*International Journal of Research in Engineering and Science (IJRES)*
*ISSN (Online): 2320-9364, ISSN (Print): 2320-9356*
*www.ijres.org Volume 10 Issue 8 ||August 2022 ||PP. 01-186*

# Improving Efficiency and Efficacy in Reinforcement Learning through Deep Model-Based Algorithms: An Exploration of Online, Expressive, Offline, and Safe Learning Approaches

Junxia Deng
University of Southern California
California, USA

Ceil Hong. Zhang
Massachusetts Institute of Technology
Massachusetts, USA
zhanghongceil@pku.org.cn

# Abstract

Recent advances in deep reinforcement learning have demonstrated its great potential for real-world problems. However, two concerns prevent reinforcement learning from being applied: *Efficiency* and *Efficacy*. This dissertation studies how to improve the efficiency and efficacy of reinforcement learning by designing *deep model-based* algorithms. The access to dynamics models empowers the algorithms to plan, which is key to sequential decision making. This dissertation covers four topics: online reinforcement learning, the expressivity of neural networks in deep reinforcement learning, offline reinforcement learning, and safe reinforcement learning. For online reinforcement learning, we present an algorithmic framework with theoretical guarantees by utilizing a lower bound of performance the policy learned in the learned environment can obtain in the real environment. We also empirically verify the efficiency of our proposed method. For expressivity of neural networks in deep reinforcement learning, we prove that in some scenarios, the model-based approaches can require much less representation power to approximate a near-optimal policy than model-free approaches, and empirically show that this can be an issue in simulated robotics environments and a model-based planner can help. For offline reinforcement learning, we devise an algorithm that enables the policy to stay close to the provided expert demonstration set to reduce distribution shift, and we also conduct experiments to demonstrate the efficacy of our methods to improve the success rate for robotic arm manipulation tasks in simulated environments. For safe reinforcement learning, we propose a method that uses the learned dynamics model to certify safe states, and our experiments show that our method can learn a decent policy without a single safety violation during training in a set of simple but challenging tasks, while baseline algorithms have hundreds of safety violations.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Reinforcement learning (RL) is a natural formulation of sequential decision making. Unlike supervised learning which requires a labeled dataset, reinforcement learning works by interacting with the environment and learning from the signals received during the interactions. In the past few years, reinforcement learning has received a tremendous amount of attention due to its empirical success.

Among these RL algorithms, two classes are of special interest to researchers:

- *Model-based* RL algorithms extend RL algorithms by a dynamics model, which predicts the change of the environment after taking one action. The additional dynamics model empowers the algorithm to plan effectively, which is key to sequential decision making.

- *Deep* RL algorithms employ artificial neural networks, which are powerful function approximators, to deal with continuous state space and action space. Although we lack theoretical understanding of deep RL algorithms, they work surprisingly well in practice.

*Deep model-based* RL algorithms take the intersection of these two classes, hoping to combine the best of two worlds. What is more interesting is that instead of relying on

a given dynamics model, artificial neural networks enable RL algorithms to learn a dynamics model, which is highly non-linear in practice.

When applying RL algorithms to real-world problems, two significant aspects might impede the application: *Efficiency* and *Efficacy*. Efficiency indicates how good RL algorithms are at using the inputs, such as online data, offline data, and domain knowledge, while efficacy indicates how good RL algorithms have the desired properties, such as high total rewards, safety, and scalability. Without efficiency, RL algorithms may be immoderate, consuming too many resources; without efficacy, RL algorithms may be impotent, failing to produce expected properties. How can we design RL algorithms that are both efficient and effective?

In this dissertation, we focus on the following two concerns by designing deep model-based RL algorithms.

- *Low sample efficiency.* Current reinforcement learning algorithms require many samples to learn the policy, but samples can be expensive to obtain. A few solutions have been proposed for this concern. For example, one might consider using existing offline data or using online data more efficiently. The model-based approaches can use the dynamics model to plan, so it can often improve the sample efficiency.

- *Training-time safety violations.* In some scenarios, safety is critical, and violations lead to very undesired consequences. Thus, we would like to design RL algorithms that focus on safety. Here we not only care about the safety of the final policy but also care about the whole training process. The model-based methods can have great potential as they may foresee the possible danger and take actions to prevent it from happening.

## 1.1 Overview

This dissertation covers four topics in RL: online RL, the expressivity of neural networks in deep RL, offline RL, and safe RL. We will give an overview of each topic in the rest of the section.

One problem for model-based RL is how the dynamics model should be chosen, so that the policy learned on the learned dynamics models can generalize, as the wrong prediction of critical transitions might be exploited by the policy. The conventional loss for model learning is the mean squared error (MSE), but a small MSE doesn't mean good generalization as it cares about the average performance and doesn't emphasize critical transitions. To resolve the concern for generalization, in Chapter 2 we theoretically design an algorithmic framework to learn the model and policy simultaneously under the principle of optimism-in-face-of-the-uncertainty. We instantiate our algorithmic framework into a practical algorithm and conduct experiments to show that it can achieve high total rewards on a set of common continuous control benchmark tasks when the number of collected samples is limited.

Chapter 2 proposes a general theory for deep model-based RL, but one essential part of deep model-based methods, the neural network, remains mysterious. A deeper understanding of neural networks sheds light on designing better deep model-based RL algorithms. In Chapter 3, we study model-based RL through the lens of expressivity of neural networks. The more units a network has, the better it can approximate a function. In model-based RL algorithms, the dynamics model is approximated, while in model-free RL algorithms, the value function and the policy are approximated. Do these two classes of algorithms require the same level of expressivity to get near-optimal performance? The answer to the question helps us find out when model-based approaches have an advantage over model-free approaches. We answer this question by showing that model-based approaches require far less expressivity to achieve near-optimal performance in certain scenarios. We construct cases where the value function

and policy require an exponential (in horizon) large neural network with a constant depth to get near-optimal performance, while the dynamics model requires only a constant large neural network. This finding hints that expressivity might be an issue for model-free approaches. To mitigate this issue, we propose an algorithm that simply uses the dynamics model to bootstrap the weak value function to a stronger policy. Our algorithm is only involved during test and can be applied on top of many existing model-based/model-free algorithms. The empirical results show that it can improve the underlying algorithm on continuous control benchmark tasks.

With the theoretical understanding in Chapters 2 and 3, one might seek other methods to reduce the required samples. One popular choice is to make use of existing offline data. The main issue for traditional methods is distribution shift, which means that the distribution of states in training is different from the distribution of states during test. As a result, the errors of the policy accumulate and prevent the policy from obtaining high total rewards. To this end, in Chapter 4 we propose an algorithm to make local corrections so that the distribution of states during test stays close to the training distribution. Our algorithm can run both with and without interactions with the environment. Our empirical result shows that our algorithm can achieve higher success rates given the same offline data than baselines when the interactions with the environment are prohibited, and it can learn faster and have better sample efficiency when the interactions with the environment are permitted.

Finally we move on to the safety concern. The previous chapters, along with many other concurrent works, improve the sample efficiency of deep model-based RL algorithms by a lot, but it might be insufficient to address the safety concern as safety violations can have very serious consequences. Moreover, safety during the training process is also crucial. In Chapter 5, we aim to design a deep model-based RL algorithm to learn a policy without any training-time safety violations. One challenge is unknown irrecoverable states, which leads to a safety violation no matter

what actions to take. Our idea is that we use the learned dynamics model to certify "recoverable" states and limit ourselves to only exploring the certified states. We propose an algorithm to co-train a policy and the certificate to a growing set of states. Our experiments show that in a set of simple but challenging tasks, the proposed algorithm can learn a decent policy without a single safety violation, while the baselines have hundreds of safety violations.

## 1.2 Previously Published Works

Chapter 2 is based on the joint work with Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell and Tengyu Ma, and is published in ICLR 2019 [Luo et al., 2019a].

Chapter 3 is based on the joint work with Kefan Dong and Tengyu Ma, and is published in ICML 2021 [Dong et al., 2020].

Chapter 4 is based on the joint work with Huazhe Xu and Tengyu Ma, and is published in ICLR 2020 [Luo et al., 2019b].

Chapter 5 is based on the joint work with Tengyu Ma, and is published in NeurIPS 2021 [Luo and Ma, 2021].

## 1.3 Notations and General Setup

Throughout the dissertation, we consider the standard RL setup with an infinite-horizon Markov Decision Process (MDP). An MDP is specified by a tuple $(\mathcal{S}, \mathcal{A}, M, r, \gamma, \mu_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $M : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function which maps a state action pair to a probability distribution of the next state, $\gamma$ is the discount factor and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mu_0$ is the distribution of the initial state. Here $\Delta(\mathcal{X})$ denotes the family of distributions over a set $\mathcal{X}$. Unless otherwise specified, we assume the transition function is deterministic, which means $M(\cdot|s, a)$ is a Dirac measure. In this case, we use $M(s, a)$ to denote

the unique value of $s'$ and view $M$ as a function from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{S}$. In model-based reinforcement learning, we typically have two models, one of which is $M^*$, the true transition model in the environment. The other transition model is either given or learned, which we denote by $\widehat{M}$.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps a state to a distribution of actions, while a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ maps a state to an action. The value function $V$ for the policy is defined as is defined

$$V^{\pi,M}(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_t \sim \pi(s_t), s_{t+1} \sim M(s_t, a_t)\right].$$

Moreover, we define the state-action value function $Q$:

$$Q^{\pi,M}(s,a) \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_0 = a, a_t \sim \pi(s_t), s_{t+1} \sim M(s_t, a_t)\right].$$

The superscript $\pi$ and/or $M$ might be omitted for clarity.

An RL agent aims to find a policy $\pi$ that maximizes the expected total reward defined as

$$\eta(\pi) \triangleq \mathbb{E}_{s_0 \sim \mu_0}\left[V^{\pi}(s_0)\right],$$

where $\mu_0$ is the distribution of the initial state. The optimal policy $\pi^*$ is the one that maximizes $\eta(\pi)$:

$$\pi^* \triangleq \operatorname*{argmax}_{\pi} \eta(\pi),$$

and we use $Q^*$ (and $V^*$) as the shorthand of $Q^{\pi^*}$ (and $V^{\pi^*}$).

We use $\mathbb{I}[\cdot]$ to denote the indicator function. Let $\lfloor x \rfloor$ denote the floor function of $x$, that is, the greatest integer less than or equal to $x$. The Kullback-Leibler (KL) divergence between two distributions $p$ and $q$ are defined as

$$D_{\mathrm{KL}}(p \parallel q) = \mathbb{E}_{x \sim p}\left[\log \frac{p(x)}{q(x)}\right].$$

We use $\|\cdot\|$ to denote a norm in Euclidean space $\mathbb{R}^d$.

# Chapter 2

# Algorithmic Framework with Theoretical Guarantees

Model-based reinforcement learning (RL) is considered to be a promising approach to reduce the sample complexity that hinders model-free RL. However, the theoretical understanding of such methods has been rather limited. This chapter introduces a novel algorithmic framework for designing and analyzing model-based RL algorithms with theoretical guarantees. We design a meta-algorithm with a theoretical guarantee of monotone improvement to a local maximum of the expected reward. The meta-algorithm iteratively builds a lower bound of the expected reward based on the estimated dynamics model and sample trajectories, and then maximizes the lower bound jointly over the policy and the model. The framework extends the optimism-in-face-of-uncertainty principle to non-linear dynamics models in a way that requires *no explicit* uncertainty quantification. Instantiating our framework with simplification gives a variant of model-based RL algorithms Stochastic Lower Bounds Optimization (SLBO). Experiments demonstrate that SLBO achieves high performance when only one million or fewer samples are permitted on a range of continuous control benchmark tasks.

## 2.1 Background

In recent years deep reinforcement learning has achieved strong empirical success, including super-human performances on Atari games and Go [Mnih et al., 2015, Silver et al., 2017a] and learning locomotion and manipulation skills in robotics [Levine et al., 2016, Schulman et al., 2015b, Lillicrap et al., 2016]. Many of these results are achieved by model-free RL algorithms that often require a massive number of samples, and therefore their applications are mostly limited to simulated environments. Model-based deep reinforcement learning, in contrast, exploits the information from state observations explicitly — by planning with an estimated dynamics model — and is considered to be a promising approach to reduce the sample complexity. Indeed, empirical results [Deisenroth and Rasmussen, 2011, Deisenroth et al., 2013, Levine et al., 2016, Nagabandi et al., 2018, Kurutach et al., 2018, Pong et al., 2018a] have shown strong improvements in sample efficiency.

Despite promising empirical findings, many of *theoretical* properties of model-based deep reinforcement learning are not well-understood. For example, how does the error of the estimated model affect the estimation of the value function and the planning? Can model-based RL algorithms be guaranteed to improve the policy monotonically and converge to a local maximum of the value function? How do we quantify the uncertainty in the dynamics models?

It's challenging to address these questions theoretically in the context of deep RL with continuous state and action space and non-linear dynamics models. Due to the high-dimensionality, learning models from observations in one part of the state space and extrapolating to another part sometimes involves a leap of faith. The uncertainty quantification of the non-linear parameterized dynamics models is difficult — even without the RL components, it is an active but widely-open research area. Prior work in model-based RL mostly quantifies uncertainty with either heuristics or simpler models [Moldovan et al., 2015, Xie et al., 2016, Deisenroth and Rasmussen, 2011].

Previous theoretical work on model-based RL mostly focuses on either the finite-state MDPs [Jaksch et al., 2010, Bartlett and Tewari, 2009, Fruit et al., 2018, Lakshmanan et al., 2015, Hinderer, 2005, Pirotta et al., 2015, 2013], or the linear parametrization of the dynamics, policy, or value function [Abbasi-Yadkori and Szepesvári, 2011, Simchowitz et al., 2018, Deisenroth and Rasmussen, 2011, Sutton et al., 2012, Tamar et al., 2012], but not much on non-linear models. Even with an oracle prediction intervals[1] or posterior estimation, to the best of our knowledge, there was no previous algorithm with convergence guarantees for model-based deep RL.

Towards addressing these challenges, the main contribution of this chapter is to propose a novel algorithmic framework for model-based deep RL with theoretical guarantees. Our meta-algorithm (Algorithm 1) extends the optimism-in-face-of-uncertainty principle to non-linear dynamics models in a way that requires *no explicit* uncertainty quantification of the dynamics models.

In this chapter, we denote $V^{\pi} = V^{\pi, M^*}$ as the value function of a policy $\pi$ on the true environment $M^*$, and denote $\widehat{V}^{\pi} = V^{\pi, \widehat{M}}$ as the value function of the policy $\pi$ on the estimated model $\widehat{M}$. We design provable upper bounds, denoted by $D^{\pi, \widehat{M}}$, on how much the error can compound and divert the expected value $\widehat{V}^{\pi}$ of the imaginary rollouts from their real value $V^{\pi}$, in a neighborhood of some reference policy. Such upper bounds capture the intrinsic difference between the estimated and real dynamics model with respect to the particular reward function under consideration.

The discrepancy bounds $D^{\pi, \widehat{M}}$ naturally leads to a lower bound for the true value function:

$$V^{\pi} \geq \widehat{V}^{\pi} - D^{\pi, \widehat{M}}. \tag{2.1.1}$$

[1] We note that the confidence interval of parameters are likely meaningless for over-parameterized neural networks models.

Our algorithm iteratively collects batches of samples from the interactions with environments, builds the lower bound above, and then maximizes it over both the dynamics model $\widehat{M}$ and the policy $\pi$. We can use any RL algorithms to optimize the lower bounds, because it will be designed to only depend on the sample trajectories from a fixed reference policy (as opposed to requiring new interactions with the policy iterate.)

We show that the performance of the policy is guaranteed to monotonically increase, assuming the optimization within each iteration succeeds (see Theorem 2.4.1.) To the best of our knowledge, this is the first theoretical guarantee of monotone improvement for model-based deep RL.

Readers may have realized that optimizing a robust lower bound is reminiscent of robust control and robust optimization. The distinction is that we optimistically and iteratively maximize the RHS of (2.1.1) jointly over the model and the policy. The iterative approach allows the algorithms to collect higher quality trajectory adaptively, and the optimism in model optimization encourages explorations of the parts of space that are not covered by the current discrepancy bounds.

To instantiate the meta-algorithm, we design a few valid discrepancy bounds in Section 2.5. In Section 2.5.1, we recover the norm-based model loss by imposing the additional assumption of a Lipschitz value function. The result suggests a norm is preferred compared to the square of the norm. Indeed in Section 2.6.2, we show that experimentally learning with $\ell_2$ loss significantly outperforms the mean-squared error loss ($\ell_2^2$).

In Section 2.5.2, we design a discrepancy bound that is *invariant* to the representation of the state space. Here we measure the loss of the model by the difference between the value of the predicted next state and the value of the true next state. Such a loss function is shown to be invariant to one-to-one transformation of the state space. Thus we argue that the loss is an intrinsic measure for the model error

without any information beyond observing the rewards. We also refine our bounds in Section 2.5.3 by utilizing some mathematical tools of measuring the difference between policies in $\chi^2$-divergence (instead of KL divergence or TV distance).

Our analysis also sheds light on the comparison between model-based RL and on-policy model-free RL algorithms such as policy gradient or TRPO [Schulman et al., 2015a]. The RHS of Equation (2.1.1) is likely to be a good approximator of $V^\pi$ in a larger neighborhood than the linear approximation of $V^\pi$ used in policy gradient is (see Remark 2.5.5.)

Finally, inspired by our framework and analysis, we design a variant of model-based RL algorithms Stochastic Lower Bounds Optimization (SLBO). Experiments demonstrate that SLBO achieves high performance when only one million samples are permitted on a range of continuous control benchmark tasks.

## 2.2 Related Work

Model-based reinforcement learning is expected to require fewer samples than model-free algorithms [Deisenroth et al., 2013] and has been successfully applied to robotics in both simulation and in the real world [Deisenroth and Rasmussen, 2011, Morimoto and Atkeson, 2003, Deisenroth et al., 2011] using dynamics models ranging from Gaussian process [Deisenroth and Rasmussen, 2011, Ko and Fox, 2009], time-varying linear models [Levine and Koltun, 2013, Lioutikov et al., 2014, Levine and Abbeel, 2014, Yip and Camarillo, 2014], mixture of Gaussians [Khansari-Zadeh and Billard, 2011], to neural networks [Hunt et al., 1992, Nagabandi et al., 2018, Kurutach et al., 2018, Tangkaratt et al., 2014, Sanchez-Gonzalez et al., 2018, Pascanu et al., 2017]. In particular, the work of Kurutach et al. [2018] uses an ensemble of neural networks to learn the dynamics model, and significantly reduces the sample complexity compared to model-free approaches. The work of Chua et al. [2018b] makes further improvement

by using a probabilistic model ensemble. Clavera et al. [2018] extended this method with meta-policy optimization and improve the robustness to model error. In contrast, we focus on theoretical understanding of model-based RL and the design of new algorithms, and our experiments use a *single* neural network to estimate the dynamics model.

Our discrepancy bound in Section 2.5 is closely related to the work of Farahmand et al. [2017] on the value-aware model loss. Our approach differs from it in three details: a) we use the absolute value of the value difference instead of the squared difference; b) we use the imaginary value function from the estimated dynamics model to define the loss, which makes the loss purely a function of the estimated model and the policy; c) we show that the iterative algorithm, using the loss function as a building block, can converge to a local maximum, partly by cause of the particular choices made in a) and b). Asadi et al. [2018] also study the discrepancy bounds under Lipschitz condition of the MDP.

Prior work explores a variety of ways of combining model-free and model-based ideas to achieve the best of the two methods [Sutton, 1991, 1990b, Racanière et al., 2017, Mordatch et al., 2016, Sun et al., 2018b]. For example, estimated models [Levine and Koltun, 2013, Gu et al., 2016b, Kalweit and Boedecker, 2017] are used to enrich the replay buffer in the model-free off-policy RL. Pong et al. [2018b] proposes goal-conditioned value functions trained by model-free algorithms and uses it for model-based controls. Feinberg et al. [2018b], Buckman et al. [2018] use dynamics models to improve the estimation of the value functions in the model-free algorithms.

On the control theory side, Dean et al. [2018, 2020] provide strong finite sample complexity bounds for solving linear quadratic regulator using model-based approach. Boczar et al. [2018] provide finite-data guarantees for the "coarse-ID control" pipeline, which is composed of a system identification step followed by a robust controller synthesis procedure. Our method is inspired by the general idea of maximizing a low

bound of the reward in Dean et al. [2020]. By contrast, our work applies to non-linear dynamical systems. Our algorithms also estimate the models iteratively based on trajectory samples from the learned policies.

Strong model-based and model-free sample complexity bounds have been achieved in the tabular case (finite state space). We refer the readers to Kakade et al. [2018], Dann et al. [2017], Szita and Szepesvári [2010], Kearns and Singh [2002], Jaksch et al. [2010], Agrawal and Jia [2017] and the reference therein. Our work focus on continuous and high-dimensional state space (though the results also apply to tabular case).

Another line of work of model-based reinforcement learning is to learn a dynamic model in a hidden representation space, which is especially necessary for pixel state spaces [Kakade et al., 2018, Dann et al., 2017, Szita and Szepesvári, 2010, Kearns and Singh, 2002, Jaksch et al., 2010]. Srinivas et al. [2018] shows the possibility to learn an abstract transition model to imitate expert policy. Oh et al. [2017] learns the hidden state of a dynamics model to predict the value of the future states and applies RL or planning on top of it. Serban et al. [2018], Ha and Schmidhuber [2018] learns a bottleneck representation of the states. Our framework can be potentially combined with this line of research.

## 2.3 Preliminaries

We use the same setup in Section 1.3. The target applications are problems with the continuous state and action space, although the results apply to discrete state or action space as well. Let $\mathcal{M}$ denote a (parameterized) family of models that we are interested in, and $\Pi$ denote a (parameterized) family of policies.

Unless otherwise stated, for random variable $X$, we will use $p_X$ to denote its density function.

Let $S_0$ be the random variable for the initial state. Let $S_t^{\pi,M}$ to denote the random variable of the states at steps $t$ when we execute the policy $\pi$ on the dynamic model $M$ stating with $S_0$. Note that $S_0^{\pi,M} = S_0$ unless otherwise stated. We will omit the subscript when it's clear from the context. We use $A_t$ to denote the actions at step $t$ similarly. We often use $\tau$ to denote the random variable for the trajectory $(S_0, A_1, \ldots, S_t, A_t, \ldots)$. We assume the reward function $R$ is *known* throughout the chapter, although $R$ can be also considered as part of the model if unknown.

Recall that $V^{\pi,M}$ be the value function on the model $M$ and policy $\pi$ is defined as:

$$V^{\pi,M}(s) \triangleq \mathop{\mathbb{E}}_{\substack{\forall t \geq 0, A_t \sim \pi(\cdot|S_t) \\ S_{t+1} \sim M(\cdot|S_t, A_t)}} \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) \mid S_0 = s \right] \tag{2.3.1}$$

We define $V^{\pi,M} \triangleq \mathbb{E}\left[V^{\pi,M}(S_0)\right]$ as the expected reward-to-go at Step 0 (averaged over the random initial states). Our goal is to maximize the reward-to-go on the true dynamics model, that is, $V^{\pi,M^\star}$, over the policy $\pi$. For simplicity, throughout the chapter, we set $\kappa = \gamma(1-\gamma)^{-1}$ since it occurs frequently in our equations. Every policy $\pi$ induces a distribution of states visited by policy $\pi$:

**Definition 2.3.1.** For a policy $\pi$, define $\rho^{\pi,M}$ as the discounted distribution of the states visited by $\pi$ on $M$. Let $\rho^\pi$ be a shorthand for $\rho^{\pi,M^\star}$ and we omit the superscript $M^\star$ throughout the chapter. Concretely, we have $\rho^\pi \triangleq (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \cdot p_{S_t^\pi}$

## 2.4 Algorithmic Framework

As mentioned in the introduction, towards optimizing $V^{\pi,M^\star}$,[2] our plan is to build a lower bound for $V^{\pi,M^\star}$ of the following type and optimize it iteratively:

$$V^{\pi,M^\star} \geq V^{\pi,\widehat{M}} - D(\widehat{M}, \pi) \tag{2.4.1}$$

[2]Note that in the introduction we used $V^\pi$ for simplicity, and in the rest of the chapter we will make the dependency on $M^\star$ explicit.

where $D(\widehat{M}, \pi) \in \mathbb{R}_{\geq 0}$ bounds from above the discrepancy between $V^{\pi,\widehat{M}}$ and $V^{\pi,M^\star}$. Building such an optimizable discrepancy bound globally that holds for all $\widehat{M}$ and $\pi$ turns out to be rather difficult, if not impossible. Instead, we shoot for establishing such a bound over the neighborhood of a reference policy $\pi_{\text{ref}}$.

$$V^{\pi,M^\star} \geq V^{\pi,\widehat{M}} - D_{\pi_{\text{ref}},\delta}(\widehat{M}, \pi), \qquad \forall \pi \text{ s.t. } d(\pi, \pi_{\text{ref}}) \leq \delta \tag{R1}$$

Here $d(\cdot, \cdot)$ is a function that measures the closeness of two policies, which will be chosen later in alignment with the choice of $D$. We will mostly omit the subscript $\delta$ in $D$ for simplicity in the rest of the chapter. We will require our discrepancy bound to vanish when $\widehat{M}$ is an accurate model:

$$\widehat{M} = M^\star \Longrightarrow D_{\pi_{\text{ref}}}(\widehat{M}, \pi) = 0, \quad \forall \pi, \pi_{\text{ref}} \tag{R2}$$

The third requirement for the discrepancy bound $D$ is that it can be estimated and optimized in the sense that

$$D_{\pi_{\text{ref}}}(\widehat{M}, \pi) \text{ is of the form } \mathop{\mathbb{E}}_{\tau \sim \pi_{\text{ref}}, M^\star}[f(\widehat{M}, \pi, \tau)] \tag{R3}$$

where $f$ is a known differentiable function. We can estimate such discrepancy bounds for *every* $\pi$ in the neighborhood of $\pi_{\text{ref}}$ by sampling empirical trajectories $\tau^{(1)}, \ldots, \tau^{(n)}$ from executing policy $\pi_{\text{ref}}$ on the real environment $M^\star$ and compute the average of $f(\widehat{M}, \pi, \tau^{(i)})$'s. We would have to insist that the expectation cannot be over the randomness of trajectories from $\pi$ on $M^\star$, because then we would have to re-sample trajectories for every possible $\pi$ encountered.

For example, assuming the dynamics models are all deterministic, one of the valid discrepancy bounds (under some strong assumptions) that will prove in Section 2.5 is

a multiple of the error of the prediction of $\widehat{M}$ on the trajectories from $\pi_{\text{ref}}$:

$$D_{\pi_{\text{ref}}}(\widehat{M}, \pi) = L \cdot \mathop{\mathbb{E}}_{S_0,\ldots,S_t,\sim\pi_{\text{ref}},M^\star} \left[ \|\widehat{M}(S_t) - S_{t+1}\| \right] \tag{2.4.2}$$

Suppose we can establish such an discrepancy bound $D$ (and the distance function $d$) with properties (R1), (R2), and (R3), — which will be the main focus of Section 2.5 —, then we can devise the following meta-algorithm (Algorithm 1). We iteratively optimize the lower bound over the policy $\pi_{k+1}$ and the model $M_{k+1}$, subject to the constraint that the policy is not very far from the reference policy $\pi_k$ obtained in the previous iteration. For simplicity, we only state the population version with the exact computation of $D_{\pi_{\text{ref}}}(\widehat{M}, \pi)$, though empirically it is estimated by sampling trajectories.

**Algorithm 1** Meta-Algorithm for Model-based RL

**Inputs:** Initial policy $\pi_0$. Discrepancy bound $D$ and distance function $d$ that satisfy (R1) and (R2).

**For** $k = 0$ to $T$:

$$\pi_{k+1}, M_{k+1} = \mathop{\text{argmax}}_{\pi\in\Pi,\ M\in\mathcal{M}} \quad V^{\pi,M} - D_{\pi_k,\delta}(M, \pi) \tag{2.4.3}$$

$$\text{s.t.} \quad d(\pi, \pi_k) \le \delta \tag{2.4.4}$$

We first remark that the discrepancy bound $D_{\pi_k}(M, \pi)$ in the objective plays the role of learning the dynamics model by ensuring the model to fit to the sampled trajectories. For example, using the discrepancy bound in the form of Equation (2.4.2), we roughly recover the standard objective for model learning, with the caveat that we only have the norm instead of the square of the norm in MSE. Such distinction turns out to be empirically important for better performance (see Section 2.6.2).

Second, our algorithm can be viewed as an extension of the optimism-in-face-of-uncertainty (OFU) principle to non-linear parameterized setting: jointly optimizing $M$ and $\pi$ encourages the algorithm to choose the most optimistic model among those

that can be used to accurately estimate the value function. (See [Jaksch et al., 2010, Bartlett and Tewari, 2009, Fruit et al., 2018, Lakshmanan et al., 2015, Pirotta et al., 2015, 2013] and references therein for the OFU principle in finite-state MDPs.) The main novelty here is to optimize the lower bound directly, without explicitly building any confidence intervals, which turns out to be challenging in deep learning. In other words, the uncertainty is measured straightforwardly by how the error would affect the estimation of the value function.

Thirdly, the maximization of $V^{\pi,M}$, when $M$ is fixed, can be solved by any model-free RL algorithms with $M$ as the environment without querying any real samples. Optimizing $V^{\pi,M}$ jointly over $\pi, M$ can be also viewed as another RL problem with an extended actions space using the known "extended MDP technique". See Jaksch et al. [2010, Section 3.1] for details.

Our main theorem shows formally that the policy performance in the real environment is non-decreasing under the assumption that the real dynamics belongs to our parameterized family $\mathcal{M}$.[3]

**Theorem 2.4.1.** *Suppose that $M^\star \in \mathcal{M}$, that $D$ and $d$ satisfy Equations* (R1) *and* (R2)*, and the optimization problem in Equation* (2.4.3) *is solvable at each iteration. Then, Algorithm 1 produces a sequence of policies $\pi_0, \ldots, \pi_T$ with monotonically increasing values:*

$$V^{\pi_0,M^\star} \leq V^{\pi_1,M^\star} \leq \cdots \leq V^{\pi_T,M^\star} \tag{2.4.5}$$

*Moreover, as $k \to \infty$, the value $V^{\pi_k,M^\star}$ converges to some $V^{\bar{\pi},M^\star}$, where $\bar{\pi}$ is a local maximum of $V^{\pi,M^\star}$ in domain $\Pi$.*

[3]We note that such an assumption, though restricted, may not be very far from reality: optimistically speaking, we only need to approximate the dynamics model accurately on the trajectories of the optimal policy. This might be much easier than approximating the dynamics model globally.

The theorem above can also be extended to a finite sample complexity result with standard concentration inequalities. We show in Theorem 2.4.3 that we can obtain an approximate local maximum in $O(1/\varepsilon)$ iterations with sample complexity (in the number of trajectories) that is polynomial in dimension and accuracy $\varepsilon$ and is logarithmic in certain smoothness parameters.

*Proof of Theorem 2.4.1.* Since $D$ and $d$ satisfy (R1), we have that

$$V^{\pi_{k+1},M^\star} \geq V^{\pi_{k+1},M_{k+1}} - D_{\pi_k}(M_{k+1},\pi_{k+1})$$

By the definition that $\pi_{k+1}$ and $M_{k+1}$ are the optimizers of Equation (2.4.3), we have that

$$V^{\pi_{k+1},M_{k+1}} - D_{\pi_k}(M_{k+1},\pi_{k+1}) \geq V^{\pi_k,M^\star} - D_{\pi_k}(M^\star,\pi_k) = V^{\pi_k,M^\star} \qquad \text{(by (R2))}$$

Combing the two equations above we complete the proof of Equation (2.4.5).

For the second part of the theorem, by compactness, we have that a subsequence of $\pi_k$ converges to some $\bar{\pi}$. By the monotonicity we have $V^{\pi_k,M^\star} \leq V^{\bar{\pi},M^\star}$ for every $k \geq 0$. For the sake of contradiction, we assume $\bar{\pi}$ is a not a local maximum, then in the neighborhood of $\bar{\pi}$ there exists $\pi'$ such that $V^{\pi',M^\star} > V^{\bar{\pi},M^\star}$ and $d(\bar{\pi},\pi') < \delta/2$. Let $t$ be such that $\pi_t$ is in the $\delta/2$-neighborhood of $\bar{\pi}$. Then we see that $(\pi', M^\star)$ is a better solution than $(\pi_{t+1}, M_{t+1})$ for the optimization problem (2.4.3) in iteration $t$ because $V^{\pi',M^\star} > V^{\bar{\pi},M^\star} \geq V^{\pi_{t+1},M^\star} \geq V^{\pi_{t+1},M_{t+1}} - D_{\pi_t}(M_{t+1},\pi_{t+1})$. (Here the last inequality uses (R1) with $\pi_t$ as $\pi_{\text{ref}}$.) The fact $(\pi', M^\star)$ is a strictly better solution than $(\pi_{t+1}, M_{t+1})$ contradicts the fact that $(\pi_{t+1}, M_{t+1})$ is defined to be the optimal solution of (2.4.3) . Therefore $\bar{\pi}$ is a local maximum and we complete the proof.

□

Particularly, the condition (R2) and (R3) are at odds with each other—(R3) essentially says that the discrepancy bound $D$ can only carry information about $M^\star$ through the sampled data, and the non-trivial uncertainty about $M^\star$ from observing only sampled data implies that the discrepancy bound $D_{\pi_{\text{ref}}}(\widehat{M}, \pi)$ can never be zero (regardless of $M^\star = \widehat{M}$ or not). We can formally see this from the linear bandit setting (which corresponds to the horizon $H = 1$ case.)

In the linear bandit setting, with a bit abuse of notation, we still use $M \in \mathbb{R}^d$ for the model parameters and use $\pi$ for the action in $\mathbb{R}^d$. We assume that the reward is linear: $V^{\pi,M} = \langle \pi, M \rangle$. In the sequel, we will show that (R3) and (R1) imply the "$\Leftarrow$" direction in (R2), and therefore we have,

$$\widehat{M} = M^* \iff \forall \pi, \pi_{\text{ref}}, D_{\pi_{\text{ref}}}(\widehat{M}, \pi) = 0, \tag{2.4.6}$$

Note that (2.4.6) is statistically impossible because it allows us to find the true model $M^*$ directly through checking if $D$ is zero for all policies $\pi, \pi_{\text{ref}}$, which is not possible given finite data (at least not before we have sufficient data).

Now we prove the reverse direction of (R2). For the sake of contradiction, we assume that there exists $\widehat{M} \neq M^*$ such that for all policies $\pi, \pi_{\text{ref}}$, we have $d(\pi, \pi_{\text{ref}}) \leq \delta$ and $D_{\pi_{\text{ref}}}(\widehat{M}, \pi) = 0$. There exists a policy $\pi$ such that $V^{\pi,\widehat{M}} = \langle \pi, \widehat{M} \rangle > \langle \pi, M^\star \rangle = V^{\pi,M^*}$ because we can take a policy $\pi$ that correlates with $\widehat{M}$ more than $M^\star$ (and this is the only place that we use linearity.) By (R1), we have $0 \geq D_{\pi_{\text{ref}}}(\widehat{M}, \pi) \geq V^{\pi,\widehat{M}} - V^{\pi,M^*}$, which is a contradiction.

### 2.4.1 Sample Complexity Bounds

In this subsection, we extend Theorem 2.4.1 to a final sample complexity result. For simplicity, let $L^{\pi,M}_{\pi_{\text{ref}},\delta} = V^{\pi,M} - D_{\pi_k,\delta}(M, \pi)$ be the lower bound of $V^{\pi,M^\star}$. We omit the

subscript $\delta$ when it's clear from contexts. When $D$ satisfies (R1), we have that,

$$V^{\pi,M^\star} \geq L^{\pi,M}_{\pi_{\mathrm{ref}},\delta} \qquad \forall \pi \text{ s.t. } d(\pi,\pi_{\mathrm{ref}}) \leq \delta \tag{2.4.7}$$

When $D$ satisfies (R3), we use $\widehat{L}^{\pi,M}_{\pi_{\mathrm{ref}},\delta}$ to denotes its empirical estimates. Namely, we replace the expectation in (R3) by empirical samples $\tau^{(1)},\ldots,\tau^{(n)}$. In other words, we optimize

$$\pi_{k+1}, M_{k+1} = \operatorname*{argmax}_{\pi\in\Pi,\ M\in\mathcal{M}} \widehat{L}^{\pi,M}_{\pi_{\mathrm{ref}},\delta} = \quad V^{\pi,M} - \frac{1}{n}\sum_{i=1}^{n} f(\widehat{M},\pi,\tau^{(i)}) \tag{2.4.8}$$

instead of Equation (2.4.3).

Let $p$ be the total number of parameters in the policy and model parameterization. We assume that we have a discrepancy bound $D_{\pi_{\mathrm{ref}}}(\pi, M)$ satisfying (R3) with a function $f$ that is bounded with $[-B_f, B_f]$ and that is $L_f$-Lipschitz in the parameters of $\pi$ and $M$. That is, suppose $\pi$ is parameterized by $\theta$ and $M$ is parameterized by $\phi$, then we require $|f(M_\phi,\phi_\theta,\tau) - f(M_{\phi'},\phi_{\theta'},\tau)| \leq L_f(\|\phi-\phi'\|_2^2 + \|\theta-\theta'\|^2)$ for all $\tau$, $\theta,\theta',\phi,\phi'$. We note that $L_f$ is likely to be exponential in dimension due to the recursive nature of the problem, but our bounds only depends on its logarithm. We also restrict our attention to parameters in an Euclidean ball $\{\theta : \|\theta\|_2 \leq B\}$ and $\{\phi : \|\phi\|_2 \leq B\}$. Our bounds will be logarithmic in $B$.

We need the following definition of approximate local maximum since with sampling error we cannot hope to converge to the exact local maximum.

**Definition 2.4.2.** We say $\pi$ is a $(\delta,\varepsilon)$-local maximum of $V^{\pi,M^\star}$ with respect to the constraint set $\Pi$ and metric $d$, if for any $\pi' \in \Pi$ with $d(\pi,\pi') \leq \delta$, we have $V^{\pi,M^\star} \geq V^{\pi',M^\star} - \varepsilon$.

We show a sample complexity bounds that scales linearly in $p$ and logarithmically in $L_f, B$ and $B_f$.

**Theorem 2.4.3.** *Let $\varepsilon > 0$. In the setting of Theorem 2.4.1, under the additional assumptions above, suppose we use $n = O(B_f p \log(BL_f/\varepsilon)/\varepsilon^2)$ trajectories to estimate the discrepancy bound in Algorithm 1. Then, for any $t$, if $\pi_t$ is not a $(\delta, \varepsilon)$-local maximum, then the total reward will increase in the next step: with high probability,*

$$V^{\pi_{t+1}, M^\star} \geq V^{\pi_t, M^\star} + \varepsilon/2 \tag{2.4.9}$$

*As a direct consequence, suppose the maximum possible total reward is $B_R$ and the initial total reward is 0, then for some $T = O(B_R/\varepsilon)$, we have that $\pi_T$ is a $(\delta, \varepsilon)$-local maximum of the $V^{\pi, M^\star}$.*

*Proof of Theorem 2.4.3.* By Hoeffiding's inequality, we have for fix $\pi$ and $\widehat{M}$, with probability $1 - n^{O(1)}$ over the randomness of $\tau^{(1)}, \ldots, \tau^{(n)}$,

$$\left| \frac{1}{n} \sum_{i=1}^{n} f(\widehat{M}, \pi, \tau^{(i)}) - \mathop{\mathbb{E}}_{\tau \sim \pi_{\mathrm{ref}}, M^\star} [f(\widehat{M}, \pi, \tau)] \right| \leq 4\sqrt{\frac{B_f \log n}{n}}. \tag{2.4.10}$$

In more succinct notations, we have $|\widehat{D}_{\pi_k, \delta}(M, \pi) - D_{\pi_k, \delta}(M, \pi)| \leq 4\sqrt{\frac{B_f \log n}{n}}$, and therefore

$$|\widehat{L}^{\pi, M} - L^{\pi, M}| \leq 4\sqrt{\frac{B_f \log n}{n}}. \tag{2.4.11}$$

By a standard $\varepsilon$-cover + union bound argument, we can prove the uniform convergence: with high probability (at least $1 - n^{O(1)}$) over the choice of $\tau^{(1)}, \ldots, \tau^{(n)}$, for all policy and model, for all policy $\pi$ and dynamics $M$,

$$|\widehat{L}^{\pi, M} - L^{\pi, M}| \leq 4\sqrt{\frac{B_f p \log(nBL_f)}{n}} = \varepsilon/4. \tag{2.4.12}$$

Suppose at iteration $t$, we are at policy $\pi_t$ which is not a $(\delta, \varepsilon)$-local maximum of $V^{\pi, M^\star}$. Then, there exists $\pi'$ such that $d(\pi', \pi_t) \leq \delta$ and

$$V^{\pi', M^\star} \geq V^{\pi_t, M^\star} + \varepsilon. \tag{2.4.13}$$

Then, we have that

$$\begin{aligned}
V^{\pi_{t+1}, M^\star} &\geq L_{\pi_t}^{\pi_{t+1}, M_{t+1}} && \text{(by Equation (2.4.7))} \\
&\geq \widehat{L}_{\pi_t}^{\pi_{t+1}, M_{t+1}} - \varepsilon/4 && \text{(by uniform convergence, Equation (2.4.12))} \\
&\geq \widehat{L}_{\pi_t}^{\pi', M^\star} - \varepsilon/4 && \text{(by the definition of } \pi_{t+1}, M_{t+1}) \\
&\geq L_{\pi_t}^{\pi', M^\star} - \varepsilon/2 && \text{(by uniform convergence, Equation (2.4.12))} \\
&= V^{\pi', M^\star} - \varepsilon/2 && \text{(by (R2))} \\
&= V^{\pi_t, M^\star} + \varepsilon/2 && \text{(by Equation (2.4.13))}
\end{aligned}$$

Note that the total reward can only improve by $\varepsilon/2$ for at most $O(B_R/\varepsilon)$ steps. Therefore, in the first $O(B_R/\varepsilon)$ iterations, we must have hit a solution that is a $(\delta, \varepsilon)$-local maximum. This completes the proof.

□

## 2.5 Discrepancy Bounds Design

In this section, we design discrepancy bounds that can provably satisfy the requirements (R1), (R2), and (R3). We design increasingly stronger discrepancy bounds from Section 2.5.1 to Section 2.5.3.

