	SVHN test

6.4.4 Network Architectures

For the superconductor regression problem, we used a 4-layer network with 4 128-unit wide linear layers for the ensemble methods and a 4-layer network with 4 512-unit wide linear layers for the other methods. For the MNIST classification problem, we used a 5-layer network with 2 convolution layers and 3 128-unit wide linear layers for our the ensemble methods and a 5-layer network with 2 convolution layers and 3 512-unit wide linear layers for our the other methods. For the CIFAR classification problem, we used a 7-layer network with 3 convolution layers and 4 256-unit wide linear layers for our the ensemble methods and a 7-layer network with 3 convolution layers and 4 512-unit wide linear layers for our the other methods.

6.4.5 Hyper parameters

The non-network architecture hyper-parameters we used for GPN were roughly consistent across all experiments, except for the regularization coefficient, β. As stated in our experiments section, we performed a small search over 5 values of β for each experiment. The full set of hyper-

Table 6.5: GPN Hyper-parameters used for our experiments.

Hyper-parameter	Value
# of sampled embeddings (k)	100
Embedding dimension	10
Regularization coefficient (β)	0.1
Bootstrap network ($\boldsymbol{\theta}$)	2-layer linear model
Prior variance ($\boldsymbol{\Sigma}_{\text{prior}}$)	40 (layer 1), 10 (layer 2)
Bootstrap activation	Tanh

parameters are shown in Table 6.5

6.5 Results and Discussion

Figure 6.2 shows the results of our small scale regression task. We can see our performs just as well or better than the ensemble method at predicting the ground truth posterior distribution and performs significantly better than dropout. The GP method seems to outperform our method in this small scale setting. However, this trend does not extend to the higher dimensional problems.

Tables 6.1 to 6.3 show the results of our high-dimensional regression and classification tasks and Figure 6.3 shows the full ROC curves for OOD prediction on these tasks. While all methods are able to achieve high In Distribution performance (loss or accuracy), the GPN is able to consistently achieve the highest OOD-detection AUC. The SNGP and OR-ensemble are also able to achieve high OOD-detection AUC, but have other drawbacks. Specifically, SNGP has a very high CI-width on In Distribution data for the regression task and a very small contrast in sample entropy between the In Distribution and OOD data for the classification tasks. And, while the OR-ensemble performs well, Figure 6.4 shows GPN scales much better.

Figure 6.5 shows two informative test images, one from the In Distribution dataset and one from the OOD dataset, for both MNIST/Fashion MNIST and CIFAR-10/SVHN, along with box plots of sampled PMFs from each of the learned models. While all methods predict the correct label with high precision on the in distribution example, ours has a uniquely wide sample distribution on the OOD example, suggesting high predicted epistemic uncertainty. This plot illustrates our method's ability to form credible posterior distributions.

6.6 Conclusion

In this chapter we introduce Generative Posterior Networks (GPNs), a method that uses unlabeled data to learn a generative model of the Bayesian posterior distribution. We prove that under mild assumptions, GPNs approximate samples from the true posterior. We then show empirically that our method significantly outperforms competing epistemic uncertainty predictions techniques on high-dimensional classification tasks and scales much better than ensembling methods, the closest

performing baseline.

Chapter 7

Conclusions and Future Directions

In this thesis, we introduced novel methods for ensuring RL algorithms, both at training time and deployment time. The goal of these approaches is facilitate the application of RL algorithms on the real-world with high potential for positive societal impact, like Tokamak control for nuclear fusion power.

In Chapter 3, we introduced a new technique for constructing non-linear, neural network policy classes for which we can provide robustness guarantees. We do this by constructing a convex set of all robustly stabalizable linear controllers and projecting the policy onto this set. We prove that this projection does indeed produce policies that are robustly stabalizable to any disturbance in a given disturbance class. Through various experiments on both synthetic domains and a simulated quadrotor and microgrid domain, we shows empirically that our method is able to out-perform other robust control methods in the average case and is robust to worst-case disturbances.

In Chapter 4, we propose Analogous Safe-state Exploration (ASE), a new safe exploration method which is uniquely able to achieve three criteria simultaneously: (1) ASE is able account for unknown, stochastic dynamics, (2) ASE provably maintains safety throughout the entire learning process and is provably optimal (in the PAC-MDP sense), and (3) ASE guides exploration, significantly improving the empirical sample efficiency. We test our method against both unsafe and safe exploration methods on a grid-world and platformer game and show that ASE improves sample efficiency over other safe methods never taking an unsafe action through the entire traing process.