### 2.5.1 Norm-based Prediction Error Bounds

In this subsection, we assume the dynamics model $M^\star$ is deterministic and we also learn with a deterministic model $\widehat{M}$. Under assumptions defined below, we derive a discrepancy bound $D$ of the form $\|\widehat{M}(S,A) - M^\star(S,A)\|$ averaged over the observed state-action pair $(S,A)$ on the dynamics model $\widehat{M}$. This suggests that the norm is a better metric than the mean-squared error for learning the model, which is empirically shown in Section 2.6.2. Through the derivation, we will also introduce a telescoping lemma, which serves as the main building block towards other finer discrepancy bounds.

We make the (strong) assumption that the value function $V^{\pi,\widehat{M}}$ on the estimated dynamics model is $L$-Lipschitz w.r.t to some norm $\|\cdot\|$ in the sense that

$$\forall s, s' \in \mathcal{S}, \left|V^{\pi,\widehat{M}}(s) - V^{\pi,\widehat{M}}(s')\right| \le L \cdot \|s - s'\| \tag{2.5.1}$$

In other words, nearby starting points should give reward-to-go under the same policy $\pi$. We note that not every real environment $M^\star$ has this property, let alone the estimated dynamics models. However, once the real dynamics model induces a Lipschitz value function, we may penalize the Lipschitz-ness of the value function of the estimated model during the training.

We start off with a lemma showing that the expected prediction error is an upper bound of the discrepancy between the real and imaginary values.

**Lemma 2.5.1.** *Suppose $V^{\pi,\widehat{M}}$ is $L$-Lipschitz (in the sense of Equation* (2.5.1)*). Recall* $\kappa = \gamma(1-\gamma)^{-1}$.

$$\left|V^{\pi,\widehat{M}} - V^{\pi,M^\star}\right| \le \kappa L \mathop{\mathbb{E}}_{\substack{S\sim\rho^\pi \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] \tag{2.5.2}$$

The proof of Lemma 2.5.1 is deferred to the end of the subsection.

However, in RHS in Equation (2.5.2) cannot serve as a discrepancy bound because it does not satisfy the requirement (R3) — to optimize it over $\pi$ we need to collect samples from $\rho^\pi$ for every iterate $\pi$ — the state distribution of the policy $\pi$ on the *real* model $M^\star$. The main proposition of this subsection stated next shows that for every $\pi$ in the neighborhood of a reference policy $\pi_{\text{ref}}$, we can replace the distribution $\rho^\pi$ be a fixed distribution $\rho^{\pi_{\text{ref}}}$ with incurring only a higher order approximation. We use the expected KL divergence between two $\pi$ and $\pi_{\text{ref}}$ to define the neighborhood:

$$d^{\text{KL}}(\pi, \pi_{\text{ref}}) \triangleq \mathop{\mathbb{E}}_{S\sim\rho^\pi} \left[D_{\text{KL}}(\pi(\cdot|S) \parallel \pi_{\text{ref}}(\cdot|S))^{1/2}\right] \tag{2.5.3}$$

**Proposition 2.5.2.** *In the same setting of Lemma 2.5.1, assume in addition that $\pi$ is close to a reference policy $\pi_{\text{ref}}$ in the sense that $d^{\text{KL}}(\pi, \pi_{\text{ref}}) \le \delta$, and that the states in $\mathcal{S}$ are uniformly bounded in the sense that $\|s\| \le B, \forall s \in \mathcal{S}$. Then,*

$$\left|V^{\pi,\widehat{M}} - V^{\pi,M^\star}\right| \le \kappa L \mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi_{\text{ref}}} \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] + 2\kappa^2\delta B \tag{2.5.4}$$

In a benign scenario, the second term in the RHS of Equation (2.5.4) should be dominated by the first term when the neighborhood size $\delta$ is sufficiently small. Moreover, the term $B$ can also be replaced by $\max_{S,A}\|\widehat{M}(S,A) - M^\star(S,A)\|$ (see the proof at the end of this subsection.). The dependency on $\kappa$ may not be tight for real-life instances, but we note that most analysis of similar nature loses the additional $\kappa$ factor [Schulman et al., 2015a, Achiam et al., 2017], and it's inevitable in the worst-case.

**A telescoping lemma** Towards proving Proposition 2.5.2 and deriving stronger discrepancy bound, we define the following quantity that captures the discrepancy

between $\widehat{M}$ and $M^\star$ on a single state-action pair $(s, a)$.

$$G^{\pi,\widehat{M}}(s,a) \triangleq \mathop{\mathbb{E}}_{\hat{s}'\sim\widehat{M}(\cdot|s,a)} V^{\pi,\widehat{M}}(\hat{s}') - \mathop{\mathbb{E}}_{s'\sim M^\star(\cdot|s,a)} V^{\pi,\widehat{M}}(s') \tag{2.5.5}$$

Note that if $M, \widehat{M}$ are deterministic, then $G^{\pi,\widehat{M}}(s,a) = V^{\pi,\widehat{M}}(\widehat{M}(s,a)) - V^{\pi,\widehat{M}}(M^\star(s,a))$. We give a telescoping lemma that decompose the discrepancy between $V^{\pi,M}$ and $V^{\pi,M^\star}$ into the expected single-step discrepancy $G$.

**Lemma 2.5.3.** *[Telescoping Lemma] Recall that $\kappa \triangleq \gamma(1-\gamma)^{-1}$. For any policy $\pi$ and dynamics models $M, \widehat{M}$, we have that*

$$V^{\pi,\widehat{M}} - V^{\pi,M} = \kappa \mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi,M}\\A\sim\pi(\cdot|S)}} \left[G^{\pi,\widehat{M}}(S,A)\right] \tag{2.5.6}$$

The proof is reminiscent of the telescoping expansion in Kakade and Langford [2002] (c.f. Schulman et al. [2015a]) for characterizing the value difference of two policies, but we apply it to deal with the discrepancy between models. With the telescoping Lemma 2.5.3, Lemma 2.5.1 follows straightforwardly from Lipschitzness of the imaginary value function. Proposition 2.5.2 follows from that $\rho^\pi$ and $\rho^{\pi_{\text{ref}}}$ are close.

*Proof of Lemma 2.5.3.* Let $W_j$ be the cumulative reward when we use dynamics model $M$ for $j$ steps and then $\widehat{M}$ for the rest of the steps, that is,

$$W_j \triangleq \mathop{\mathbb{E}}_{\substack{\forall t\geq 0, A_t\sim\pi(\cdot|S_t)\\ \forall j>t\geq 0, S_{t+1}\sim M(\cdot|S_t,A_t)\\ \forall t\geq j, S_{t+1}\sim\widehat{M}(\cdot|S_t,A_t)}} \left[\sum_{t=0}^{\infty}\gamma^t R(S_t,A_t) \mid S_0 = s\right]$$

By definition, we have that $W_\infty = V^{\pi,M}(s)$ and $W_0 = V^{\pi,\widehat{M}}(s)$. Then, we decompose the target into a telescoping sum,

$$V^{\pi,M}(s) - V^{\pi,\widehat{M}}(s) = \sum_{j=0}^{\infty}(W_{j+1} - W_j) \tag{2.5.7}$$

Now we re-write each of the summands $W_{j+1} - W_j$. Comparing the trajectory distribution in the definition of $W_{j+1}$ and $W_j$, we see that they only differ in the dynamics model applied in $j$-th step. Concretely, $W_j$ and $W_{j+1}$ can be rewritten as $W_j = R + \mathbb{E}_{S_j,A_j\sim\pi,M}\left[\mathbb{E}_{\hat{S}_{j+1}\sim\widehat{M}(\cdot|S_j,A_j)}\left[\gamma^{j+1}V^{\pi,\widehat{M}}(\hat{S}_{j+1})\right]\right]$ and $W_{j+1} = R + \mathbb{E}_{S_j,A_j\sim\pi,M^\star}\left[\mathbb{E}_{S_{j+1}\sim M(\cdot|S_j,A_j)}\left[\gamma^{j+1}V^{\pi,\widehat{M}}(S_{j+1})\right]\right]$ where $R$ denotes the reward from the first $j$ steps from policy $\pi$ and model $M^\star$. Canceling the shared term in the two equations above, we get

$$W_{j+1} - W_j = \gamma^{j+1} \mathop{\mathbb{E}}_{S_j,A_j\sim\pi,M}\left[\mathop{\mathbb{E}}_{\substack{\hat{S}_{j+1}\sim\widehat{M}(\cdot|S_j,A_j)\\ S_{j+1}\sim M(\cdot|S_j,A_j)}}\left[V^{\pi,\widehat{M}}(S_{j+1}) - V^{\pi,\widehat{M}}(\widehat{S}_{j+1})\right]\right]$$

Combining the equation above with Equation (2.5.7) concludes that

$$V^{\pi,M} - V^{\pi,\widehat{M}} = \frac{\gamma}{1-\gamma}\mathop{\mathbb{E}}_{S\sim\rho^\pi,A\sim\pi(S)}\left[\mathop{\mathbb{E}}_{S'\sim M^\star(\cdot|S,A)} V^{\pi,\widehat{M}}(S') - \mathop{\mathbb{E}}_{\hat{S}'\sim\widehat{M}(\cdot|S,A)} V^{\pi,\widehat{M}}(\hat{S}')\right]$$

□

*Proof of Lemma 2.5.1 and proposition 2.5.2 .* By definition of $G$ and the Lipschitzness of $V^{\pi,\widehat{M}}$, we have that $|G^{\pi,\widehat{M}}(s,a)| \le L|\widehat{M}(s,a) - M^\star(s,a)|$. Then, by Lemma 2.5.3 and triangle inequality, we have that

$$\left|V^{\pi,\widehat{M}} - V^{\pi,M^\star}\right| = \kappa \cdot \Big|\mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi,M}\\ A\sim\pi(\cdot|S)}}\left[G^{\pi,\widehat{M}}(S,A)\right]\Big| \le \kappa \mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi,M}\\ A\sim\pi(\cdot|S)}}\left[\left|G^{\pi,\widehat{M}}(S,A)\right|\right]$$

$$\leq \kappa \mathop{\mathbb{E}}_{\substack{S\sim\rho^\pi \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right]. \tag{2.5.8}$$

Next we prove the main part of the proposition. Thus we proved Lemma 2.5.1. Note that for any distribution $\rho$ and $\rho'$ and function $f$, we have $\mathbb{E}_{S\sim\rho} f(S) = \mathbb{E}_{S\sim\rho'} f(S) + \langle \rho - \rho', f\rangle \leq \mathbb{E}_{S\sim\rho'} f(S) + \|\rho - \rho'\|_1 \|f\|_\infty$. Thus applying this inequality with $f(S) = \mathbb{E}_{A\sim\pi(\cdot|S)}\left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right]$, we obtain that

$$\begin{aligned}
\mathop{\mathbb{E}}_{\substack{S\sim\rho^\pi \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] &\leq \mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi_{\text{ref}}} \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] \\
&\quad + \|\rho^{\pi_{\text{ref}}} - \rho\|_1 \max_S \mathop{\mathbb{E}}_{A\sim\pi(\cdot|S)} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] \\
&\leq \mathop{\mathbb{E}}_{\substack{S\sim\rho^{\pi_{\text{ref}}} \\ A\sim\pi(\cdot|S)}} \left[\|\widehat{M}(S,A) - M^\star(S,A)\|\right] + 2\delta\kappa B
\end{aligned} \tag{2.5.9}$$

where the last inequality uses the inequalities (see Corollary 2.5.13) that $\|\rho^\pi - \rho^{\pi_{\text{ref}}}\|_1 \leq \frac{\gamma}{1-\gamma}\mathbb{E}_{S\sim\rho^{\pi_{\text{ref}}}}\left[D_{\text{KL}}(\pi(S) \parallel \pi_{\text{ref}}(S))^{1/2}|S\right] = \delta\kappa$ and that $\|\widehat{M}(S,A) - M^\star(S,A)\| \leq 2B$. Combining Equations (2.5.8) and (2.5.9) we complete the proof of Proposition 2.5.2. □

### 2.5.2 Representation-invariant Discrepancy Bounds

The main limitation of the norm-based discrepancy bounds in previous subsection is that it depends on the state representation. Let $\mathcal{T}$ be a one-to-one map from the state space $\mathcal{S}$ to some other space $\mathcal{S}'$, and for simplicity of this discussion let's assume a model $M$ is deterministic. Then if we represent every state $s$ by its transformed representation $\mathcal{T}s$, then the transformed model $M^{\mathcal{T}}$ defined as $M^{\mathcal{T}}(s,a) \triangleq \mathcal{T}M(\mathcal{T}^{-1}s, a)$ together with the transformed reward $R^{\mathcal{T}}(s,a) \triangleq R(\mathcal{T}^{-1}s, a)$ and transformed policy $\pi^{\mathcal{T}}(s) \triangleq \pi(\mathcal{T}^{-1}s)$ is equivalent to the original set of the model, reward, and policy in terms of the performance (Lemma 2.5.6). Thus such transformation $\mathcal{T}$ is not identifiable

from only observing the reward. However, the norm in the state space is a notion that depends on the hidden choice of the transformation $\mathcal{T}$. [4]

Another limitation is that the loss for the model learning should also depend on the state itself instead of only on the difference $\widehat{M}(S,A) - M^\star(S,A)$. It is possible that when $S$ is at a critical position, the prediction error needs to be highly accurate so that the model $\widehat{M}$ can be useful for planning. On the other hand, at other states, the dynamics model is allowed to make bigger mistakes because they are not essential to the reward.

We propose the following discrepancy bound towards addressing the limitations above. Recall the definition of $G^{\pi,\widehat{M}}(s,a) \triangleq V^{\pi,\widehat{M}}(\widehat{M}(s,a)) - V^{\pi,\widehat{M}}(M^\star(s,a))$ which measures the difference between $\widehat{M}(s,a))$ and $M^\star(s,a)$ according to their imaginary rewards. We construct a discrepancy bound using the absolute value of $G$. Let's define $\varepsilon_1$ and $\varepsilon_{\max}$ as the average of $|G^{\pi,\hat{M}}|$ and its maximum: $\varepsilon_1 \triangleq \mathbb{E}_{S\sim\rho^{\pi_{\rm ref}}}\left[\left|G^{\pi,\widehat{M}}(S,A)\right|\right]$ and $\varepsilon_{\max} \triangleq \max_S \left|G^{\pi,\widehat{M}}(S)\right|$ where $G^{\pi,\widehat{M}}(S) \triangleq \mathbb{E}_{A\sim\pi}\left[G^{\pi,\widehat{M}}(S,A)\right]$. We will show that the following discrepancy bound $D^G_{\pi_{\rm ref}}(\widehat{M},\pi)$ satisfies the property (R1), (R2).

$$D^G_{\pi_{\rm ref}}(\widehat{M},\pi) = \kappa\cdot\varepsilon_1 + \kappa^2\delta\varepsilon_{\max} \tag{2.5.10}$$

**Proposition 2.5.4.** *Let $d^{\rm KL}$ and $D^G$ be defined as in Equations* (2.5.3) *and* (2.5.10)*. Then the choice $d = d^{\rm KL}$ and $D = D^G$ satisfies the basic requirements (Equations* (R1) *and* (R2)*). Moreover, $G$ is invariant w.r.t any one-to-one transformation of the state space (in the sense of Equation* (2.5.13) *in the proof).*

The proof follows from the telescoping lemma (Lemma 2.5.3) and is deferred to the end of the subsection. We remark that the first term $\kappa\varepsilon_1$ can in principle be estimated and optimized approximately: the expectation be replaced by empirical

[4] That said, in many cases the reward function itself is known, and the states have physical meanings, and therefore we may be able to use the domain knowledge to figure out the best norm.

samples from $\rho^{\pi_{\text{ref}}}$, and $G^{\pi,\hat{M}}$ is an analytical function of $\pi$ and $\widehat{M}$ when they are both deterministic, and therefore can be optimized by back-propagation through time (BPTT). (When $\pi$ and $\widehat{M}$ and are stochastic with a re-parameterizable noise such as Gaussian distribution [Kingma and Welling, 2013], we can also use back-propagation to estimate the gradient.) The second term in Equation (2.5.10) is difficult to optimize because it involves the maximum. However, it can be in theory considered as a second-order term because $\delta$ can be chosen to be a fairly small number. (In the refined bound in Section 2.5.3, the dependency on $\delta$ is even milder.)

*Remark* 2.5.5. Proposition 2.5.4 intuitively suggests a technical reason of why model-based approach can be more sample-efficient than policy gradient based algorithms such as TRPO or PPO [Schulman et al., 2015a, 2017]. The approximation error of $V^{\pi,\widehat{M}}$ in model-based approach decreases as the model error $\varepsilon_1, \varepsilon_{\max}$ decrease or the neighborhood size $\delta$ decreases, whereas the approximation error in policy gradient only linearly depends on the the neighborhood size [Schulman et al., 2015a]. In other words, model-based algorithms can trade model accuracy for a larger neighborhood size, and therefore the convergence can be faster (in terms of outer iterations.) This is consistent with our empirical observation that the model can be accurate in a descent neighborhood of the current policy so that the constraint (2.4.4) can be empirically dropped. We also refine our bonds in Section 2.5.3, where the discrepancy bounds is proved to decay faster in $\delta$.

**Proof of Proposition 2.5.4**

Towards proving the second part of Proposition 2.5.4 regarding the invariance, we state the following lemma:

**Lemma 2.5.6.** *Suppose for simplicity the model and the policy are both deterministic. For any one-to-one transformation from $\mathcal{S}$ to $\mathcal{S}'$, let $M^{\mathcal{T}}(s,a) \triangleq \mathcal{T}M(\mathcal{T}^{-1}s,a)$, $R^{\mathcal{T}}(s,a) \triangleq R(\mathcal{T}^{-1}s,a)$, and $\pi^{\mathcal{T}}(s) \triangleq \pi(\mathcal{T}^{-1}s)$ be a set of transformed model, reward*

*and policy. Then we have that* $(M, \pi, R)$ *is equivalent to* $(M^{\mathcal{T}}, \pi^{\mathcal{T}}, R^{\mathcal{T}})$ *in the sense that*

$$V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(\mathcal{T}s) = V^{\pi, M}(s)$$

*where the value function* $V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}$ *is defined with respect to* $R^{\mathcal{T}}$*.*

*Proof of Proposition 2.5.4.* Let $s_0^{\mathcal{T}} = \mathcal{T}s, \ldots,$ be the sequence of states visited by policy $\pi^{\mathcal{T}}$ on model $M^{\mathcal{T}}$ starting from $s$. We have that $s_0^{\mathcal{T}} = \mathcal{T}s = \mathcal{T}s_0$. We prove by induction that $s_t^{\mathcal{T}} = \mathcal{T}s_t$. Assume this is true for some value $t$, then we prove that $s_{t+1}\mathcal{T} = \mathcal{T}s_{t+1}$ holds:

$$\begin{aligned}
s_{t+1}^{\mathcal{T}} &= M^{\mathcal{T}}(s_t^{\mathcal{T}}, \pi^{\mathcal{T}}(s_t^{\mathcal{T}})) = M^{\mathcal{T}}(\mathcal{T}s_t, \pi^{\mathcal{T}}(\mathcal{T}s_t)) && \text{(by inductive hypothesis)} \\
&= \mathcal{T}M(s_t, \pi(s_t)) && \text{(by defintion of } M^{\mathcal{T}}, \pi^{\mathcal{T}}) \\
&= \mathcal{T}s_{t+1}
\end{aligned}$$

Thus we have $R^{\mathcal{T}}(s_t^{\mathcal{T}}, a_t^{\mathcal{T}}) = R(s_t, a_t)$. Therefore $V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(\mathcal{T}s) = V^{\pi, M}(s)$. □

Next, we will introduce a few supporting lemmas regarding to $\chi^2$ distances.

**Definition 2.5.7** ($\chi^2$ distance, c.f. Nielsen and Nock [2014], Cover and Thomas [2012])**.** The Neyman $\chi^2$ distance between two distributions $p$ and $q$ is defined as

$$\chi^2(p, q) \triangleq \int \frac{(p(x) - q(x))^2}{q(x)} dx = \int \frac{p(x)^2}{q(x)} dx - 1$$

For notational simplicity, suppose two random variables $X$ and $Y$ has distributions $p_X$ and $p_Y$, we often write $\chi^2(X, Y)$ as a simplification for $\chi^2(p_X, p_Y)$.

**Theorem 2.5.8** (Sason and Verdú [2016])**.** *The Kullback-Leibler (KL) divergence between two distributions* $p, q$ *is bounded from above by the* $\chi^2$ *distance:*

$$D_{\mathrm{KL}}(p \parallel q) \leq \chi^2(p, q)$$

*Proof of Theorem 2.5.8.* Since log is a concave function, by Jensen inequality we have

$$\begin{aligned} D_{\mathrm{KL}}(p \parallel q) &= \int p(x) \log \frac{p(x)}{q(x)} dx \leq \log \int p(x) \cdot \frac{p(x)}{q(x)} dx \\ &= \log(\chi^2(p, q) + 1) \leq \chi^2(p, q) \end{aligned}$$

□

**Definition 2.5.9** ($\chi^2$ distance between transitions)**.** Given two transition kernels $P, P'$. For any distribution $\mu$, we define $\chi^2_\mu(P', P)$ as:

$$\chi^2_\mu(P', P) \triangleq \int \mu(x) \chi^2(P'(\cdot|X = x), P(\cdot|X = x)) dx$$

**Theorem 2.5.10.** *Suppose random variables $(X, Y)$ and $(X', Y')$ satisfy that $p_{Y|X} = p_{Y'|X'}$. Then*

$$\chi^2(Y, Y') \leq \chi^2(X, X')$$

*Or equivalently, for any transition kernel $P$ and distribution $\mu, \mu'$, we have*

$$\chi^2(P\mu, P\mu') \leq \chi^2(\mu, \mu')$$

*Proof of Theorem 2.5.10.* Denote $p_{Y|X}(y \mid x) = p_{Y'|X'}(y \mid x)$ by $p(y \mid x)$, and we rewrite $p_X$ as $p$ and $p_{X'}$ as $p'$. By Cauchy-Schwarz inequality, we have:

$$\begin{aligned} p_Y(y)^2 = \left( \int p(y|x) p(x) dx \right)^2 &\leq \left( \int p(y|x) p'(x) dx \right) \left( \int p(y|x) \frac{p(x)^2}{p'(x)} dx \right) \\ &= p_{Y'}(y) \left( \int p(y|x) \frac{p(x)^2}{p'(x)} dx \right) \end{aligned} \tag{2.5.11}$$

It follows that

$$\chi^2(Y, Y') = \int \frac{p_Y(y)^2}{p_{Y'}(y)} dy - 1 \leq \int dy \int p(y|x) \frac{p(x)^2}{p'(x)} dx - 1 = \chi^2(X, X')$$

□

**Theorem 2.5.11.** *Let $X, Y, Y'$ are three random variables. Then,*

$$\chi^2(Y, Y') \leq \mathbb{E}\left[\chi^2(Y|X, Y'|X)\right]$$

*We note that the expectation on the right hand side is over the randomness of $X$.*[5] *As a direct corollary, we have for transition kernel $P'$ and $P$ and distribution $\mu$,*

$$\chi^2(P'\mu, P\mu) \leq \chi^2_\mu(P', P)$$

*Proof of Theorem 2.5.11.* We denote $p_{Y'|X}(y|x)$ by $p'(y \mid x)$ and $p_{Y|X}(y|x)$ by $p(y|x)$, and let $p(x)$ be a simplification for $p_X(x)$. We have by Cauchy-Schwarz,

$$\frac{p_Y(y)^2}{p_{Y'}(y)} = \frac{\left(\int p(y|x)p(x)dx\right)^2}{\int p'(y \mid x)p(x)dx} \leq \int \frac{p(y|x)^2}{p'(y|x)} p(x)dx$$

It follows that

$$\chi^2(Y, Y') = \int \frac{p_Y(y)^2}{p_{Y'}(y)} dy - 1 \leq \int \frac{p(y|x)^2}{p'(y|x)} p(x)dxdy - 1 = \mathbb{E}\left[\chi^2(Y|X, Y'|X) \mid X\right]$$

□

**Lemma 2.5.12.** *Let $\mu$ be a distribution over the state space $\mathcal{S}$. Let $P$ and $P'$ be two transition kernels. $G \triangleq \sum_{k=0}^{\infty}(\gamma P)^k = (\mathrm{Id} - \gamma P)^{-1}$ and $G' \triangleq \sum_{k=0}^{\infty}(\gamma P')^k = (\mathrm{Id} - \gamma P')^{-1}$. Let $d \triangleq (1-\gamma)G\mu$ and $d' \triangleq (1-\gamma)G'\mu$ be the discounted distribution*

[5] Observe $\chi^2(Y|X, Y'|X)$ deterministically depends on $X$.

*starting from $\mu$ induced by the transition kernels $G$ and $G'$. Then,*

$$|d - d'|_1 \leq \frac{1}{1-\gamma}|\Delta d|_1$$

*Moreover, let $\gamma(P' - P) = \Delta$. Then, we have*

$$G' - G = \sum_{k=1}^{\infty}(G\Delta)^k G$$

*Proof of Lemma 2.5.12.* With algebraic manipulation, we obtain,

$$\begin{aligned} G' - G &= (\mathrm{Id} - \gamma P')^{-1}((\mathrm{Id} - \gamma P) - (\mathrm{Id} - \gamma P')(\mathrm{Id} - \gamma P)^{-1} \\ &= G'\Delta G \end{aligned} \tag{2.5.12}$$

It follows that

$$\begin{aligned} |d - d'|_1 &= (1-\gamma)|G'\Delta G\mu|_1 \leq |\Delta G\mu|_1 \qquad \text{(since } (1-\gamma)|G'|_{1\to 1} \leq 1\text{)} \\ &= \frac{1}{1-\gamma}|\Delta d|_1 \end{aligned}$$

Replacing $G'$ in the RHS of the Equation (2.5.12) by $G' = G + G'\Delta G$, and doing this recursively gives

$$G' - G = \sum_{k=1}^{\infty}(G\Delta)^k G$$

□

**Corollary 2.5.13.** *Let $\pi$ and $\pi'$ be two policies and let $\rho^\pi$ be defined as in Definition 2.3.1. Then,*

$$|\rho^\pi - \rho'|_1 \leq \frac{\gamma}{1-\gamma} \mathop{\mathbb{E}}_{S\sim\rho^\pi}\left[D_{\mathrm{KL}}(\pi(S) \parallel \pi'(S))^{1/2} \mid S\right]$$

*Proof of Corollary 2.5.13.* Let $P$ and $P'$ be the state-state transition matrix under policy $\pi$ and $\pi'$ and $\Delta \triangleq \gamma(P' - P)$. By Lemma 2.5.12, we have that

$$\begin{aligned}
|\rho^{\pi} - \rho^{\pi'}|_1 \leq \frac{1}{1-\gamma}|\Delta\rho^{\pi}|_1 &= \frac{\gamma}{1-\gamma} \mathop{\mathbb{E}}_{S\sim\rho^{\pi}} \left[|p_{M^\star(S,\pi(S))|S} - p_{M^\star(S,\pi'(S))|S}|_1\right] \\
&\leq \frac{\gamma}{1-\gamma} \mathop{\mathbb{E}}_{S\sim\rho^{\pi}} \left[|p_{\pi(S)|S} - p_{\pi'(S)|S}|_1\right] \\
&\leq \frac{\gamma}{1-\gamma} \mathop{\mathbb{E}}_{S\sim\rho^{\pi}} \left[D_{\mathrm{KL}}(\pi(S) \parallel \pi'(S))^{1/2} \mid S\right] \\
&\qquad \text{(by Pinkser's inequality)}
\end{aligned}$$

□

Now we present the proof of Proposition 2.5.4;

*Proof of Proposition 2.5.4.* We first show the invariant of $G$ under deterministic models and policies. The same result applies to stochastic policies with slight modification. Let $s^{\mathcal{T}} = \mathcal{T}s$. We consider the transformation applied to $M$ and $M^\star$ and the resulting $G$ function

$$G^{\mathcal{T}}(s^{\mathcal{T}}, a) \triangleq |V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(M^{\mathcal{T}}(s^{\mathcal{T}}, a)) - V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(M^{\star,\mathcal{T}}(s^{\mathcal{T}}, a))|$$

Note that by Lemma 2.5.6, we have that $V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(M^{\mathcal{T}}(s^{\mathcal{T}}, a)) = V^{\pi, M}(\mathcal{T}^{-1}M^{\mathcal{T}}(s^{\mathcal{T}}, a)) = V^{\pi, M}(M(s, a))$. Similarly, $V^{\pi^{\mathcal{T}}, M^{\mathcal{T}}}(M^{\star,\mathcal{T}}(s^{\mathcal{T}}, a)) = V^{\pi, M}(M^\star(s, a))$. Therefore we obtain that

$$G^{\mathcal{T}}(s^{\mathcal{T}}, a) = G(s, a) \tag{2.5.13}$$

By Lemma 2.5.3 and triangle inequality, we have that

$$\begin{aligned}
\frac{1-\gamma}{\gamma}\left|V^{\pi,M} - V^{\pi,\widehat{M}}\right| &\leq \mathop{\mathbb{E}}_{S\sim\rho^{\pi}}\left[\left|G^{\pi,\widehat{M}}(S)\right|\right] && \text{(triangle inequality)}
\end{aligned}$$

$$\leq \mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\mathrm{ref}}}}\left[\left|G^{\pi,\widehat{M}}(S)\right|\right] + |\rho^{\pi} - \rho^{\pi_{\mathrm{ref}}}|_1 \cdot \max_S \left|G^{\pi,\widehat{M}}(S)\right|$$

(Holder inequality)

By Corollary 2.5.13 we have that $|\rho^{\pi} - \rho^{\pi_{\mathrm{ref}}}|_1 \leq \frac{\gamma}{1-\gamma}\,\mathbb{E}_{S\sim\rho^{\pi_{\mathrm{ref}}}}\left[D_{\mathrm{KL}}(\pi(S) \parallel \pi_{\mathrm{ref}}(S))^{1/2}|S\right] = \frac{\delta\gamma}{1-\gamma}$. Combining this with the equation above, we complete the proof. □

### 2.5.3 Refined Bounds

The theoretical limitation of the discrepancy bound $D^G(\widehat{M}, \pi)$ is that the second term involving $\varepsilon_{\max}$ is not rigorously optimizable by stochastic samples. In the worst case, there seem to exist situations where such infinity norm of $G^{\pi,\widehat{M}}$ is inevitable. In this section we tighten the discrepancy bounds with a different closeness measure $d$, $\chi^2$-divergence, in the policy space, and the dependency on the $\varepsilon_{\max}$ is smaller (though not entirely removed.) We note that $\chi^2$-divergence has the same second order approximation as KL divergence around the local neighborhood the reference policy and thus locally affects the optimization much.

We start by defining a re-weighted version $\beta^{\pi}$ of the distribution $\rho^{\pi}$ where examples in later step are slightly weighted up. We can effectively sample from $\beta^{\pi}$ by importance sampling from $\rho^{\pi}$

**Definition 2.5.14.** For a policy $\pi$, define $\beta^{\pi}$ as the *re-weighted* version of discounted distribution of the states visited by $\pi$ on $M^{\star}$. Recall that $p_{S_t^{\pi}}$ is the distribution of the state at step $t$, we define $\beta^{\pi} \triangleq (1-\gamma)^2 \sum_{t=1}^{\infty} t\gamma^{t-1} p_{S_t^{\pi}}$.

Then we are ready to state our discrepancy bound. Let

$$d^{\chi^2}(\pi, \pi_{\mathrm{ref}}) \triangleq \max\{\mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\mathrm{ref}}}}\left[\chi^2(\pi(\cdot|S), \pi_{\mathrm{ref}}(\cdot|S))\right], \mathop{\mathbb{E}}_{S\sim\beta^{\pi_{\mathrm{ref}}}}\left[\chi^2(\pi(\cdot|S), \pi_{\mathrm{ref}}(\cdot|S))\right]\} \tag{2.5.14}$$

$$D^{\chi^2}_{\pi_{\text{ref}}}(\widehat{M}, \pi) \triangleq (1-\gamma)^{-1}\varepsilon_1 + (1-\gamma)^{-2}\delta\varepsilon_2 + (1-\gamma)^{-5/2}\delta^{3/2}\varepsilon_{\max} \tag{2.5.15}$$

where $\varepsilon_2 \triangleq \mathbb{E}_{S\sim\beta^{\pi_{\text{ref}}}}\left[G^{\pi,\widehat{M}}(S,A)^2\right]$ and $\varepsilon_1, \varepsilon_{\max}$ are defined in Section 2.5.2.

**Proposition 2.5.15.** *The discrepancy bound $D^{\chi^2}$ and closeness measure $d^{\chi^2}$ satisfies requirements* (R1) *and* (R2).

To prove Proposition 2.5.15, we start by presenting our toolboxes.

### $\chi^2$-Divergence Based Inequalities

**Lemma 2.5.16.** *Let $S$ be a random variable over the domain $\mathcal{S}$. Let $\pi$ and $\pi'$ be two policies and and $A \sim \pi(\cdot \mid S)$ and $A' \sim \pi'(\cdot \mid S)$. Let $Y \sim M(\cdot \mid S, A)$ and $Y' \sim M(\cdot \mid S, A')$ be the random variables for the next states under two policies. Then,*

$$\mathbb{E}\left[\chi^2(Y|S, Y'|S)\right] \leq \mathbb{E}\left[\chi^2(A|S, A'|S)\right]$$

*Proof of Lemma 2.5.16.* By definition, we have that $Y|S = s, A = a$ has the same density as $Y'|S = s, A' = a$ for any $a$ and $s$. Therefore by Theorem 2.5.10 (setting $X, X', Y, Y'$ in Theorem 2.5.10 by $A|S = s$, $A'|S = s$, $Y|S = s$, $Y'|S = s$ respectively), we have

$$\chi^2(Y|S = s, Y'|S = s) \leq \chi^2(A|S = s, A'|S = s)$$

Taking expectation over the randomness of $S$ we complete the proof.

□

### Properties of Markov Processes

We consider bounded the difference of the distributions induced by two markov process starting from the same initial distributions $\mu$. Let $P, P'$ be two transition kernels.

Let $G \triangleq \sum_{k=0}^{\infty} \gamma^k P^k$ and $\bar{G} \triangleq (1-\gamma)G$. Define $G'$ and $\bar{G}'$ similarly. Therefore we have that $\bar{G}\mu$ is the discounted distribution of states visited by the markov process starting from distribution $\mu$. In other words, if $\mu$ is the distribution of $S_0$, and $P$ is the transition kernel induced by some policy $\pi$, then $\bar{G}\mu = \rho^\pi$.

First of all, let $\Delta = \gamma(P' - P)$ and we note that with simple algebraic manipulation,

$$\bar{G}' - \bar{G} = (1-\gamma)^{-1}\bar{G}'\Delta\bar{G} \tag{2.5.16}$$

Let $f$ be some function. We will mostly interested in the difference between $\mathbb{E}_{S\sim\bar{G}\mu}[f]$ and $\mathbb{E}_{S\sim\bar{G}'\mu}[f]$, which can be rewritten as $\langle(\bar{G}' - G)\mu, f\rangle$. We will bound this quantity from above by some divergence measure between $P'$ and $P$.

We start off with a simple lemma that controls the form $\langle p - q, f\rangle$ by the $\chi^2$ divergence between $p$ and $q$. With this lemma we can reduce our problem of bounding $\langle(\bar{G}' - G)\mu, f\rangle$ to characterizing the $\chi^2$ divergence between $\bar{G}'\mu$ and $\bar{G}\mu$.

**Lemma 2.5.17.** *Let $p$ and $q$ be probability distributions. Then we have*

$$\langle q - p, f\rangle^2 \le \chi^2(q,p) \cdot \langle p, f^2\rangle$$

*Proof of Lemma 2.5.17.* By Cauchy-Schwartz inequality, we have

$$\langle q - p, f\rangle^2 \le \left(\int \frac{(q(x) - p(x))^2}{p(x)} dx\right)\left(\int p(x) f(x)^2\right) = \chi^2(q,p) \cdot \langle p, f^2\rangle$$

□

The following Lemma is a refinement of the lemma above. It deals with the distributions $p$ and $q$ with the special structure $p = WP'\mu$ and $q = WP\mu$.

**Lemma 2.5.18.** *Let $W, P', P$ be transition kernels and $\mu$ be a distribution. Then,*

$$\langle W(P' - P)\mu, f\rangle^2 \le \chi^2_\mu(P', P)\langle WP\mu, f^2\rangle$$

*where $\chi^2_\mu(P', P)$ is a divergence between transitions defined in Definition 2.5.9.*

*Proof of Lemma 2.5.18.* By Lemma 2.5.17 with $p = WP\mu$ and $q = WP'\mu$, we conclude that

$$\langle W(P' - P)\mu, f\rangle^2 \le \chi^2(q, p) \cdot \langle p, f^2\rangle \le \chi^2(WP'\mu, WP\mu)\langle WP\mu, f^2\rangle$$

By Theorem 2.5.10 and Theorem 2.5.11 we have that $\chi^2(WP'\mu, WP\mu) \le \chi^2(P'\mu, P\mu) \le \chi^2_\mu(P', P)$, plugging this into the equation above we complete the proof. □

Now we are ready to state the main result of this subsection.

**Lemma 2.5.19.** *Let $\bar{G}, \bar{G}', P', P, f$ as defined in the beginning of this section. Let $\delta_1 = (1-\gamma)^{-1}\chi^2_{\bar{G}\mu}(P', P)^{1/2}$ and $\delta_2 = (1-\gamma)^{-1}\chi^2_{\bar{G}P\bar{G}\mu}(P', P)^{1/2}$. Then,*

$$\begin{aligned} \left|\langle \bar{G}'\mu, f\rangle - \langle \bar{G}\mu, f\rangle\right| &\le \delta_1\|f\|_\infty \\ \left|\langle \bar{G}'\mu, f\rangle - \langle \bar{G}\mu, f\rangle\right| &\le \delta_1\langle \bar{G}P\bar{G}\mu, f^2\rangle^{1/2} + \delta_1\delta_2^{1/2}\|f\|_\infty \end{aligned} \tag{2.5.17}$$

*Proof of Lemma 2.5.19.* Recall by Equation (2.5.16), we have

$$\langle(\bar{G}' - \bar{G})\mu, f\rangle = (1-\gamma)^{-1}\langle \bar{G}'\Delta\bar{G}\mu, f\rangle \tag{2.5.18}$$

By Lemma 2.5.18,

$$\langle \bar{G}'\Delta\bar{G}\mu, f\rangle \le \chi^2_{\bar{G}\mu}(P', P)^{1/2}\langle \bar{G}'P\bar{G}\mu, f^2\rangle^{1/2} \tag{2.5.19}$$

By Holder inequality and the fact that $\|\bar{G}\|_{1\to1} = 1$, $\|\bar{G}'\|_{1\to1} = 1$ and $\|P\|_{1\to1} = 1$, we have

$$\langle \bar{G}'\Delta\bar{G}\mu, f\rangle \le \chi^2_{\bar{G}\mu}(P', P)^{1/2}\langle \bar{G}'P\bar{G}\mu, f^2\rangle^{1/2}$$

$$
\begin{aligned}
&\leq \chi^2_{\bar{G}\mu}(P', P)^{1/2} \|\bar{G}' P \bar{G}\mu\|_1^{1/2} \|f^2\|_\infty^{1/2} \\
&\leq \chi^2_{\bar{G}\mu}(P', P)^{1/2} \|f\|_\infty \\
&\quad \text{(by } \|\bar{G}' P \bar{G}\mu\|_1 \leq \|\bar{G}'\|_{1\to 1} \|P\|_{1\to 1} \|\bar{G}\|_{1\to 1} \|\mu\|_1 \leq 1) \\
&\leq (1-\gamma)\delta_1 \|f\|_\infty \qquad (2.5.20)
\end{aligned}
$$

Combining Equations (2.5.18) and (2.5.20) we complete the proof of Equation (2.5.17).

Next we bound $\langle \bar{G}' P \bar{G}\mu, f^2 \rangle^{1/2}$ in a more refined manner. By Equation (2.5.16), we have

$$
\begin{aligned}
\langle \bar{G}' P \bar{G}\mu, f^2 \rangle^{1/2} &= \left( \langle \bar{G} P \bar{G}\mu, f^2 \rangle + \frac{1}{1-\gamma} \langle \bar{G}' \Delta \bar{G} P \bar{G}\mu, f^2 \rangle \right)^{1/2} \\
&\leq \langle \bar{G} P \bar{G}\mu, f^2 \rangle^{1/2} + (1-\gamma)^{-1/2} \langle \bar{G}' \Delta \bar{G} P \bar{G}\mu, f^2 \rangle^{1/2} \qquad (2.5.21)
\end{aligned}
$$

By Lemma 2.5.18 again, we have that

$$
\langle \bar{G}' \Delta \bar{G} P \bar{G}, f^2 \rangle^2 \leq \chi^2_{\bar{G} P \bar{G}\mu}(P', P) \langle \bar{G}' P \bar{G} P \bar{G}\mu, f^4 \rangle \qquad (2.5.22)
$$

By Holder inequality and the fact that $\|\bar{G}\|_{1\to 1} = 1$, $\|\bar{G}'\|_{1\to 1} = 1$ and $\|P\|_{1\to 1} = 1$, we have

$$
\langle \bar{G}' P \bar{G} P \bar{G}\mu, f^4 \rangle \leq \|\bar{G}' P \bar{G} P \bar{G}\mu\|_1 \|f^4\|_\infty \leq \|f\|_\infty^4 \qquad (2.5.23)
$$

Combining Equation (2.5.21), (2.5.23) gives

$$
(1-\gamma)^{-1/2} \langle \bar{G}' \Delta \bar{G} P \bar{G}, f^2 \rangle^{1/2} \leq ((1-\gamma)^{-1} \chi^2_{\bar{G} P \bar{G}\mu}(P', P)^{1/2})^{1/2} \|f\|_\infty = \delta_2^{1/2} \|f\|_\infty \qquad (2.5.24)
$$

Then, combining Equations (2.5.18), (2.5.19) and (2.5.24), we have

$$\begin{aligned}
\langle(\bar{G}' - \bar{G})\mu, f\rangle &= (1-\gamma)^{-1}\chi^2_{\bar{G}\mu}(P', P)^{1/2}\langle \bar{G}' P \bar{G}\mu, f^2\rangle^{1/2} \\
&\qquad\qquad \text{(by Equations (2.5.18) and (2.5.19))} \\
&= \delta_1 \langle \bar{G}' P \bar{G}\mu, f^2\rangle^{1/2} \\
&\leq \delta_1 \langle \bar{G} P \bar{G}\mu, f^2\rangle^{1/2} + \delta_1 (1-\gamma)^{-1/2}\langle \bar{G}'\Delta \bar{G} P \bar{G}\mu, f^2\rangle^{1/2} \\
&\qquad\qquad \text{(by Equation (2.5.21))} \\
&\leq \delta_1 \langle \bar{G} P \bar{G}\mu, f^2\rangle^{1/2} + \delta_1 \delta_2^{1/2}\|f\|_\infty \qquad \text{(by Equation (2.5.24))}
\end{aligned}$$

□

The following lemma is a stronger extension of Lemma 2.5.19, which can be used to future improve Proposition 2.5.15, and may be of other potential independent interests. We state it for completeness.

**Lemma 2.5.20.** *Let $\bar{G}, \bar{G}', P', P, f$ as defined in the beginning of this section. Let $d_k \triangleq (\bar{G}P)^k \bar{G}\mu$ and $\delta_k \triangleq (1-\gamma)^{-1}\chi^2_{d_{k-1}}(P', P)^{1/2}$, then we have that for any $K$,*

$$\begin{aligned}
\left|\langle \bar{G}'\mu, f\rangle - \langle \bar{G}\mu, f\rangle\right| &\leq \delta_1 \langle d_1, f^2\rangle^{-1/2} + \delta_1\delta_2^{1/2}\langle d_2, f^4\rangle^{-1/4} \\
&\quad + \delta_1 \ldots \delta_K^{2^{-K+1}} \langle d_k, f^{2^K}\rangle^{2^{-K}} + \delta_1 \ldots \delta_K^{2^{-K+1}}\|f\|_\infty
\end{aligned}$$

*Proof of Lemma 2.5.20.* We first use induction to prove that:

$$\begin{aligned}
|\langle(\bar{G}' - \bar{G})\mu, f\rangle| &\leq \sum_{k=1}^{K} \left(\prod_{0\leq s\leq k-1} \delta_{s+1}^{2^{-s}}\right) \langle d_k, f^{2^k}\rangle^{2^{-k}} \\
&\quad + \left(\prod_{0\leq s\leq K-1} \delta_{s+1}^{2^{-s}}\right) \langle \bar{G}'\Delta(\bar{G}P)^K \bar{G}\mu, f^{2^K}\rangle^{2^{-K}} \qquad (2.5.25)
\end{aligned}$$

By the first equation of Lemma 2.5.19, we got the case for $K = 1$. Assuming we have proved the case for $K$, then applying

$$\langle \bar{G}' \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle = \langle \bar{G} \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle + (1-\gamma)^{-1} \langle \bar{G} \Delta (\bar{G}P)^{K+1} \bar{G}\mu, f^{2^K} \rangle \tag{2.5.26}$$

$$\begin{aligned} &\leq \langle \bar{G} \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle \\ &\qquad + (1-\gamma)^{-1} \chi^2_{d_K}(P', P)^{1/2} \langle \bar{G}' \Delta (\bar{G}P)^{K+1} \bar{G}\mu, f^{2^{K+1}} \rangle^{1/2} \\ &\leq \langle \bar{G} \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle + \delta_{K+1} \langle \bar{G}' \Delta (\bar{G}P)^{K+1} \bar{G}\mu, f^{2^{K+1}} \rangle^{1/2} \end{aligned}$$

By Cauchy-Schwartz inequality, we obtain that

$$\begin{aligned} \langle \bar{G}' \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle^{2^{-K}} \leq \langle \bar{G} \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle^{2^{-K}} \\ + \delta_{K+1} \langle \bar{G}' \Delta (\bar{G}P)^{K+1} \bar{G}\mu, f^{2^{K+1}} \rangle^{2^{-K-1}} \end{aligned}$$

Plugging the equation above into Equation (2.5.25), we provide the induction hypothesis for the case with $K + 1$.

Now applying $\langle \bar{G}' \Delta (\bar{G}P)^K \bar{G}\mu, f^{2^K} \rangle^{2^{-K}} \leq \|f\|_\infty$ with Equation (2.5.25) we complete the proof.

□

With all supporting lemma above, we may prove Proposition 2.5.15.

*Proof of Proposition 2.5.15.* Let $\mu$ be the distribution of the initial state $S_0$, and let $P'$ and $P$ be the state-to-state transition kernel under policy $\pi$ and $\pi_{\text{ref}}$. Let $\bar{G} \triangleq (1-\gamma) \sum_{k=0}^{\infty} \gamma^k P^k$ and $\bar{G}' \triangleq (1-\gamma) \sum_{k=0}^{\infty} \gamma^k P'^k$. Under these notations, we can re-write $\rho^{\pi_{\text{ref}}} = \bar{G}\mu$ and $\rho^{\pi} = \bar{G}'\mu$. Moreover, we observe that $\beta^{\pi_{\text{ref}}} = \bar{G}P\bar{G}\mu$.

Let $\delta_1 \triangleq (1-\gamma)^{-1} \chi^2_{\bar{G}\mu}(P', P)^{1/2}$ and $\delta_2 \triangleq (1-\gamma)^{-1} \chi^2_{\bar{G}P\bar{G}\mu}(P', P)^{1/2}$ by the $\chi^2$ divergence between $P'$ and $P$, measured with respect to distributions $\bar{G}\mu = \rho^{\pi_{\text{ref}}}$ and

$\bar{G}P\bar{G}\mu = \beta^{\pi_{\text{ref}}}$. By Lemma 2.5.16, we have that the $\chi^2$-divergence between the states can be bounded by the $\chi^2$-divergence between the actions in the sense that:

$$\chi^2_{\bar{G}\mu}(P', P)^{1/2} = \chi^2_{\rho^{\pi_{\text{ref}}}}(P', P)^{1/2} \leq \mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\text{ref}}}} \left[\chi^2(\pi(\cdot|S), \pi_{\text{ref}}(\cdot|S))\right]^{1/2}$$
$$\chi^2_{\bar{G}P\bar{G}\mu}(P', P)^{1/2} = \chi^2_{\beta^{\pi_{\text{ref}}}}(P', P)^{1/2} \leq \mathop{\mathbb{E}}_{S\sim\beta^{\pi_{\text{ref}}}} \left[\chi^2(\pi(\cdot|S), \pi_{\text{ref}}(\cdot|S))\right]^{1/2}$$

Therefore we obtain that $\delta_1 \leq (1-\gamma)^{-1}\delta$, $\delta_2 \leq (1-\gamma)^{-1}\delta$. Let $f(s) = G^{\pi,\widehat{M}}(s)$. By Lemma 2.5.19, we can control the difference between $\langle \rho^{\pi_{\text{ref}}}, f\rangle$ and $\langle \rho^{\pi}, f\rangle$ by

$$\begin{aligned}
\left|\mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\text{ref}}}}\left[G^{\pi,\widehat{M}}(S)\right] - \mathop{\mathbb{E}}_{S\sim\rho^{\pi}}\left[G^{\pi,\widehat{M}}(S)\right]\right| &= |\langle \rho^{\pi_{\text{ref}}}, f\rangle - \langle \rho^{\pi}, f\rangle| \\
&\leq \delta_1 \langle \bar{G}P\bar{G}\mu, f^2\rangle^{1/2} + \delta_1\delta_2^{1/2}\|f\|_\infty \\
&\leq \delta_1\varepsilon_2 + \delta_1\delta_2^{1/2}\varepsilon_{\max}
\end{aligned}$$

It follows that

$$\begin{aligned}
\left|V^{\pi,\widehat{M}} - V^{\pi,M}\right| &\leq \gamma(1-\gamma)^{-1}\left|\mathop{\mathbb{E}}_{S\sim\rho^{\pi}}\left[G^{\pi,\widehat{M}}(S)\right]\right| && \text{(by Lemma 2.5.3)} \\
&\leq \gamma(1-\gamma)^{-1}\left(\left|\mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\text{ref}}}}\left[G^{\pi,\widehat{M}}(S)\right]\right| + \left|\mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\text{ref}}}}\left[G^{\pi,\widehat{M}}(S) - \mathop{\mathbb{E}}_{S\sim\rho^{\pi}}\left[G^{\pi,\widehat{M}}(S)\right]\right]\right|\right) \\
&\leq \gamma(1-\gamma)^{-1}\left|\mathop{\mathbb{E}}_{S\sim\rho^{\pi_{\text{ref}}}}\left[G^{\pi,\widehat{M}}(S)\right]\right| + \gamma(1-\gamma)^{-1}\delta_1\varepsilon_2 + \gamma(1-\gamma)^{-1}\delta_1\delta_2^{1/2}\varepsilon_{\max} \\
&\leq (1-\gamma)^{-1}\varepsilon_1 + (1-\gamma)^{-2}\delta\varepsilon_2 + (1-\gamma)^{-5/2}\delta^{3/2}\varepsilon_{\max}
\end{aligned}$$

□

## 2.6 Practical Implementation and Experiments

### 2.6.1 Practical Implementation

We design with simplification of our framework a variant of model-based RL algorithms, Stochastic Lower Bound Optimization (SLBO). First, we removed the constraints (2.4.4). Second, we stop the gradient w.r.t $M$ (but not $\pi$) from the occurrence of $M$ in $V^{\pi,M}$ in Equation (2.4.3) (and thus our practical implementation is not optimism-driven.)

Extending the discrepancy bound in Section 2.5.1, we use a multi-step prediction loss for learning the models with $\ell_2$ norm. For a state $s_t$ and action sequence $a_{t:t+h}$, we define the $h$-step prediction $\hat{s}_{t+h}$ as $\hat{s}_t = s_t,$ and for $h \geq 0, \hat{s}_{t+h+1} = \widehat{M}_\phi(\hat{s}_{t+h}, a_{t+h})$, The *$H$-step loss* is then defined as

$$\mathcal{L}_\phi^{(H)}((s_{t:t+h}, a_{t:t+h}); \phi) = \frac{1}{H}\sum_{i=1}^{H} \|(\hat{s}_{t+i} - \hat{s}_{t+i-1}) - (s_{t+i} - s_{t+i-1})\|_2. \tag{2.6.1}$$

A similar loss is also used in Nagabandi et al. [2018] for validation. We note that motivation by the theory in Section 2.5.1, we use $\ell_2$-norm instead of the square of $\ell_2$ norm. The loss function we attempt to optimize at iteration $k$ is thus[6]

$$\max_{\phi,\theta} \quad V^{\pi_\theta, \mathrm{sg}(\widehat{M}_\phi)} - \lambda \mathop{\mathbb{E}}_{(s_{t:t+h}, a_{t:t+h}) \sim \pi_k, M^\star} \left[\mathcal{L}_\phi^{(H)}((s_{t:t+h}, a_{t:t+h}); \phi)\right] \tag{2.6.2}$$

where $\lambda$ is a tunable parameter and sg denotes the stop gradient operation.

We note that the term $V^{\pi_\theta, \mathrm{sg}(\widehat{M}_\phi)}$ depends on both the parameter $\theta$ and the parameter $\phi$ but there is no gradient passed through $\phi$, whereas $\mathcal{L}_\phi^{(H)}$ only depends on the $\phi$. We optimize Equation (2.6.2) by alternatively maximizing $V^{\pi_\theta, \mathrm{sg}(\widehat{M}_\phi)}$ and minimizing $\mathcal{L}_\phi^{(H)}$: for the former, we use TRPO with samples from the estimated dynamics model $\widehat{M}_\phi$ (by treating $\widehat{M}_\phi$ as a fixed simulator), and for the latter we

[6]This is technically not a well-defined mathematical objective. The sg operation means identity when the function is evaluated, whereas when computing the update, $\mathrm{sg}(M_\phi)$ is considered fixed.

use standard stochastic gradient methods. Algorithm 2 gives a pseudo-code for the algorithm. The $n_{\text{model}}$ and $n_{\text{policy}}$ iterations are used to balance the number of steps of TRPO and Adam updates within the loop indexed by $n_{\text{inner}}$.[7]

**Algorithm 2** Stochastic Lower Bound Optimization (SLBO)

1: Initialize model network parameters $\phi$ and policy network parameters $\theta$
2: Initialize dataset $\mathcal{D} \leftarrow \emptyset$
3: **for** $n_{\text{outer}}$ iterations **do**
4: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{$ collect $n_{\text{collect}}$ samples from real environment using $\pi_\theta$ with noises $\}$
5: $\quad$ **for** $n_{\text{inner}}$ iterations **do** ▷ *optimize* (2.6.2) *with stochastic alternating updates*
6: $\quad\quad$ **for** $n_{\text{model}}$ iterations **do**
7: $\quad\quad\quad$ optimize (2.6.1) over $\phi$ with sampled data from $\mathcal{D}$ by one step of Adam
8: $\quad\quad$ **for** $n_{\text{policy}}$ iterations **do**
9: $\quad\quad\quad \mathcal{D}' \leftarrow \{$ collect $n_{\text{trpo}}$ samples using $\widehat{M}_\phi$ as dynamics $\}$
10: $\quad\quad\quad$ optimize $\pi_\theta$ by running TRPO on $\mathcal{D}'$

**Power of stochasticity and connection to standard model-based RL.** We identify the main advantage of our algorithms over standard model-based RL algorithms is that we alternate the updates of the model and the policy within an outer iteration. By contrast, most of the existing model-based RL methods only optimize the models once (for a lot of steps) after collecting a batch of samples (see Algorithm 3 for an example). The stochasticity introduced from the alternation with stochastic samples seems to dramatically reduce the overfitting (of the policy to the estimated dynamics model) in a way similar to that SGD regularizes ordinary supervised training. [8] Another way to view the algorithm is that the model obtained from line 7 of Algorithm 2 at different inner iteration serves as an ensemble of models. We do believe that a cleaner and easier instantiation of our framework (with optimism)

[7] In principle, to balance the number of steps, it suffices to take one of $n_{\text{model}}$ and $n_{\text{policy}}$ to be 1. However, empirically we found the optimal balance is achieved with larger $n_{\text{model}}$ and $n_{\text{policy}}$, possibly due to complicated interactions between the two optimization problem.

[8] Similar stochasticity can potentially be obtained by an extreme hyperparameter choice of the standard MB RL algorithm: in each outer iteration of Algorithm 3, we only sample a very small number of trajectories and take a few model updates and policy updates. We argue our interpretation of stochastic optimization of the lower bound (2.6.2) is more natural in that it reveals the regularization from stochastic optimization.

exists, and the current version, though performing very well, is not necessarily the best implementation.

**Entropy regularization.** An additional component we apply to SLBO is the commonly-adopted entropy regularization in policy gradient method [Williams and Peng, 1991, Mnih et al., 2016], which was found to significantly boost the performance in our experiments (ablation study in Section 2.6.3). Specifically, an additional entropy term is added to the objective function in TRPO. We hypothesize that entropy bonus helps exploration, diversifies the collected data, and thus prevents overfitting.

### 2.6.2 Experimental Results

We evaluate our algorithm SLBO (Algorithm 2) on five continuous control tasks from rllab [Duan et al., 2016], including Swimmer, Half Cheetah, Humanoid, Ant, Walker. All environments that we test have a maximum horizon of 500, which is longer than most of the existing model-based RL work [Nagabandi et al., 2018, Kurutach et al., 2018]. (Environments with longer horizons are commonly harder to train.) More details can be found in Section 2.6.4.

**Baselines.** We compare our algorithm with 3 other algorithms including: (1) Soft Actor-Critic (SAC) [Haarnoja et al., 2018], one of the state-of-the-art model-free off-policy algorithm in sample efficiency; (2) Trust-Region Policy Optimization (TRPO) [Schulman et al., 2015a], a policy-gradient based algorithm; and (3) Model-Based TRPO, a standard model-based algorithm described in Algorithm 3. Details of these algorithms can be found in Section 2.6.4.[9]

[9]We did not have the chance to implement the competitive random search algorithms in [Mania et al., 2018] yet, although our test performance with 500 episode length is higher than theirs with 1000 episode on Half Cheetach (3950 by ours vs 2345 by theirs) and Walker (3650 by ours vs 894 by theirs).

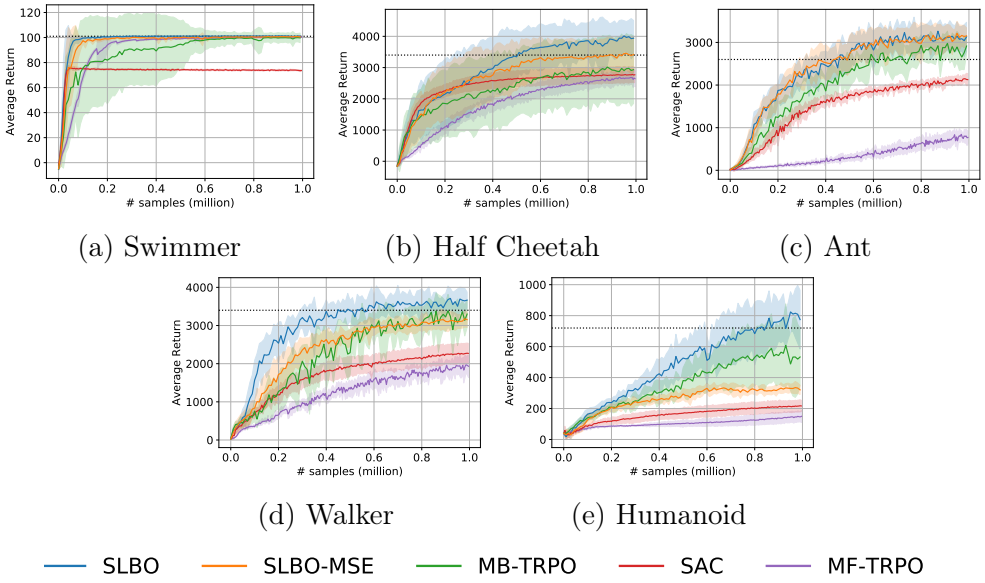
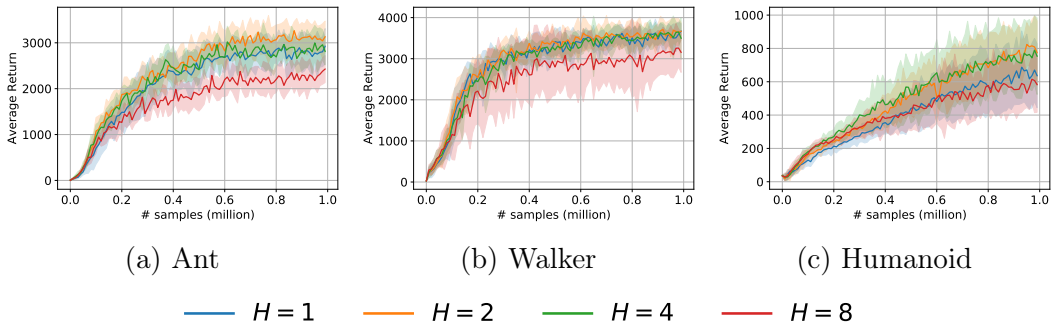



Figure 2.1: Comparison between SLBO (ours), SLBO with squared $\ell^2$ model loss (SLBO-MSE), vanilla model-based TRPO (MB-TRPO), model-free TRPO (MF-TRPO), and Soft Actor-Critic (SAC). We average the results over 10 different random seeds, where the solid lines indicate the mean and shaded areas indicate one standard deviation. The dotted reference lines are the total rewards of MF-TRPO after 8 million steps.

The result is shown in Figure 2.1. In Figure 2.1, our algorithm shows superior convergence rate (in number of samples) than all the baseline algorithms while achieving better final performance with 1M samples. Specifically, we mark model-free TRPO performance after 8 million steps by the dotted line in Figure 2.1 and find out that our algorithm can achieve comparable or better final performance in one million steps. For ablation study, we also add the performance of SLBO-MSE, which corresponds to running SLBO with squared $\ell_2$ model loss instead of $\ell_2$. SLBO-MSE performs significantly worse than SLBO on four environments, which is consistent with our derived model loss in Section 2.5.1. [10]

[10]Videos demonstrations are available at `https://sites.google.com/view/algombrl/home`https://sites.google.com/view/algombrl/home. A link to the codebase is available at `https://github.com/roosephu/slbo`https://github.com/roosephu/slbo.




(a) Ant (b) Walker (c) Humanoid

Figure 2.2: Ablation study on multi-step model training. All the experiments are average over 10 random seeds. The x-axis shows the total amount of real samples from the environment. The y-axis shows the averaged return from execution of our learned policy. The solid line is the mean of the total rewards from each seed. The shaded area is one-standard deviation.

### 2.6.3 Ablation Study

In this subsection, we study the performance of SLBO and baselines with 4 million training samples in Section 2.6.3 and the ablation study of multi-step model training can be found in Section 2.6.3.

**Multi-step model training.** We compare multi-step model training with single-step model training and the results are shown on Figure 2.2. Note that $H = 1$ means we use single-step model training. We observe that small $H$ (e.g., 2 or 4) can be beneficial, but larger $H$ (e.g., 8) can hurt. We hypothesize that smaller $H$ can help the model learn the uncertainty in the input and address the error-propagation issue to some extent. Pathak et al. [2018] uses an auto-regressive recurrent model to predict a multi-step loss on a trajectory, which is closely related to ours. However, theirs differs from ours in the sense that they do not use the predicted output $x_{t+1}$ as the input for the prediction of $x_{t+2}$, and so on and so forth.

**Entropy regularization.** Figure 2.3 shows that entropy reguarization can improve both sample efficiency and final performance. More entropy regularization leads to better sample efficiency and higher total rewards. We observe that in the late

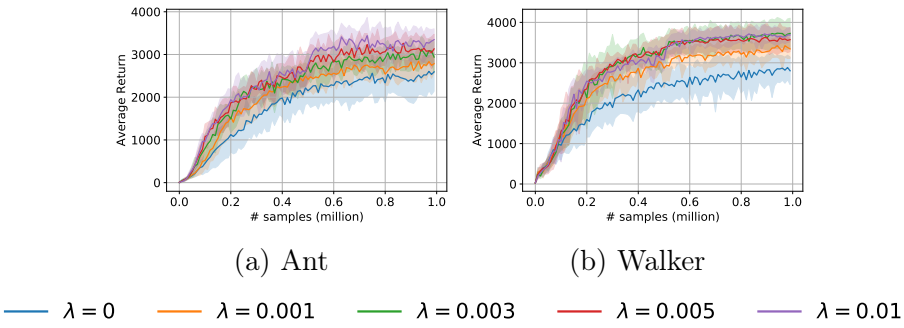



Figure 2.3: Ablation study on entropy regularization. $\lambda$ is the coefficient of entropy regularization in the TRPO's objective. All the experiments are averaged over 10 random seeds. The x-axis shows the total amount of real samples from the environment. The y-axis shows the averaged return from execution of our learned policy. The solid line is the mean of the total rewards from each seed. The shaded area is one-standard deviation.

iterations of training, entropy regularization may hurt the performance thus we stop using entropy regularization in the second half of training.

**SLBO with four million training steps.** Figure 2.4 shows that SLBO is superior to SAC and MF-TRPO in Swimmer, Half Cheetah, Walker and Humanoid when 4 million samples or fewer samples are allowed. For Ant environment , although SLBO with less than one million samples reaches the performance of MF-TRPO with 8 million samples, SAC's performance surpasses SLBO after 2 million steps of training. Since model-free TRPO almost stops improving after 8M steps and our algorithms uses TRPO for optimizing the estimated environment, we don't expect SLBO can significantly outperform the reward of TRPO at 8M steps. The result shows that SLBO is also satisfactory in terms of asymptotic convergence (compared to TRPO.) It also indicates a better planner or policy optimizer instead of TRPO might be necessary to further improve the performance.

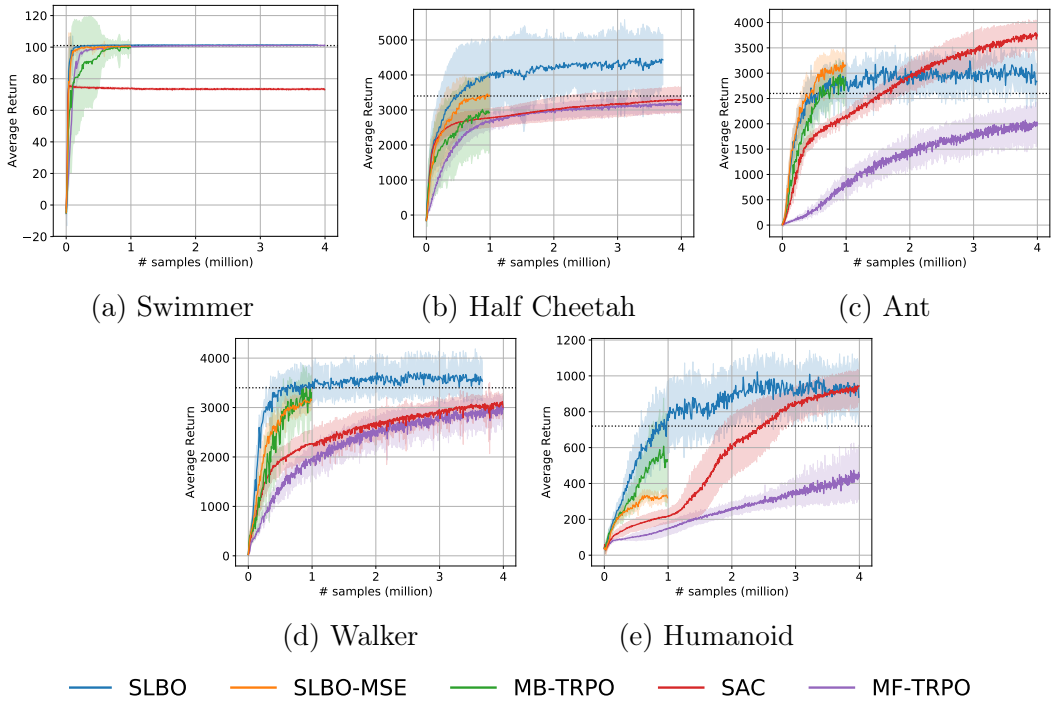



Figure 2.4: Comparison among SLBO (ours), SLBO with squared $\ell^2$ model loss (SLBO-MSE), vanilla model-based TRPO (MB-TRPO), model-free TRPO (MF-TRPO), and Soft Actor-Critic (SAC) with more samples than in Figure 2.1. SLBO, SAC, MF-TRPO are trained with 4 million real samples. We average the results over 10 different random seeds, where the solid lines indicate the mean and shaded areas indicate one standard deviation. The dotted reference lines are the total rewards of MF-TRPO after 8 million steps.

### 2.6.4 Implementation Details

**Environment setup.** We benchmark our algorithm on six tasks based on physics simulator Mujoco [Todorov et al., 2012b]. We use rllab's implementation [Duan et al., 2016] [11] to interact with Mujoco. All the environments we use have a maximum horizon of 500 steps. We remove all contact information from observation. To compute reward from states, we put the velocity of center of mass into the states.

**Network architecture and model learning.** We use the same reward function as in rllab, except that all the coefficients $C_{\text{contact}}$ in front of the contact force s are set to 0 in our case. We refer the readers to Duan et al. [2016] Supp Material 1.2 for more

[11] commit b3a2899 in `https://github.com/rll/rllab/`

details. All actions are projected to the action space by clipping. We normalize all observations by $s' = \frac{s-\mu}{\sigma}$ where $\mu, \sigma \in \mathbb{R}^{d_{\text{observation}}}$ are computed from all observations we collect from the real environment. Note that $\mu, \sigma$ may change as we collect new data. Our policy will always produce an action $a$ in $[-1, 1]^{d_{\text{action}}}$ and the action $a'$, which is fed into the environment, is scaled linearly by $a' = \frac{1-a}{2}a_{\min} + \frac{1+a}{2}a_{\max}$, where $a_{\min}, a_{\max}$ are the min or max values allowed at each entry.

**SLBO.** The dynamics model is represented by a feed-forward neural network with two hidden layers, each of which contains 500 hidden units. The activation function at each layer is ReLU. We use Adam to optimize the loss function with learning rate $10^{-3}$ and $L_2$ regularization $10^{-5}$. The network does not predict the next state directly; instead, it predicts the normalized difference of $s_{t+1} - s_t$. The normalization scheme and statistics are the same as those of observations: We maintain $\mu, \sigma$ from collected data in the real environment and may change them as we collect more, and the normalized difference is $\frac{s_{t+1}-s_t-\sigma}{\mu}$.

The policy network is a feed-forward network with two hidden layers, each of which contains 32 hidden units. The policy network uses tanh as activation function and outputs a Gaussian distribution $\mathcal{N}(\mu(s), \text{diag}(\sigma^2))$ where $\sigma$ a state-independent trainable vector.

During our evaluation, we use $H = 2$ for multi-step model training and the batch size is given by $\frac{256}{H} = 128$, i.e., we enforce the model to see 256 transitions at each batch.

We run our algorithm $n_{\text{outer}} = 100$ iterations. We collect $n_{\text{train}} = 10000$ steps of real samples from the environment at the start of each iteration using current policy with Ornstein-Uhlunbeck noise (with parameter $\theta = 0.15, \sigma = 0.3$) for better exploration. At each iteration, we optimize dynamics model and policy alternatively

for $n_{\text{inner}} = 20$ times. At each iteration, we optimize dynamics model for $n_{\text{model}} = 100$ times and optimize policy for $n_{\text{policy}} = 40$ times.

We tune multi-step model training parameter $H \in \{1, 2, 4, 8\}$, entropy regularization coefficient $\lambda \in \{0, 0.001, 0.003, 0.005\}$ and $n_{\text{policy}} \in \{10, 20, 40\}$ on Ant and find $H = 2, \lambda = 0.005, n_{\text{policy}} = 40$ work best, then we fix them in all environments, though environment-specific hyperparameters may work better. The other hyperparameters, including $n_{\text{inner}}, n_{\text{model}}$ and network architecture, are never tuned. We observe that at the first several iterations, the policy overfits to the learnt model so a reduction of $n_{\text{policy}}$ at the beginning can further speed up convergence but we omit this for simplicity.

The most important hyperparameters we found are $n_{\text{policy}}$ and the coefficient in front of the entropy regularizer $\lambda$. It seems that once $n_{\text{model}}$ is large enough we don't see any significant changes. We did have a held-out set for model prediction (with the same distribution as the training set) and found out the model doesn't overfit much. As mentioned in Section 2.6.4, we also found out normalizing the state helped a lot since the raw entries in the state have different magnitudes; if we don't normalize them, the loss will be dominated by the loss of some large entries.

**TRPO.** TRPO hyperparameters are listed at Table 2.1, which are the same as OpenAI Baselines' implementation. These hyperparameters are fixed for all experiments where TRPO is used, including ours, MB-TRPO and MF-TRPO. We do not tune these hyperparameters. We also normalize observations as our algorithm and OpenAI Baselines do.

We use a neural network as the value function to reduce variance, which has 2 hidden layers of units 64 and uses tanh as activation functions. We use Generalized Advantage Estimator (GAE) [Schulman et al., 2015b] to estimate advantages. Both

Table 2.1: TRPO Hyperparameters.

| **Hyperparameters** | **Values** |
|---|---|
| batch size | 4000 |
| max KL divergence | 0.01 |
| discount $\gamma$ | 0.99 |
| GAE $\lambda$ | 0.95 |
| CG iterations | 10 |
| CG damping | 0.1 |

TRPO used in our algorithm and that in model-free algorithm share the same set of hyperparameters.

**SAC.** For fair comparison, we do not use a large policy network (2 hidden layers, one of which has 256 hidden units) as the authors suggest, but use exactly the same policy network as ours. All other hyperparameters are kept the same as the authors'. Note that Q network and value network have 256 hidden units at each hidden layers, which is more than TRPO's. We refer the readers to Haarnoja et al. [2018] Appendix D for more details.

**MB-TRPO.** Model-Based TRPO (MB-TRPO) is similar to our algorithm SLBO but does not optimize model and policy alternatively during one iteration. We do not tune the hyperparameter $n_{\text{model}}$ since any number beyond a certain threshold would bring similar results. For $n_{\text{policy}}$ we try $\{100, 200, 400, 800\}$ on Ant and find $n_{\text{policy}} = 200$ works best in Ant so we use it for all other environments. Note that when Algorithm 2 uses 800 Adam updates (per outer iteration), it has the same amount of updates (per outer iteration) as in Algorithm 3. As suggested by Section 2.6.1, we use 0.005 as the coefficient of entropy bonus for all experiments.

**Algorithm 3** Model-Based Trust Region Policy Optimization (MB-TRPO)

initialize model network parameters $\phi$ and policy network parameters $\theta$
initialize dataset $\mathcal{D} \leftarrow \emptyset$
**for** $n_{\text{outer}}$ iterations **do**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{$ collect $n_{\text{collect}}$ samples from real environment using $\pi_\theta$ with noises $\}$
    **for** $n_{\text{model}}$ iterations **do**
        optimize (2.6.1) over $\phi$ with sampled data from $\mathcal{D}$ by one step of Adam
    **for** $n_{\text{policy}}$ iterations **do**
        $\mathcal{D}' \leftarrow \{$ collect $n_{\text{trpo}}$ samples using $\widehat{M}_\phi$ as dynamics $\}$
        optimize $\pi_\theta$ by running TRPO on $\mathcal{D}'$

## 2.7 Conclusion

We devise a novel algorithmic framework for designing and analyzing model-based RL algorithms with the guarantee to convergence monotonically to a local maximum of the reward. Experimental results show that our proposed algorithm (SLBO) achieves high performance on several mujoco benchmark tasks when one million or fewer samples are permitted.

A compelling (but obvious) empirical open question then given rise to is whether model-based RL can achieve near-optimal reward on other more complicated tasks or real-world robotic tasks with fewer samples. We believe that understanding the trade-off between optimism and robustness is essential to design more sample-efficient algorithms. Currently, we observed empirically that the optimism-driven part of our proposed meta-algorithm (optimizing $V^{\pi,\widehat{M}}$ over $\widehat{M}$) may lead to instability in the optimization, and therefore don't in general help the performance. It's left for future work to find practical implementation of the optimism-driven approach.

# Chapter 3

# Expressivity of Neural Networks in Deep Reinforcement Learning

We compare the model-free reinforcement learning with the model-based approaches through the lens of the expressive power of neural networks for policies, $Q$-functions, and dynamics. We show, theoretically and empirically, that even for one-dimensional continuous state space, there are many MDPs whose optimal $Q$-functions and policies are much more complex than the dynamics. We hypothesize many real-world MDPs also have a similar property. For these MDPs, model-based planning is a favorable algorithm, because the resulting policies can approximate the optimal policy significantly better than a neural network parameterization can, and model-free or model-based policy optimization rely on policy parameterization. Motivated by the theory, we apply a simple multi-step model-based bootstrapping planner (BOOTS) to bootstrap a weak $Q$-function into a stronger policy. Empirical results show that applying BOOTS on top of model-based or model-free policy optimization algorithms at the test time improves the performance on MuJoCo benchmark tasks.

## 3.1 Background

Model-based deep reinforcement learning (RL) algorithms offer a lot of potentials in achieving significantly better sample efficiency than the model-free algorithms for continuous control tasks. We can largely categorize the model-based deep RL algorithms into two types: 1. model-based policy optimization algorithms which learn policies or $Q$-functions, parameterized by neural networks, on the estimated dynamics, using off-the-shelf model-free algorithms or their variants [Luo et al., 2019a, Janner et al., 2019, Kaiser et al., 2019, Kurutach et al., 2018, Feinberg et al., 2018a, Buckman et al., 2018], and 2. model-based planning algorithms, which plan with the estimated dynamics [Nagabandi et al., 2018, Chua et al., 2018a, Wang and Ba, 2019].

A deeper theoretical understanding of the pros and cons of model-based and the model-free algorithms in the continuous state space case will provide guiding principles for designing and applying new sample-efficient methods. The prior work on the comparisons of model-based and model-free algorithms mostly focuses on their sample efficiency gap, in the case of tabular MDPs [Zanette and Brunskill, 2019, Jin et al., 2018], linear quadratic regulator [Tu and Recht, 2018], and contextual decision process with sparse reward [Sun et al., 2019].

In this chapter, we theoretically compare model-based RL and model-free RL in the continuous state space through the lens of *approximability* by neural networks, and then use the insight to design practical algorithms. What is the representation power of neural networks for expressing the $Q$-function, the policy, and the dynamics? How do the model-based and model-free algorithms utilize the expressivity of neural networks?

Our main finding is that even for the case of one-dimensional continuous state space, there can be a massive gap between the approximability of $Q$-function and the policy and that of the dynamics:

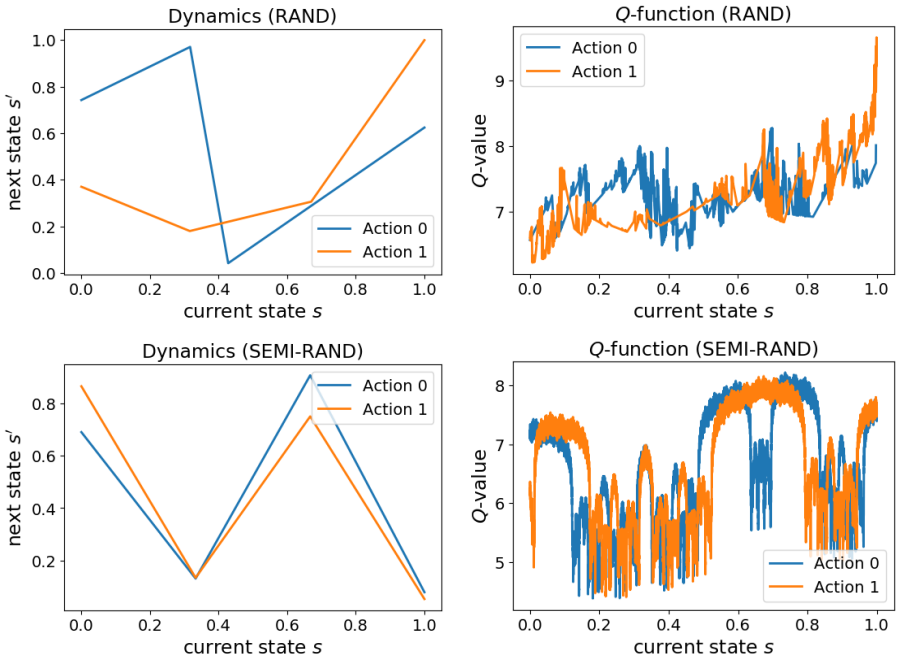



Figure 3.1: **Left:** The dynamics of two randomly generated MDPs (from the RAND, and SEMI-RAND methods detailed in Section 3.4.5). **Right:** The corresponding $Q$-functions which are more complex than the dynamics (more details in Section 3.4.5).

*The optimal Q-function and policy can be significantly more complex than the dynamics.*

We construct environments where the dynamics are simply piecewise linear functions with constant pieces, but the optimal $Q$-functions and the optimal policy require an exponential (in the horizon) number of linear pieces, or exponentially wide neural networks, to approximate.[1] The approximability gap can also be observed empirically on (semi-) randomly generated piecewise linear dynamics with a decent chance. (See Figure 3.1 for two examples.)

When the approximability gap occurs, any deep RL algorithms with policies parameterized by neural networks will suffer from a sub-optimal performance. These algorithms include both model-free algorithms such as DQN [Mnih et al., 2015] and

[1]In turn, the dynamics can also be much more complex than the $Q$-function. Consider the following situation: a subset of the coordinates of the state space can be arbitrarily difficult to express by neural networks, but the reward function can only depend on the rest of the coordinates and remain simple.

SAC [Haarnoja et al., 2018], and model-based policy optimization algorithms such as SLBO [Luo et al., 2019a] and MBPO [Janner et al., 2019]. To validate the intuition, we empirically apply these algorithms to the constructed or the randomly generated MDPs. Indeed, they fail to converge to the optimal rewards even with sufficient samples, which suggests that they suffer from the lack of expressivity.

However, in such cases, model-based planning algorithms should not suffer from the lack of expressivity, because they only use the learned, parameterized dynamics, which are easy to express. The policy obtained from the planning is the maximizer of the total future reward on the learned dynamics, and can have an exponential (in the horizon) number of pieces even if the dynamics has only a constant number of pieces. In fact, even a partial planner can help improve the expressivity of the policy. If we plan for $k$ steps and then resort to some $Q$-function for estimating the total reward of the remaining steps, we can obtain a policy with $2^k$ more pieces than what $Q$-function has.

We hypothesize that the real-world continuous control tasks also have a more complex optimal $Q$-function and a policy than the dynamics. The theoretical analysis of the synthetic dynamics suggests that a model-based few-steps planner on top of a parameterized $Q$-function will outperform the original $Q$-function because of the addtional expressivity introduced by the planning. We empirically verify the intuition on MuJoCo benchmark tasks. We show that applying a model-based planner on top of $Q$-functions learned from model-based or model-free policy optimization algorithms in the test time leads to significant gains over the original $Q$-function or policy.

In summary, our contributions are:

1. We construct continuous state space MDPs whose $Q$-functions and policies are proved to be more complex than the dynamics (Sections 3.4.1 and 3.4.3.)

2. We empirically show that with a decent chance, (semi-)randomly generated piecewise linear MDPs also have complex $Q$-functions (Section 3.4.5.)

3. We show theoretically and empirically that the model-free RL or model-based policy optimization algorithms suffer from the lack of expressivity for the constructed MDPs (Section 3.4.5), whereas model-based planning solve the problem efficiently (Section 3.6.)

4. Inspired by the theory, we propose a simple model-based bootstrapping planner (BOOTS), which can be applied on top of any model-free or model-based $Q$-learning algorithms at the test time. Empirical results show that BOOTS improves the performance on MuJoCo benchmark tasks.

## 3.2 Related Work

**Comparisons with Prior Theoretical Work.** Model-based RL has been extensively studied in the tabular case (see Zanette and Brunskill [2019], Azar et al. [2017] and the references therein), but much less so in the context of deep neural networks approximators and continuous state space. Luo et al. [2019a] give sample complexity and convergence guarantees suing principle of optimism in the face of uncertainty for non-linear dynamics.

Below we review several prior results regarding model-based versus model-free dichotomy in various settings. We note that our work focuses on the angle of expressivity, whereas the work below focuses on the sample efficiency.

**Tabular MDPs.** The extensive study in tabular MDP setting leaves little gap in their sample complexity of model-based and model-free algorithms, whereas the space complexity seems to be the main difference [Strehl et al., 2006]. The best sample complexity bounds for model-based tabular RL [Azar et al., 2017, Zanette and Brunskill, 2019] and model-free tabular RL [Jin et al., 2018] only differ by a poly($H$) multiplicative factor (where $H$ is the horizon.)

**Linear Quadratic Regulator.** Dean et al. [2018] and Dean et al. [2017] provided sample complexity bound for model-based LQR. Recently, Tu and Recht [2018] analyzed sample efficiency of the model-based and model-free problem in the setting of Linear Quadratic Regulator, and proved an $O(d)$ gap in sample complexity, where $d$ is the dimension of state space. Unlike tabular MDP case, the space complexity of model-based and model-free algorithms has little difference. The sample-efficiency gap mostly comes from that dynamics learning has $d$-dimensional supervisions, whereas $Q$-learning has only one-dimensional supervision.