In Chapter 5, we dive into offline RL and introduce Projected Off-Policy Q-Learning (POP-QL), a new method for improving performance of offline TD-based RL algorithms. By projecting both the policy and sampling distribution onto the convex set defined by the Bellman contraction mapping condition, POP-QL is able to prevent divergence of the approximate value function caused by off-policy TD updates. In practice, we show that POP-QL significantly improves offline performance over vanilla Soft Actor-Critic (SAC) and even out-performs the state-of-the-art method, Conservative Q-Learning (CQL), on tasks where the dataset is severly sub-optimal.

Finally, in Chapter 6 we propose an new epistemic uncertainty estimation method for deep neural

networks, Gaussian Posterior Networks (GPNs). Using unlabeled data, GPNs approximate samples from the Bayesian posterior distribution. Empirically, we find our method out-performs other methods on out-of-distribution detection tasks without sacrificing in-distribution performance on both regression and classification tasks. This type of epistemic uncertainty predictor could be very helpful for use in offline Model-Based RL.

The ultimate goal of this thesis is to provide techniques and algorithms that allow RL to be applied on highly impactful real-world problems. However, through my experience with these approaches, I see some of these approaches to be more promising directions forward than others. In order guarantee safety throughout the training process, safe RL techniques require significant prior knowledge that are very challenging to construct in practice, especially on complex domains such Tokamak control on power-grid optimization. For this reason I think offline RL approaches (potentially with online fine-tuning) seem to be the most promising path forward for safely training RL policies in practice. There have already been successful applications of offline RL [70, 71]. However, I think there is still significant room for improvement on performance in these domains. Our approach, POP-QL, makes progress in moving this field forward. The more progress made in this area of offline RL and robustness, the easier it becomes to take advantage of RL algorithms for highly impactful real-world problems.

Bibliography

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006. 2.2

[2] David Abel, D Ellis Hershkowitz, and Michael L Littman. Near optimal behavior via approximate state abstraction. *arXiv preprint arXiv:1701.04113*, 2017. 2.2.1

[3] Murad Abu-Khalaf, Frank L Lewis, and Jie Huang. Policy Iterations on the Hamilton–Jacobi–Isaacs Equation for H_∞ State Feedback Control With Input Saturation. *IEEE Transactions on Automatic Control*, 51(12):1989–1995, 2006. 2.2.2

[4] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017. 2.2.2

[5] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable Convex Optimization Layers. In *Advances in Neural Information Processing Systems*, pages 9558–9570, 2019. 3.4.1, 3.4.2

[6] Anayo K. Akametalu, Shahab Kaynama, Jaime F. Fisac, Melanie Nicole Zeilinger, Jeremy H. Gillula, and Claire J. Tomlin. Reachability-based safe learning with Gaussian processes. In *53rd IEEE Conference on Decision and Control, CDC 2014*, 2014. 2.2.1, 4.2.1, 4.2.1

[7] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving Rubik's Cube with a Robot Hand. *arXiv preprint arXiv:1910.07113*, 2019. 2.2.2

[8] Eitan Altman. *Constrained Markov Decision Processes*, volume 7. CRC Press, 1999. 2.2.2

[9] Brandon Amos and J Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 136–145, 2017. 3.4.3, 3.5.2

[10] Karl J Astrom. *Introduction to Stochastic Control Theory*. Elsevier, 1971. 1

[11] Tamer Başar and Pierre Bernhard. *H_∞-Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach*. Springer Science & Business Media, 2008. 2.2.2

[12] Heinz H Bauschke. *Projection algorithms and monotone operators*. PhD thesis, Dept. of Mathematics and Statistics, Simon Fraser University, 1996. 3.5.1

[13] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016. 2.1.3

[14] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017. 2.2.1, 4.2.1, 4.2.1, 4.2.1

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 2.2

[16] Erdem Bıyık, Jonathan Margoliash, Shahrouz Ryan Alimo, and Dorsa Sadigh. Efficient and safe exploration in deterministic markov decision processes with unknown transition models. *arXiv preprint arXiv:1904.01068*, 2019. 4.2.1

[17] Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C. Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE Trans. Robotics*, 2010. 2.2.1

[18] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. 2.3.1

[19] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015. 2.3.1

[20] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 3.4.4

[21] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15. Siam, 1994. 2.2.2, 3.2.1, 3.2.1, 3.3.1, 3.3.2, 3.6.7