**Contextual Decision Process (with function approximator).** Sun et al. [2019] prove an exponential information-theoretical gap between mode-based and model-free algorithms in the factored MDP setting. Their definition of model-free algorithms requires an exact parameterization: the value-function hypothesis class should be exactly the family of optimal value-functions induced by the MDP family. This limits the application to deep reinforcement learning where over-parameterized neural networks are frequently used. Moreover, a crucial reason for the failure of the model-free algorithms is that the reward is designed to be sparse.

**Related Empirical Work.** A large family of model-based RL algorithms uses existing model-free algorithms of its variant on the learned dynamics. MBPO [Janner et al., 2019], STEVE [Buckman et al., 2018], and MVE [Feinberg et al., 2018a] are model-based $Q$-learning-based policy optimization algorithms, which can be viewed as modern extensions and improvements of the early model-based $Q$-learning framework, Dyna [Sutton, 1990a]. SLBO [Luo et al., 2019a] is a model-based policy optimization algorithm using TRPO as the algorithm in the learned environment.

Another way to exploit the dynamics is to use it to perform model-based planning. Racanière et al. [2017] and Du and Narasimhan [2019] use the model to generated additional extra data to do planning implicitly. Chua et al. [2018a] study how to

combine an ensemble of probabilistic models and planning, which is followed by Wang and Ba [2019], which introduces a policy network to distill knowledge from a planner and provides a prior for the planner. Piché et al. [2018] uses methods in Sequential Monte Carlo in the context of control as inference. Oh et al. [2017] trains a $Q$-function and then perform lookahead planning. Nagabandi et al. [2018] uses random shooting as the planning algorithm.

Heess et al. [2015] backprops through a stochastic computation graph with a stochastic gradient to optimize the policy under the learned dynamics. Levine and Koltun [2013] distills a policy from trajectory optimization. Rajeswaran et al. [2016] trains a policy adversarially robust to the worst dynamics in the ensemble. Clavera et al. [2018] reformulates the problem as a meta-learning problem and using meta-learning algorithms. Predictron [Silver et al., 2017b] learns a dynamics and value function and then use them to predict the future reward sequences.

Another line of work focus on how to improve the learned dynamics model. Many of them use an ensemble of models [Kurutach et al., 2018, Rajeswaran et al., 2016, Clavera et al., 2018], which are further extended to an ensemble of probabilistic models [Chua et al., 2018a, Wang and Ba, 2019]. Luo et al. [2019a] designs a discrepancy bound for learning the dynamics model. Talvitie [2014] augments the data for model training in a way that the model can output a real observation from its own prediction. Malik et al. [2019] calibrates the model's uncertainty so that the model's output distribution should match the frequency of predicted states. Oh et al. [2017] learns a representation of states by predicting rewards and future returns using representation.

## 3.3 Preliminaries

**Bellman Equation.** Recall that $\pi^\star$ is the optimal policy, and $V^\star$ is the optimal value function (that is, the value function for policy $\pi^\star$). The value function $V^\pi$ for policy $\pi$

and optimal value function $V^\star$ satisfy the Bellman equation and Bellman optimality equation, respectively. Let $Q^\pi$ and $Q^\star$ defines the state-action value function for policy $\pi$ and optimal state-action value function. Then, for a deterministic dynamics $f$, we have

$$\begin{cases} V^\pi(s) = Q^\pi(s, \pi(s)), \\ Q^\pi(s,a) = r(s,a) + \gamma V^\pi(M(s,a)), \end{cases} \tag{3.3.1}$$

and

$$\begin{cases} V^\star(s) = \max_{a \in \mathcal{A}} Q^\star(s,a), \\ Q^\star(s,a) = r(s,a) + \gamma V^\star(M(s,a)). \end{cases} \tag{3.3.2}$$

Denote the Bellman operator for dynamics $M$ by $\mathcal{B}_M$: $\left(\mathcal{B}_M[Q]\right)(s,a) = r(s,a) + \max_{a'} \gamma Q(M(s,a), a')$.

**Neural Networks.** We focus on fully-connected neural nets with ReLU function as activations. A one-dimensional input and one-dimensional output ReLU neural net represents a piecewise linear function. A two-layer ReLU neural net with $d$ hidden neurons represents a piecewise linear function with at most $(d+1)$ pieces. Similarly, an $H$-layer neural net with $d$ hidden neurons in each layer represents a piecewise linear function with at most $(d+1)^H$ pieces [Pascanu et al., 2013].

**Problem Setting and Notations.** In this chapter, we focus on continuous state space, discrete action space MDPs with $\mathcal{S} \subset \mathbb{R}$. We assume the dynamics is deterministic (that is, $s_{t+1} = M(s_t, a_t)$), and the reward is known to the agent.

## 3.4 Approximability of $Q$-functions and Dynamics

We show that there exist MDPs in one-dimensional continuous state space that have simple dynamics but complex $Q$-functions and policies. Moreover, any polynomial-size

neural network function approximator of the $Q$-function or policy will result in a sub-optimal expected total reward, and learning $Q$-functions parameterized by neural networks requires fundamentally an exponential number of samples (Section 3.4.3). Section 3.4.5 illustrates the phenomena that $Q$-function is more complex than the dynamics occurring frequently and naturally even with random MDP, beyond the theoretical construction.

### 3.4.1 A Provable Construction of MDPs with Complex $Q$

Recall that we consider the infinite horizon case and $0 < \gamma < 1$ is the discount factor. Let $H = (1-\gamma)^{-1}$ be the "effective horizon" — the rewards after $\gg H$ steps become negligible due to the discount factor. For simplicity, we assume that $H > 3$ and it is an integer. (Otherwise we take just take $H = \lfloor (1-\gamma)^{-1} \rfloor$.) Throughout this section, we assume that the state space $\mathcal{S} = [0, 1)$ and the action space $\mathcal{A} = \{0, 1\}$.

**Definition 3.4.1.** Given the effective horizon $H = (1-\gamma)^{-1}$, we define an MDP $M_H$ as follows. Let $\kappa = 2^{-H}$. The dynamics $M$ by the following piecewise linear functions with at most three pieces.

$$M(s,0) \triangleq \begin{cases} 2s & \text{if } s < 1/2 \\ 2s-1 & \text{if } s \geq 1/2 \end{cases} \qquad M(s,1) \triangleq \begin{cases} 2s+\kappa & \text{if } s < (1-\kappa)/2 \\ 2s+\kappa-1 & \text{if } (1-\kappa)/2 \leq s \leq (2-\kappa)/2 \\ 2s+\kappa-2 & \text{otherwise.} \end{cases}$$

The reward function is defined as

$$r(s,0) \triangleq \mathbb{I}[1/2 \leq s < 1]$$
$$r(s,1) \triangleq \mathbb{I}[1/2 \leq s < 1] - 2(\gamma^{H-1} - \gamma^H)$$

The initial state distribution $\mu$ is uniform distribution over the state space $[0, 1)$.


(a) Visualization of dynamics for action $a = 0, 1$.

(b) The reward function $r(s, 0)$ and $r(s, 1)$.

(c) Approximation of optimal $Q$-function $Q^\star(s, a)$

Figure 3.2: A visualization of the dynamics, the reward function in the MDP defined in Definition 3.4.1, and the approximation of its optimal $Q$-function for the effective horizon $H = 4$. We can also construct slightly more involved construction with Lipschitz dynamics and very similar properties. Please see Section 3.4.2.

The dynamics and the reward function for $H = 4$ are visualized in Figures 3.2a and 3.2b. Note that by the definition, the transition function for a fixed action $a$ is a piecewise linear function with at most 3 pieces. Our construction can be modified so that the dynamics is Lipschitz and the same conclusion holds, which we will show in Section 3.4.2.

Attentive readers may also realize that the dynamics can be also be written succinctly as $M(s, 0) = 2s \mod 1$ and $M(s, 1) = 2s + \kappa \mod 1$[2], which are key properties that we use in the proof of Theorem 3.4.2 below.

**Optimal $Q$-function $Q^\star$ and the optimal policy $\pi^\star$.** Even though the dynamics of the MDP constructed in Definition 3.4.1 has only a constant number of pieces, the $Q$-function and policy are very complex: (1) the policy is a piecewise linear function with exponentially number of pieces, (2) the optimal $Q$-function $Q^\star$ and the optimal value function $V^\star$ are actually *fractals* that are not continuous anywhere. These are formalized in the theorem below.

[2]The mod function is defined as: $x \mod 1 \triangleq x - \lfloor x \rfloor$. More generally, for positive real $k$, we define $x \mod k \triangleq x - k\lfloor x/k \rfloor$.

**Theorem 3.4.2.** *For $s \in [0,1)$, let $s^{(k)}$ denotes the $k$-th bit of $s$ in the binary representation.*[3] *The optimal policy $\pi^\star$ for the MDP defined in Definition 3.4.1 has $2^{H+1}$ number of pieces. In particular,*

$$\pi^\star(s) = \mathbb{I}[s^{(H+1)} = 0]. \tag{3.4.1}$$

*And the optimal value function is a fractal with the expression:*

$$V^\star(s) = \sum_{h=1}^{H} \gamma^{h-1} s^{(h)} + \sum_{h=H+1}^{\infty} \gamma^{h-1} \left(1 + 2(s^{(h+1)} - s^{(h)})\right) + \gamma^{H-1} \left(2s^{(H+1)} - 2\right). \tag{3.4.2}$$

*The close-form expression of $Q^\star$ can be computed by $Q^\star(s,a) = r(s,a) + V^\star(M(s,a))$, which is also a fractal.*

We approximate the optimal $Q$-function by truncating the infinite sum to $2H$ terms, and visualize it in Figure 3.2c. We discuss the main intuitions behind the construction in the following proof sketch of the Theorem.

*Proof Sketch.* The key observation is that the dynamics $M$ essentially shift the binary representation of the states with some addition. We can verify that the dynamics satisfies $M(s,0) = 2s \mod 1$ and $M(s,1) = 2s + \kappa \mod 1$ where $\kappa = 2^{-H}$. In other words, suppose $s = 0.s^{(1)}s^{(2)}\cdots$ is the binary representation of $s$, and let left-shift$(s) = 0.s^{(2)}s^{(3)}\cdots$.

$$M(s,0) = \text{left-shift}(s) \tag{3.4.3}$$

$$M(s,1) = (\text{left-shift}(s) + 2^{-H}) \mod 1 \tag{3.4.4}$$

[3] Or more precisely, we define $s^{(h)} \triangleq \lfloor 2^h s \rfloor \mod 2$.

Moreover, the reward function is approximately equal to the first bit of the binary representation

$$r(s,0) = s^{(1)}, \quad r(s,a) \approx s^{(1)} \tag{3.4.5}$$

(Here the small negative drift of reward for action $a = 1$, $-2(\gamma^{H-1} - \gamma^H)$, is only mostly designed for the convenience of the proof, and casual readers can ignore it for simplicity.) Ignoring carries, the policy pretty much can only affect the $H$-th bit of the next state $s' = M(s,a)$: the $H$-th bit of $s'$ is either equal to $(H+1)$-th bit of $s$ when action is 0, or equal its flip when action is 1. Because the bits will eventually be shifted left and the reward is higher if the first bit of a future state is 1, towards getting higher future reward, the policy should aim to create more 1's. Therefore, the optimal policy should choose action 0 if the $(H+1)$-th bit of $s$ is already 1, and otherwise choose to flip the $(H+1)$-th bit by taking action 1.

A more delicate calculation that addresses the carries properly would lead us to the form of the optimal policy (Equation (3.4.1).) Computing the total reward by executing the optimal policy will lead us to the form of the optimal value function (Equation (3.4.2).) (This step does require some elementary but sophisticated algebraic manipulation.)

With the form of the $V^\star$, a shortcut to a formal, rigorous proof would be to verify that it satisfies the Bellman equation, and verify $\pi^\star$ is consistent with it. □

*Proof of Theorem 3.4.2.* Since the solution to Bellman optimal equations is unique, we only need to verify that $V^\star$ and $\pi^\star$ defined in Equation (3.3.2) satisfy the following,

$$V^\star(s) = r(s, \pi^\star(s)) + \gamma V^\star(M(s, \pi^\star(s))), \tag{3.4.6}$$

$$V^\star(s) \geq r(s,a) + \gamma V^\star(M(s,a)), \quad \forall a \neq \pi^\star(s). \tag{3.4.7}$$

Recall that $s^{(i)}$ is the $i$-th bit in the binary representation of $s$, that is, $s^{(i)} = \lfloor 2^i s \rfloor \bmod 2$. Let $\hat{s} = M(s, \pi^\star(s))$. Since $\pi^\star(s) = \mathbb{I}[s^{(H+1)} = 0]$, which ensures the $H$-bit of the next state is 1, we have

$$\hat{s}^{(i)} = \begin{cases} s^{(i+1)}, & i \neq H, \\ 1, & i = H. \end{cases} \tag{3.4.8}$$

For simplicity, define $\varepsilon = 2(\gamma^{H-1} - \gamma^H)$. The definition of $r(s, a)$ implies that

$$r(s, \pi^\star(s)) = \mathbb{I}[1/2 \leq s < 1] - \mathbb{I}[\pi^\star(s) = 1]\varepsilon = s^{(1)} - \left(1 - s^{(H+1)}\right)\varepsilon.$$

By elementary manipulation, Equation (3.4.2) is equivalent to

$$V^\star(s) = \sum_{i=1}^{H} \gamma^{i-1} s^{(i)} + \sum_{i=H+1}^{\infty} \left(\gamma^{i-1} - 2(\gamma^{i-2} - \gamma^{i-1})\left(1 - s^{(i)}\right)\right), \tag{3.4.9}$$

Now, we verify Equation (3.4.6) by plugging in the proposed solution (namely, Equation (3.4.9)). As a result,

$$\begin{aligned} r(s, \pi^\star(s)) + \gamma V^\star(\hat{s}) &= s^{(1)} - \left(1 - s^{(H+1)}\right)\varepsilon + \gamma \sum_{i=1}^{H} \gamma^{i-1} \mathbb{I}[\hat{s}^{(i)} = 1] \\ &\quad + \gamma \sum_{i=H+1}^{\infty} \left(\gamma^{i-1} - \left(1 - \hat{s}^{(i)}\right) 2(\gamma^{i-2} - \gamma^{i-1})\right) \\ &= s^{(1)} - \left(1 - s^{(H+1)}\right)\varepsilon + \sum_{i=2}^{H} \gamma^{i-1} s^{(i)} + \gamma^H \\ &\quad + \sum_{i=H+2}^{\infty} \left(\gamma^{i-1} - \left(1 - s^{(i)}\right) 2(\gamma^{i-2} - \gamma^{i-1})\right) \\ &= \sum_{i=1}^{H} \gamma^{i-1} s^{(i)} + \sum_{i=H+1}^{\infty} \left(\gamma^{i-1} - \left(1 - s^{(i)}\right) 2(\gamma^{i-2} - \gamma^{i-1})\right) \\ &= V^\star(s), \end{aligned}$$

which verifies Equation (3.4.6).

In the following we verify Equation (3.4.7). Consider any $a \neq \pi^\star(s)$. Let $\bar{s} = f(s,a)$ for shorthand. Note that $\bar{s}^{(i)} = s^{(i+1)}$ for $i > H$. As a result,

$$\begin{aligned}
&V^\star(s) - \gamma V^\star(\bar{s}) \\
&= \sum_{i=1}^{H} \gamma^{i-1} s^{(i)} + \sum_{i=H+1}^{\infty} \left(\gamma^{i-1} - \left(1 - s^{(i)}\right) 2(\gamma^{i-2} - \gamma^{i-1})\right) \\
&\quad - \sum_{i=1}^{H} \gamma^{i-1} \bar{s}^{(i)} - \sum_{i=H+1}^{\infty} \left(\gamma^{i-1} - \left(1 - \bar{s}^{(i)}\right) 2(\gamma^{i-2} - \gamma^{i-1})\right) \\
&= s^{(1)} + \sum_{i=1}^{H-1} \gamma^i \left(s^{(i+1)} - \bar{s}^{(i)}\right) - \gamma^H \bar{s}^{(H)} + \gamma^H - 2\left(1 - s^{(H+1)}\right)\left(\gamma^{H-1} - \gamma^H\right)
\end{aligned}$$

For the case where $s^{(H+1)} = 0$, we have $\pi^\star(s) = 1$. For $a = 0$, $\bar{s}^{(i)} = s^{(i+1)}$ for all $i \geq 1$. Consequently,

$$V^\star(s) - \gamma V^\star(\bar{s}) = s^{(1)} + \gamma^H - \varepsilon > s^{(1)} = r(s,0),$$

where the last inequality holds when $\gamma^H - \varepsilon > 0$, or equivalently, $\gamma > 2/3$.

For the case where $s^{(H+1)} = 1$, we have $\pi^\star(s) = 0$. For $a = 1$, we have $s^{(H+1)} = 1$ and $\bar{s}^{(H)} = 0$. Let $p = \max\{i \leq H : s^{(i)} = 0\}$, where we define the max of an empty set is 0. The dynamics $M(s,1)$ implies that

$$\bar{s}^{(i)} = \begin{cases} s^{(i+1)}, & i+1 < p \text{ or } i > H, \\ 1, & i+1 = p, \\ 0, & p < i+1 \leq H+1. \end{cases}$$

Therefore,

$$V^\star(s) - \gamma V^\star(\bar{s}) = s^{(1)} + \gamma^H + \sum_{i=1}^{H-1} \gamma^i \left(s^{(i+1)} - \bar{s}^{(i)}\right) > s^{(1)} - \varepsilon = r(s,1).$$

In both cases, we have $V^\star - \gamma V^\star(\bar{s}) > r(s,a)$ for $a \neq \pi^\star(s)$, which proves Equation (3.4.7). □

### 3.4.2 Extension of the Constructed Family

In this subsection, we present an extension to our construction in Section 3.4.1 such that the dynamics is Lipschitz. The action space is $\mathcal{A} = \{0,1,2,3,4\}$. We define $\textsc{clip}(x) = \max\{\min\{x,1\},0\}$.

**Definition 3.4.3.** Given effective horizon $H = (1-\gamma)^{-1}$, we define an MDP $M'_H$ as follows. Let $\kappa = 2^{-H}$. The dynamics is defined as

$$\begin{aligned}
M(s,0) &\triangleq \textsc{clip}(2s),\\
M(s,1) &\triangleq \textsc{clip}(2s-1),\\
M(s,2) &\triangleq \textsc{clip}(2s+\kappa),\\
M(s,3) &\triangleq \textsc{clip}(2s+\kappa-1),\\
M(s,4) &\triangleq \textsc{clip}(2s+\kappa-2).
\end{aligned}$$

Reward function is given by

$$\begin{aligned}
r(s,0) \triangleq r(s,1) &\triangleq \mathbb{I}[1/2 \le s < 1]\\
r(s,2) \triangleq r(s,3) \triangleq r(s,4) &\triangleq \mathbb{I}[1/2 \le s < 1] - 2(\gamma^{H-1} - \gamma^H)
\end{aligned}$$

The intuition behind the extension is that, we perform the mod operation manually. The following theorem is an analog to Theorem 3.4.2.

**Theorem 3.4.4.** *The optimal policy $\pi^\star$ for $M'_H$ is defined by,*

$$\pi^\star(s) \triangleq \begin{cases} 0, & \mathbb{I}[s^{(H+1)} = 0] \text{ and } 2s < 1, \\ 1, & \mathbb{I}[s^{(H+1)} = 0] \text{ and } 1 \le 2s < 2, \\ 2, & \mathbb{I}[s^{(H+1)} = 1] \text{ and } 2s + \theta < 1, \\ 3, & \mathbb{I}[s^{(H+1)} = 1] \text{ and } 1 \le 2s + \theta < 2, \\ 4, & \mathbb{I}[s^{(H+1)} = 1] \text{ and } 2 < 2s + \theta. \end{cases} \tag{3.4.10}$$

*And the corresponding optimal value function is,*

$$V^\star(s) = \sum_{h=1}^{H} \gamma^{h-1} s^{(h)} + \sum_{h=H+1}^{\infty} \gamma^{h-1} \left(1 + 2(s^{(h+1)} - s^{(h)})\right) + \gamma^{H-1} \left(2s^{(H+1)} - 2\right). \tag{3.4.11}$$

We can obtain a similar upper bound on the performance of policies with polynomial pieces.

**Theorem 3.4.5.** *Let $M_H$ be the MDP constructed in Definition 3.4.3. Suppose a piecewise linear policy $\pi$ has a near optimal reward in the sense that $\eta(\pi) \ge 0.99 \cdot \eta(\pi^\star)$, then it has to have at least $\Omega\left(\exp(cH)/H\right)$ pieces for some universal constant $c > 0$.*

The proof is very similar to that for Theorem 3.4.8. One of the difference here is to consider the case where $M(s, a) = 0$ or $M(s, a) = 1$ separately. Attentive readers may notice that the dynamics where $M(s, a) = 0$ or $M(s, a) = 1$ may destroy the "near uniform" behavior of state distribution $\mu_h^\pi$ (see Lemma 3.4.13). Here we show that such destroy comes with high cost. Formally speaking, if the clip is triggered in an interval, then the averaged single-step suboptimality gap is $0.1/(1 - \gamma)$.

**Lemma 3.4.6.** *Let $\ell_k \triangleq [k/2^{H/2}, (k + 1)/2^{H/2})$. For $k \in [2^{H/2}]$, if policy $\pi$ does not change its action at interval $\ell_k$ (that is, $|\{\pi(s) : s \in \ell_k\}| = 1$) and $M(s, \pi(s)) =$*

$0,\ \forall s \in \ell_k$ *or* $M(s, \pi(s)) = 1,\ \forall s \in \ell_k$. *We have*

$$\frac{1}{|\ell_k|} \int_{s \in \ell_k} (V^\star(s) - Q^\star(s, \pi(s)))\, ds \geq \frac{0.1}{1-\gamma} \tag{3.4.12}$$

*for large enough* $H$.

*Proof of Lemma 3.4.6.* Without loss of generality, we consider the case where $M(s, \pi(s)) = 0$. The proof for $M(s, \pi(s)) = 1$ is essentially the same.

By elementary manipulation, we have

$$V^\star(s) - V^\star(0) \geq \sum_{i=1}^{H} \gamma^{i-1} s^{(i)}.$$

Let $\hat{s} = M(s, \pi^\star(s))$. It follows from Bellman Equation (3.3.2) that

$$V^\star(s) = r(s, \pi^\star(s)) + \gamma V^\star(\hat{s}),$$
$$Q^\star(s, \pi(s)) = r(s, \pi(s)) + \gamma V^\star(0).$$

Recall that we define $\epsilon \triangleq 2\left(\gamma^{H-1} - \gamma^H\right)$. As a consequence,

$$\begin{aligned}(V^\star(s) - Q^\star(s, \pi(s))) &> r(s, \pi^\star(s)) - r(s, \pi(s)) + \gamma(V^\star(\hat{s}) - V^\star(0)) \\ &\geq -\epsilon + \gamma \sum_{i=1}^{H} \gamma^{i-1} \hat{s}^{(i)}.\end{aligned}$$

Plugging into Equation (3.4.12), we have

$$\begin{aligned}&\frac{1}{|\ell_k|} \int_{s \in \ell_k} (V^\star(s) - Q^\star(s, \pi(s)))\, ds \geq\ -\epsilon + \frac{1}{|\ell_k|} \int_{s \in \ell_k} \left( \sum_{i=1}^{H} \gamma^i \right) \hat{s}^{(i)}\, ds \\ \geq\ &-\epsilon + \sum_{i=1}^{H} \gamma^i \left( \frac{1}{|\ell_k|} \int_{s \in \ell_k} \hat{s}^{(i)}\, ds \right) \geq -\epsilon + \frac{\gamma^{H/2} - \gamma^H}{1-\gamma}.\end{aligned}$$

Equation (3.4.12) is proved by noticing for large enough $H$,

$$-\epsilon + \frac{\gamma^{H/2} - \gamma^H}{1-\gamma} > \frac{0.1}{1-\gamma}.$$

□

Let $D = \{0, 1\}$ for simplicity. For any policy $\pi$, we define a transition operator $\hat{\mathcal{T}}^\pi$, such that

$$\left(\hat{\mathcal{T}}^\pi \mu\right)(Z) \triangleq \mu\left(\{s : p(s,a) \in Z, M(s, \pi(s)) \notin D\right),$$

and the state distribution induced by it, defined recursively by

$$\begin{aligned}\hat{\mu}_1^\pi(s) &\triangleq 1,\\ \hat{\mu}_h^\pi &\triangleq \widehat{\mathcal{T}}^\pi \mu_{h-1}^\pi.\end{aligned}$$

We also define the density function for states that are truncated as follows,

$$\hat{\rho}_h^\pi(s) \triangleq \mathbb{I}[M(s, \pi(s)) \in D]\hat{\mu}_h^\pi(s).$$

Following advantage decomposition lemma (Corollary 3.4.11), the key step for proving Theorem 3.4.5 is

$$\begin{aligned}\eta(\pi^\star) - \eta(\pi) \geq \sum_{h=1}^{\infty} \gamma^{h-1} \mathbb{E}_{s\sim\hat{\mu}_h^\pi}\left[V^\star(s) - Q^\star(s, \pi(s))\right]\\ + \sum_{h=1}^{\infty} \gamma^h \mathbb{E}_{s\sim\rho_h^\pi}\left[V^\star(s) - Q^\star(s, \pi(s))\right].\end{aligned} \tag{3.4.13}$$

Similar to Lemma 3.4.13, the following lemma shows that the density for most of the small intervals is either uniformly clipped, or uniformly spread over this interval.

**Lemma 3.4.7.** *Let $z(\pi)$ be the number of pieces of policy $\pi$. For $k \in [2^{H/2}]$, define interval $\ell_k \triangleq [k/2^{H/2}, (k+1)/2^{H/2})$. Let $\nu_h(k) \triangleq \inf_{s\in\ell_k} \hat{\mu}_h^\pi(s)$ and $\omega_h(k) \triangleq \inf_{s\in\ell_k} \hat{\rho}_h^\pi(s)$. If the initial state distribution $\mu$ is uniform distribution, then for any $h \geq 1$,*

$$\sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \nu_h(k) + \sum_{h'=1}^{h-1} \sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \omega_{h'}(k) \geq 1 - 2h\frac{z(\pi)+10}{2^{H/2}}. \tag{3.4.14}$$

*Proof of Lemma 3.4.7.* Omitted. The proof is similar to Lemma 3.4.13. □

Now we present the proof for Theorem 3.4.5.

*Proof of Theorem 3.4.5.* For any $k \in [2^{H/2}]$, consider the interval $\ell_k = [k/2^{H/2}, (k+1)/2^{H/2})$.. If $\pi$ does not change at interval $\ell_k$ (that is, $|\{\pi(s) : s \in \ell_k\}| = 1$), by Lemma 3.4.12 we have

$$\int_{s\in\ell_k} (V^\star(s) - Q^\star(s, \pi(s)))\, ds \geq 0.075 \cdot 2^{-H/2}. \tag{3.4.15}$$

By Equations (3.4.12), (3.4.13) and (3.4.15), we have

$$\begin{aligned}
&\eta(\pi^\star) - \eta(\pi) \\
&\geq \sum_{h=1}^{H} \gamma^{h-1} \left( \sum_{k=0}^{2^{H/2}} 0.075 \cdot 2^{-H/2} \cdot \nu_h(k) \right) + \sum_{h=1}^{H} \sum_{k=0}^{2^{H/2}} \gamma^h \cdot 2^{-H/2} \cdot \omega_h(k) \cdot \frac{0.1}{1-\gamma}.
\end{aligned} \tag{3.4.16}$$

By Lemma 3.4.7, we get

$$\sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \nu_h(k) + \sum_{h'=1}^{h-1} \sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \omega_{h'}(k) \geq 1 - 2h\frac{z(\pi)+10}{2^{H/2}}. \tag{3.4.17}$$

For the sake of contradiction, we assume $z(\pi) = o\left(\exp(cH)/H\right)$, then for large enough $H$ we have,

$$1 - 2\frac{Hz(\pi)+10}{2^{H/2}} > 0.8.$$

Consequently,

$$\sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \nu_h(k) > 0.8 - \sum_{h'=1}^{h-1} \sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \omega_{h'}(k). \tag{3.4.18}$$

Plugging in Equation (3.4.16), we get

$$\begin{aligned}
&\eta(\pi^\star) - \eta(\pi) \\
&\geq \sum_{h=1}^{H} 0.075\gamma^{h-1} \left( \sum_{k=0}^{2^{H/2}} 2^{-H/2} \nu_h(k) \right) + \sum_{h=1}^{H} \sum_{k=0}^{2^{H/2}} \gamma^h \cdot 2^{-H/2} \cdot \omega_h(k) \cdot \frac{0.1}{1-\gamma}. \\
&\geq \sum_{h=1}^{H} 0.075\gamma^{h-1} \left( 0.8 - \sum_{h'=1}^{h-1} \sum_{k=0}^{2^{H/2}} 2^{-H/2} \cdot \omega_{h'}(k) \right) + \sum_{h=1}^{H} \sum_{k=0}^{2^{H/2}} \gamma^h \cdot 2^{-H/2} \cdot \omega_h(k) \cdot \frac{0.1}{1-\gamma} \\
&\geq 0.06 \frac{1-\gamma^H}{1-\gamma} + \sum_{h=1}^{H} \sum_{k=0}^{2^{H/2}} \cdot 2^{-H/2} \cdot \omega_h(k) \left( \frac{0.1\gamma^h}{1-\gamma} - 0.075 \sum_{h'=h}^{H} \gamma^{h'-1} \right) \\
&\geq 0.06 \frac{1-\gamma^H}{1-\gamma} + \sum_{h=1}^{H} \sum_{k=0}^{2^{H/2}} \cdot 2^{-H/2} \cdot \omega_h(k) \frac{\gamma^{h-1}}{1-\gamma} \left( 0.1\gamma - 0.075 \left( 1 - \gamma^{H-h} \right) \right)
\end{aligned}$$

When $\gamma > 1/4$, we have $0.1\gamma - 0.075(1-\gamma^{H-h}) > 0$. As a consequence,

$$\eta(\pi^\star) - \eta(\pi) > 0.06 \frac{1-\gamma^H}{1-\gamma} \geq \frac{0.01}{1-\gamma}.$$

Now, since $\eta(\pi^\star) \leq 1/(1-\gamma)$, we have $\eta(\pi) < 0.99\eta(\pi^\star)$. Therefore for near-optimal policy $\pi$, $z(\pi) = \Omega\left(\exp(cH)/H\right)$. □

### 3.4.3 Approximability of $Q$-function

A priori, the complexity of $Q^\star$ or $\pi^\star$ does not rule out the possibility that there exists an approximation of them that do an equally good job in terms of maximizing the rewards. However, we show that in this section, indeed, there is no neural network approximation of $Q^\star$ or $\pi^\star$ with a polynomial width. We prove this by showing any

piecewise linear function with a sub-exponential number of pieces cannot approximate either $Q^\star$ or $\pi^\star$ with a near-optimal total reward.

**Theorem 3.4.8.** *Let $M_H$ be the MDP constructed in Definition 3.4.1. Suppose a piecewise linear policy $\pi$ has a near optimal reward in the sense that $\eta(\pi) \geq 0.99 \cdot \eta(\pi^\star)$, then it has to have at least $\Omega\left(\exp(cH)/H\right)$ pieces for some universal constant $c > 0$. As a corollary, no constant depth neural networks with polynomial width (in $H$) can approximate the optimal policy with near optimal rewards.*

Consider a policy $\pi$ induced by a value function $Q$, that is, $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$. Then,when there are two actions, the number of pieces of the policy is bounded by twice the number of pieces of $Q$. This observation and the theorem above implies the following inapproximability result of $Q^\star$.

**Corollary 3.4.9.** *In the setting of Theorem 3.4.8, let $\pi$ be the policy induced by some $Q$. If $\pi$ is near-optimal in a sense that $\eta(\pi) \geq 0.99 \cdot \eta(\pi^\star)$, then $Q$ has at least $\Omega\left(\exp(cH)/H\right)$ pieces for some universal constant $c > 0$.*

The intuition behind the proof of Theorem 3.4.8 is as follows. Recall that the optimal policy has the form $\pi^\star(s) = \mathbb{I}[s^{(H+1)} = 0]$. One can expect that any polynomial-pieces policy $\pi$ behaves suboptimally in most of the states, which leads to the suboptimality of $\pi$.

### Proof of Theorem 3.4.8

For a fixed parameter $H$, let $z(\pi)$ be the number of pieces in $\pi$. For a policy $\pi$, define the state distribution when acting policy $\pi$ at step $h$ as $\mu_h^\pi$.

In order to prove Theorem 3.4.8, we show that if $1/2 - 2Hz(\pi)/2^H < 0.3$, then $\eta(\pi) < 0.99\eta(\pi^\star)$. The proof is based on the advantage decomposition lemma.

**Lemma 3.4.10** (Advantage Decomposition Lemma [Schulman et al., 2015a, Kakade and Langford, 2002])**.** *Define $A^\pi(s, a) \triangleq r(s, a) + \gamma V^\pi(M(s, a)) - V^\pi(s) = Q^\pi(s, a) -$*

$V^{\pi}(s)$. *Given policies* $\pi$ *and* $\tilde{\pi}$*, we have*

$$\eta(\pi) = \eta(\tilde{\pi}) + \sum_{h=1}^{\infty} \gamma^{h-1} \mathbb{E}_{s\sim\mu_h^{\pi}} \left[ A^{\tilde{\pi}}(s, \pi(s)) \right]. \tag{3.4.19}$$

**Corollary 3.4.11.** *For any policy* $\pi$*, we have*

$$\eta(\pi^{\star}) - \eta(\pi) = \sum_{h=1}^{\infty} \gamma^{h-1} \mathbb{E}_{s\sim\mu_h^{\pi}} \left[ V^{\star}(s) - Q^{\star}(s, \pi(s)) \right]. \tag{3.4.20}$$

Intuitively speaking, since $\pi^{\star} = \mathbb{I}[s^{(H+1)} = 0]$, the a policy $\pi$ with polynomial pieces behaves suboptimally in most of the states. Lemma 3.4.12 shows that the single-step suboptimality gap $V^{\star}(s) - Q^{\star}(s, \pi(s))$ is large for a constant portion of the states. On the other hand, Lemma 3.4.13 proves that the state distribution $\mu_h^{\pi}$ is near uniform, which means that suboptimal states can not be avoided. Combining with Corollary 3.4.11, the suboptimal gap of policy $\pi$ is large.

The next lemma shows that, if $\pi$ does not change its action for states from a certain interval, the average advantage term $V^{\star}(s) - Q^{\star}(s, \pi(s))$ in this interval is large.

**Lemma 3.4.12.** *Let* $\ell_k \triangleq [k/2^H, (k+1)/2^H)$*, and* $\mathcal{K} \triangleq \{0 \le k < 2^H : k \mod 2 = 1\}$*. Then for* $k \in \mathcal{K}$*, if policy* $\pi$ *does not change its action at interval* $\ell_k$ *(that is,* $|\{\pi(s) : s \in \ell_k\}| = 1$*), we have*

$$\frac{1}{|\ell_k|} \int_{s\in\ell_k} \left( V^{\star}(s) - Q^{\star}(s, \pi(s)) \right) ds \ge 0.15. \tag{3.4.21}$$

*Proof of Lemma 3.4.12.* Note that for any $k \in \mathcal{K}$, $s^{(H)} = 1, \forall s \in \ell_k$. Now fix a parameter $k \in \mathcal{K}$. Suppose $\pi(s) = a_i$ for $s \in \ell_k$. Then for any $s$ such that $s^{(H+1)} + i \ne 1$, we have

$$V^{\star}(s) - Q^{\star}(s, \pi(s)) \ge \gamma^H - \varepsilon.$$

For $H > 15$, we have $\gamma^H - \varepsilon > 0.3$. Therefore,

$$\int_{s\in\ell_k} (V^\star(s) - Q^\star(s,\pi(s)))\, ds \geq \int_{s\in\ell_k} 0.3\cdot\mathbb{I}[s^{(H+1)} \neq 1-i]\, ds \geq 0.3\cdot 2^{-H-1} = 0.15\cdot 2^{-H}.$$

□

Next lemma shows that when the number of pieces in $\pi$ is not too large, the distribution $\mu_h^\pi$ is close to uniform distribution for step $1 \leq h \leq H$.

**Lemma 3.4.13.** *Let $z(\pi)$ be the number of pieces of policy $\pi$. For $k \in [2^H]$, define interval $\ell_k \triangleq [k/2^H, (k+1)/2^H)$. Let $\nu_h(k) \triangleq \inf_{s\in\ell_k} \mu_h^\pi(s)$, If the initial state distribution $\mu$ is uniform distribution, then for any $h \geq 1$,*

$$\sum_{0\leq k<2^H} 2^{-H}\cdot\nu_h(k) \geq 1 - 2h\frac{z(\pi)}{2^H}. \tag{3.4.22}$$

*Proof of Lemma 3.4.13.* Now let us fix a parameter $H$ and policy $\pi$. For every $h$, we prove by induction that there exists a function $\xi_h(s)$, such that

(a) $0 \leq \xi_h(s) \leq \min\{\mu_h^\pi(s), 1\}$,

(b) $\inf_{s\in\ell_k} \xi_h(s) = \sup_{s\in\ell_k} \xi_h(s), \quad \forall k \in [A^H]$,

(c) $\int_{s\in[0,1)} d\xi_h(s) \geq 1 - h\cdot z(\pi)/2^{H-1}$.

For the base case $h = 1$, we define $\xi_h(s) = \mu_h^\pi(s) = 1$ for all $s \in [0,1)$. Now we construct $\xi_{h+1}$ from $\xi_h$.

For a fixed $k \in [2^H]$, define $l_k \triangleq k\cdot 2^{-H}, r_k \triangleq (k+1)\cdot 2^{-H}$ as the left and right endpoints of interval $\ell_k$. Let $\{x_k^{(i)}\}_{i=1}^2$ be the set of 2 solutions of equation

$$2x + 2^{-H} \equiv l_k \mod 1$$

where $0 \leq x < 1$, and we define $y_k^{(i)} \triangleq x_k^{(i)} + 2^{-H} \mod 1$. By definition, only states from the set $\cup_{i=1}^2 [x_k^{(i)}, y_k^{(i)})$ can reach states in interval $\ell_k$ by a single transition. We define a

set $I_k \triangleq \{i : 1 \le i \le 2, |\{\pi(s) : s \in [x_k^{(i)}, y_k^{(i)})\}| = 1\}$. That is, the intervals where policy $\pi$ acts unanimously. Consequently, for $i \in I_k$, the set $\{s : s \in [x_k^{(i)}, y_k^{(i)}), M(s, \pi(s)) \in \ell_k\}$ is an interval of length $2^{-H-1}$, and has the form

$$u_k^{(i)} \triangleq [x_k^{(i)} + w_k^{(i)} \cdot 2^{-H-1}, x_k^{(i)} + (w_k^{(i)} + 1) \cdot 2^{-H-1})$$

for some integer $w_k^{(i)} \in \{0, 1\}$. By statement (b) of induction hypothesis,

$$\inf_{s \in u_k^{(i)}} \xi_h(s) = \sup_{s \in u_k^{(i)}} \xi_h(s). \tag{3.4.23}$$

Now, the density $\xi_{h+1}(s)$ for $s \in \ell_k$ is defined as,

$$\xi_{h+1}(s) \triangleq \sum_{i \in I_k} \frac{1}{2} \cdot \xi_h(x_k^{(i)} + w_k^{(i)} \cdot 2^{-H-1}).$$

The intuition of the construction is that, we discard those density that cause non-uniform behavior (that is, the density in intervals $[x_k^{(i)}, y_k^{(i)})$ where $i \notin I_k$). When the number of pieces of $\pi$ is small, we can keep most of the density. Now, statement (b) is naturally satisfied by definition of $\xi_{h+1}$. We verify statement (a) and (c) below.

For any set $B \subseteq \ell_k$, let $(\mathcal{T}^\pi)^{-1}(B) \triangleq \{s \in \mathcal{S} : M(s, \pi(s)) \in B\}$ be the inverse of Markov transition $\mathcal{T}^\pi$. Then we have,

$$\begin{aligned}
&(\mathcal{T}^\pi \xi_h)(B) \triangleq \xi_h\left((\mathcal{T}^\pi)^{-1}(B)\right) = \sum_{i \in \{1,2\}} \xi_h\left((\mathcal{T}^\pi)^{-1}(B) \cap [x_k^{(i)}, y_k^{(i)})\right) \\
\ge &\sum_{i \in I_k} \xi_h\left((\mathcal{T}^\pi)^{-1}(B) \cap [x_k^{(i)}, y_k^{(i)})\right) \\
= &\sum_{i \in I_k} \left|(\mathcal{T}^\pi)^{-1}(B) \cap [x_k^{(i)}, y_k^{(i)})\right| \xi_h\left(x_k^{(i)} + w_k^{(i)} \cdot 2^{-H-1}\right) \qquad \text{(By Equation (3.4.23))} \\
= &\sum_{i \in I_k} \frac{|B|}{2} \xi_h\left(x_k^{(i)} + w_k^{(i)} \cdot 2^{-H-1}\right),
\end{aligned}$$

where $|\cdot|$ is the shorthand for standard Lebesgue measure.

By definition, we have

$$\xi_{h+1}(B) = \sum_{i \in I_k} \frac{|B|}{2} \xi_h \left( x_k^{(i)} + w_k^{(i)} \cdot 2^{-H-1} \right) \leq (\mathcal{T}^\pi \xi_h)(B) \leq (\mathcal{T}^\pi \mu_h^\pi)(B) = \mu_{h+1}^\pi(B),$$

which verifies statement (a).

For statement (c), recall that $\mathcal{S} = [0, 1)$ is the state space. Note that $\mathcal{T}^\pi$ preserve the overall density. That is $(\mathcal{T}^\pi \xi_h)(\mathcal{S}) = \xi_h(\mathcal{S})$. We only need to prove that

$$(\mathcal{T}^\pi \xi_h)(\mathcal{S}) - \xi_{h+1}(\mathcal{S}) \leq h \cdot z(\pi)/2^{H-1} \tag{3.4.24}$$

and statement (c) follows by induction.

By definition of $\xi_{h+1}(s)$ and the induction hypothesis that $\xi_h(s) \leq 1$, we have

$$(\mathcal{T}^\pi \xi_h)(\ell_k) - \xi_{h+1}(\ell_k) \leq (2 - |I_k|)2^{-H}.$$

On the other hand, for any $s \in \mathcal{S}$, the set $\{k \in [2^H] : s \in \cup_{i=1}^2 [x_k^{(i)}, y_k^{(i)})\}$ has cardinality 2, which means that one intermittent point of $\pi$ can correspond to at most 2 intervals that are not in $I_k$ for some $k$. Thus, we have

$$\sum_{0 \leq k < 2^H} |I_k| \geq 2^{H+1} - \sum_{s: \pi^-(s) \neq \pi^+(s)} \left| \{k \in [2^H] : s \in \cup_{i=1}^2 [x_k^{(i)}, y_k^{(i)})\} \right| \geq 2^{H+1} - 2 \cdot z(\pi).$$

Consequently

$$(\mathcal{T}^\pi \xi_h)(\mathcal{S}) - \xi_{h+1}(\mathcal{S}) = \sum_{0 \leq k < 2^H} ((\mathcal{T}^\pi \xi_h)(\ell_k) - \xi_{h+1}(\ell_k)) \leq z(\pi) 2^{-H+1},$$

which proves statement (c). □

Now we present the proof for Theorem 3.4.8.

*Proof of Theorem 3.4.8.* For any $k \in [2^H]$, consider the interval $\ell_k = [k/2^H, (k+1)/2^H)$. Let $\mathcal{K} \triangleq \{k \in [A^H] : k \mod 2 = 1\}$. If $\pi$ does not change at interval $\ell_k$ (that is, $|\{\pi(s) : s \in \ell_k\}| = 1$), by Lemma 3.4.12 we have

$$\int_{s\in\ell_k} (V^\star(s) - Q^\star(s, \pi(s)))\, ds \geq 0.15 \cdot 2^{-H}. \tag{3.4.25}$$

Let $\nu_h(k) \triangleq \inf_{s\in\ell_k} \mu_h^\pi(s)$, then by advantage decomposition lemma (namely, Corollary 3.4.11), we have

$$\begin{aligned}
\eta(\pi^\star) - \eta(\pi) &= \sum_{h=1}^{\infty} \gamma^{h-1} \left( \int_{s\in[0,1)} (V^*(s) - Q^\star(s, \pi(s)))\, d\mu_h^\pi(s) \right) \\
&\geq \sum_{h=1}^{H} \gamma^{h-1} \left( \sum_{k\in\mathcal{K}} \int_{s\in\ell_k} (V^*(s) - Q^\star(s, \pi(s)))\, d\mu_h^\pi(s) \right) \\
&\geq \sum_{h=1}^{H} \gamma^{h-1} \left( \sum_{k\in\mathcal{K}} \int_{s\in\ell_k} \nu_h(k)(V^*(s) - Q^\star(s, \pi(s)))\, ds \right) \\
&\geq \sum_{h=1}^{H} \gamma^{h-1} \left( \sum_{k\in\mathcal{K}} 0.15 \cdot 2^{-H} \cdot \nu_h(k) \right).
\end{aligned}$$

By Lemma 3.4.13 and union bound, we get

$$\sum_{k\in\mathcal{K}} 2^{-H} \cdot \nu_h(k) \geq \frac{1}{2} - 2h\frac{z(\pi)}{2^H}. \tag{3.4.26}$$

For the sake of contradiction, we assume $z(\pi) = o\left(\exp(cH)/H\right)$, then for large enough $H$ we have,

$$1/2 - \frac{2Hz(\pi)}{2^H} \geq 0.3.$$

Consequently,

$$\eta(\pi^\star) - \eta(\pi) \geq \sum_{h=1}^{H} 0.045\gamma^{h-1} = 0.045 \cdot \frac{1-\gamma^H}{1-\gamma} \geq \frac{0.01}{1-\gamma}.$$

Now, since $\eta(\pi^\star) \le 1/(1-\gamma)$, we have $\eta(\pi) < 0.99\eta(\pi^\star)$. Therefore for near-optimal policy $\pi$, $z(\pi) = \Omega\left(\exp(cH)/H\right)$. □

### 3.4.4 Sample Complexity Lower Bound of Q-learning

Beyond the expressivity lower bound, we also provide an exponential sample complexity lower bound for Q-learning algorithms parameterized with neural networks.

Recall that Corollary 3.4.9 says that in order to find a near-optimal policy by a Q-learning algorithm, an exponentially large $Q$-network is required. In this subsection, we show that even if an exponentially large $Q$-network is applied for $Q$ learning, still we need to collect an exponentially large number of samples, ruling out the possibility of efficiently solving the constructed MDPs with Q-learning algorithms.

Towards proving the sample complexity lower bound, we consider a stronger family of Q-learning algorithm, *Q-learning with Oracle* (Algorithm 4). We assume that the algorithm has access to a Q-Oracle, which returns the optimal $Q$-function upon querying any pair $(s, a)$ during the training process. *Q-learning with Oracle* is conceptually a stronger computation model than the vanilla Q-learning algorithm, because it can directly fit the $Q$ functions with supervised learning, without relying on the rollouts or the previous $Q$ function to estimate the target $Q$ value. Theorem 3.4.14 proves a sample complexity lower bound for Q-learning algorithm on the constructed example.

**Theorem 3.4.14** (Informal Version of Theorem 3.4.16)**.** *Suppose $\mathcal{Q}$ is an infinitely-wide two-layer neural networks, and $R(Q)$ is $\ell_1$ norm of the parameters and serves as a tiebreaker. Then, any instantiation of the* Q-learning with oracle *algorithm requires exponentially many samples to find a policy $\pi$ such that $\eta(\pi) > 0.99\eta(\pi^\star)$.*

The proof of Theorem 3.4.14 is to exploit the sparsity of the solution found by minimal-norm tie-breaker. It can be proven that there are at most $O(n)$ non-zero

**Algorithm 4** Q-LEARNING WITH ORACLE

**Require:** A hypothesis space $\mathcal{Q}$ of $Q$-function parameterization.

1: Sample $s_0 \sim \mu$ from the initial state distribution $\mu$
2: **for** $i = 1, 2, \cdots, n$ **do**
3: Decide whether to restart the trajectory by setting $s_i \sim \mu$ based on historical information
4: Query Q-ORACLE to get the function $Q^\star(s_i, \cdot)$.
5: Apply any action $a_i$ (according to any rule) and sample $s_{i+1} \sim M(s_i, a_i)$.
6: Learn the $Q$-function that fit all the data the best:

$$Q \leftarrow \arg\min_{Q \in \mathcal{Q}} \frac{1}{n} \sum_{i=1}^{n} \left(Q(s_i, a_i) - Q^\star(s_i, a_i)\right)^2 + \lambda R(Q)$$

7: Return the greedy policy according to $Q$.

neurons in the minimal-norm solution, where $n$ is the number of data points. The proof is completed by combining with Theorem 3.4.8.

### Proof of Theorem 3.4.14

A two-layer ReLU neural net $Q(s, \cdot)$ with input $s$ is of the following form,

$$Q(s, a) = \sum_{i=1}^{d} w_{i,a} \left[k_i s + b_i\right]_+ + c_a, \tag{3.4.27}$$

where $d$ is the number of hidden neurons. $w_{i,a}, c_a, k_i, b_i$ are parameters of this neural net, where $c_{i,a}, b_i$ are bias terms. $[x]_+$ is a shorthand for ReLU activation $\mathbb{I}[x > 0]x$. Now we define the norm of a neural net.

**Definition 3.4.15** (Norm of a Neural Net)**.** The norm of a two-layer ReLU neural net is defined as,

$$\sum_{i=1}^{d} |w_{i,a}| + |k_i|. \tag{3.4.28}$$

Recall that the *Q-learning with oracle* algorithm finds the solution by the following supervised learning problem,

$$\min_{Q\in\mathcal{Q}} \frac{1}{n}\sum_{t=1}^{n} \left(Q(s_t,a_t) - Q^\star(s_t,a_t)\right)^2. \tag{3.4.29}$$

Then, we present the formal version of Theorem 3.4.14.

**Theorem 3.4.16.** *Let $Q$ be the minimal $\ell_1$ norm solution to Equation* (3.4.29)*, and $\pi$ the greedy policy according to $Q$. When $n = o(\exp(cH)/H)$, we have $\eta(\pi) < 0.99\eta(\pi^\star)$.*

The proof of Theorem 3.4.14 is by characterizing the minimal-norm solution, namely the sparsity of the minimal-norm solution as stated in the next lemma.

**Lemma 3.4.17.** *The minimal-norm solution to Equation* (3.4.29) *has at most $32n+1$ non-zero neurons. That is, $|\{i : k_i \neq 0\}| \leq 32n+1$.*

We first present the proof of Theorem 3.4.16, followed by the proof of Lemma 3.4.17.

*Proof of Theorem 3.4.16.* Recall that the policy is given by $\pi(s) = \arg\max_{a\in\mathcal{A}} Q(s,a)$. For a $Q$-function with $32n+2$ pieces, the greedy policy according to $Q(s,a)$ has at most $64n+4$ pieces. Combining with Theorem 3.4.8, in order to find a policy $\pi$ such that $\eta(\pi) > 0.99\eta(\pi^\star)$, $n$ needs to be exponentially large (in effective horizon $H$). □

Proof of Lemma 3.4.17 is based on merging neurons. Let $x_i = -b_i/k_i, \mathbf{w}_i = (w_{i,1}, w_{i,2})$, and $\mathbf{c} = (c_1, c_2)$. In vector form, neural net defined in Equation (3.4.27) can be written as,

$$Q(s,\cdot) = \sum_{i=1}^{d} \mathbf{w}_i \left[k_i(s-x_i)\right]_+ + \mathbf{c}.$$

First we show that neurons with the same $x_i$ can be merged together.

**Lemma 3.4.18.** *Consider the following two neurons,*

$$k_1 \left[s-x_1\right]_+ \mathbf{w}_1, \quad k_2 \left[s-x_2\right]_+ \mathbf{w}_2.$$

*with $k_1 > 0$, $k_2 > 0$. If $x_1 = x_2$, then we can replace them with one single neuron of the form $k' [x - x_1]_+ \mathbf{w}'$ without changing the output of the network. Furthermore, if $\mathbf{w}_1 \neq 0, \mathbf{w}_2 \neq 0$, the norm strictly decreases after replacement.*

*Proof of Lemma 3.4.18.* We set $k' = \sqrt{|k_1\mathbf{w}_1 + k_2\mathbf{w}_2|_1}$, and $w' = (k_1\mathbf{w}_1 + k_2\mathbf{w}_2)/k'$, where $|\mathbf{w}|_1$ represents the 1-norm of vector $\mathbf{w}$. Then, for all $s \in \mathbb{R}$,

$$k' [x - x_1]_+ \mathbf{w}' = (k_1\mathbf{w}_1 + k_2\mathbf{w}_2) [s - x_1]_+ = k_1 [s - x_1]_+ \mathbf{w}_1 + k_2 [s - x_1]_+ \mathbf{w}_2.$$

The norm of the new neuron is $|k'| + |\mathbf{w}'|_1$. By calculation we have,

$$\begin{aligned} |k'| + |\mathbf{w}'|_1 &= 2\sqrt{|k_1\mathbf{w}_1 + k_2\mathbf{w}_2|_1} \leq 2\sqrt{|k_1\mathbf{w}_1|_1 + |k_2\mathbf{w}_2|_1} \\ &\overset{(a)}{\leq} 2\left(\sqrt{|k_1\mathbf{w}_1|_1} + \sqrt{|k_2\mathbf{w}_2|_1}\right) \leq |k_1| + |\mathbf{w}_1|_1 + |k_2| + |\mathbf{w}_2|_1. \end{aligned}$$

Note that the inequality (a) is strictly less when $|k_1\mathbf{w}_1|_1 \neq 0$ and $|k_2\mathbf{w}_2|_1 \neq 0$. □

Next we consider merging two neurons with different intercepts between two data points. Without loss of generality, assume the data points are listed in ascending order. That is, $s_i \leq s_{i+1}$.

**Lemma 3.4.19.** *Consider two neurons*

$$k_1 [s - x_0]_+ \mathbf{w}_1, \quad k_2 [s - x_0 - \delta]_+ \mathbf{w}_2.$$

*with $k_1 > 0, k_2 > 0$. If $s_i \leq x_0 < x_0 + \delta \leq s_{i+1}$ for some $1 \leq i \leq n$, then the two neurons can replaced by a set of three neurons,*

$$k' [s - x_0]_+ \mathbf{w}', \quad \tilde{k} [s - s_i]_+ \tilde{\mathbf{w}}, \quad \tilde{k} [s - s_{i+1}]_+ (-\tilde{\mathbf{w}})$$

*such that for $s \leq s_i$ or $s \geq s_{i+1}$, the output of the network is unchanged. Furthermore, if $\delta \leq (s_{i+1} - s_i)/16$ and $|\mathbf{w}_1|_1 \neq 0, |\mathbf{w}_2|_1 \neq 0$, the norm decreases strictly.*

Before proving Lemma 3.4.19, we present a technical lemma.

**Lemma 3.4.20.** *For $A, B, C, D \geq 0$ and $AC \geq BD$, we have*

$$A + C + \frac{1}{2}(B + D) \geq 2\sqrt{AC + BD}.$$

*Furthermore, when $BD > 0$, the inequality is strict.*

*Proof of Lemma 3.4.20.* Note that $A+B+\frac{1}{2}(C+D) \geq 2\sqrt{AC}+\sqrt{BD}$. And we have,

$$\left(2\sqrt{AC} + \sqrt{BD}\right)^2 - \left(2\sqrt{AC + BD}\right)^2 = 4\sqrt{AC \cdot BD} - 3BD \geq BD \geq 0.$$

And when $BD > 0$, the inequality is strict. □

Now we are going to prove Lemma 3.4.19.

*Proof of Lemma 3.4.19.* For simplicity, define $\Delta = s_{i+1} - s_i$. We set

$$\begin{aligned}
k' &= \sqrt{|k_1\mathbf{w}_1 + k_2\mathbf{w}_2|_1}, \\
\mathbf{w}' &= (k_1\mathbf{w}_1 + k_2\mathbf{w}_2)/k', \\
\tilde{k} &= \sqrt{|k_2\mathbf{w}_2|_1\delta/\Delta}, \\
\tilde{\mathbf{w}} &= -k_2\mathbf{w}_2\delta/(\Delta\tilde{k}).
\end{aligned}$$

Note that for $s \leq s_i$, all of the neurons are inactive. For $s \geq s_{i+1}$, all of the neurons are active, and

$$\begin{aligned}
&k'\mathbf{w}'(s - x_0) + \tilde{k}\tilde{\mathbf{w}}(s - s_i) - \tilde{k}\tilde{\mathbf{w}}(s - s_{i+1}) \\
&= (k_1\mathbf{w}_1 + k_2\mathbf{w}_2)(s - x_0) - k_2\mathbf{w}_2\delta \\
&= k_1(s - x_0)\mathbf{w}_1 + k_2(s - x_0 - \delta)\mathbf{w}_2,
\end{aligned}$$

which means that the output of the network is unchanged. Now consider the norm of the two networks. Without loss of generality, assume $|k_1\mathbf{w}_1|_1 > |k_2\mathbf{w}_2|_1$. The original network has norm $|k_1| + |\mathbf{w}_1|_1 + |k_2| + |\mathbf{w}_2|_1$. And the new network has norm

$$\begin{aligned}
&|k'| + |\mathbf{w}'|_1 + 2|\tilde{k}| + 2|\tilde{\mathbf{w}}|_1 = 2\sqrt{|k_1\mathbf{w}_1 + k_2\mathbf{w}_2|_1} + 4\sqrt{|k_2\mathbf{w}_2|_1\delta/\Delta} \\
\overset{(a)}{\leq}\ &|k_1| + |\mathbf{w}_1|_1 + |k_2| + |\mathbf{w}_2|_1 + \left(4\sqrt{|k_2\mathbf{w}_2|_1\delta/\Delta} - \frac{1}{2}(|k_2| + |\mathbf{w}_2|_1)\right),
\end{aligned}$$

where the inequality (a) is a result of Lemma 3.4.20, and is strictly less when $|\mathbf{w}_1|_1 \neq 0, |\mathbf{w}_2|_1 \neq 0$.

When $\delta/\Delta < 1/16$, we have $\left(4\sqrt{|k_2\mathbf{w}_2|_1\delta/\Delta} - \frac{1}{2}(|k_2| + |\mathbf{w}_2|_1)\right) < 0$, which implies that

$$|k'| + |\mathbf{w}'|_1 + 2|\tilde{k}| + 2|\tilde{\mathbf{w}}|_1 < |k_1| + |\mathbf{w}_1|_1 + |k_2| + |\mathbf{w}_2|_1.$$

Similarly, two neurons with $k_1 < 0$ and $k_2 < 0$ can be merged together. □

Now we are ready to prove Lemma 3.4.17. As hinted by previous lemmas, we show that between two data points, there are at most 34 non-zero neurons in the minimal norm solution.

*Proof of Lemma 3.4.17.* Consider the solution to Equation (3.4.29). Without loss of generality, assume that $s_i \leq s_{i+1}$. In the minimal norm solution, it is obvious that $|\mathbf{w}_i|_1 = 0$ if and only if $k_i = 0$. Therefore we only consider those neurons with $k_i \neq 0$, denoted by index $1 \leq i \leq d'$.

Let $\mathcal{B}_t \triangleq \{-b_i/k_i : 1 \leq i \leq d', s_t < -b_i/k_i < s_{t+1}, k_i > 0\}$. Next we prove that in the minimal norm solution, $|\mathcal{B}_t| \leq 15$. For the sake of contradiction, suppse $|\mathcal{B}_t| > 15$. Then there exists $i, j$ such that, $s_t < -b_i/k_i < s_{t+1}, s_t < -b_j/k_j < s_{t+1}, |b_i/k_i - b_j/k_j| < (s_{t+1} - s_i)/16$, and $k_i > 0, k_j > 0$. By Lemma 3.4.19, we can obtain a neural net with smaller norm by merging neurons $i, j$ together without violating Equation (3.4.29), which leads to contradiction.

By Lemma 3.4.18, $|\mathcal{B}_t| \leq 15$ implies that there are at most 15 non-zero neurons with $s_t < -b_i/k_i < s_{t+1}$ and $k_i > 0$. For the same reason, there are at most 15 non-zero neurons with $s_t < -b_i/k_i < s_{t+1}$ and $k_i < 0$.

On the other hand, there are at most 2 non-zero neurons with $s_t = -b_i/k_i$ for all $t \leq n$, and there are at most 1 non-zero neurons with $-b_i/k_i < s_1$. Therefore, we have $d' \leq 32n + 1$. □

### 3.4.5 Approximability of $Q$-functions of Randomly Generated MDPs

In this subsection, we show the phenomena that the $Q$-function is more complex does not only occurs in the crafted cases as in the previous subsection, but also occurs more robustly with a decent chance for (semi-) randomly generated MDPs. (Mathematically, this says that the family of MDPs with such a property is not a degenerate measure-zero set.)

It is challenging and perhaps requires deep math to characterize the fractal structure of $Q$-functions for random dynamics, which is beyond the scope of this chapter. Instead, we take an empirical approach here. We generate random piecewise linear and Lipschitz dynamics, and compute their $Q$-functions for the finite horizon, and then visualize the $Q$-functions or count the number of pieces in the $Q$-functions. We also use DQN algorithm [Mnih et al., 2015] with a finite-size neural network to learn the $Q$-function.

We set horizon $H = 10$ for simplicity and computational feasibility. The state and action space are $[0, 1)$ and $\{0, 1\}$ respectively. We design two methods to generate random or semi-random piecewise dynamics with at most four pieces.

First, we have a uniformly random method, called RAND, where we independently generate two piecewise linear functions for $f(s, 0)$ and $f(s, 1)$, by generating random positions for the kinks, generating random outputs for the kinks, and connecting the

kinks by linear lines. In this method, the generated MDPs are with less structure. The details are shown as follows.

- State space $\mathcal{S} = [0, 1)$.
- Action space $\mathcal{A} = \{0, 1\}$.
- Number of pieces is fixed to 3. The positions of the kinks are generated by, $x_i \sim U(0, 1)$ for $i = 1, 2$ and $x_0 = 0, x_1 = 1$. The values are generated by $x_i' \sim U(0, 1)$.
- The reward function is given by $r(s, a) = s,\ \forall s \in \mathcal{S}, a \in \mathcal{A}$.
- The horizon is fixed as $H = 10$.
- Initial state distribution is $U(0, 1)$.

Figure 3.1 visualizes one of the RAND-generated MDPs with complex Q-functions.

In the second method, called SEMI-RAND, we introduce a bit more structure in the generation process, towards increasing the chance to see the phenomenon. The functions $f(s, 0)$ and $f(s, 1)$ have 3 pieces with shared kinks. We also design the generating process of the outputs at the kinks so that the functions have more fluctuations. In this method, we add some structures to the dynamics, resulting in a more significant probability that the optimal policy is complex. We generate dynamics with fix and shared kinks, generate the output at the kinks to make the functions fluctuating. The details are shown as follows.

- State space $\mathcal{S} = [0, 1)$.
- Action space $\mathcal{A} = \{0, 1\}$.
- Number of pieces is fixed to 3. The positions of the kinks are generated by, $x_i = i/3,\ \forall 0 \leq i \leq 3$. And the values are generated by $x_i' \sim 0.65 \times \mathbb{I}[i \mod 2 = 0] + 0.35 \times U(0, 1)$.

- The reward function is $r(s, a) = s$ for all $a \in \mathcal{A}$.
- The horizon is fixed as $H = 10$.
- Initial state distribution is $U(0, 1)$.

Figure 3.1 visualizes one of the MDPs generated by SEMI-RAND method.

**The optimal policy and $Q$ can have a large number of pieces.** Because the state space has one dimension, and the horizon is 10, we can compute the exact $Q$-functions by recursively applying Bellman operators, and count the number of pieces. We found that, 8.6% fraction of the 1000 MDPs independently generated from the RAND method has policies with more than 100 pieces, much larger than the number of pieces in the dynamics (which is 4). Using the SEMI-RAND method, a 68.7% fraction of the MDPs has polices with more than $10^3$ pieces. We also plot the histogram of the number of pieces of the $Q$-functions. Figure 3.1 visualize the $Q$-functions and dynamics of two MDPs generated from RAND and SEMI-RAND method. These results suggest that the phenomenon that $Q$-function is more complex than dynamics is not a degenerate phenomenon and can occur with non-zero measure. As shown in Figure 3.3, even for horizon $H = 10$, the optimal policy tends to have much more pieces than the dynamics.

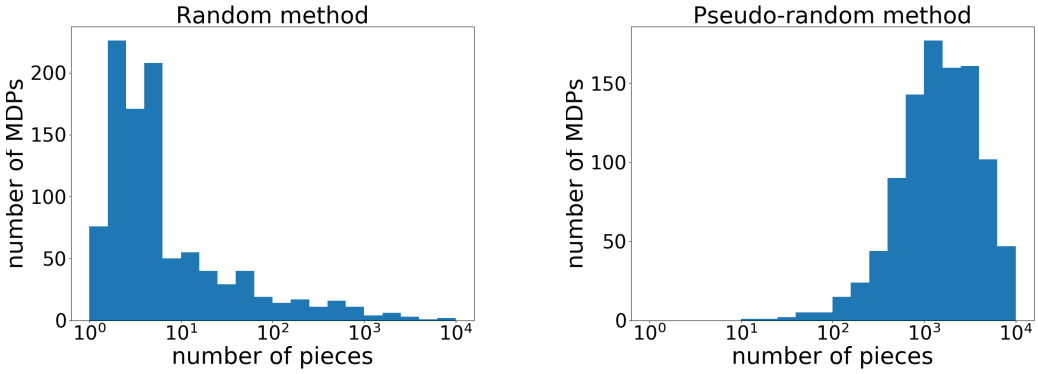



Figure 3.3: The histogram of number of pieces in optimal policy $\pi^\star$ in random method (left) and semi-random method(right).

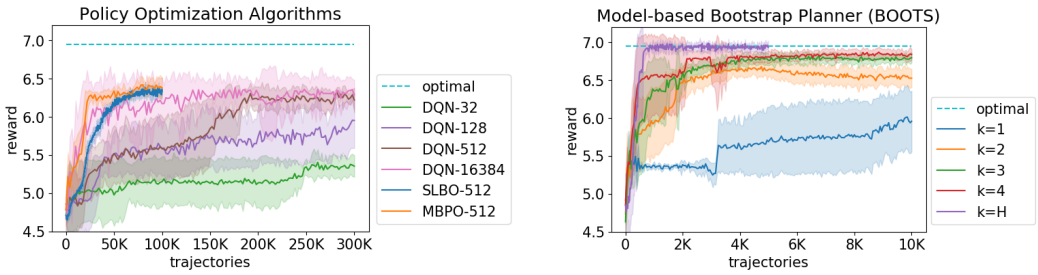



Figure 3.4: **(Left):** The performance of DQN, SLBO, and MBPO on the bottom dynamics in Figure 3.1. The number after the acronym is the width of the neural network used in the parameterization of $Q$. We see that even with sufficiently large neural networks and sufficiently many steps, these algorithms still suffers from bad approximability and cannot achieve optimal reward. **(Right):** Performance of BOOTS + DQN with various planning steps. A near-optimal reward is achieved with even $k = 3$, indicating that the bootstrapping with the learned dynamics improves the expressivity of the policy significantly.

**Model-based policy optimization methods also suffer from a lack of expressivity.** As an implication of our theory in the previous section, when the $Q$-function or the policy are too complex to be approximated by a reasonable size neural network, both model-free algorithms or model-based policy optimization algorithms will suffer from the lack of expressivity, and as a consequence, the sub-optimal rewards. We verify this claim on the randomly generated MDPs discussed in Section 3.4.5, by running DQN [Mnih et al., 2015], SLBO [Luo et al., 2019a], and MBPO [Janner et al., 2019] with various architecture size.

For the ease of exposition, we use the MDP visualized in the bottom half of Figure 3.1. The optimal policy for this specific MDP has 765 pieces, and the optimal $Q$-function has about $4 \times 10^4$ number of pieces, and we can compute the optimal total rewards.

First, we apply DQN to this environment by using a two-layer neural network with various widths to parameterize the $Q$-function. The training curve is shown in Figure 3.4 (Left). Model-free algorithms can not find near-optimal policy even with $2^{14}$ hidden neurons and 1M trajectories, which suggests that there is a fundamental

**Algorithm 5** Model-based Bootstrapping Planner (BOOTS) + RL Algorithm X

**training:** run Algorithm X, store the all samples in the set $R$, store the learned $Q$-function $Q$, and the learned dynamics $\widehat{M}$ if it is available in Algorithm X.
**testing:**
    if $\widehat{M}$ is not available, learn $\widehat{M}$ from the data in $R$
    execute the policy BOOTS(s) at every state $s$

**function** BOOTS(s)
    **Given:** query oracle for function $Q$ and $\widehat{M}$
    Compute

$$\pi^{\text{boots}}_{k,Q,\widehat{M}}(s) = \operatorname*{argmax}_{a} \max_{a_1,\ldots,a_k} r(s,a) + \cdots + \gamma^{k-1} r(s_{k-1}, a_{k-1}) + \gamma^k Q(s_k, a_k)$$

    using a zero-th order optimization algorithm (which only requires oracle query of the function value) such as cross-entropy method or random shooting.

approximation issue. This result is consistent with Fu et al. [2019], in a sense that enlarging Q-network improves the performance of DQN algorithm at convergence.

Second, we apply SLBO and MBPO in the same environment. Because the policy network and $Q$-function in SLOBO and MBPO cannot approximate the optimal policy and value function, we see that they fail to achieve near-optimal rewards, as shown in Figure 3.4 (Left).

## 3.5 Model-based Bootstrapping Planner

Our theory and experiments in Sections 3.4.3 and 3.4.5 demonstrate that when the $Q$-function or the policy is complex, model-free or model-based policy optimization algorithms will suffer from the lack of expressivity. The intuition suggests that model-based planning algorithms will not suffer from the lack of expressivity because the final policy is not represented by a neural network. For the construction in Section 3.4.1, we can actually prove that even a few-steps planner can bootstrap the expressivity of the $Q$-function (formalized in Theorem 3.5.1 below).

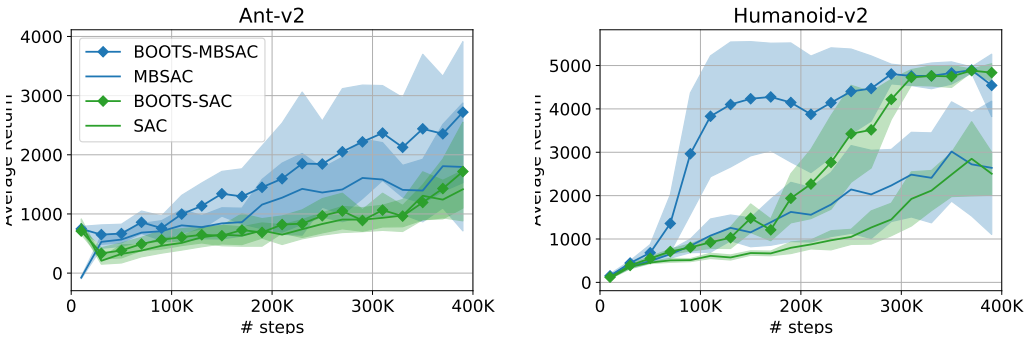



Figure 3.5: Comparison of BOOTS-MBSAC vs MBSAC and BOOTS-SAC vs SAC on Ant and Humanoid. Particularly on the Humanoid environment, BOOTS improves the performance significantly. The test policies for MBSAC and SAC are the deterministic policy that takes the mean of the output of the policy network, because the deterministic policy performs better than the stochastic policy in the test time.

Inspired the theoretical result, we apply a simple $k$-step model-based bootstrapping planner on top of existing $Q$-functions (trained from either model-based or model-free approach) *in the test time*, on either the one-dimensional MDPs considered in Section 3.4 or the continuous control benchmark tasks in MuJoCo. The bootstrapping planner is reminiscent of MCTS using in AlphaGo [Silver et al., 2016, 2018]. However, here, we use the learned dynamics and deal with the continuous state space.

Given a function $Q$ that is potentially not expressive enough to approximate the optimal $Q$-function, we can apply the Bellman operator with a learned dynamics $\widehat{M}$ for $k$ times to get a bootstrapped version of $Q$:

$$\mathcal{B}^k_{\widehat{M}}[Q] \triangleq \underbrace{\mathcal{B}_{\widehat{M}}[\cdots[\mathcal{B}_{\widehat{M}}[Q]]]}_{k \text{ times}} \tag{3.5.1}$$

$$\text{or} \qquad \mathcal{B}^k_{\widehat{M}}[Q](s,a) \triangleq \max_{a_1,\cdots,a_k} r(s_0,a_0) + \cdots + \gamma^{k-1} r(s_{k-1},a_{k-1}) + \gamma^k Q(s_k,a_k) \tag{3.5.2}$$

where $s_0 = s, a_0 = a$ and $s_{h+1} \sim \widehat{M}(s_h, a_h)$.

Given the bootstrapped version, we can derive a greedy policy w.r.t it:

$$\pi^{\text{boots}}_{k,Q,\widehat{M}}(s) \triangleq \max_a \mathcal{B}^k_{\widehat{M}}[Q](s,a). \tag{3.5.3}$$

Algorithm 5, called BOOTS summarizes how to apply the planner on top of any RL algorithm with a $Q$-function (straightforwardly).

For the MDPs constructed in Section 3.4.1, we can prove that representing the optimal $Q$-function by $\mathcal{B}^k_{\widehat{M}}[Q]$ requires fewer pieces in $Q$ than representing the optimal $Q$-function by $Q$ directly.

**Theorem 3.5.1.** *Consider the MDP $M_H$ defined in Definition 3.4.1. There exists a constant-piece piecewise linear dynamics $\widehat{M}$ and a $2^{H-k+1}$-piece piecewise linear function $Q$, such that the bootstrapped policy $\pi^{\text{boots}}_{k,Q,\widehat{M}}(s)$ achieves the optimal total rewards.*

By contrast, recall that in Theorem 3.4.8, we show that approximating the optimal $Q$-function directly with a piecewise linear function requires $\approx 2^H$ piecewise. Thus we have a multiplicative factor of $2^k$ gain in the expressivity by using the bootstrapped policy. Here the exponential gain is only magnificent enough when $k$ is close to $H$ because the gap of approximability is huge. However, in more realistic settings — the randomly-generated MDPs and the MuJoCo environment — the bootstrapping planner improves the performance significantly as shown in the next subsection.

*Proof of Theorem 3.5.1.* First we define the true trajectory estimator

$$\eta(s_0, a_0, a_1, \cdots, a_k) \triangleq \sum_{j=0}^{k-1} \gamma^j r(s_j, a_j) + \gamma^k Q^\star(s_k, a_k),$$

the true optimal action sequence

$$a_0^\star, a_1^\star, \cdots, a_k^\star \triangleq \arg\max_{a_0,a_1,\cdots,a_k} \eta(s_0, a_0, a_1, \cdots, a_k),$$

and the true optimal trajectory

$$s_0^\star = s_0,\ s_j^\star = M(s_{j-1}^\star, a_{j-1}^\star), \forall j > 1.$$

It follows from the definition of optimal policy that, $a_j^\star = \pi^\star(s_j)$. Consequently we have

$$s_k{}^{(H-k+1)} = s_k{}^{(H-k+2)} = \cdots = s_k{}^{(H)} = 1.$$

Define the set $G \triangleq \{s : s^{(H-k+1)} = s^{(H-k+2)} = \cdots = s^{(H)} = 1\}$. We claim that the following function satisfies the statement of Theorem 3.5.1

$$Q(s,a) = \mathbb{I}[s \in G] \cdot \frac{2}{1-\gamma}.$$

Since $s_k^\star \in G$, and $s_k \notin G$ for $s_k$ generated by non-optimal action sequence, we have

$$Q(s_k^\star, a) >\ Q^\star(s_k^\star, a) \geq\ Q^\star(s_k, a) >\ Q(s_k, a),$$

where the second inequality comes from the optimality of action sequence $a_h^\star$. As a consequence, for any $(a_0, a_1, \cdots, a_k) \neq (a_0^\star, a_1^\star, \cdots, a_k^\star)$

$$\begin{aligned}
\hat{\eta}(s_0, a_0^\star, a_1^\star, \cdots, a_k^\star) &> \eta(s_0, a_0^\star, a_1^\star, \cdots, a_k^\star) \\
&\geq \eta(s_0, a_0, a_1, \cdots, a_k) \\
&> \hat{\eta}(s_0, a_0, a_1, \cdots, a_k).
\end{aligned}$$

Therefore, $(\hat{a}_0^\star, \hat{a}_1^\star, \cdots, \hat{a}_k^\star) = (a_0^\star, a_1^\star, \cdots, a_k^\star)$. □

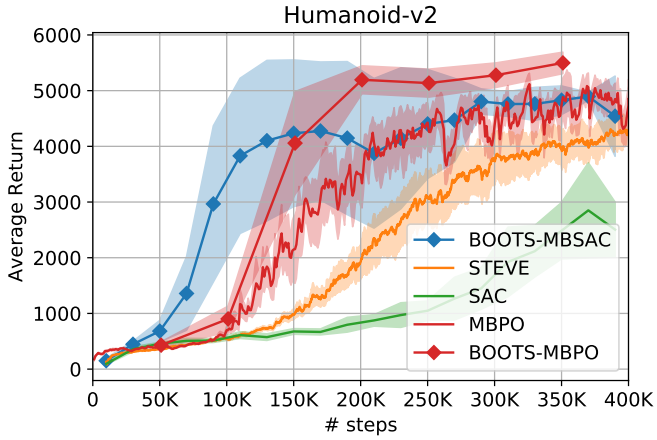



Figure 3.6: BOOTS-MBSAC or BOOTS-MBPO outperforms previous high-performance algorithms on Humanoid. The results are averaged over 5 random seeds and shadow area indicates a single standard deviation from the mean.

## 3.6 Experiments

**BOOTS on random piecewise linear MDPs.** We implement BOOTS (Algorithm 5) with various steps of planning and with the learned dynamics. The planner is an exponential-time planner which enumerates all the possible future sequence of actions. We also implement bootstrapping with partial planner with varying planning horizon. As shown in Figure 3.4, BOOTS + DQN not only has the best sample-efficiency, but also achieves the optimal reward. In the meantime, even a partial planner helps to improve both the sample-efficiency and performance. More details of this experiment are deferred to Section 3.6.2.

**BOOTS on MuJoCo environments.** We work with the OpenAI Gym environments [Brockman et al., 2016] based on the Mujoco simulator [Todorov et al., 2012b] with maximum horizon 1000 and discount factor 1. We apply BOOTS on top of three algorithms: (a) **SAC** [Haarnoja et al., 2018], a model-free RL algorithm; (b) **MBPO** [Janner et al., 2019], a model-based Q-learning algorithm, and an extension of Dyna [Sutton, 1990a]; (c) a computationally efficient variant of MBPO that we develop using ideas from SLBO [Luo et al., 2019a], which is called **MBSAC**. The main

difference here from MBPO and other works such as [Wang and Ba, 2019, Kurutach et al., 2018] is that we don't use model ensemble. Instead, we occasionally optimize the dynamics by one step of Adam to introduce stochasticity in the dynamics, following the technique in SLBO [Luo et al., 2019a]. Our algorithm is a few times faster than MBPO in wall-clock time. It performs similarly to MBPO on Humanoid, but generally worse than MBPO on other environments. See Section 3.6.2 for details.

We use $k = 4$ steps of planning unless explicitly mentioned otherwise in the ablation study (Section 3.6.1). In Figure 3.5, we compare BOOTS+SAC with SAC, and BOOTS + MBSAC with MBSAC on Gym Ant and Humanoid environments, and demonstrate that BOOTS can be used on top of existing strong baselines. We found that BOOTS has little help for other simpler environments, and we suspect that those environments have much less complex $Q$-functions so that our theory and intuitions do not necessarily apply.

In Figure 3.6, we compare BOOTS+MBSAC and BOOTS+MBPO with other MBPO, SAC, and STEVE [Buckman et al., 2018][4] on the Humanoid environment.

### 3.6.1 Ablation Study

**Planning with oracle dynamics and more environments.** We found that BOOTS has smaller improvements on top of MBSAC and SAC for the environment Cheetah and Walker. To diagnose the issue, we also plan with an oracle dynamics (the true dynamics). This tells us whether the lack of improvement comes from inaccurate learned dynamics. The results are presented in two ways in Figure 3.7 and Figure 3.8. In Figure 3.7, we plot the mean rewards and the standard deviation of various methods across the randomness of multiple seeds. However, the randomness from the seeds somewhat obscures the gains of BOOTS on each individual run. Therefore, for

[4]For STEVE, we use the official code at `https://github.com/tensorflow/models/tree/master/research/steve`

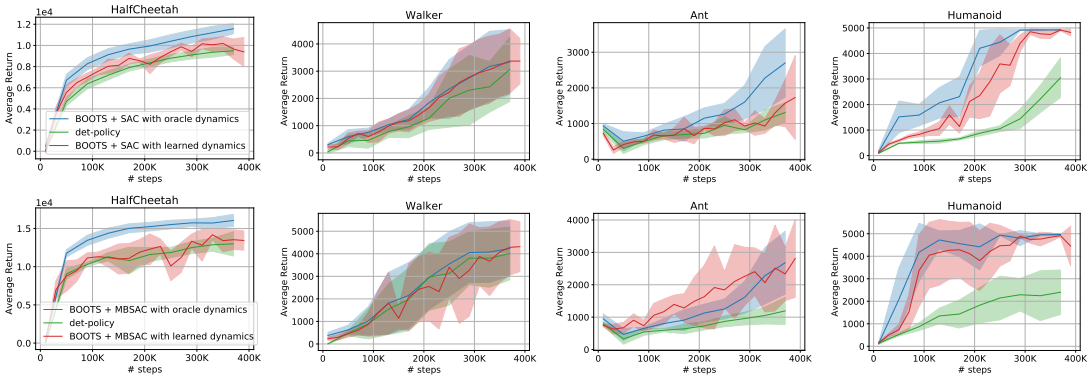



Figure 3.7: BOOTS with oracle dynamics on top of SAC (top) and MBSAC (bottom) on HalfCheetah, Walker, Ant and Humanoid. The solid lines are average over 5 runs, and the shadow areas indicate the standard deviation.

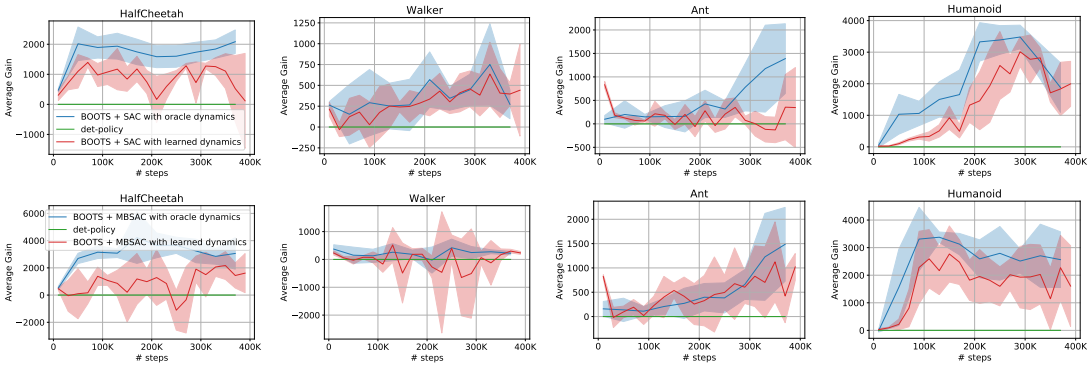



Figure 3.8: The relative gains of BOOTS over SAC (top) and MBSAC (bottom) on HalfCheetah, Walker, Ant and Humanoid. The solid lines are average over 5 runs, and the shadow areas indicate the standard deviation.

completeness, we also plot the relative gain of BOOTS on top of MBSAC and SAC, and the standard deviation of the gains in Figure 3.8.

From Figure 3.8 we can see planning with the oracle dynamics improves the performance in most of the cases (but with various amount of improvements). However, the learned dynamics sometimes not always can give an improvement similar to the oracle dynamics. This suggests the learned dynamics is not perfect, but oftentimes can lead to good planning. This suggests the expressivity of the $Q$-functions varies depending on the particular environment. How and when to learn and use a learned dynamics for planning is a very interesting future open question.

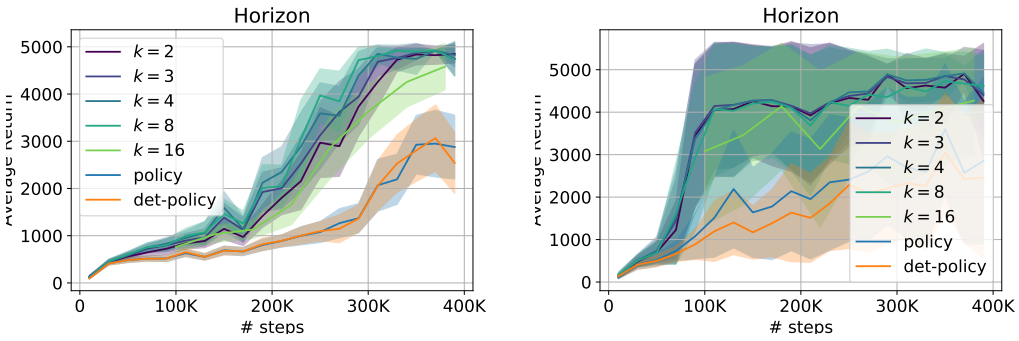



Figure 3.9: Different BOOTS planning horizon $k$ on top of SAC (left) and MBSAC (right) on Humanoid. The solid lines are average over 5 runs, and the shadow areas indicate the standard deviation.