[22] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct): 213–231, 2002. 2.1.3, 2.1.4, 4.3, 4.7, 4.7.3

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 5.4

[24] Samuel Burer and Renato DC Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2): 329–357, 2003. 5.3.3

[25] Lucian Buşoniu, Tim de Bruin, Domagoj Tolić, Jens Kober, and Ivana Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28, 2018. 2.2.2

[26] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov Control. In *Advances in Neural Information Processing Systems*, pages 3245–3254, 2019. 2.2.2

[27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple

framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 6.1

[28] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602 (7897):414–419, 2022. 1

[29] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R Arabnia. A brief review of domain adaptation. *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020*, pages 877–894, 2021. 6.1

[30] Yantao Feng, Brian DO Anderson, and Michael Rotkowitz. A game theoretic algorithm to compute local stabilizing solutions to HJBI equations in nonlinear H_∞ control. *Automatica*, 45(4):881–888, 2009. 2.2.2

[31] Claude-Nicolas Fiechter. Efficient reinforcement learning. In *Proceedings of the seventh annual conference on Computational learning theory*, pages 88–97. ACM, 1994. 2.1.4

[32] Wendell H. Fleming and William M. McEneaney. Risk-sensitive control on an infinite time horizon. *SIAM J. Control Optim.*, 1995. 2.2.1

[33] Jeffrey P Freidberg. *Plasma physics and fusion energy*. Cambridge university press, 2008. 1

[34] Stefan R Friedrich and Martin Buss. A robust stability approach to robot reinforcement learning based on a parameterization of stabilizing controllers. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3365–3372. IEEE, 2017. 2.2.2

[35] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020. 3, 5.4, 5.1, 5.2

[36] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 5.5

[37] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019. 2.3.3, 5.4

[38] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016. 6.4.2

[39] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015. 6.1

[40] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018. 6.4.2

[41] Carles Gelada and Marc G Bellemare. Off-policy deep reinforcement learning by bootstrap-

ping the covariate shift. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3647–3655, 2019. 2.3.3

[42] Xinyang Geng. Jaxcql: a simple implementation of sac and cql in jax. *GitHub repository*, 2022. URL `https://github.com/young-geng/JaxCQL`. 5.4

[43] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003. 2.2.1

[44] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017. 6.4.3

[45] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019. 5.4

[46] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. 2.1.2, 2.1.3, 5.1, 5.4

[47] Wassim M Haddad and VijaySekhar Chellaboina. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton University Press, 2011. 3.2.1, 2

[48] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019. 2.2

[49] Assaf Hallak and Shie Mannor. Consistent on-line off-policy evaluation. In *International Conference on Machine Learning*, pages 1372–1383. PMLR, 2017. 2.3.3

[50] Kam Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346–354, 2018. 6.1, 6.4.1

[51] Minghao Han, Yuan Tian, Lixian Zhang, Jun Wang, and Wei Pan. H_∞ Model-free Reinforcement Learning with Robust Stability Guarantee. *CoRR*, 2019. 2.2.2

[52] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN 2008, 16th European Symposium on Artificial Neural Networks, Proceedings*, 2008. 2.2.1

[53] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016. 5.5

[54] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. Bayesian deep ensembles via the neural tangent kernel. *arXiv preprint arXiv:2007.05864*, 2020. 2.3.1, 6.1

[55] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013. 2.3.1

[56] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010. 2.1.4

[57] Ray Jiang, Tom Zahavy, Zhongwen Xu, Adam White, Matteo Hessel, Charles Blundell, and Hado Van Hasselt. Emphatic algorithms for deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5023–5033. PMLR, 7 2021. 2.3.3

[58] Ming Jin and Javad Lavaei. Stability-certified reinforcement learning: A control-theoretic perspective. *arXiv preprint arXiv:1810.11505*, 2018. 2.2.2

[59] Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 306–312, 2003. 2.2.1

[60] Julian Kates-Harbeck, Alexey Svyatkovskiy, and William Tang. Predicting disruptive instabilities in controlled fusion plasmas through deep learning. *Nature*, 568(7753):526–531, 2019. 1

[61] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002. 4.6.5

[62] Hassan K Khalil and Jessy W Grizzle. *Nonlinear Systems*, volume 3. Prentice Hall Upper Saddle River, NJ, 2002. 3.3.3