**The effect of planning horizon.** We experimented with different planning horizons in Figure 3.9. By planning with a longer horizon, we can earn slightly higher total rewards for both MBSAC and SAC. Planning horizon $k = 16$, however, does not work well. We suspect that it's caused by the compounding effect of the errors in the dynamics.

### 3.6.2 Implementation Details

**SEMI-RAND MDP.** The MDP where we run the experiment is given by the SEMI-RAND method, described in Section 3.4.5. We list the dynamics of this MDP in the following.

$$r(s,a) = s, \quad \forall s \in \mathcal{S}, a \in \mathcal{A},$$

$$M(s,0) = \begin{cases} (0.131 - 0.690) \cdot x/0.333 + 0.690, & 0 \le x < 0.333, \\ (0.907 - 0.131) \cdot (x - 0.333)/0.334 + 0.131, & 0.333 \le x < 0.667, \\ (0.079 - 0.907) \cdot (x - 0.667)/0.333 + 0.907, & 0.667 \le x, \end{cases}$$

$$M(s,1) = \begin{cases} (0.134 - 0.865) \cdot x/0.333 + 0.865, & 0 \le x < 0.333, \\ (0.750 - 0.134) \cdot (x - 0.333)/0.334 + 0.134, & 0.333 \le x < 0.667, \\ (0.053 - 0.750) \cdot (x - 0.667)/0.333 + 0.750, & 0.667 \le x, \end{cases}$$

**Implementation details of DQN algorithm.** We present the hyper-parameters of DQN algorithm. Our implementation is based on PyTorch tutorials[5].

- The Q-network is a fully connected neural net with one hidden-layer. The width of the hidden-layer is varying.
- The optimizer is SGD with learning rate 0.001 and momentum 0.9.
- The size of replay buffer is $10^4$.
- Target-net update frequency is 50.
- Batch size in policy optimization is 128.
- The behavior policy is greedy policy according to the current Q-network with $\epsilon$-greedy. $\epsilon$ exponentially decays from 0.9 to 0.01. Specifically, $\epsilon = 0.01 + 0.89\exp(-t/200)$ at the $t$-th episode.

**Implementation details of MBPO algorithm.** For the model-learning step, we use $\ell_2$ loss to train our model, and we use Soft Actor-Critic (SAC) [Haarnoja et al., 2018] in the policy optimization step. The parameters are set as,

- number of hidden neurons in model-net: 32,
- number of hidden neurons in value-net: 512,
- optimizer for model-learning: Adam with learning rate 0.001.

[5] `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`

- temperature: $\tau = 0.01$,
- the model rollout steps: $M = 5$,
- the length of the rollout: $k = 5$,
- number of policy optimization step: $G = 5$.

Other hyper-parameters are kept the same as DQN algorithm.

**Implementation details of TRPO algorithm.** For the model-learning step, we use $\ell_2$ loss to train our model. Instead of TRPO [Schulman et al., 2015a], we use PPO [Schulman et al., 2017] as policy optimizer. The parameters are set as,

- number of hidden neurons in model-net: 32,
- number of hidden neurons in policy-net: 512,
- number of hidden neurons in value-net: 512,
- optimizer: Adam with learning rate 0.001,
- number of policy optimization step: 5.
- The behavior policy is $\epsilon$-greedy policy according to the current policy network. $\epsilon$ exponential decays from 0.9 to 0.01. Specifically, $\epsilon = 0.01 + 0.89 \exp(-t/20000)$ at the $t$-th episode.

**Implementation details of Model-based Planning algorithm.** The perfect model-based planning algorithm iterates between learning the dynamics from sampled trajectories, and planning with the learned dynamics (with an exponential time algorithm which enumerates all the possible future sequence of actions). The parameters are set as,

- number of hidden neurons in model-net: 32,

- optimizer for model-learning: Adam with learning rate 0.001.

**Implementation details of bootstrapping.** The training time behavior of the algorithm is exactly like DQN algorithm, except that the number of hidden neurons in the Q-net is set to 64. Other parameters are set as,

- number of hidden neurons in model-net: 32,
- optimizer for model-learning: Adam with learning rate 0.001.
- planning horizon varies.

**Model-based SAC (MBSAC).** Here we describe our MBSAC algorithm in Algorithm 6, which is a model-based policy optimization and is used in BOOTS-MBSAC. As mentioned in Section 3.6, the main difference from MBPO and other works such as Wang and Ba [2019], Kurutach et al. [2018] is that we don't use model ensemble. Instead, we occasionally optimize the dynamics by one step of Adam to introduce stochasticity in the dynamics, following the technique in SLBO [Luo et al., 2019a]. As argued in [Luo et al., 2019a], the stochasticity in the dynamics can play a similar role as the model ensemble. Our algorithm is a few times faster than MBPO in wall-clock time. It performs similarlty to MBPO on Humanoid, but a bit worse than MBPO in other environments. In MBSAC, we use SAC to optimize the policy $\pi_\beta$ and the $Q$-function $Q_\varphi$. We choose SAC due to its sample-efficiency, simplicity and off-policy nature. We mix the real data from the environment and the virtual data which are always fresh and are generated by our learned dynamics model $\widehat{M}_\theta$.[6]

For Ant, we modify the environment by adding the $x$ and $y$ axis to the observation space to make it possible to compute the reward from observations and actions. For Humanoid, we add the position of center of mass. We don't have any other modifications. All environments have maximum horizon 1000.

[6]In the chapter of MBPO [Janner et al., 2019], the authors don't explicitly state their usage of real data in SAC; the released code seems to make such use of real data, though.

**Algorithm 6** MBSAC

Parameterize the policy $\pi_\beta$, dynamics $\widehat{M}_\theta$, and the $Q$-function $Q_\varphi$ by neural networks. Initialize replay buffer $\mathcal{B}$ with $n_{\text{init}}$ steps of interactions with the environments by a random policy, and pretrain the dynamics on the data in the replay buffer.
$t \leftarrow 0$, and sample $s_0$ from the initial state distribution.
**for** $n_{\text{iter}}$ iterations **do**
    Perform action $a_t \sim \pi_\beta(\cdot|s_t)$ in the environment, obtain $s'$ as the next state from the environment.
    $s_{t+1} \leftarrow s'$, and add the transition $(s_t, a_t, s_{t+1}, r_t)$ to $\mathcal{B}$.
    $t \leftarrow t+1$. If $t = T$ or the trajectory is done, reset to $t = 0$ and sample $s_0$ from the initial state distribution.
    **for** $n_{\text{policy}}$ iterations **do**
        **for** $n_{\text{model}}$ iterations **do**
            Optimize $\widehat{M}_\theta$ with a mini-batch of data from $\mathcal{B}$ by one step of Adam.
        Sample $n_{\text{real}}$ data $\mathcal{B}_{\text{real}}$ and $n_{\text{start}}$ data $\mathcal{B}_{\text{start}}$ from $\mathcal{B}$.
        Perform $q$ steps of **virtual** rollouts using $\widehat{M}_\theta$ and policy $\pi_\beta$ starting from states in $\mathcal{B}_{\text{start}}$; obtain $\mathcal{B}_{\text{virtual}}$.
        Update $\pi_\beta$ and $Q_\varphi$ using the mini-batch of data in $\mathcal{B}_{\text{real}} \cup \mathcal{B}_{\text{virtual}}$ by SAC.

For the policy network, we use an MLP with ReLU activation function and two hidden layers, each of which contains 256 hidden units. For the dynamics model, we use a network with 2 Fixup blocks [Zhang et al., 2019], with convolution layers replaced by a fully connected layer. We found out that with similar number of parameters, fixup blocks leads to a more accurate model in terms of validation loss. Each fixup block has 500 hidden units. We follow the model training algorithm in Luo et al. [2019a] in which non-squared $\ell_2$ loss is used instead of the standard MSE loss.

## 3.7 Conclusion

Our study suggests that there exists a significant representation power gap of neural networks between for expressing $Q$-function, the policy, and the dynamics in both constructed examples and empirical benchmarking environments. We show that our model-based bootstrapping planner BOOTS helps to overcome the approximation

issue and improves the performance in synthetic settings and in the difficult MuJoCo environments. We raise some interesting open questions.

- Can we theoretically generalize our results to high-dimensional state space, or continuous actions space? Can we theoretically analyze the number of pieces of the optimal $Q$-function of a stochastic dynamics?

- In this chapter, we measure the complexity by the size of the neural networks. It's conceivable that for real-life problems, the complexity of a neural network can be better measured by its weights norm. Could we build a more realistic theory with another measure of complexity?

- The BOOTS planner comes with a cost of longer test time. How do we efficiently plan in high-dimensional dynamics with a long planning horizon?

- The dynamics can also be more complex (perhaps in another sense) than the $Q$-function in certain cases. How do we efficiently identify the complexity of the optimal $Q$-function, policy, and the dynamics, and how do we deploy the best algorithms for problems with different characteristics?

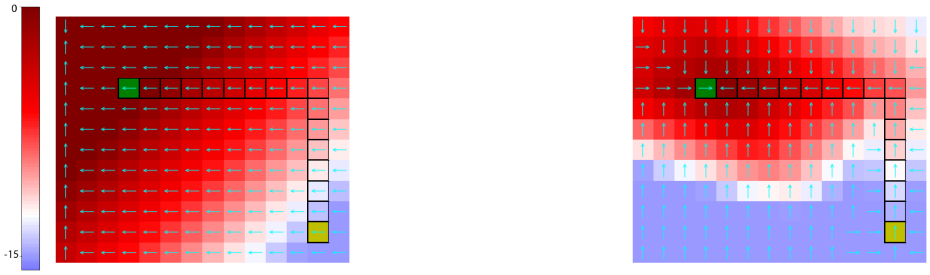
# Chapter 4

# Imitation Learning via Negative Sampling

Imitation learning, followed by reinforcement learning algorithms, is a promising paradigm to solve complex control tasks sample-efficiently. However, learning from demonstrations often suffers from the covariate shift problem, which results in cascading errors of the learned policy. We introduce a notion of conservatively-extrapolated value functions, which provably lead to policies with self-correction. We design an algorithm Value Iteration with Negative Sampling (VINS) that practically learns such value functions with conservative extrapolation. We show that VINS can correct mistakes of the behavioral cloning policy on simulated robotics benchmark tasks. We also propose the algorithm of using VINS to initialize a reinforcement learning algorithm, which is shown to outperform prior works in sample efficiency.

## 4.1 Background

Reinforcement learning (RL) algorithms, especially with sparse rewards, often require a large amount of trial-and-errors. Imitation learning from a small number of demonstrations followed by RL fine-tuning is a promising paradigm to improve the sample




(a) The value function learned from the standard Bellman equation (or supervised learning) on the demonstration states. The value function falsely extrapolates to the unseen states. For example, the top left corner has erroneously the largest value. As a result, once the policy induced by the value function makes a mistake, the error will compound.

(b) The conservatively-extrapolated value function (defined in Equation (4.4.2)) learned with negative sampling (VINS, Algorithm 8 in Section 4.5). The values at unseen states tend to be lower than their nearby states in the demonstrations, and therefore the corresponding policy tend to correct itself towards the demonstration trajectories.

Figure 4.1: A toy environment where the agent aims to walk from a starting state (the yellow entry) to a goal state (the green entry). The reward is sparse: $R(s, a) = -1$ unless $s$ is at the goal (which is also the terminal state.) The colors of the entries show the learned value functions. Entries in black edges are states in demonstrations. The cyan arrows show the best actions according to the value functions.

efficiency [Rajeswaran et al., 2017, Večerík et al., 2017, Hester et al., 2018, Nair et al., 2018, Gao et al., 2018].

The key technical challenge of learning from demonstrations is the covariate shift: the distribution of the states visited by the demonstrations often has a low-dimensional support; however, knowledge learned from this distribution may not necessarily transfer to other distributions of interests. This phenomenon applies to both learning the policy and the value function. The policy learned from behavioral cloning has compounding errors after we execute the policy for multiple steps and reach unseen states [Bagnell, 2015, Ross and Bagnell, 2010]. The value function learned from the demonstrations can also extrapolate falsely to unseen states. See Figure 4.1a for an illustration of the false extrapolation in a toy environment.

We develop an algorithm that learns a value function that extrapolates to unseen states more conservatively, as an approach to attack the optimistic extrapolation problem [Fujimoto et al., 2018a]. Consider a state $s$ in the demonstration and its nearby state $\tilde{s}$ that is not in the demonstration. The key intuition is that $\tilde{s}$ should have a lower value than $s$, because otherwise $\tilde{s}$ likely should have been visited by the demonstrations in the first place. If a value function has this property for most of the pair $(s, \tilde{s})$ of this type, the corresponding policy will tend to correct its errors by driving back to the demonstration states because the demonstration states have locally higher values. We formalize the intuition in Section 4.4 by defining the so-called conservatively-extrapolated value function, which is guaranteed to induce a policy that stays close to the demonstrations states (Theorem 4.4.4).

In Section 4.5, we design a practical algorithm for learning the conservatively-extrapolated value function by a negative sampling technique inspired by work on learning embeddings [Mikolov et al., 2013, Gutmann and Hyvärinen, 2012]. We also learn a dynamics model by standard supervised learning so that we compute actions by maximizing the values of the predicted next states. This algorithm does not use any additional environment interactions, and we show that it empirically helps correct errors of the behavioral cloning policy.

When additional environment interactions are available, we use the learned value function and the dynamics model to initialize an RL algorithm. This approach relieves the inefficiency in the prior work [Hester et al., 2018, Nair et al., 2018, Rajeswaran et al., 2017] that the randomly-initialized $Q$ functions require a significant amount of time and samples to be warmed up, even though the initial policy already has a non-trivial success rate. Empirically, the proposed algorithm outperforms the prior work in the number of environment interactions needed to achieve near-optimal success rate.

In summary, our main contributions are:

1. We formalize the notion of values functions with conservative extrapolation which are proved to induce policies that stay close to demonstration states and achieve near-optimal performances.

2. We propose the algorithm Value Iteration with Negative Sampling (VINS) that outperforms behavioral cloning on three simulated robotics benchmark tasks with sparse rewards.

3. We show that initializing an RL algorithm from VINS outperforms prior work in sample efficiency on the same set of benchmark tasks.

## 4.2 Related Work

**Imitation learning.** Imitation learning is commonly adopted as a standard approach in robotics [Pomerleau, 1989, Schaal, 1997, Argall et al., 2009, Osa et al., 2017, Ye and Alterovitz, 2017, Aleotti and Caselli, 2006, Lawitzky et al., 2012, Torabi et al., 2018, Le et al., 2017, 2018] and many other areas such as playing games [Mnih et al., 2013]. Behavioral cloning [Bain and Sommut, 1999] is one of the underlying central approaches. See Osa et al. [2018] for a thorough survey and more references therein. If we are allowed to access an expert policy (instead of trajectories) or an approximate value function, in the training time or in the phase of collecting demonstrations, then, stronger algorithms can be designed, such as DAgger [Ross et al., 2011], AggreVaTe [Ross and Bagnell, 2014], AggreVaTeD [Sun et al., 2017], DART [Laskey et al., 2017], THOR [Sun et al., 2018a]. Our setting is that we have only clean demonstrations trajectories and a sparse reward (but we still hope to learn the self-correctable policy.)

Ho and Ermon [2016], Wang et al. [2017], Schroecker et al. [2018] successfully combine generative models in the setting where a large amount of environment interaction without rewards are allowed. The sample efficiency of Ho and Ermon

[2016] has been improved in various ways, including maximum mean discrepancy minimization [Kim and Park, 2018], a Bayesian formulation of GAIL [Jeon et al., 2018], using an off-policy RL algorithm and solving reward bias problem [Kostrikov et al., 2018], and bypassing the learning of reward function [Sasaki et al., 2018]. By contrast, we would like to minimize the amount of environment interactions needed, but are allowed to access a sparse reward. The work of Schroecker and Isbell [2017] also aims to learn policies that can stay close to the demonstration sets, but through a uite different approach of estimating the true MAP estimate of the policy. The algorithm also requires environment interactions, whereas one of our main goals is to improve upon behavioral cloning without any environment interactions.

Inverse reinforcement learning (e.g., see [Abbeel and Ng, 2004, Ng et al., 2000, Ziebart et al., 2008, Finn et al., 2016a,b, Fu et al., 2017]) is another important and successful line of ideas for imitation learning. It relates to our approach in the sense that it aims to learn a reward function that the expert is optimizing. In contrast, we construct a model to learn the value function (of the trivial sparse reward $R(s, a) = -1$), rather than the reward function. Some of these works (e.g., Finn et al. [2016a,b], Fu et al. [2017]) use techniques that are reminiscent of negative sampling or contrastive learning, although unlike our methods, they use "negative samples" that are sampled from the environments.

**Leveraging demonstrations for sample-efficient reinforcement learning.** Demonstrations have been widely used to improve the efficiency of RL [Kim et al., 2013, Chemali and Lazaric, 2015, Piot et al., 2014, Sasaki et al., 2018], and a common paradigm for continuous state and action space is to initialize with RL algorithms with a good policy or $Q$ function [Rajeswaran et al., 2017, Nair et al., 2018, Večerík et al., 2017, Hester et al., 2018, Gao et al., 2018]. We experimentally compare with the algorithm in Nair et al. [2018] on the same type of tasks. Gao et al. [2018]

has introduced soft version of actor-critic to tackle the false extrapolation of $Q$ in the argument of $a$ when the action space is discrete. In contrast, we deal with the extrapolation of the states in a continuous state and action space.

**Model-based reinforcement learning.** Even though we will learn a dynamics model in our algorithms, we do not use it to generate fictitious samples for planning. Instead, the learned dynamics are only used in combination with the value function to get a $Q$ function. Therefore, we do not consider our algorithm as model-based techniques. We refer to Kurutach et al. [2018], Clavera et al. [2018], Sun et al. [2018b], Chua et al. [2018b], Sanchez-Gonzalez et al. [2018], Pascanu et al. [2017], Khansari-Zadeh and Billard [2011], Luo et al. [2019a] and the reference therein for recent work on model-based RL.

**Off-policy reinforcement learning** There is a large body of prior works in the domain of off-policy RL, including extensions of policy gradient [Gu et al., 2016a, Degris et al., 2012, Wang et al., 2016] or Q-learning [Watkins and Dayan, 1992, Haarnoja et al., 2018, Munos et al., 2016]. Fujimoto et al. [2018a] propose to solve off-policy reinforcement learning by constraining the action space, and Fujimoto et al. [2018b] use double Q-learning [Van Hasselt et al., 2016] to alleviate the optimistic extrapolation issue. In contrast, our method adjusts the erroneously extrapolated value function by explicitly penalizing the unseen states (which is customized to the particular demonstration off-policy data). For most of the off-policy methods, their convergence are based on the assumption of visiting each state-action pair sufficiently many times. In the learning from demonstration setting, the demonstrations states are highly biased or structured; thus off-policy method may not be able to learn much from the demonstrations.

## 4.3 Problem Setup and Challenges

We consider the setting in Section 1.3. At test time, a random initial state $s_0$ is generated from some distribution $\mu_0$. We assume $\mu_0$ has a low-dimensional bounded support because typically initial states have special structures. We aim to find a policy $\pi$ such that executing $\pi$ from state $s_0$ will lead to a set of goal states $\mathcal{G}$. All the goal states are terminal states, and we run the policy for at most $T$ steps if none of the goal states is reached.

Let $\tau \triangleq (s_0, a_1, s_1, \ldots, )$ be the trajectory obtained by executing a deterministic policy $\pi$ from $s_0$, where $a_t = \pi(s_t)$, and $s_{t+1} = M^\star(s_t, a_t)$. The success rate of the policy $\pi$ is defined as

$$\mathrm{succ}(\pi) \triangleq \mathbb{E}\left[\mathbb{I}[\exists t \leq T, s_t \in \mathcal{G}]\right] \tag{4.3.1}$$

where the expectation is taken over the randomness of $s_0$. Note that the problem comes with a natural sparse reward: $R(s, a) = -1$ for every $s$ and $a$. This will encourage reaching the goal with as small number of steps as possible: the total payoff of a trajectory is equal to negative the number of steps if the trajectory succeeds, or $-T$ otherwise.

Let $\pi_{\mathrm{e}}$ be an expert policy [1] from which a set of $n$ demonstrations are sampled. Concretely, $n$ independent initial states $\{s_0^{(i)}\}_{i=1}^n$ from $D_{s_0}$ are generated, and the expert executes $\pi_{\mathrm{e}}$ to collect a set of $n$ trajectories $\{\tau^{(i)}\}_{i=1}^n$. We only have the access to the trajectories but not the expert policy itself.

We will design algorithms for two different settings:

[1] In this work, we only consider deterministic expert policies.

**Imitation learning without environment interactions.** The goal is to learn a policy $\pi$ from the demonstration trajectories $\{\tau^{(i)}\}_{i=1}^{n}$ without having any additional interactions with the environment.

**Leveraging demonstrations in reinforcement learning.** Here, in addition to the demonstrations, we can also interact with the environment (by sampling $s_0 \sim D_{s_0}$ and executing a policy) and observe if the trajectory reaches the goal. We aim is to minimize the amount of environment interactions by efficiently leveraging the demonstrations.

Let $\mathcal{U}$ be the set of states that can be visited by the demonstration policy from a random state $s_0$ with positive probability. Throughout this chapter, *we consider the situation where the set $\mathcal{U}$ is only a small subset or a low-dimensional manifold of the entire state space.* This is typical for continuous state space control problems in robotics, because the expert policy may only visit a very special kind of states that are the most efficient for reaching the goal. For example, in the toy example in Figure 4.1, the set $\mathcal{U}$ only contains those entries with black edges.[2]

To put our theoretical motivation in Section 4.4 into context, next we summarize a few challenges of imitation learning that are particularly caused by that $\mathcal{U}$ is only a small subset of the state space.

**Cascading errors for behavioral cloning.** As pointed out by Bagnell [2015], Ross and Bagnell [2010], the errors of the policy can compound into a long sequence of mistakes and in the worst case cascade quadratically in the number of time steps $T$. From a statistical point of view, the fundamental issue is that the distribution of the states that a learned policy may encounter is different from the demonstration state

[2]One may imagine that $\mathcal{U}$ can be a more diverse set if the demonstrations are more diverse, but an expert will not visit entries on the top or bottom few rows, because they are not on any optimal routes to the goal state.

distribution. Concretely, the behavioral cloning $\pi_{\mathrm{BC}}$ performs well on the states in $\mathcal{U}$ but not on those states far away from $\mathcal{U}$. However, small errors of the learned policy can drive the state to leave $\mathcal{U}$, and then the errors compound as we move further and further away from $\mathcal{U}$. As shown in Section 4.4, our key idea is to design policies that correct themselves to stay close to the set $\mathcal{U}$.

**Degeneracy in learning value or $Q$ functions from only demonstrations.** When $\mathcal{U}$ is a small subset or a low-dimensional manifold of the state space, off-policy evaluation of $V^{\pi_{\mathrm{e}}}$ and $Q^{\pi_{\mathrm{e}}}$ is fundamentally problematic in the following sense. The expert policy $\pi_{\mathrm{e}}$ is not uniquely defined outside $\mathcal{U}$ because any arbitrary extension of $\pi_{\mathrm{e}}$ outside $\mathcal{U}$ would not affect the performance of the expert policy (because those states outside $\mathcal{U}$ will never be visited by $\pi_{\mathrm{e}}$ from $s_0 \sim D_{s_0}$). As a result, the value function $V^{\pi_{\mathrm{e}}}$ and $Q^{\pi_{\mathrm{e}}}$ is not uniquely defined outside $\mathcal{U}$. In Section 4.4, we will propose a conservative extrapolation of the value function that encourages the policy to stay close to $\mathcal{U}$. Fitting $Q^{\pi_{\mathrm{e}}}$ is in fact even more problematic: there exists a function $Q(s, a)$ that does not depend on $a$ at all, which can still match $Q^{\pi_{\mathrm{e}}}$ on all possible demonstration data. Consider $Q(s, a) \triangleq Q^{\pi_{\mathrm{e}}}(s, \pi_{\mathrm{e}}(s))$. We can verify that for any $(s, a)$ pair in the demonstrations satisfying $a = \pi_{\mathrm{e}}(s)$, it holds that $Q(s, a) = Q^{\pi_{\mathrm{e}}}(s, a)$. However, $Q(s, a)$ cannot be accurate for other choices of $a$'s because by its definition, it does not use any information from the action $a$.

**Success and challenges of initializing RL with imitation learning.** A successful paradigm for sample-efficient RL is to initialize the RL policy by some coarse imitation learning algorithm such as BC [Rajeswaran et al., 2017, Večerík et al., 2017, Hester et al., 2018, Nair et al., 2018, Gao et al., 2018]. However, the authors suspect that the method can still be improved, because the value function or the $Q$ function are only randomly initialized so that many samples are burned to warm them up. As alluded before and shown in Section 4.4, we will propose a way to learn a value

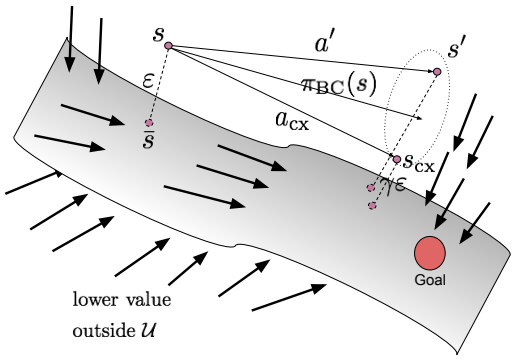



Figure 4.2: Illustration of the correction effect. A conservatively-extrapolated value function $V$, as shown in the figure, has lower values further away from $\mathcal{U}$, and therefore the gradients of $V$ point towards $\mathcal{U}$. With such a value function, suppose we are at state $s$ which is $\varepsilon$-close to $\mathcal{U}$. The locally-correctable assumption of the dynamics assumes the existence of $a_{\mathrm{cx}}$ that will drive us to state $s_{\mathrm{cx}}$ that is closer to $\mathcal{U}$ than $s$. Since $s_{\mathrm{cx}}$ has a relatively higher value compared to other possible future states that are further away from $\mathcal{U}$ (e.g., $s'$ shown in the figure), $s_{\mathrm{cx}}$ will be preferred by the optimization (4.4.3). In other words, if an action $a$ leads to state $s$ with large distance to $\mathcal{U}$, the action won't be picked by Equation (4.4.3) because it cannot beat $a_{\mathrm{cx}}$.

function from the demonstrations so that the following RL algorithm can be initialized by a policy, value function, and $Q$ function (which is a composition of value and dynamics model) and thus converge faster.

## 4.4 Theoretical Motivations

In this section, we formalize our key intuition that the ideal extrapolation of the value function $V^{\pi_{\mathrm{e}}}$ should be that the values should decrease as we get further and further from the demonstrations. Recall that we use $\mathcal{U}$ to denote the set of states reachable by the expert policy from any initial state $s_0$ drawn with positive probability from $D_{s_0}$. [3] Let $\Pi_{\mathcal{U}}(s)$ be the projection of $s \in \mathbb{R}^d$ to a set $\mathcal{U} \subset \mathbb{R}^d$ (according to the norm $\|\cdot\|$) [4].

We introduce the notion of value functions with conservative extrapolation which matches $V^{\pi_{\mathrm{e}}}$ on the demonstration states $\mathcal{U}$ and has smaller values outside $\mathcal{U}$. As

[3] Recall that we assume that $D_{s_0}$ has a low-dimensional support and thus typically $\mathcal{U}$ will also be a low-dimensional subset of the ambient space.

[4] Any tiebreaker can be used if there are multiple closest points.

formally defined in Equations (4.4.1) and (4.4.2) in Algorithm 7, we extrapolate $V^{\pi_e}$ in a way that the value at $s \notin \mathcal{U}$ is decided by the value of its nearest neighbor in $\mathcal{U}$ (that is $V^{\pi_e}(\Pi_{\mathcal{U}}(s))$, and its distance to the nearest neighbor (that is, $\|s - \Pi_{\mathcal{U}}(s)\|$). We allow a $\delta_V > 0$ error because exact fitting inside or outside $\mathcal{U}$ would be impossible.

**Algorithm 7** Self-correctable policy induced from a value function with conservative extrapolation

**Require: conservatively-extrapolated values** $V$ satisfying

$$V(s) = V^{\pi_e}(s) \pm \delta_V, \qquad \text{if } s \in \mathcal{U} \tag{4.4.1}$$

$$V(s) = V^{\pi_e}(\Pi_{\mathcal{U}}(s)) - \lambda\|s - \Pi_{\mathcal{U}}(s)\| \pm \delta_V \qquad \text{if } s \notin \mathcal{U} \tag{4.4.2}$$

and a locally approximately correct dynamics $M$ and BC policy $\pi_{\mathrm{BC}}$ satisfying Assumption 4.4.1.

**Self-correctable policy $\pi$:**

$$\pi(s) \triangleq \operatorname*{argmax}_{a:\|a-\pi_{\mathrm{BC}}(s)\|\le\zeta} V(M(s,a)) \tag{4.4.3}$$

Besides a conservatively-extrapolated value function $V$, our Algorithm 7 relies on a learned dynamics model $M$ and a behavioral cloning policy $\pi_{\mathrm{BC}}$. With these, the policy returns the action with the maximum value of the predicted next state in around the action of the BC policy. In other words, the policy $\pi$ attempts to re-adjust the BC policy locally by maximizing the value of the next state.

Towards analyzing Algorithm 7, we will make a few assumptions. We first assume that the BC policy is correct in the set $\mathcal{U}$, and the dynamics model $M$ is locally correct around the set $\mathcal{U}$ and the BC actions. Note that these are significantly weaker than assuming that the BC policy is globally correct (which is impossible to ensure) and that the model $M$ is globally correct.

**Assumption 4.4.1** (Local errors in learned dynamics and BC policy)**.** We assume the BC policy $\pi_{\mathrm{BC}}$ makes at most $\delta_\pi$ error in $\mathcal{U}$: for all $s \in \mathcal{U}$, we have $\|\pi_{\mathrm{BC}}(s) - \pi_e(s)\| \le \delta_\pi$. We also assume that the learned dynamics $M$ has $\delta_M$ error locally around $\mathcal{U}$ and the

BC actions in the sense that for all $s$ that is $\varepsilon$-close to $\mathcal{U}$, and any action that is $\zeta$-close to $\pi_{\mathrm{BC}}(s)$, we have $\|M(s,a) - M^\star(s,a)\| \le \delta_M$.

We make another crucial assumption on the stability/correctability of the true dynamics. The following assumption essentially says that if we are at a state that is near the demonstration set, then there exists an action that can drive us closer to the demonstration set. This assumption rules out certain dynamics that does not allow corrections even after the policy making a small error. For example, if a robot, unfortunately, falls off a cliff, then fundamentally it cannot recover itself — our algorithm cannot deal with such pathological situations.

**Assumption 4.4.2** (Locally-correctable dynamics)**.** For some $\gamma \in (0,1)$ and $\varepsilon > 0, L_c > 0$, we assume that the dynamics $M^\star$ is $(\gamma, L_c, \varepsilon)$-locally-correctable w.r.t to the set $\mathcal{U}$ in the sense that for all $\varepsilon_0 \in (0, \varepsilon]$ and any tuple $(\bar{s}, \bar{a}, \bar{s}')$ satisfying $\bar{s}, \bar{s}' \in \mathcal{U}$ and $\bar{s}' = M^\star(\bar{s}, \bar{a})$, and any $\varepsilon_0$-perturbation $s$ of $\bar{s}$ (that is, $s \in N_{\varepsilon_0}(\bar{s})$), there exists an action $a_{\mathrm{cx}}$ that is $L_c\varepsilon_0$ close to $\bar{a}$, such that it makes a correction in the sense that the resulting state $s'$ is $\gamma\varepsilon_0$-close to the set $\mathcal{U}$: $s' = M^\star(s, a_{\mathrm{cx}}) \in N_{\gamma\varepsilon_0}(\mathcal{U})$. Here $N_\delta(K)$ denotes the set of points that are $\delta$-close to $K$.

Finally, we will assume the BC policy, the value function, and the dynamics are all Lipschitz in their arguments.[5] We also assume the projection operator to the set $\mathcal{U}$ is locally Lipschitz. These are regularity conditions that provide loose local extrapolation of these functions, and they are satisfied by parameterized neural networks that are used to model these functions.

**Assumption 4.4.3** (Lipschitz-ness of policy, value function, and dynamics)**.** We assume that the policy $\pi_{\mathrm{BC}}$ is $L_\pi$-Lipschitz. That is, $\|\pi_{\mathrm{BC}}(s) - \pi_{\mathrm{BC}}(\tilde{s})\| \le L_\pi \|s - \tilde{s}\|$ for all $s, \tilde{s}$. We assume the value function $V^{\pi_{\mathrm{e}}}$ and the learned value function $V$ are

[5] We note that technically when the reward function is $R(s,a) = -1$, the value function is not Lipschitz. This can be alleviated by considering a similar reward $R(s,a) = -\alpha - \beta\|a\|^2$ which does not require additional information.

$L_V$-Lipschitz, the model $M^\star$ is $L_{M,a}$-Lipschitz w.r.t to the action and $L_{M,s}$-Lipschitz w.r.t to the state $s$. We also assume that the set $\mathcal{U}$ has $L_\Pi$-Lipschitz projection locally: for all $s, \hat{s}$ that is $\varepsilon$-close to $\mathcal{U}$, $\|\Pi_{\mathcal{U}}(s) - \Pi_{\mathcal{U}}(\hat{s})\| \le L_\Pi \|s - \hat{s}\|$.

Under these assumptions, now we are ready to state our main theorem. It claims that 1) the induced policy $\pi$ in Algorithm 7 stays close to the demonstration set and performs similarly to the expert policy $\pi_{\mathrm{e}}$, and 2) following the induced policy $\pi$, we will arrive at a state with a near-optimal value.

**Theorem 4.4.4.** *Suppose Assumptions 4.4.1 to 4.4.3 hold with sufficiently small $\varepsilon > 0$ and errors $\delta_M$, $\delta_\pi$,$\delta_\pi > 0$ so that they satisfy $\zeta \ge L_c\varepsilon + \delta_\pi + L_\pi$. Let $\lambda$ be sufficiently large so that $\lambda \ge \frac{2L_V L_\Pi L_M \zeta + 2\delta_V + 2L_V \delta_M}{(1-\gamma)\varepsilon}$. Then, the policy $\pi$ from Equation* (4.4.3) *satisfies the following:*

1. *Starting from $s_0 \in \mathcal{U}$ and executing policy $\pi$ for $T_0 \le T$ steps, the resulting states $s_1, \ldots, s_{T_0}$ are all $\varepsilon$-close to the demonstrate states set $\mathcal{U}$.*

2. *In addition, suppose the expert policy makes at least $\rho$ improvement every step in the sense that for every $s \in \mathcal{U}$, either $V^{\pi_{\mathrm{e}}}(M^\star(s, \pi_{\mathrm{e}}(s))) \ge V^{\pi_{\mathrm{e}}}(s) + \rho$ or $M^\star(s, \pi_{\mathrm{e}}(s))$ reaches the goal.*[6] *Assume $\varepsilon$ and $\delta_M, \delta_V, \delta_\pi$ are small enough so that they satisfy $\rho \gtrsim \varepsilon + \delta_\pi$.*

   *Then, the policy $\pi$ will achieve a state $s_T$ with $T \le 2|V^{\pi_{\mathrm{e}}}(s_0)|/\rho$ steps which is $\varepsilon$-close to a state $\bar{s}_T$ with value at least $V^{\pi_{\mathrm{e}}}(s_T) \gtrsim -(\varepsilon + \delta_\pi)$.*[7]

We defer the proof to the end of the subsection. The first bullet follows inductively invoking the following lemma which states that if the current state is $\varepsilon$-close to $\mathcal{U}$, then so is the next state. The proof of the Lemma is the most mathematically involved part of the chapter and is deferred to the end of the section. We demonstrate the key idea of the proof in Figure 4.2 and its caption.

[6] $\rho$ is 1 when the reward is always $-1$ before achieving the goal.

[7] Here $\gtrsim$ hides multiplicative constant factors depending on the Lipschitz parameters $L_{M,a}, L_{M,s}, L_\pi, L_V$.

**Lemma 4.4.5.** *In the setting of Theorem 4.4.4, suppose $s$ is $\varepsilon$-close to the demonstration states set $\mathcal{U}$. Suppose $\mathcal{U}$, and let $a = \pi(s)$ and $s' = M^\star(s, a)$. Then, $s'$ is also $\varepsilon$-close to the set $\mathcal{U}$.*

We effectively represent the $Q$ function by $V(M(s, a))$ in Algorithm 7. And we argue below that this helps address the degeneracy issue when there are random goal states (which is the case in our experiments.)

**Coping with the degeneracy with learned dynamics model.** Cautious reader may realize that the degeneracy problem discussed in Section 4.3 about learning $Q$ function from demonstrations with deterministic policies may also occur with learning the model $M$. However, we will show that when the problem has the particular structure of reaching a random but given goal state $g$, learning $Q$ still suffers the degeneracy but learning the dynamics does not.

The typical way to deal with a random goal state $g$ is to consider goal-conditioned value function $V(s, g)$, policy $\pi(s, g)$, and $Q$ function $Q(s, a, g)$.[8] However, the dynamics model does not have to condition on the goal. Learning $Q$ function still suffers from the degeneracy problem because $Q(s, a, g) \triangleq Q^{\pi_e}(s, \pi_e(s, g), g)$ matches $Q^{\pi_e}$ on the demonstrations but does not use the information from $a$ at all. However, learning $M$ does not suffer from such degeneracy because given a single state $s$ in the demonstration, there is still a variety of action $a$ that can be applied to state $s$ because there are multiple possible goals $g$. (In other words, we cannot construct a pathological $M(s, a) = M^\star(s, \pi_e(s))$ because the policy also takes in $g$ as an input). As a result, parameterizing $Q$ by $Q(s, a, g) = V(M(s, a), g)$ do not suffer from the degeneracy either.

[8] This is equivalent to viewing the random goal $g$ as part of an extended state $\check{s} = (s, g)$. Here the second part of the extended state is randomly chosen during sampling the initial state, but never changed by the dynamics. Thus all of our previous work does apply to this situation via this reduction.

**Discussion: can we learn conservatively-extrapolated $Q$-function?** We remark that we do not expect a conservative-extrapolated $Q$-functions would be helpful. The fundamental idea here is to penalize the value of unseen states so that the policy can self-correct. However, to learn a $Q$ function that induces self-correctable policies, we should *encourage* unseen actions that can correct the trajectory, instead of penalize them just because they are not seen before. Therefore, it is crucial that the penalization is done on the unseen states (or $V$) but not the unseen actions (or $Q$).

### Proofs of Lemma 4.4.5 and theorem 4.4.4

*Proof of Lemma 4.4.5.* Let $\bar{s} \triangleq \Pi_{\mathcal{U}} s$ and $\bar{a} \triangleq \pi_{\mathrm{e}}(\bar{s})$. Because $s$ is $\varepsilon$-close to the set $\mathcal{U}$, we have $\|s - \bar{s}\| \leq \varepsilon$. By the $(\gamma, L_c, \varepsilon)$-locally-correctable assumption of the dynamics, we have that there exists an action $a_{\mathrm{cx}}$ such that a) $\|a_{\mathrm{cx}} - \bar{a}\| \leq L_c \varepsilon$ and b) $s'_{\mathrm{cx}} \triangleq M^\star(s, a_{\mathrm{cx}})$ is $\gamma\varepsilon$-close to the set $\mathcal{U}$. Next we show that $a_{\mathrm{cx}}$ belongs to the constraint set $\{a : \|a - \pi_{\mathrm{BC}}(s)\| \leq \zeta\}$ in Equation (4.4.3). Note that $\|a_{\mathrm{cx}} - \pi_{\mathrm{BC}}(s)\| \leq \|a_{\mathrm{cx}} - \bar{a}\| + \|\bar{a} - \pi_{\mathrm{BC}}(\bar{s})\| + \|\pi_{\mathrm{BC}}(\bar{s}) - \pi_{\mathrm{BC}}(s)\| \leq L_c\varepsilon + \delta_\pi + L_\pi \varepsilon$ because of triangle inequality, the closeness of $a_{\mathrm{cx}}$ and $\bar{a}$, the assumption that $\pi_{\mathrm{BC}}$ has $\delta$ error in the demonstration state set $\mathcal{U}$, and the Lipschitzness of $\pi_{\mathrm{BC}}$. Since $\zeta$ is chosen to be bigger than $L_c\varepsilon + \delta_\pi + L_\pi\varepsilon$, we conclude that $a_{\mathrm{cx}}$ belongs to the constraint set of the optimization in Equation (4.4.3).