[63] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020. 2.3.3

[64] J Kolter. The fixed points of off-policy td. *Advances in Neural Information Processing Systems*, 24:2169–2177, 2011. (document), 3, 2.3.3, 5.1, 5.1, 5.2.2, 5.3.1, 5.3.2, 5.5

[65] Mayuresh V Kothare, Venkataramanan Balakrishnan, and Manfred Morari. Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 32(10): 1361–1379, 1996. 2.2.2, 3.6.2

[66] Roman Kuiava, Rodrigo A Ramos, and Hemanshu R Pota. A New Method to Design Robust Power Oscillation Dampers for Distributed Synchronous Generation Systems. *Journal of Dynamic Systems, Measurement, and Control*, 135(3), 2013. 3.6.7

[67] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019. 2.3.3, 5.4

[68] Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*, 2020. 2.3.3

[69] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 2.3.3, 5, 5.4

[70] Aviral Kumar, Amir Yazdanbakhsh, Milad Hashemi, Kevin Swersky, and Sergey Levine.

Data-driven offline optimization for architecting hardware accelerators. *arXiv preprint arXiv:2110.11346*, 2021. 7

[71] Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*, 2022. 7

[72] Quang Linh Lam, Antoneta Iuliana Bratcu, and Delphine Riu. Frequency Robust Control in Stand-alone Microgrids with PV Sources: Design and Sensitivity Analysis. In *Symposium de Genie Electrique*, 2016. 3.6.1, 3.6.6

[73] Tor Lattimore and Marcus Hutter. Pac bounds for discounted mdps. In *International Conference on Algorithmic Learning Theory*, pages 320–334. Springer, 2012. 4.2

[74] Jongmin Lee, Wonseok Jeon, Byungjun Lee, Joelle Pineau, and Kee-Eung Kim. Optidice: Offline policy optimization via stationary distribution correction estimation. In *International Conference on Machine Learning*, pages 6120–6130. PMLR, 2021. 2.3.3

[75] Derong Liu, Hongliang Li, and Ding Wang. Neural-network-based zero-sum game for discrete-time nonlinear systems via iterative adaptive dynamic programming algorithm. *Neurocomputing*, 110:92–100, 2013. 2.2.2

[76] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33: 7498–7512, 2020. 2.3.1, 6.4, 6.4.2, 6.4.3

[77] Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *Advances in Neural Information Processing Systems*, 31, 2018. 2.3.3

[78] Biao Luo, Huai-Ning Wu, and Tingwen Huang. Off-Policy Reinforcement Learning for H_∞ Control Design. *IEEE Transactions on Cybernetics*, 45(1):65–76, 2014. 2.2.2

[79] Sridhar Mahadevan, Bo Liu, Philip Thomas, Will Dabney, Steve Giguere, Nicholas Jacek, Ian Gemp, and Ji Liu. Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. *arXiv preprint arXiv:1405.6757*, 2014. 2.3.3

[80] Gaurav Manek and J Zico Kolter. The pitfalls of regularization in off-policy TD learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 2.1, 2.3.2, 2.3.3, 5.3

[81] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 2.1.2, 2.2, 2.2.2, 5.5

[82] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2012. 2.2.1, 4.2.1

[83] Jun Morimoto and Kenji Doya. Robust Reinforcement Learning. *Neural Computation*, 17 (2):335–359, 2005. 2.2.2

[84] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in Neural Information Processing Systems*, 32, 2019. 2.3.3

[85] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019. 2.3.3, 5.4

[86] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer Science & Business Media, 2013. 3.5.1, 3.5.1

[87] Masahiro Ono, Marco Pavone, Yoshiaki Kuwata, and J. Balaram. Chance-constrained dynamic programming with application to risk-aware robotic space exploration. *Auton. Robots*, 2015. 2.2.1

[88] Ian Osband, Zheng Wen, Mohammad Asghari, Morteza Ibrahimi, Xiyuan Lu, and Benjamin Van Roy. Epistemic neural networks. *arXiv preprint arXiv:2107.08924*, 2021. 2.3.1

[89] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010. 6.1

[90] Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, pages 234–244. PMLR, 2020. 2.3.1, 6.1, 6.2, 6.2, 1, 6.4.2

[91] Theodore J. Perkins and Andrew G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3, 2002. 2.2.1

[92] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust Adversarial Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2817–2826. JMLR. org, 2017. 2.2.2, 3.6.2