This suggests that the maximum value of the optimization (4.4.3) is bigger than the corresponding value of $a_{\mathrm{cx}}$:

$$V(M(s, a)) \geq V(M(s, a_{\mathrm{cx}})) \tag{4.4.4}$$

Note that $a$ belongs to the constraint set by definition and therefore $\|a - a_{\mathrm{cx}}\| \leq 2\zeta$. By Lipschitzness of the dynamics model, and the value function $V^{\pi_{\mathrm{e}}}$, we have that $\|M^\star(s, a) - M^\star(s, a_{\mathrm{cx}})\| \leq L_M \|a - a_{\mathrm{cx}}\| \leq 2L_M\zeta$. Let $s' = M^\star(s, a)$ and $s'_{\mathrm{cx}} =$

$M^\star(s, a_{\text{cx}})$. We have $\|s' - s'_{\text{cx}}\| \leq 2L_M\zeta$. By the Lipschitz projection assumption, we have that $\|\Pi_{\mathcal{U}} s' - \Pi_{\mathcal{U}} s'_{\text{cx}}\| \leq L_\Pi \|s' - s'_{\text{cx}}\| \leq 2L_\Pi L_M \zeta$, which in turns implies that $|V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s') - V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s'_{\text{cx}})| \leq 2L_V L_\Pi L_M \zeta$ by Lipschitzness of $V^{\pi_{\text{e}}}$. It follows that

$$
\begin{aligned}
V(s') &\leq V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s') - \lambda \|s' - \Pi_{\mathcal{U}} s'\| + \delta_V && \text{(by assumption (4.4.1))} \\
&\leq V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s'_{\text{cx}}) + |V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s'_{\text{cx}}) - V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s')| - \lambda \|s' - \Pi_{\mathcal{U}} s'\| + \delta_V \\
& && \text{(by triangle inequality)} \\
&\leq V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} s'_{\text{cx}}) + 2L_V L_\Pi L_M \zeta - \lambda \|s' - \Pi_{\mathcal{U}} s'\| + \delta_V \\
& && \text{(by equations in paragraph above)} \\
&\leq V(s'_{\text{cx}}) + \lambda \|s'_{\text{cx}} - \Pi s'_{\text{cx}}\| + 2L_V L_\Pi L_M \zeta - \lambda \|s' - \Pi_{\mathcal{U}} s'\| + 2\delta_V \\
& && \text{(by assumption (4.4.2))}
\end{aligned}
$$

Note that by the Lipschitzness of the value function and the assumption on the error of the dynamics model,

$$
\begin{aligned}
|V(s') - V(M(s,a))| &= |V(M^\star(s,a)) - V(M(s,a))| \\
&\leq L_v \|M^\star(s,a) - M(s,a)\| \leq L_V \delta_M
\end{aligned} \tag{4.4.5}
$$

Simlarly

$$
\begin{aligned}
|V(s'_{\text{cx}}) - V(M(s,a_{\text{cx}}))| &= |V(M^\star(s,a_{\text{cx}})) - V(M(s,a_{\text{cx}}))| \\
&\leq L_v \|MT^\star(s,a_{\text{cx}}) - M(s,a_{\text{cx}})\| \leq L_V \delta_M
\end{aligned} \tag{4.4.6}
$$

Combining the three equations above, we obtain that

$$
\begin{aligned}
&\lambda \|s'_{\text{cx}} - \Pi s'_{\text{cx}}\| + 2L_V L_\Pi L_M \zeta - \lambda \|s' - \Pi_{\mathcal{U}} s'\| + 2\delta_V \\
&\geq V(s') - V(s'_{\text{cx}}) \\
&\geq V(M(s,a)) - V(M(s,a_{\text{cx}})) - 2L_V \delta_M && \text{(by Equations (4.4.5) and (4.4.6))}
\end{aligned}
$$

$$\geq -2L_V\delta_M \qquad \text{(by Equation (4.4.4))}$$

Let $\kappa = 2L_V L_\Pi L_M \zeta + 2\delta_V + 2L_V\delta_M$ and use the assumption that $s'_{\text{cx}}$ is $\gamma\varepsilon$-close to the set $\mathcal{U}$ (which implies that $\|s'_{\text{cx}} - \Pi s'_{\text{cx}}\| \leq \gamma\varepsilon$), we obtain that

$$\lambda\|s' - \Pi_{\mathcal{U}} s'\| \leq \lambda\|s'_{\text{cx}} - \Pi s'_{\text{cx}}\| + \kappa \leq \lambda\gamma\varepsilon + \kappa \tag{4.4.7}$$

Note that $\lambda \geq \frac{\kappa}{(1-\gamma)\varepsilon}$, we have that $\|s' - \Pi_{\mathcal{U}} s'\| \leq \varepsilon$.

□

*Proof of Theorem 4.4.4.* To prove bullet 1, we apply Lemma 4.4.5 inductively for $T$ steps. To prove bullet 2, we will prove that as long as $s_i$ is $\varepsilon$-close to $\mathcal{U}$, then we can improve the value function by at least $\rho$ in one step. Consider $\bar{s}_i = \Pi_{\mathcal{U}}(s_i)$. We triangle inequality, we have that $\|\pi(s_i) - \pi_{\text{e}}(\bar{s}_i)\| \leq \|\pi(s_i) - \pi_{\text{BC}}(s_i)\| + \|\pi_{\text{BC}}(s_i) - \pi_{\text{BC}}(\bar{s}_i)\| + \|\pi_{\text{BC}}(\bar{s}_i) - \pi_{\text{e}}(\bar{s}_i)\|$. These three terms can be bounded respectively by $\zeta$, $L_\pi\|s_i - \bar{s}_i\| \leq L_\pi\varepsilon$, and $\delta_\pi$, using the definition of $\pi$, the Lipschitzness of $\pi_{\text{BC}}$, and the error assumption of $\pi_{\text{BC}}$ on the demonstration state set $\mathcal{U}$, respectively. It follows that $\|M^\star(s_i, \pi(s_i)) - M^\star(\bar{s}_i, \pi_{\text{e}}(\bar{s}_i))\| \leq L_{M,s}\|s_i - \bar{s}_i\| + L_{M,a}\|\pi(s_i) - \pi_{\text{e}}(\bar{s}_i)\| \leq L_{M,s}\varepsilon + L_{M,a}(\zeta + L_\pi\varepsilon + \delta_\pi)$. It follows by the Lipschitzness of the projection that

$$\begin{aligned}
\|\bar{s}_{i+1} - \Pi_{\mathcal{U}} M^\star(\bar{s}_i, \pi_{\text{e}}(\bar{s}_i))\| &= \|\Pi_{\mathcal{U}} M^\star(s_i, \pi(s_i)) - \Pi_{\mathcal{U}} M^\star(\bar{s}_i, \pi_{\text{e}}(\bar{s}_i))\| && (4.4.8)\\
&= |\Pi_{\mathcal{U}} M^\star(s_i, \pi(s_i)) - M^\star(\bar{s}_i, \pi_{\text{e}}(\bar{s}_i)) && (4.4.9)\\
&\leq L_\Pi(L_{M,s}\varepsilon + L_{M,a}(\zeta + L_\pi\varepsilon + \delta_\pi)) && (4.4.10)
\end{aligned}$$

This implies that

$$|V^{\pi_{\text{e}}}(\bar{s}_{i+1}) - V^{\pi_{\text{e}}}(\Pi_{\mathcal{U}} M^\star(\bar{s}_i, \pi_{\text{e}}(\bar{s}_i))| \leq L_V L_\Pi(L_{M,s}\varepsilon + L_{M,a}(\zeta + L_\pi\varepsilon + \delta_\pi)) \tag{4.4.11}$$

Note that we assumed that $V(M^\star(\bar{s}_i, \pi_e(\bar{s}_i)) \geq V(\bar{s}_i) + \rho$ or $M^\star(\bar{s}_i, \pi_e(\bar{s}_i))$ reaches the goal. If the former, it follows that $V^{\pi_e}(\bar{s}_{i+1}) \geq V^{\pi_e}(\bar{s}_i) + \rho - L_V L_\Pi (L_{M,s}\varepsilon + L_{M,a}(\zeta + L_\pi \varepsilon + \delta_\pi)) \geq V^{\pi_e}(\bar{s}_i) + \rho/2$. Otherwise, or $s_{i+1}$ is $\varepsilon$-close to $\bar{s}_{i+1}$ whose value is at most $-L_V L_\Pi (L_{M,s}\varepsilon + L_{M,a}(\zeta + L_\pi \varepsilon + \delta_\pi)) = -O(\varepsilon + \delta_\pi)$

□

## 4.5 Main Approach

**Learning value functions with negative sampling from demonstration trajectories.** As motivated in Section 4.4 by Algorithm 7 and Theorem 4.4.4, we first develop a practical method that can learn a value function with conservative extrapolation, without environment interaction. Let $V_\phi$ be a value function parameterized by $\phi$. Using the standard TD learning loss, we can ensure the value function to be accurate on the demonstration states $\mathcal{U}$ (i.e., to satisfy Equation (4.4.1)). Let $\bar{\phi}$ be the target value function,[9] the TD learning loss is defined as

$$\mathcal{L}_{td}(\phi) = \mathbb{E}_{(s,a,s')\sim\rho^{\pi_e}} \left[ \left( r(s,a) + V_{\bar{\phi}}(s') - V_\phi(s) \right)^2 \right]$$

where $r(s,a)$ is the (sparse) reward, $\bar{\phi}$ is the parameter of the target network, $\rho^{\pi_e}$ is the distribution of the states-action-states tuples of the demonstrations. The crux of the ideas in this chapter is to use a negative sampling technique to enforce the value function to satisfy conservative extrapolation requirement (4.4.2). It would be infeasible to enforce condition (4.4.2) for every $s \notin \mathcal{U}$. Instead, we draw random "negative samples" $\tilde{s}$ from the neighborhood of $\mathcal{U}$, and enforce the condition (4.4.2). This is inspired by the negative sampling approach widely used in NLP for training word embeddings [Mikolov et al., 2013, Gutmann and Hyvärinen, 2012]. Concretely,

[9] A target value function is widely used in RL to improve the stability of the training [Lillicrap et al., 2016, Mnih et al., 2015].

we draw a sample $s \sim \rho^{\pi_e}$, create a random perturbation of $s$ to get a point $\tilde{s} \notin \mathcal{U}$. and construct the following loss function:[10]

$$\mathcal{L}_{ns}(\phi) = \mathbb{E}_{s\sim\rho^{\pi_e},\tilde{s}\sim\mathsf{perturb}(s)} \left(V_{\bar{\phi}}(s) - \lambda\|s - \tilde{s}\| - V_\phi(\tilde{s})\right)^2.$$

The rationale of the loss function can be best seen in the situation when $\mathcal{U}$ is assumed to be a low-dimensional manifold in a high-dimensional state space. In this case, $\tilde{s}$ will be outside the manifold $\mathcal{U}$ with probability 1. Moreover, the random direction $\tilde{s} - s$ is likely to be almost orthogonal to the tangent space of the manifold $\mathcal{U}$, and thus $s$ is a reasonable approximation of the projection of $\tilde{s}$ back to the $\mathcal{U}$, and $\|s - \tilde{s}\|$ is an approximation of $\|\Pi_{\mathcal{U}}\tilde{s} - \tilde{s}\|$. If $\mathcal{U}$ is not a manifold but a small subset of the state space, these properties may still likely to hold for a good fraction of $s$.

We only attempt to enforce condition (4.4.2) for states near $\mathcal{U}$. This likely suffices because the induced policy is shown to always stay close to $\mathcal{U}$. Empirically, we perturb $s$ by adding a Gaussian noise. The loss function to learn $V_\phi$ is defined as $\mathcal{L}(\phi) = \mathcal{L}_{td}(\phi) + \mu\mathcal{L}_{ns}(\phi)$ for some constant $\mu > 0$. For a mini-batch $\mathcal{B}$ of data, we define the corresponding empirical loss by $\mathcal{L}(\phi; \mathcal{B})$ (similarly we define $\mathcal{L}_{td}(\phi; \mathcal{B})$ and $\mathcal{L}_{ns}(\phi; \mathcal{B})$). The concrete iterative learning algorithm is described in line 1-7 of Algorithm 8 (except line 6 is for learning the dynamics model, described below.)

**Learning the dynamics model.** We use standard supervised learning to train the model. We use $\ell_2$ norm as the loss for model parameters $\theta$ instead of the more commonly used MSE loss, following the success of Luo et al. [2019a]: $\mathcal{L}_{\text{model}}(\theta) = \mathbb{E}_{(s,a,s')\sim\rho^{\pi_e}} \left[\|M_\theta(s, a) - s'\|_2\right]$.

**Optimization for policy.** We don't maintain an explicit policy but use an induced policy from $V_\phi$ and $M_\theta$ by optimizing Equation (4.4.3). A natural choice would be

[10] With slight abuse of notation, we use $\rho^{\pi_e}$ to denote both the distribution of $(s, a, s')$ tuple and the distribution of $s$ of the expert trajectories.

**Algorithm 8** Value Iteration on Demonstrations with Negative Sampling (VINS)

1: $\mathcal{R} \leftarrow$ demonstration trajectories ▷ *No environment interaction will be used*
2: Initialize value parameters $\bar{\phi} = \phi$ and model parameters $\theta$ randomly
3: **for** $i = 1, \ldots, T$ **do**
4: sample mini-batch $\mathcal{B}$ of $N$ transitions $(s, a, r, s')$ from $\mathcal{R}$
5: update $\phi$ to minimize $\mathcal{L}_{td}(\phi; \mathcal{B}) + \mathcal{L}_{ns}(\phi; \mathcal{B})$
6: update $\theta$ to minimize loss $\mathcal{L}_{\text{model}}(\theta; \mathcal{B})$
7: update target network: $\bar{\phi} \leftarrow \bar{\phi} + \tau(\phi - \bar{\phi})$
8:
9: **function** POLICY($s$)
10: Option 1: $a = \pi_{\text{BC}}(s)$; Option 2: $a = 0$
11: sample $k$ noises $\xi_1, \ldots, \xi_k$ from Uniform$[-1, 1]^m$ ▷ *$m$ is the dimension of action space*
12: $i^* = \text{argmax}_i \, V_\phi(M_\theta(s, a + \alpha\xi_i))$ ▷ *$\alpha > 0$ is a hyper-parameter*
13: **return** $a + \alpha\xi_{i^*}$

using projected gradient ascent to optimize Equation (4.4.3). It's also possible to use cross-entropy methods in Kalashnikov et al. [2018] to optimize it. However, we found the random shooting suffices because the action space is relatively low-dimensional in our experiments. Moreover, the randomness introduced appears to reduce the overfitting of the model and value function slightly. As shown in line 10-13 of Algorithm 8, we sample $k$ actions in the feasible set and choose the one with maximum $V_\phi(M_\theta(s, a))$.

**Value iteration with environment interaction.** As alluded before, when more environment interactions are allowed, we initialize an RL algorithm by the value function, dynamics learned from VINS. Given that we have $V$ and $M$ in hand, we alternate between fitted value iterations for updating the value function and supervised learning for updating the models. A pseudocode our algorithm VINS+RL can be found in Algorithm 9. We do not use negative sampling here since the RL algorithms

already collect bad trajectories automatically. We also do not hallucinate any goals as in HER [Andrychowicz et al., 2017].

**Algorithm 9** Value Iteration with Environment Interactions Initialized by VINS (VINS+RL)

**Require:** Initialize parameters $\phi, \theta$ from the result of VINS (Algorithm 8)

1: $\mathcal{R} \leftarrow$ demonstration trajectories;
2: **for** stage $t = 1, \ldots$ **do**
3: collect $n_1$ samples using the induced policy $\pi$ in Algorithm 8 (with Option 2 in Line 10) and add them to $\mathcal{R}$
4: **for** $i = 1, \ldots, n_{\text{inner}}$ **do**
5: sample mini-batch $\mathcal{B}$ of $N$ transitions $(s, a, r, s')$ from $\mathcal{R}$
6: update $\phi$ to minimize $\mathcal{L}_{td}(\phi; \mathcal{B})$
7: update target value network: $\bar{\phi} \leftarrow \bar{\phi} + \tau(\phi - \bar{\phi})$
8: update $\theta$ to minimize loss $\mathcal{L}_{\text{model}}(\theta; \mathcal{B})$

## 4.6 Experiments

### 4.6.1 Experimental Setup

**Environments.** We evaluate our algorithms in three simulated robotics environments[11] designed by Plappert et al. [2018] based on OpenAI Gym [Brockman et al., 2016] and MuJoCo [Todorov et al., 2012a]: Reach, Pick-And-Place, and Push. In the three environments, a 7-DoF robotics arm is manipulated for different goals. In Reach, the task is to reach a randomly sampled target location; In Pick-And-Place, the goal is to grasp a box in a table and reach a target location, and in Push, the goal is to push a box to a target location.

The reward function is 0 if the goal is reached; otherwise -1. Intuitively, an optimal agent should complete the task in a shortest possible path. The environment will stop once the agent achieves the goal or max step number have been reached. Reaching max step will be regarded as failure.

For more details, we refer the readers to Plappert et al. [2018].

[11] Available at `https://github.com/openai/gym/tree/master/gym/envs/robotics`.

**Demonstrations.** For each task, we use Hindsight Experience Replay (HER) [Andrychowicz et al., 2017] to train a policy until convergence. The policy rolls out to collect 100/200 successful trajectories as demonstrations except for Reach environment where 100 successful trajectories are sufficient for most of the algorithms to achieve optimal policy. We filtered out unsuccessful trajectories during data collection.

We consider two settings: imitation learning from only demonstrations data, and leveraging demonstration in RL with a *limited* amount of interactions. We compare our algorithm with Behavioral Cloning and multiple variants of our algorithms in the first setting. We compare with Nair et al. [2018], and GAIL [Ho and Ermon, 2016] in the second setting. We do not compare with Gao et al. [2018] because it cannot be applied to the case with continuous actions and we do not compare with many new algorithms [Wu et al., 2019, Kumar et al., 2020, Yu et al., 2020] as they were not available when this work was done.

**Behavioral Cloning [Bain and Sommut, 1999].** Behavioral Cloning (BC) learns a mapping from a state to an action on demonstration data using supervised learning. We use MSE loss for predicting the actions.

**Nair *et al.*'18 [Nair et al., 2018].** A previous algorithm from Nair et al. [2018] combines HER [Andrychowicz et al., 2017] with BC and a few techniques: 1) an additional replay buffer filled with demonstrations, 2) an additional behavioral cloning loss for the policy, 3) a $Q$-filter for non-optimal demonstrations, 4) resets to states in the demonstrations to deal with long horizon tasks. We note that reseting to an arbitrary state may not be realistic for real-world applications in robotics. In contrast, our algorithm does not require resetting to a demonstration state.

**GAIL [Ho and Ermon, 2016].** Generative Adversarial Imitation Learning (GAIL) imitates the expert by matching the state-action distribution with a GAN-like framework.

**HER [Andrychowicz et al., 2017].** Hindsight Experience Replay (HER) is the one of the best techniques that deal with sparse-reward environments with multiple *goals* and can be combined with any off-policy RL algorithm. The key idea is that HER extends the replay buffer by changing the goals. With reasonable chosen goals, the underlying off-policy RL algorihtm can receive more signals from the generated experience, making policy optimization more complete.

**DAC [Kostrikov et al., 2018].** Discriminator-Actor-Critic (DAC) is a sample-efficient imitation learning algorihtm built on the top of GAIL. It addresses the reward bias problem by adapting AIRL reward function and introducing an absorbing state. Furthermore, it replaces the underlying RL algorithm in GAIL by TD3 [Fujimoto et al., 2018b] to make it more sample efficient.

**VINS.** As described in Section 4.5, in the setting without environment interaction, we use Algorithm 8; otherwise we use it to initialize an RL algorithm (see Algorithm 9). We use neural networks to parameterize the value function and the dynamics model. The granularity of the HER demonstration policy is very coarse, and we argument the data with additional linear interpolation between consecutive states. We also use only a subset of the states as inputs to the value function and the dynamics model, which apparently helps improve the training and generalization of them. Implementation details can be found in Section 4.6.4.

| | VINS (ours) | BC |
|---|---|---|
| Reach 10 | $\mathbf{99.3 \pm 0.1}\%$ | $98.6 \pm 0.1\%$ |
| Pick 100 | $\mathbf{75.7 \pm 1.0}\%$ | $66.8 \pm 1.1\%$ |
| Pick 200 | $\mathbf{84.0 \pm 0.5}\%$ | $82.0 \pm 0.8\%$ |
| Push 100 | $\mathbf{44.0 \pm 1.5}\%$ | $37.3 \pm 1.1\%$ |
| Push 200 | $\mathbf{55.2 \pm 0.7}\%$ | $51.3 \pm 0.6\%$ |

Table 4.1: The success rates of achieving the goals for VINS and BC in the setting without any environment interactions. A random policy has about 5% success rate at Pick and Push.

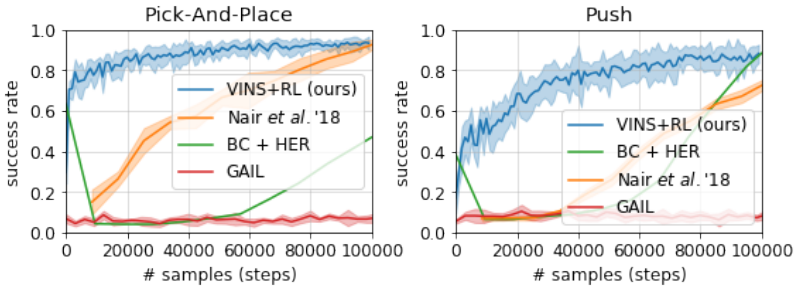



Figure 4.3: The learning curves of VINS+RL (Algorithm 9) vs Nair *et al.*'18 on Pick-And-Place and Push. Shaded areas indicates one standard error estimated from 10 random seeds.[13]

### 4.6.2 Experiment Results

Our main results are reported in Table 4.1 [12] for the setting with no environment interaction and Figure 4.3 for the setting with environment interactions. Table 4.1 shows that the Reach environment is too simple so that we do not need to run the RL algorithm. On the harder environments Pick-And-Place and Push, our algorithm VINS outperforms BC. We believe this is because our conservatively-extrapolated value function helps correct the mistakes in the policy. Here we use 2k trials to estimate the success rate (so that the errors in the estimation is negligible), and we run the algorithms with 10 different seeds. The error bars are for 1 standard error.

Figure 4.3 shows that VINS initialized RL algorithm outperforms the baselines in sample efficiency. We believe the main reason is that due to the initialization of

[12]The standard error in the chapter means the standard error of average success rate over 10 (100 for Reach 10) different random seeds by the same algorithm, that is, the standard deviation of 10 numbers over $\sqrt{10}$ (or 10, respectively).

value and model, we pay less samples for warming up the value function. We note that our initial success rate in RL is slightly lower than the final result of VINS in Table 4.1. This is because in RL we implemented a slightly worse variant of the policy induced by VINS: in the policy of Algorithm 8, we use option 2 to search the action uniformly. This suffices because the additional interactions quickly allows us to learn a good model and the BC constraint is no longer needed.

### 4.6.3 Ablation Study

Towards understanding the effect of each component of VINS, we perform three ablative experiments to show the importance of negative sampling, searching in the neighborhood of Behavioral Cloned actions (option 1 in line 10 or Algorithm 8), and a good dynamics model. The results are shown in Table 4.2. We study three settings: (1) VINS without negative sampling (VINS w/o NS), where the loss $\mathcal{L}_{ns}$ is removed; (2) VINS without BC (VINS w/o BC), where option 2 in line 10 or Algorithm 8 is used; (3) VINS with oracle model without BC (VINS w/ oracle w/o BC), where we use the true dynamics model to replace line 12 of Algorithm 8. Note that the last setting is only synthetic for ablation study because in the real-world we don't have access to the true dynamics model. Please see the caption of Table 4.2 for more interpretations. We use the same set of hyperparameters for the same environment, which may not be optimal: for example, with more expert trajectories, the negative sampling loss $\mathcal{L}_{ns}$, which can be seen as a regularziation, should be assigned a smaller coefficient $\mu$.

### 4.6.4 Implementation Details

**Behavioral Cloning.** We use a feed-forward neural network with 3 hidden layers, each containinig 256 hidden units, and ReLU activation functions. We train the network until the test success rate plateaus.

| | Pick 100 | Pick 200 | Push 100 | Push 200 |
|---|---|---|---|---|
| BC | $66.8 \pm 1.1\%$ | $82.0 \pm 0.8\%$ | $37.3 \pm 1.1\%$ | $51.3 \pm 0.6\%$ |
| VINS | $75.7 \pm 1.0\%$ | $84.0 \pm 0.5\%$ | $44.0 \pm 0.8\%$ | $55.2 \pm 0.7\%$ |
| VINS w/o BC | $28.5 \pm 1.1\%$ | $43.6 \pm 1.2\%$ | $14.3 \pm 0.5\%$ | $24.9 \pm 1.3\%$ |
| VINS w/ oracle w/o BC | $51.4 \pm 1.4\%$ | $62.3 \pm 1.1\%$ | $40.7 \pm 1.4\%$ | $42.9 \pm 1.3\%$ |
| VINS w/ oracle | $76.3 \pm 1.4\%$ | $87.0 \pm 0.7\%$ | $48.7 \pm 1.2\%$ | $63.8 \pm 1.3\%$ |
| VINS w/o NS | $48.5 \pm 2.1\%$ | $71.6 \pm 0.9\%$ | $29.3 \pm 1.2\%$ | $38.7 \pm 1.5\%$ |

Table 4.2: Ablation study of components of VINS in the setting without environment interactions. We reported the average performance of 10 runs (with different random seeds) and the empirical standard error of the estimator of the average performance. The success rate of VINS w/o NS is consistently worse than VINS, which suggests that NS is crucial for tackling the false extrapolation. From comparisons between VINS w/o BC and VINS w/ oracle w/o BC, and between VINS and VINS w/ oracle, we observe that if the learning of the dynamics can be improved (potentially by e.g., by collecting data with random actions), then VINS or VINS w/o BC can be improved significantly. We also suspect that the reason why we need to search over the neighborhood of BC actions is that the dynamics is not accurate at state-action pairs far away from the demonstration set (because the dynamics is only learned on the demonstration set.)

**Nair et al.'18 [Nair et al., 2018].** We use the implementation from `https://github.com/jangirrishabh/Overcoming-exploration-from-demos`. We don't change the default hyperparameters, except that we're using 17 CPUs.

**GAIL [Ho and Ermon, 2016].** We use the implementation from OpenAI Baselines [Dhariwal et al., 2017]. We don't change the default hyperparameters.

**HER [Andrychowicz et al., 2017].** We also use the code from OpenAI Baselines and keep the default hyperparameters.

**Discriminator-Actor-Critic [Kostrikov et al., 2018].** We use the implementation from the official implementation `https://github.com/google-research/google-research/tree/master/dac`.

**VINS.**

- Architecture: We use feed-forward neural networks as function approximators for values and dynamics models. For the $V_\phi$, the network has one hidden layer which has 256 hidden units and a layer normalization layer [Lei Ba et al., 2016]. The dynamics model is a feed-forward neural network with two hidden layers and ReLU activation function. Each hidden layer has 500 units. The model uses the reduced states and actions to predict the next reduced states.

- Value augmentation: We augment the dataset by a linear interpolation between two consecutive states, i.e., for a transition $(s, a, r, s')$ in the demonstration, it's augmented to $(s + \lambda(s' - s), a, \lambda r, s')$ for a random real $\lambda \sim \text{Uniform}[0, 1]$. To minimize the losses, we use the Adam optimizer [Kingma and Ba, 2014] with learning rate $3 \times 10^{-4}$. We remove some less relevant coordinates from the state space to make the dimension smaller. (But we maintain the full state space for BC. BC will perform worse with reduce state space.) Specifically, the states of our algorithm for different environment are a) Reach: the position of the arm, b) Pick: the position of the block, the gripper, and the arm, and c) Push: the position of the block and the arm.

- States representation and perturbation: Both our model $M$ and value function $V$ are trained on the space of reduced states. The perturb function is also designed separately for each task. For Reach and Push, we perturb the arm only; For Pick-And-Place, we perturb the arm or the gripper. In the implementation, we perturb the state $s$ by adding Gaussian noise from a distribution $\mathcal{N}(0, \rho^2\Sigma)$, where $\Sigma$ is a diagonal matrix that contains the variances of each coordinate of the states from the demonstration trajectories. Here $\rho > 0$ is a hyper-parameter to tune.

## 4.7 Conclusion

We devise a new algorithm, VINS, that can learn self-correctable by learning value function and dynamics model from demonstrations. The key idea is a theoretical formulation of conservatively-extrapolated value functions that provably leads to self-correction. The empirical results show a promising performance of VINS and an algorithm that initializes RL with VINS. It's a fascinating direction to study other algorithms that may learn conservatively-extrapolated value functions in other real-world applications beyond the proof-of-concepts experiments in this chapter. For example, the negative sampling by Gaussian perturbation technique in this chapter may not make sense for high-dimensional pixel observation. The negative sampling can perhaps be done in the representation space (which might be learned by unsupervised learning algorithms) so that we can capture the geometry of the state space better.

# Chapter 5

# Safe Reinforcement Learning with Zero Training-time Violations

Training-time safety violations have been a major concern when we deploy reinforcement learning algorithms in the real world. This chapter explores the possibility of safe RL algorithms with *zero* training-time safety violations in the challenging setting where we are only given a safe but trivial-reward initial policy without any prior knowledge of the dynamics model and additional offline data. We propose an algorithm, **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS),which iteratively *learns* barrier certificates, dynamics models, and policies. The barrier certificates, learned via adversarial training, ensure the policy's safety assuming calibrated learned dynamics model. We also add a regularization term to encourage larger certified regions to enable better exploration. Empirical simulations show that zero safety violations are already challenging for a suite of simple environments with only 2-4 dimensional state space, especially if high-reward policies have to visit regions near the safety boundary. Prior methods require hundreds of violations to achieve decent rewards on these tasks, whereas our proposed algorithms incur zero violations.

## 5.1 Background

Researchers have demonstrated that reinforcement learning (RL) can solve complex tasks such as Atari games [Mnih et al., 2015], Go [Silver et al., 2017a], dexterous manipulation tasks [Akkaya et al., 2019], and many more robotics tasks in simulated environments [Haarnoja et al., 2018]. However, deploying RL algorithms to real-world problems still faces the hurdle that they require many unsafe environment interactions. For example, a robot's unsafe environment interactions include falling and hitting other objects, which incur physical damage costly to repair. Many recent deep RL works reduce the number of environment interactions significantly (e.g., see Haarnoja et al. [2018], Fujimoto et al. [2018b], Janner et al. [2019], Dong et al. [2020], Luo et al. [2019a], Chua et al. [2018b] and reference therein), but the number of unsafe interactions is still prohibitive for safety-critical applications such as robotics, medicine, or autonomous vehicles [Berkenkamp et al., 2017].

Reducing the number of safety violations may not be sufficient for these safety-critical applications—we may have to eliminate them. This chapter explores the possibility of safe RL algorithms with *zero safety violations* in both training time and test time. We also consider the challenging setting where we are only given a safe but trivial-reward initial policy.

A recent line of works on safe RL design novel actor-critic based algorithms under the constrained policy optimization formulation [Thananjeyan et al., 2021, Srinivasan et al., 2020, Bharadhwaj et al., 2020, Yang et al., 2020, Stooke et al., 2020]. They significantly reduce the number of training-time safety violations. However, these algorithms fundamentally learn the safety constraints by contrasting the safe and unsafe trajectories. In other words, because the safety set is only specified through the safety costs that are observed *postmortem*, the algorithms only learn the concept of safety through seeing unsafe trajectories. Therefore, these algorithms cannot achieve zero training-time violations. For example, even for the simple 2D inverted pendulum

environment, these methods still require at least 80 unsafe trajectories (see Figure 5.2 in Section 5.7).

Another line of work utilizes ideas from control theory and model-based approach [Cheng et al., 2019, Berkenkamp et al., 2017, Taylor et al., 2019, Zeng et al., 2020]. These works propose sufficient conditions involving certain Lyapunov functions or control barrier functions that can certify the safety of a subset of states or policies [Cheng et al., 2019]. These conditions assume access to calibrated dynamics models. They can, in principle, permit safety guarantees without visiting any unsafe states because, with the calibrated dynamics model, we can foresee future danger. However, control barrier functions are often non-trivially *handcrafted* with prior knowledge of the environments [Ames et al., 2019, Nguyen and Sreenath, 2016].

This work aims to design model-based safe RL algorithms that empirically achieve zero training-time safety violations by *learning* the barrier certificates *iteratively*. We present the algorithm **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS), which alternates between *learning* barrier certificates that certify the safety of *larger* regions of states, optimizing the policy, collecting more data within the certified states, and refining the learned dynamics model with data.[1]

The work of Richards et al. [2018] is a closely related prior result, which learns a Lyapunov function given a fixed dynamics model via discretization of the state space. Our work significantly extends it with three algorithmic innovations. First, we use adversarial training to learn the certificates, which avoids discretizing state space and can potentially work with higher dimensional state space than the two-dimensional problems in Richards et al. [2018]. Second, we do not assume a given, globally accurate dynamics model; instead, we learn the dynamics model from safe explorations. We achieve this by co-learning the certificates, dynamics model, and policy to iteratively

[1]We note that our goal is not to provide end-to-end formal guarantees of safety, which might be extremely challenging—nonconvex minimax optimizations and uncertainty quantification for neural networks are used as sub-procedures, and it's challenging to have worst-case guarantees for them.

grow the certified region and improve the dynamics model and still maintain zero violations. Thirdly, the work Richards et al. [2018] only certifies the safety of some states and does not involve learning a policy. In contrast, our work learns a policy and tailors the certificates to the learned policies. In particular, our certificates aim to certify only states near the trajectories of the current and past policies—this allows us to not waste the expressive power of the certificate parameterization on irrelevant low-reward states.

We evaluate our algorithms on a suite of tasks, including a few where achieving high rewards requires careful exploration near the safety boundary. For example, in the ***Swing*** environment, the goal is to swing a rod with the largest possible angle under the safety constraints that the angle is less than 90°. We show that our method reduces the number of safety violations from several hundred to zero on these tasks.

## 5.2 Related Work

Prior works about Safe RL take very different approaches. Dalal et al. [2018] adds an additional layer, which corrects the output of the policy locally. Some of them use Lagrangian methods to solve CMDP, while the Lagrangian multiplier is controlled adaptively [Tessler et al., 2018] or by a PID [Stooke et al., 2020]. Achiam et al. [2017], Yang et al. [2020] build a trust-region around the current policy, and Zanger et al. [2021] further improved Achiam et al. [2017] by learning the dynamics model. Eysenbach et al. [2017] learns a reset policy so that the policy only explores the states that can go back to the initial state. Turchetta et al. [2020] introduces a learnable teacher, which keeps the student safe and helps the student learn faster in a curriculum manner. Srinivasan et al. [2020] pre-trains a policy in a simpler environment and fine-tunes it in a more difficult environment. Bharadhwaj et al. [2020] learns conservative safety critics which underestimate how safe the policy is, and uses the conservative

safety critics for safe exploration and policy optimization. Thananjeyan et al. [2021] makes use of existing offline data and co-trains a recovery policy. Hazan et al. [2020] considers the nonstochastic control problem, and proposes an efficient algorithm to identify the linear dynamic system even with adversarial perturbations.

Another line of work involves Lyapunov functions and barrier functions. Chow et al. [2018] studies the properties of Lyapunov functions and learns them via bootstrapping with a discrete action space. Built upon Chow et al. [2018], Sikchi et al. [2021] learns the policy with Deterministic Policy Gradient theorem in environments with a continuous action space. Like TRPO [Schulman et al., 2015a], Sikchi et al. [2021] also builds a trust region of policies for optimization. Donti et al. [2020] constructs sets of stabilizing actions using a Lyapunov function, and project the action to the set, while Chow et al. [2019] projects action or parameters to ensure the decrease of Lyapunov function after a step. Ohnishi et al. [2019] is similar to ours but it constructs a barrier function manually instead of learning such one. Ames et al. [2019] gives an excellent overview of control barrier functions and how to design them. Perhaps the most related work to ours is Cheng et al. [2019], which also uses a barrier function to safeguard exploration and uses a reinforcement learning algorithm to learn a policy. However, the key difference is that we *learn* a barrier function, while Cheng et al. [2019] handcrafts one. The works on Lyapunov functions [Berkenkamp et al., 2017, Richards et al., 2018] require the discretizating the state space and thus only work for low-dimensional space.

Anderson et al. [2020] iteratively learns a neural policy which possily has higher total rewards but is more unasfe, distills the learned neural policy into a symbolic policy which is simpler and safer, and use automatic verification to certify the symbolic policy. The certification process is similar to construct a barrier function. As the certification is done on a learned policy, the region of certified states also grows. Howver, it assumes a known calibrated dynamcis model, while we also learns it. Also,

tt can only certifie states where a piecewise-linear policy is safe, while potentially we can certify more states.

## 5.3 Problem Setup and Preliminaries

We consider the general setup in Section 1.3, but emphasize on the safety of the policy $\pi$. Let $\mathcal{S}_{\text{unsafe}} \subset \mathcal{S}$ be the set of unsafe states specified by the user. The user-specified safe set $\mathcal{S}_{\text{safe}}$ is defined as $\mathcal{S} \backslash \mathcal{S}_{\text{unsafe}}$. A state $s$ is (user-specified) safe if $s \in \mathcal{S}_{\text{safe}}$. A trajectory is safe if and only if all the states in the trajectory are safe. An initial state drawn from $\mu_0$ is assumed to safe with probability 1. We say a deterministic policy $\pi$ is safe starting from state $s$, if the infinite-horizon trajectory obtained by executing $\pi$ starting from $s$ is safe. We also say a policy $\pi$ is safe if it is safe starting from an initial state drawn from $\mu$ with probability 1. A major challenge toward safe RL is the existence of irrecoverable states which are currently safe but will eventually lead to unsafe states regardless of future actions. We define the notion formally as follows.

**Definition 5.3.1.** A state $s$ is *viable* iff there exists a policy $\pi$ such that $\pi$ is safe starting from $s$, that is, executing $\pi$ starting from $s$ for infinite steps never leads to an unsafe state. A user-specified safe state that is not viable is called an *irrecoverable* state.

We remark that unlike Srinivasan et al. [2020], Roderick et al. [2020], we do not assume all safe states are viable. We rely on the extrapolation and calibration of the dynamics model to foresee risks. A calibrated dynamics model $M$ predicts a confidence region of states $M(s, a) \subseteq \mathcal{S}$, such that for any state $s$ and action $a$, we have $M^*(s, a) \in M(s, a)$.

### 5.3.1 Barrier Certificate

Barrier certificates are powerful tools to certify the stability of a dynamical system. Barrier certificates are often applied to a continuous-time dynamical system, but here we describe its discrete-time version where our work is based upon. We refer the readers to Prajna and Jadbabaie [2004], Prajna and Rantzer [2005] for more information about continuous-time barrier certificates.

Given a discrete-time dynamical system $s_{t+1} = f(s_t)$ *without control* starting from $s_0$, a function $h : \mathcal{S} \to \mathbb{R}$ is a barrier certifcate if for any $s \in \mathcal{S}$ such that $h(s) \geq 0$, $h(f(s)) \geq 0$. Zeng et al. [2020] considers a more restrictive requirement: For any state $s \in \mathcal{S}$, $h(f(s)) \geq \alpha h(s)$ for a constant $0 \leq \alpha < 1$.

it is easy to use a barrier certificate $h$ to show the stability of the dynamical system. Let $\mathcal{C}_h = \{s : h(s) \geq 0\}$ be the superlevel set of $h$. The requirement of barrier certificates directly translates to the requirement that if $s \in \mathcal{C}_h$, then $f(s) \in \mathcal{C}_h$. This property of $\mathcal{C}_h$, which is known as the *forward-invariant* property, is especially useful in safety-critical settings: suppose a barrier certificate $h$ such that $\mathcal{C}_h$ does not contain unsafe states and contains the initial state $s_0$, then it is guaranteed that $\mathcal{C}_h$ contains the entire trajectory of states $\{s_t\}_{t\geq 0}$ which are safe.

Finding barrier certificates requires a known dynamics model $f$, which often can only be approximated in practice. This issue can be resolved by using a well-calibrated dynamics model $\hat{f}$, which predicts a confidence interval containing the true output. When a calibrated dynamics model $\hat{f}$ is used, we require that for any $s \in \mathcal{S}$, $\min_{s' \in \hat{f}(s)} h(s') \geq 0$.

Control barrier functions [Ames et al., 2019] are extensions to barrier certificates in the control setting. That is, control barrier functions are often used to *find* an action to meet the safety requirement instead of certifying the stability of a closed dynamical system. In this work, we simply use barrier certificates because in Section 5.4, we view

the policy and the calibrated dynamics model as a whole closed dynamical system whose stability we are going to certify.

### 5.3.2 Metropolis-Adjusted Langevin Algorithm (MALA)

Given a probability density function $p$ on $\mathbb{R}^d$, Metropolis-Adjusted Langevin Algorithm (MALA) obtains random samples $x \sim p$ when direct sampling is difficult. it is based on Metropolis-Hastings algorithm which generates a sequence of samples $\{x_t\}_t$. Metropolis-Hastings algorithm requires a *proposal distribution* $q(x'|x)$. At step $t \geq 0$, Metropolis-Hastings algorithm generates a new sample $\hat{x}_{t+1} \sim q(\cdot|x_t)$ and accept it with probability

$$\alpha(x \to x') \triangleq \min\left(1, \frac{p(x')q(x|x')}{p(x)q(x'|x)}\right).$$

If the sample $\hat{x}_{t+1}$ is accepted, we set $x_{t+1} = \hat{x}_{t+1}$; Otherwise the old sample $x_t$ is used: $x_{t+1} = x_t$. MALA considers a special proposal function $q_\tau(x'|x) = \mathcal{N}(x+\tau\nabla p(x), 2\tau I_d)$. See Algorithm 10 for the pseudocode.

**Algorithm 10** Metropolis-Adjusted Langevin Algorithm (MALA)

**Require:** A probability density function $p$ and a step size $\tau$.

Initialize $x_0$ arbitrarily.
**for** $t$ from 0 to $\infty$ **do**
    Draw $\zeta_t \sim \mathcal{N}(0, I_d)$.
    Set $\hat{x}_{t+1} = x_t + \tau\nabla \log p(X_t) + \sqrt{2\tau}\zeta_t$.
    Draw $u_t \sim \text{Uniform}[0, 1]$.
    **if** $u_t \geq \alpha(x_t \to \hat{x}_{t+1})$ **then**
        Set $x_{t+1} = \hat{x}_{t+1}$.
    **else**
        Set $x_{t+1} = x_t$.

## 5.4 Learning Barrier Certificates via Adversarial Training

This section describes an algorithm that learns a barrier certificate for a fixed policy $\pi$ under a calibrated dynamics model $M$. Concretely, to certify a policy $\pi$ is safe, we aim to learn a (discrete-time) barrier certificate $h$ that satisfies the following three requirements.

**R.1.** For $s_0 \sim \mu_0$, $h(s_0) \geq 0$ with probability 1.

**R.2.** For every $s \in \mathcal{S}_{\text{unsafe}}$, $h(s) < 0$.

**R.3.** For any $s$ such that $h(s) \geq 0$, $\min_{s' \in M(s,\pi(s))} h(s) \geq 0$.

Requirement **R.1** and **R.3** guarantee that the policy $\pi$ will never leave the set $\mathcal{C}_h = \{s \in \mathcal{S} : h(s) \geq 0\}$ by simple induction. Moreover, **R.2** guarantees that $\mathcal{C}_h$ only contains safe states and therefore the policy never visits unsafe states.

In the rest of the section, we aim to design and train such a barrier certificate $h = h_\phi$ parametrized by neural network $\phi$.

**$h_\phi$ parametrization.** The three requirements for a barrier certificate are challenging to simultaneously enforce with constrained optimization involving neural network parameterization. Instead, we will parametrize $h_\phi$ with **R.1** and **R.2** built-in such that for any $\phi$, $h_\phi$ always satisfies **R.1** and **R.2**.