[93] Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000. 2.3.3

[94] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. 4.2

[95] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 6.1

[96] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015. 2.1.2, 2.3.3

[97] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2.1.2, 3.6.2

[98] Sumeet Singh, Spencer M Richards, Vikas Sindhwani, Jean-Jacques E Slotine, and Marco Pavone. Learning stabilizable nonlinear dynamics with contraction-based regularization. *The International Journal of Robotics Research*, page 0278364920949931, 2020. 3.6.5

[99] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation

for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008. 2.1.3, 2.1.4, 4.3, 4.3.1, 4.3.3, 4.4, 4.5.1, 4.6.4, 4.6.5, 4.6.5, 4.6.5, 4.6.5, 4.7, 4.7.3

[100] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006. 2.1.4, 4.2

[101] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009. 2.1.4

[102] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, second edition edition, 2020. 2.3.2, 5.1

[103] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999. 2.1.2

[104] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 4.3.2

[105] Richard S Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent o (n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in neural information processing systems*, 21(21):1609–1616, 2008. 2.3.3

[106] Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17:2603–2631, 2016. 2.3.3

[107] István Szita and Csaba Szepesvári. Constrained policy optimization. In *Model-based reinforcement learning with nearly tight exploration complexity bounds, ICML 2010*, 2010. 4.2

[108] Adrien Ali Taïga, Aaron Courville, and Marc G Bellemare. Approximate exploration through state abstraction. *arXiv preprint arXiv:1808.09819*, 2018. 2.2.1

[109] Majid Alkaee Taleghan and Thomas G. Dietterich. Efficient Exploration for Constrained MDPs. In *2018 AAAI Spring Symposia*, 2018. 2.2.2

[110] Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987. 2.3.1

[111] Jonathan Taylor, Doina Precup, and Prakash Panagaden. Bounding performance loss in approximate mdp homomorphisms. In *Advances in Neural Information Processing Systems*, pages 1649–1656, 2009. 2.2.1

[112] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for MIT 6.832. *Working draft edition*, 3, 2009. 3.6.4

[113] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009. 2.3.1

[114] JN Tsitsiklis and B Van Roy. An analysis of temporal-difference learning with function approximation. *Rep. LIDS-P-2322. Lab. Inf. Decis. Syst. Massachusetts Inst. Technol. Tech. Rep*, 1996. 2.1.2, 2.3.2, 5.2.2

[115] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, 2016. 2.2.1, 4.2, 4.2.1, 4.2.1

[116] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International conference on machine learning*, pages 9690–9700. PMLR, 2020. 6.4, 6.4.3

[117] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019. 2.2

[118] Akifumi Wachi, Yanan Sui, Yisong Yue, and Masahiro Ono. Safe exploration and optimization of constrained mdps using gaussian processes. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, 2018. 2.2.1

[119] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 6.1

[120] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784. PMLR, 2015. 2.3.1

[121] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016. 2.3.1, 6.4.2

[122] Huai-Ning Wu and Biao Luo. Simultaneous policy update algorithms for learning the solution of linear continuous-time H_∞ state feedback control. *Information Sciences*, 222: 472–485, 2013. 2.2.2

[123] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019. 2.3.3

[124] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-Based Constrained Policy Optimization. In *International Conference on Learning Representations*, 2020. 2.2.2

[125] David D Yao, Shuzhong Zhang, and Xun Yu Zhou. A primal-dual semi-definite programming approach to linear quadratic control. *IEEE Transactions on Automatic Control*, 46(9): 1442–1447, 2001. 3.2.2

[126] Kaichao You, Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Universal domain adaptation. In *Proceedings of the IEEE/CVF conference on computer*

vision and pattern recognition, pages 2720–2729, 2019. 6.1

[127] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020. 2.3.3

[128] Kaiqing Zhang, Bin Hu, and Tamer Basar. Policy Optimization for $\mathcal{H}_2$ Linear Control with $\mathcal{H}_\infty$ Robustness Guarantee: Implicit Regularization and Global Convergence. In *Learning for Dynamics and Control*, pages 179–190. PMLR, 2020. 2.2.2

[129] Shangtong Zhang, Bo Liu, Hengshuai Yao, and Shimon Whiteson. Provably convergent two-timescale off-policy actor-critic with function approximation. In *International Conference on Machine Learning*, pages 11204–11213. PMLR, 2020. 2.3.3

[130] Kemin Zhou and John Comstock Doyle. *Essentials of Robust Control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998. 2.2.2