We assume the initial state $s_0$ is deterministic. To capture the known user-specified safety set, we first handcraft a continuous function $\mathcal{B}_{\text{unsafe}} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ satisfying $\mathcal{B}_{\text{unsafe}}(s) \approx 0$ for typical $s \in \mathcal{S}_{\text{safe}}$ and $\mathcal{B}_{\text{unsafe}}(s) > 1$ for any $s \in \mathcal{S}_{\text{unsafe}}$ and can be seen as a smoothed indicator of $\mathcal{S}_{\text{unsafe}}$.[2] The construction of $\mathcal{B}_{\text{unsafe}}$ does not need prior knowledge of irrecoverable states, but only the user-specified safety set $\mathcal{S}_{\text{safe}}$.

[2]The function $\mathcal{B}_{\text{unsafe}}(s)$ is called a barrier function for the user-specified safe set in the optimization literature. Here we do not use this term to avoid confusion with the barrier certificate.

To further encode the user-specified safety set into $h_\phi$, we choose $h_\phi$ to be of form $h_\phi(s) = 1 - \text{Softplus}(f_\phi(s) - f_\phi(s_0)) - \mathcal{B}_{\text{unsafe}}(s)$, where $f_\phi$ is a neural network, and $\text{Softplus}(x) = \log(1 + e^x)$.

Because $s_0$ is safe and $\mathcal{B}_{\text{unsafe}}(s_0) \approx 0$, $h_\phi(s_0) \approx 1 - \text{Softplus}(0) > 0$. Therefore $h_h$ satisfies **R.1**. Moreover, for any $s \in \mathcal{S}_{\text{unsafe}}$, we have $h_\phi(s) < 1 - \mathcal{B}_{\text{unsafe}}(s) < 0$, so $h_\phi$ in our parametrization satisfies **R.2** by design.

The parameterization can also be extended to multiple initial states. For example, if the initial $s_0$ is sampled from a distribution $\mu$ that is supported on a bounded set, and suppose that we are given the indicator function $\mathcal{B}_{\text{init}} : \mathcal{S} \to \mathbb{R}$ for the support of $\mu$ (that is, $\mathcal{B}_{\text{init}}(s) = 1$ for any $s \in \text{supp}(\mu)$, and $\mathcal{B}_{\text{init}}(s) = 0$ otherwise). Then, the parametrization of $h_\phi$ can be $h_\phi(s) = 1 - \text{Softplus}(f_\phi(s))(1 - \mathcal{B}_{\text{init}}(s)) - \mathcal{B}_{\text{unsafe}}(s)$. For simplicity, we focus on the case where there is a single initial state.

**Training barrier certificates.** We now move on to training $\phi$ to satisfy **R.3**. Let

$$U(s, a, h) \triangleq \max_{s' \in M(s,a)} -h(s'). \tag{5.4.1}$$

Then, **R.3** requires $U(s, \pi(s), h_\phi) \leq 0$ for *any* $s \in \mathcal{C}_{h_\phi}$, The constraint in **R.3** naturally leads up to formulate the problem as a min-max problem. Define our objective function to be

$$C^*(h_\phi, U, \pi) \triangleq \max_{s \in \mathcal{C}_{h_\phi}} U(s, \pi(s), h_\phi) = \max_{s \in \mathcal{C}_{h_\phi}, s' \in M(s,\pi(s))} -h(s')\,, \tag{5.4.2}$$

and we want to minimize $C^*$ w.r.t. $\phi$:

$$\min_\phi C^*(h_\phi, U, \pi) = \min_\phi \max_{s \in \mathcal{C}_{h_\phi}, s' \in M(s,\pi(s))} -h(s'), \tag{5.4.3}$$

Our goal is to ensure the minimum value is less than 0. We use gradient descent to solve the optimization problem, hence we need an explicit form of $\nabla_\phi C^*$. Let $L(s, \nu; \phi)$ be the Lagrangian for the constrained optimization problem in $C^*$ where $\nu \geq 0$ is the Lagrangian multiplier of the constraint $s \in \mathcal{C}_{h_\phi}$:

$$L(s, \nu; \phi) \triangleq U(s, \pi(s), h_\phi) + \nu h_\phi(s), \quad C^* = \max_{s \in \mathcal{S}} \min_{\nu \geq 0} L(s, \nu; \phi).$$

By the Envelope Theorem (see Carter [2001, Section 6.1]), we have

$$\nabla_\phi C^* = \nabla_\phi U(s^*, \pi(s^*), h_\phi) + \nu^* \nabla_\phi h_\phi(s^*),$$

where $s^*$ and $\nu^*$ are the optimal solution for $\phi$. Once $s^*$ is known, the optimal Lagrangian multiplier $\nu^*$ can be given by KKT conditions:

$$\begin{cases} \nu^* h_\phi(s^*) = 0, \\ \nabla_s L(s^*, \nu^*; \phi) = \mathbf{0}, \end{cases} \implies \nu^* = \begin{cases} 0 & h_\phi(s^*) > 0, \\ \frac{\|\nabla_s U(s^*, \pi(s^*), h_\phi)\|_2}{\|\nabla_s h_\phi(s^*)\|_2} & h_\phi(s^*) = 0. \end{cases}$$

Now we move on to the efficient calculation of $s^*$.

**Computing the adversarial $s^*$.** Because the maximization problem with respect to $s$ is nonconcave, there could be multiple local maxima. In practice, we find that it is more efficient and reliable to use multiple local maxima to compute $\nabla_\phi C^*$ and then average the gradient.

Solving $s^*$ is highly non-trivial, as it is a non-concave optimization problem with a constraint $s \in \mathcal{C}_{h_\phi}$. To deal with the constraint, we introduce a Lagrangian multiplier $\lambda$ and optimize $U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_{h_\phi}}$ w.r.t. $s$ without any constraints. However, it is still very time-consuming to solve an optimization problem independently at each time. Based on the observation that the parameters of $h$ do not change too much by one step of gradient step, we can use the optimal solution from the last optimization

problem as the initial solution for the next one, which naturally leads to the idea of maintaining a set of candidates of $s^*$'s during the computation of $\nabla_\phi C^*$.

We use Metropolis-adjusted Langevin algorithm (MALA) [Besag, 1994] to maintain a set of candidates $\{s_1, \ldots, s_m\}$ which are supposed to sample from $\exp(\tau(U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_{h_\phi}}))$. Here $\tau$ is the temperature indicating we want to focus on the samples with large $U(s, \pi(s), h_\phi)$. Although the indicator function always have zero gradient, it is still useful in the sense that MALA will reject $s_i \notin \mathcal{C}_{h_\phi}$. A detailed description of MALA is given in Section 5.3.2.

For our purpose, as we seek to compute $C^*(h_\phi, U, \pi_\theta)$, we maintain $m = 10^4$ sequences of samples $\{\{s_t^{(i)}\}_t\}_{i \in [m]}$. Recall that $C^*$ involves a constrained optimization problem:

$$C^*(h_\phi, U, \pi_\theta) = \max_{s: h_\phi(s) \leq 1} U(s, \pi_\theta(s), h_\phi),$$

so for each $i \in [m]$, the sequence $\left\{s_t^{(i)}\right\}_t$ follows the Algorithm 10 to sample $s \sim \exp(\lambda_1 U(s, \pi_\theta(s), h_\phi) - \lambda_2 \mathbb{I}_{s \in \mathcal{C}_h})$ with $\lambda_1 = 30, \lambda_2 = 1000$. The step size $\tau$ is chosen such that the acceptance rate is approximately 0.6. In practice, when $s_t^{(i)} \notin \mathcal{C}_h$, we do not use MALA, but use gradient descent to project it back to the set $\mathcal{C}_h$.

We choose MALA over gradient descent because the maintained candidates are more diverse, approximate local maxima. If we use gradient descent to find $s^*$, then multiple runs of GD likely arrive at the same $s^*$, so that we lost the parallelism from simultaneously working with multiple local maxima. MALA avoids this issue by its intrinsic stochasticity, which can also be controlled by adjusting the hyperparameter $\tau$.

We summarize our algorithm of training barrier certificates in Algorithm 11 (which contains optional regularization that will be discussed in Section 5.5.2). At Line 2, the initialization of $s_i$'s is arbitrary, as long as they have a sort of stochasticity.

**Algorithm 11** Learning barrier certificate $h_\phi$ for a policy $\pi$ w.r.t. a calibrated dynamics model $M$.

**Require:** Temperature $\tau$, Lagrangian multiplier $\lambda$, and optionally a regularization function $\mathsf{Reg}$.

1: Let $U$ be defined as in Equation (5.4.1).
2: Initialize $m$ candidates of $s_1, \ldots, s_m \in \mathcal{S}$ randomly.
3: **for** $n$ iterations **do**
4: **for** every candidate $s_i$ **do**
5: sample $s_i \sim \exp(\tau(U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_h}))$ by MALA (Algorithm 10).
6: $W \leftarrow \{s_i : h_\phi(s_i) \geq 0, i \in [m]\}$.
7: Train $\phi$ to minimize $C^*(h_\phi, U, \pi) + \mathsf{Reg}(\phi)$ using all candidates in $W$.

## 5.5 Main Approach

In this section, we present our main algorithm, **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS), shown in Algorithm 12, to *iteratively* co-train barrier certificates, policy and dynamics model, using the algorithm in Section 5.4. In addition to parametrizing $h$ by $\phi$, we further parametrize the policy $\pi$ by $\theta$, and parametrize calibrated dynamics model $M$ by $\omega$. CRABS alternates between training a barrier certificate that certifies the policy $\pi_\theta$ w.r.t. a calibrated dynamics model $M_\omega$ (Line 5), collecting data safely using the certified policy (Line 3, details in Section 5.5.1), learning a calibrated dynamics model (Line 4, details in Section 5.5.3), and training a policy with the constraint of staying in the superlevel set of the barrier function (Line 6, details in Section 5.5.4).In the following subsections, we discuss how we implement each line in detail.

### 5.5.1 Safe Exploration with Certified Safeguard Policy

Safe exploration is challenging because it is difficult to detect irrecoverable states. The barrier certificate is designed to address this—a policy $\pi$ certified by some $h$ guarantees to stay within $\mathcal{C}_h$ and therefore can be used for collecting data. However, we may need more diversity in the collected data beyond what can be offered by the deterministic certified policy $\pi^{\text{safeguard}}$. Thanks to the contraction property **R.3**, we in

**Algorithm 12** CRABS: **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (Details in Section 5.5)

**Require:** An initial safe policy $\pi_{\text{init}}$.

Collected trajectories buffer $\widehat{D} \leftarrow \emptyset$; $\pi \leftarrow \pi_{\text{init}}$.

**for** $T$ epochs **do**

  Invoke Algorithm 13 to safely collect trajectories (using $\pi$ as the safeguard policy and a noisy version of $\pi$ as the $\pi^{\text{expl}}$). Add the trajectories to $\widehat{D}$.

  Learn a calibrated dynamics model $M$ with $\widehat{D}$.

  Learn a barrier certificate $h$ that certifies $\pi$ w.r.t. $M$ using Algorithm 11 with regularization.

  Optimize policy $\pi$ (according to the reward), using data in $\widehat{D}$, with the constraint that $\pi$ is certified by $h$.

**Algorithm 13** Safe exploration with safeguard policy $\pi^{\text{safeguard}}$

**Require:** (1) A policy $\pi^{\text{safeguard}}$ certified by barrier certificate $h$,
(2) Any proposal exploration policy $\pi^{\text{expl}}$.

**Require:** A state $s \in \mathcal{C}_{h_\phi}$.

Sample $n$ actions $a_1, \dots a_n$ from $\pi^{\text{expl}}(s)$.

**if** there exists an $a_i$ such that $U(s, a_i, h) \leq 1$ **then**

  **return:** $a_i$

**else**

  **return:** $\pi^{\text{safeguard}}(s)$.

fact know that any exploration policy $\pi^{\text{expl}}$ within the superlevel set $\mathcal{C}_h$ can be made safe with $\pi^{\text{safeguard}}$ being a safeguard policy—we can first try actions from $\pi^{\text{expl}}$ and see if they stay within the viable subset $\mathcal{C}_h$, and if none does, invoke the safeguard policy $\pi^{\text{safeguard}}$. Algorithm 13 describes formally this simple procedure that makes any exploration policy $\pi^{\text{expl}}$ safe. By a simple induction, one can see that the policy defined in Algorithm 13 maintains that all the visited states lie in $\mathcal{C}_h$. The main idea of Algorithm 13 is also widely used in policy shielding [Alshiekh et al., 2018, Jansen et al., 2018, Anderson et al., 2020], as the policy $\pi^{\text{safeguard}}$ sheilds the policy $\pi$ in $\mathcal{C}_{h_\phi}$.

The safeguard policy $\pi^{\text{safeguard}}$ is supposed to safeguard the exploration. However, activating the safeguard too often is undesirable, as it only collects data from $\pi^{\text{safeguard}}$

so there will be little exploration. To mitigate this issue, we often choose $\pi^{\text{expl}}$ to be a noisy version of $\pi^{\text{safeguard}}$ so that $\pi^{\text{expl}}$ will be roughly safe by itself. Moreover, the safeguard policy $\pi^{\text{safeguard}}$ will be trained via optimizing the reward function as shown in the next subsections. Therefore, a noisy version of $\pi^{\text{safeguard}}$ will explore the high-reward region and avoid unnecessary exploration.

Following Haarnoja et al. [2018], the policy $\pi_\theta$ is parametrized as $\tanh(\mu_\theta(s))$, and the proposal exploration policy $\pi_\theta^{\text{expl}}$ is parametrized as $\tanh(\mu_\theta(s) + \sigma_\theta(s)\zeta)$ for $\zeta \sim \mathcal{N}(0, I)$, where $\mu_\theta$ and $\sigma_\theta$ are two neural networks. Here the tanh is applied to squash the outputs to the action set $[-1, 1]$.

### 5.5.2 Regularizing Barrrier Certificates

The quality of exploration is directly related to the quality of policy optimization. In our case, the exploration is only within the learned viable set $\mathcal{C}_{h_\phi}$ and it will be hindered if $\mathcal{C}_{h_\phi}$ is too small or does not grow during training. To ensure a large and growing viable subset $\mathcal{C}_{h_\phi}$, we encourage the volume of $\mathcal{C}_{h_\phi}$ to be large by adding a regularization term

$$\mathsf{Reg}(\phi; \hat{h}) = \mathbb{E}_{s\in\mathcal{S}}[\text{relu}(\hat{h}(s) - h_\phi(s))],$$

Here $\hat{h}$ is the barrier certificate obtained in the previous epoch. In the ideal case when $\mathsf{Reg}(\phi; \hat{h}) = 0$, we have $\mathcal{C}_{h_\phi} \supset \mathcal{C}_{\hat{h}}$, that is, the new viable subset $\mathcal{C}_{h_\phi}$ is at least bigger than the reference set (which is the viable subset in the previous epoch.) We compute the expectation over $\mathcal{S}$ approximately by using the set of candidate $s$'s maintained by MALA.

In summary, to learn $h_\phi$ in CRABS, we minimize the following objective (for a small positive constant $\lambda$) over $\phi$ as shown in Algorithm 11:

$$\mathcal{L}(\phi; U, \pi_\theta, \hat{h}) = C^*(L_\phi, U, \pi_\theta) + \lambda\mathsf{Reg}(\phi; \hat{h}). \tag{5.5.1}$$

We remark that the regularization is not the only reason why the viable set $\mathcal{C}_{h_\phi}$ can grow. When the dynamics model becomes more accurate as we collect more data, the $\mathcal{C}_{h_\phi}$ will also grow. This is because an inaccurate dynamics model will typically make the $\mathcal{C}_{h_\phi}$ smaller—it is harder to satisfy **R.3** when the confidence region $M(s, \pi(s))$ in the constraint contains many possible states. Vice versa, shrinking the size of the confidence region will make it easier to certify more states.

### 5.5.3 Learning a Calibrated Dynamics Model

It is a challenging open question to obtain a dynamics model $M$ (or any supervised learning model) that is theoretically well-calibrated especially with domain shift [Zhao et al., 2020]. In practice, we heuristically approximate a calibrated dynamics model by learning an ensemble of probabilistic dynamics models, following common practice in RL [Yu et al., 2020, Janner et al., 2019, Chua et al., 2018b]. We learn $K$ probabilistic dynamics models $f_{\omega_1}, \ldots, f_{\omega_K}$ using the data in the replay buffer $\widehat{D}$. (Interestingly, prior work shows that an ensemble of probabilistic models can still capture the error of estimating a deterministic ground-truth dynamics model [Janner et al., 2019, Chua et al., 2018b].) Each probabilistic dynamics model $f_{\omega_i}$ outputs a Gaussian distribution $\mathcal{N}(\mu_{\omega_i}(s,a), \mathrm{diag}(\sigma^2_{\omega_i}(s,a)))$ with diagonal covariances, where $\mu_{\omega_i}$ and $\sigma_{\omega_i}$ are parameterized by neural networks. Given a replay buffer $\widehat{D}$, the objective for a probabilistic dynamics model $f_{\omega_i}$ is to minimize the negative log-likelihood:

$$\mathcal{L}_M(\omega_i) = -\mathbb{E}_{(s,a,s')\sim\widehat{D}}\left[-\log f_{\omega_i}(s'|s,a)\right]. \tag{5.5.2}$$

The only difference in the training procedure of these probabilistic models is the randomness in the initialization and mini-batches. We simply aggregate the means of all learn dynamics models as a coarse approximation of the confidence region, i.e., $M_\omega(s,a) = \{\mu_{\omega_i}(s,a)\}_{i\in[K]}$.

We note that we implicitly rely on the neural networks for the dynamics model to extrapolate to unseen states. However, local extrapolation suffices. The dynamics models' accuracy affects the size of the viable set—the more accurate the model is, the more likely the viable set is bigger. In each epoch, we rely on the additional data collected and the model's extrapolation to reduce the errors of the learned dynamics model on unseen states that are *near* the seen states, so that the learned viable set can grow in the next epoch. Indeed, in Section 5.7 (Figure 5.3) we show that the viable set grows gradually as the error and uncertainty of the models improves over epoch.

### 5.5.4 Policy Optimization

We describe our policy optimization algorithm in Algorithm 14. The desiderata here are (1) the policy needs certified by the current barrier certificate $h$ and (2) the policy has as high reward as possible. We break down our policy optimization algorithm into two components: First, we optimize the total rewards $R(\pi_\theta)$ of the policy $\pi_\theta$; Second, we use adversarial training to guarantee the optimized policy can be certified by $h_\phi$. The modification of SAC is to some extent non-essential and mostly for technical convenience of making SAC somewhat compatible with the constraint set. Instead, it is the adversarial step that fundamentally guarantees that the policy is certified by the current $h_\phi$.

As in original SAC, we maintain two $Q$ functions $Q_{\psi_i}$ and their target networks $Q_{\bar{\psi}_i}$ for $i \in \{1, 2\}$, together with a learnable temperature $\alpha$. The objective for the policy is to minimize

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s\sim\widehat{D}, a\sim\pi_\theta}\left[\alpha \log \pi_\theta^{\text{expl}}(a|s) - \hat{Q}_{\psi_1}(s,a)\right], \tag{5.5.3}$$

where $\hat{Q}_{\psi_1}(s,a) = Q_{\psi_1}(s,a)$ if $U(s,a,h) \leq 0$, otherwise $\hat{Q}_{\psi_1}(s,a) = -C - U(s,a,h)$ for a large enough constant $C$. The heuristics behind the design of $\hat{Q}_{\psi_1}$ is that we should lower the probability of $\pi_\theta^{\text{expl}}$ proposing an action which will possibly leave the superlevel set $\mathcal{C}_{h_\phi}$ to reduce the frequency of invoking the safeguard policy during exploration.

**Adversarial training** We use adversarial training to guarantee $\pi_\theta$ can be certified by $h_\phi$. Similar to what we've done in training $h_\phi$ adversarially, the objective for training $\pi_\theta$ is to minimize $C^*(h_\phi, U, \pi_\theta)$. Unlike the case of $\phi$, the gradient of $C^*(h_\phi, U, \pi_\theta)$ w.r.t. $\theta$ is simply $\nabla_\theta U(s^*, \pi_\theta(s^*), h_\phi)$, as the constraint $h_\phi(s)$ is unrelated to $\pi_\theta$. We also use MALA to solve $s^*$ and plug it into the gradient term $\nabla_\theta U(s^*, \pi_\theta(s^*), h_\phi)$.

**Optimizing $R(\pi_\theta)$** We use a modified SAC [Haarnoja et al., 2018] to optimize $R(\pi_\theta)$ for safety concerns.

The temporal difference objective for the $Q$ function is

$$\mathcal{L}_Q(\psi_i) = \mathbb{E}_{(s,a,r,s')\sim\widehat{D}}\mathbb{E}_{a'\sim\pi_\theta^{\text{expl}}(s')}\left[(Q_{\psi_i}(s,a) - (r + \gamma \min_{i\in\{1,2\}} Q_{\bar{\psi}_i}(s,a)))^2 \mathbb{I}_{U(s',a',h_\phi)\leq 0}\right], \tag{5.5.4}$$

We remark that we reject all $a' \sim \pi_\theta^{\text{expl}}(s')$ such that $U(s',a',h_\phi) > 0$, as our safe exploration algorithm (Algorithm 13) will reject all of them eventually. The temperature $\alpha$ is learned the same as in Haarnoja et al. [2018]:

$$\mathcal{L}_\alpha(\alpha) = \mathbb{E}_{s\sim\widehat{D}}[-\alpha \log \pi_\theta^{\text{expl}}(a|s) - \alpha\bar{\mathcal{H}}], \tag{5.5.5}$$

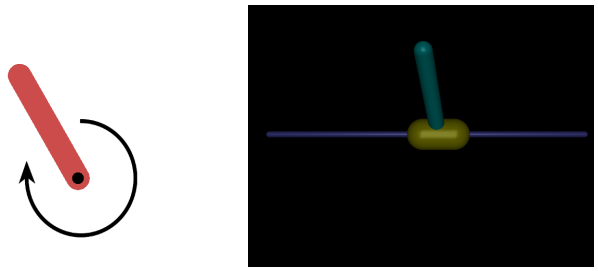
where $\bar{\mathcal{H}}$ is hyperparameter, indicating the target entropy of the policy $\pi_\theta^{\text{expl}}$.

As a side note, although we only optimize $\pi_\theta^{\text{expl}}$ here, $\pi_\theta$ is also optimized implicitly because $\pi_\theta^{\text{expl}}$ simply outputs the mean of $\pi_\theta$ deterministically.

**Algorithm 14** Modified SAC to train a policy while constraining it to stay within $\mathcal{C}_{h_\phi}$

**Require:** A policy $\pi$, the replay buffer $\widehat{D}$

1: Sample a batch $\mathcal{B}$ from buffer $\widehat{D}$.
2: Train $\theta$ to minimize $\mathcal{L}_\pi(\theta)$ using $\mathcal{B}$.
3: Train $Q$ to minimize $\mathcal{L}_Q(\psi_i)$ for $i \in \{1, 2\}$ using $\mathcal{B}$.
4: Train $\alpha$ to minimize $\mathcal{L}_\alpha(\alpha)$ using $\mathcal{B}$.
5: Invoke MALA to training $s^*$ adversarially (as in L4-5 in Algorithm 11).
6: Train $\theta$ minimize $C^*(h_\phi, U, \pi_\theta)$.
7: Update target network $\bar{\psi}_i$ for $i \in \{1, 2\}$.



(a) Pendulum (b) CartPole

Figure 5.1: Illustration of environments. The left figure illustrates the Pendulum environment, which is used by ***Upright*** and ***Tilt*** tasks. The right figer illustrates the CartPole environment, which is used by ***Move*** and ***Swing*** tasks.

## 5.6 High-risk, High-reward Environments

We design four tasks, three of which are high-risk, high-reward tasks, to check the efficacy of our algorithm. Even though they are all based on inverted pendulum or cart pole, we choose the reward function to be somewhat conflicted with the safety constraints. That is, the optimal policy needs to take a trajectory that is near the safety boundary. This makes the tasks particularly challenging and suitable for stress testing our algorithm's capability of avoiding irrecoverable states.

These tasks have state dimension dimensions between 2 to 4. We focus on the relatively low dimensional environments to avoid conflating the failure to learn accurate dynamics models from data and the failure to provide safety given a learned approximate dynamics model. Indeed, we identify that the major difficulty to scale up to high-dimensional environments is that it requires significantly more data to learn a

decent high-dimensional dynamics model that can predict long-horizon trajectories. We remark that we aim to have zero violations. This is very difficult to achieve, even if the environment is low dimensional. As shown by Section 5.7, many existing algorithms fail to do so.

(a) ***Upright***. The task is based on Pendulum-v0 in Open AI Gym [Brockman et al., 2016], as shown in Figure 5.1a. The agent can apply torque to control a pole. The environment involves the crucial quantity: the tilt angle $\theta$ which is defined to be the angle between the pole and a vertical line. The safety requirement is that the pole does not fall below the horizontal line. Technically, the user-specified safety set is $\{\theta : |\theta| \leq \theta_{\max} = 1.5\}$ (note that the threshold is very close to $\frac{\pi}{2}$ which corresponds to 90°.) The reward function $r$ is $r(s, a) = -\theta^2$, so the optimal policy minimizes the angle and angular speed by keeping the pole upright. The horizon is 200 and the initial state $s_0 = (0.3, -0.9)$.

(b) ***Tilt***. This action set, dynamics model, and horizon, and safety set are the same as in *Upright*. The reward function is different: $r(s, a) = -(\theta_{\text{limit}} - \theta)^2$. The optimal policy is supposed to stay tilting near the angle $\theta = \theta_{\text{limit}}$ where $\theta_{\text{limit}} = -0.41151684$ is the largest angle the pendulum can stay balanced. The challenge is during exploration, it is easy for the pole to overshoot and violate the safety constraints.

(c) ***Move***. The task is based on a cart pole and the goal is to move a cart (the yellow block) to control the pole (with color teal), as shown in Figure 5.1b. The cart has an $x$ position between $-1$ and 1, and the pole also has an angle $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ with the same meaning as ***Upright*** and ***Tilt***. The starting position is $x = \theta = 0$. We design the reward function to be $r(s, a) = x^2$. The user-specified safety set is $\{(x, \theta) : |\theta| \leq \theta_{\max} = 0.2, |x| \leq 0.9\}$ where 0.2 corresponds to roughly 11°. Therefore, the optimal policy needs to move the cart and the pole slowly in one direction, preventing the pole from falling down and the cart from going too far. The horizon is set to 1000.

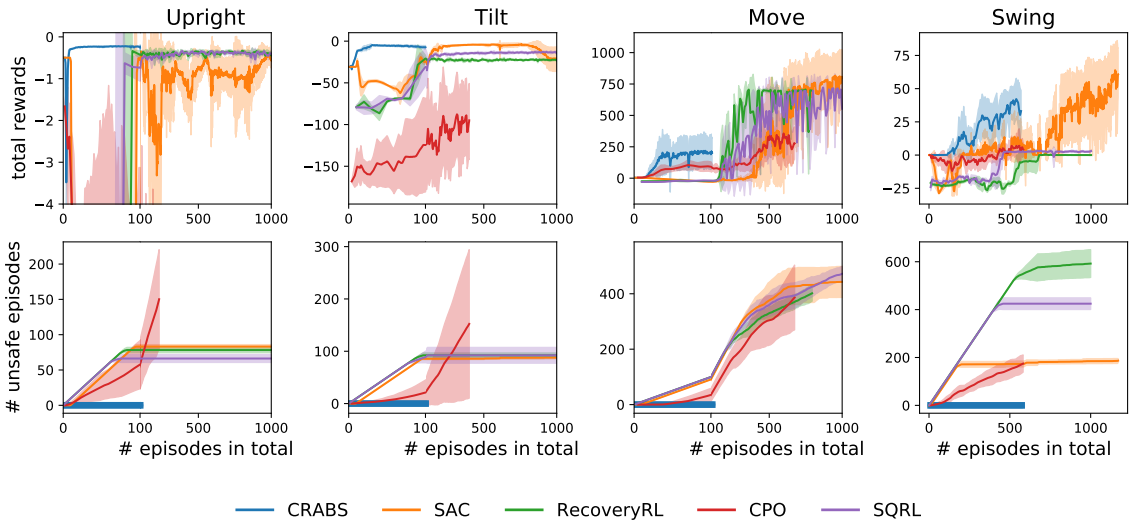



Figure 5.2: Comparision between CRABS and baselines. CRABS can learn a policy without any safety violations, while other baselines have a lot of safety violations. We run each algorithm four times with independent randomness. The solid curves indicate the mean of four runs and the shaded areas indicate one standard deviation around the mean.

(d) ***Swing***. This task is similar to ***Move***, except for a few differences: The reward function is $r(s,a) = \theta^2$; The user-specified safety set is $\{(x,\theta) : |\theta| \leq \theta_{\max} = 1.5, |x| \leq 0.9\}$. So the optimal policy will swing back and forth to some degree and needs to control the angles well so that it does not violate the safety requirement.

For all the tasks, once the safety constraint is violated, the episode will terminate immediately and the agent will receive a reward of -30 as a penalty. The number -30 is tuned by running SAC and choosing the one that SAC performs best with.

## 5.7 Experiments

In this section, we conduct experiments to answer the following question: Can CRABS learn a reasonable policy without safety violations in the designed tasks?

**Baselines.** We compare our algorithm CRABS against four baselines: (a) **Soft Actor-Critic** (SAC) [Haarnoja et al., 2018], one of the state-of-the-art RL algorithms,

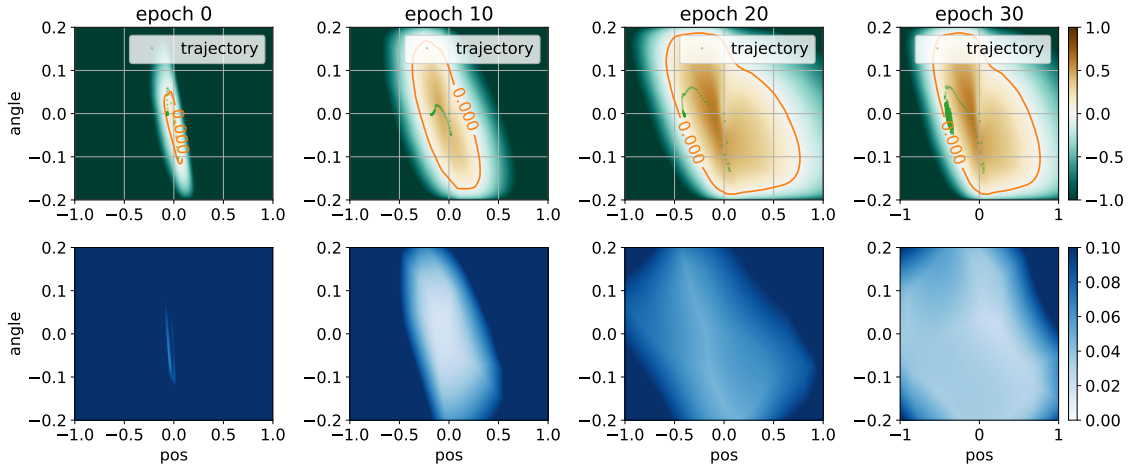



Figure 5.3: Visualization of the growing viable subsets learned by CRABS in ***Move***. To illustrate the 4-dimensional state space, we project a state from $[x, \theta, \dot{x}, \dot{\theta}]$ to $[x, \theta]$. **Top** row illustrates the learned $\mathcal{C}_{h_\phi}$ at different epochs. The red curve encloses superlevel set $\mathcal{C}_{h_\phi}$, while the green points indicate the projected trajectory of the current safe policy. We can also observe that policy $\pi$ learns to move left as required by the task. We note that shown states in the trajectory sometimes seemingly are not be enclosed by the red curve due to the projection. **Bottom** row illustrates the size of confidence region (defined by Equation (5.7.1)) of the dynamics model $M_\omega$ at different epochs. Darker states mean the learned dynamics model is less confident. We can observe that the learned dynamics model tends to be confident at more states after diverse data are collected.

(b) **Constrained Policy Optimization** (CPO) [Achiam et al., 2017], a safe RL algorithm which builds a trust-region around the current policy and optimizes the policy in the trust-region, (c) **RecoveryRL** [Thananjeyan et al., 2021] which leverages offline data to pretrain a risk-sensitive $Q$ function and also utilize two policies to achieving two goals (being safe and obtaining high rewards), and (d) **SQRL** [Srinivasan et al., 2020] which leverages offline data in an easier environment and fine-tunes the policy in a more difficult environment. SAC and CPO are given an initial safe policy for safe exploration, while RecoveryRL and SQRL are given offline data containing 40K steps from both mixed safe and unsafe trajectories which are free and are not counted. CRABS collects more data at each iteration in ***Swing*** than in other tasks to learn a better dynamics model $M$. For SAC, we use the default hyperparameters because

we found they are not sensitive. For RecoveryRL and SQRL, the hyperparameters are tuned in the same way as in Thananjeyan et al. [2021]. For CPO, we tune the step size and batch size. More details of experiment setup and the implementation of baselines can be found in Section 5.7.1.

**Results.** Our main results are shown in Figure 5.2. From the perspective of total rewards, SAC achieves the best total rewards among all of the 5 algorithms in ***Move*** and ***Swing***. In all tasks, CRABS can achieve reasonable total rewards and learns faster at the beginning of training, and we hypothesize that this is directly due to its strong safety enforcement. RecoveryRL and SQRL learn faster than SAC in ***Move***, but they suffer in ***Swing***. RecoveryRL and SQRL are not capable of learning in ***Swing***, although we observed the average return during exploration at the late stages of training can be as high as 15. CPO is quite sample-inefficient and does not achieve reasonable total rewards as well.

From the perspective of safety violations, CRABS surpasses all baselines **without a single safety violation**. The baseline algorithms always suffer from many safety violations. SAC, SQRL, and RecoveryRL have a similar number of unsafe trajectories in ***Upright, Tilt, Move***, while in ***Swing***, SAC has the fewest violations and RecoveryRL has the most violations. CPO has a lot of safety violations. We observe that for some random seeds, CPO does find a safe policy and once the policy is trained well, the safety violations become much less frequent, but for other random seeds, CPO keeps visiting unsafe trajectories before it reaches its computation budget.

**Visualization of learned viable subset $\mathcal{C}_{h_\phi}$.** To demonstrate that the algorithms work as expected, we visualized the viable set $\mathcal{C}_{h_\phi}$ in Figure 5.3. As shown in the figure, our algorithm CRABS succeeds in certifying more and more viable states and does not get stuck locally, which demonstrates the efficacy of the regularization at Section 5.5.2. We also visualized how confident the learned dynamics model is as

training goes on. More specifically, the uncertainty of a calibrated dynamics model $M$ at state $s$ is defined as

$$\mathsf{Uncertainty}(M, s) \triangleq \max_{s_1, s_2 \in M(s, \mathbf{0})} \|s_1 - s_2\|_2. \tag{5.7.1}$$

We can see from Figure 5.3 that the initial dynamics model is only locally confident around the initial policy, but becomes more and more confident after collecting more data.

**Handcrafted barrier function $h$.** To demonstrate the advantage of learning a barrier function, we also conduct experiments on a variant of CRABS, which uses a handcrafted barrier certificate by ourselves and does not train it, that is, Algorithm 12 without Line 5. The results show that this variant does not perform well: It does not achieve high rewards, and has many safety violations. We hypothesize that the policy optimization is often burdened by adversarial training, and the safeguard policy sometimes cannot find an action to stay within the superlevel set $\mathcal{C}_h$.

### 5.7.1 Implementation Details

Our code is implemented by Pytorch [Paszke et al., 2019] and runs in a single RTX-2080 GPU. Typically it takes 12 hours to run one seed for ***Upright***, ***Tilt*** and ***Move***, and for ***Swing*** it takes around 60 hours. In a typical run of ***Swing***, 33 hours are spent on learning barrier functions.

**Environment.** All the environments are based on OpenAI Gym [Brockman et al., 2016] where MuJoCo [Todorov et al., 2012b] serves as the underlying physics engine. We use discount $\gamma = 0.99$.

The tasks ***Upright*** and ***Tilt*** are based on `Pendulum-v0`. The obsevation is $[\theta, \dot{\theta}]$ where $\theta$ is the angle between the pole and a vertical line, and $\dot{\theta}$ is the angular velocity.

The agent can apply a torque to the pendulum. The task ***Move*** and ***Swing*** is based on `InvertedPendulum-v2` with observation $[x, \theta, \dot{x}, \dot{\theta}]$. The agent can control how the cart moves.

As all of the constraints are in the form of $\|\theta\| \leq \theta_{\max}$ and $|x| \leq x_{\max}$. For each type of constraint, we design $\mathcal{B}_{\text{unsafe}}$ to be

$$\mathcal{B}_{\text{unsafe}}(s) = \max\left(\omega\left(\theta/\theta_{\max}\right), \omega\left(x/x_{\max}\right)\right),$$

with $\omega(x) = \max(0, 100(|x|-1))$. If there is no constraint of $x$, we just take $\mathcal{B}_{\text{unsafe}}(s) = \omega\left(\theta/\theta_{\max}\right)$. One can easy check that $\mathcal{B}_{\text{unsafe}}(s)$ is continuous and equals to 1 at the boundary of safety set.

**Policy.** We parametrize our policy using a feed-forward neural network with ReLU activation and two hidden layers, each of which contains 256 hidden units. Similar to Haarnoja et al. [2018], the output of the policy is squashed by a tanh function.

The initial policy is obtained by running SAC for $10^5$ steps, checking the intermediate policy for every $10^4$ steps and picking the first safe intermediate policy.

In all tasks, we optimize the policy for 2000 steps in a single epoch.

**Dynamics Model.** We use an ensemble of five learned dynamics models as the calibrated dynamcis model. Each of the dynamics model contains 4 hidden layers with 400 hidden units and use Swish as the activation function [Ramachandran et al., 2017]. Following Chua et al. [2018b], we also train learnable parameters to bound the output of $\sigma_\omega$. We use Adam [Kingma and Ba, 2014] with learning rate 0.001, weight decay 0.000075 and batch size 256 to optimize the dynamics model.

In the experiment ***Move*** and ***Swing***, the initial model is obtained by traininng one a data for 20000 steps with 500 safe trajectories, obtained by adding different noises to the initial safe policy.

At each epoch, we optimize the dynamics models for 1000 steps.

**Barrier certificate $h$.** The barrier certificate is parametrized by a feed-forward neural network with ReLU activation and two hidden layers, each of which contains 256 hidden units. The coefficient $\lambda$ in Equation (5.5.1) is set to 0.001.

**Collecting data.** In ***Upright***, ***Tilt*** and ***Move***, the Line 3 in Algorithm 12 collects a single episode. In ***Swing***, the Line 3 collects six episodes, two of which are from Algorithm 13 with a uniform random policy, another two are from the current policy, and the remaining two are from the current policy but with more noises. In Algorithm 13, we first draw $n = 100$ Gaussian samples $\zeta_i \sim \mathcal{N}(0, I)$, and the sampled actions are $a_i = \tanh(\mu_\theta(s) + \zeta_i \sigma_\theta(s))$, where $\sigma_\theta(s)$ and $\mu_\theta(s)$ are the outputs of the exploration policy $\pi^{\text{expl}}$.

**RecoveryRL.** We use the code in `https://github.com/abalakrishna123/recovery-rl`. We remark that when running experiments in Recovery RL, we do not add the violation penalty for an unsafe trajectory. We set $\epsilon_{\text{risk}} = 0.5$ (chosen from $[0.1, 0.3, 0.7, 0.7]$) and discount factor $\gamma_{\text{risk}} = 0.6$ (chosen from $[0.8, 0.7, 0.6, 0.5]$). The offline dataset $\mathcal{D}_{\text{offline}}$, which is used to pretrain the $Q^{\pi}_{\text{risk}}$, contains 20K transitions from a random policy and another 20K transitions from the initial (safe) policy used by CRABS. The violations in the offline dataset is **not** counted when plotting.

Unfortunately, with chosen hyperparameters, we do not observe reasonable high reward from the policy, but we do observe that after around 400 episodes, RecoveryRL visits high reward (15-20) region in the ***Swing*** task and there are few violations since then.

**SAC.** We implement SAC ourselves with learned temperature $\alpha$, which we hypothesize is the reason of it superior performance over RecoveryRL and SQRL. The violation

penalty is chosen to be 30 from $[3, 10, 30, 100]$ by tuning in the ***Swing*** and ***Move*** task. We found out that with violation penalty being 100, SAC has slightly fewer violations (around 167), but the total reward can be quite low ($< 2$) after $10^6$ samples, so we choose to show the result of violation penalty being 30.

**SQRL.** We use code provided by RecoveryRL with the same offline data and hyperparameters. However, we found out that the $\nu$ parameter (that is, the Lagrangian multiplier) is very important and tune it by choosing the optimal one from $[3, 10, 30, 100, 300]$ in ***Swing***. The optimal $\nu$ is the same as that for SAC, which is 30. As SQRL and RecoveryRL use a fixed temperature for SAC, we find it suboptimal in some cases, e.g., for ***Swing***.

**CPO.** We use the code in `https://github.com/jachiam/cpo`. To make CPO more sample efficient and easier to compare, we reduce the batch size from 50000 to 5000 (for ***Move*** and ***Tilt***) or 1000 (for ***Tilt*** and ***Upright***). We tune the step size in $[0.02, 0.05, 0.005]$ but do not find substantial difference, while tuning the batch size can significantly reduce its sample efficiency, although it is still sample-inefficient.

## 5.8 Conclusion

In this chapter, we propose a novel algorithm CRABS for training-time safe RL. The key idea is that we co-train a barrier certificate together with the policy to certify viable states, and only explore in the learned viable subset. The empirical rseults show that CRABS can learn some tasks without a single safety violation. We consider using model-based policy optimization techniques to improve the total rewards and sample efficiency as a promising future work.

We focus on low-dimensional continuous state space in this chapter because it is already a sufficiently challenging setting for zero training-time violations, and we

leave the high-dimensional state space as an important open question. We observed in our experiments that it becomes more challenging to learn a dynamics model in higher dimensional state space that is sufficiently accurate and calibrated even under the training data distribution (the distribution of observed trajectories). Therefore, to extend our algorithms to high dimensional state space, we suspect that we either need to learn better dynamics models or the algorithm needs to be more robust to the errors in the dynamics model.

# Bibliography


Yasin Abbasi-Yadkori and Csaba Szepesvári. Regret bounds for the adaptive control of linear quadratic systems. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 1–26, 2011.

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017.

Shipra Agrawal and Randy Jia. Optimistic posterior sampling for reinforcement learning: worst-case regret bounds. In *Advances in Neural Information Processing Systems*, pages 1184–1194, 2017.

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.

Jacopo Aleotti and Stefano Caselli. Grasp recognition in virtual reality for robot pregrasp planning by demonstration. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2801–2806. IEEE, 2006.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. *arXiv preprint arXiv:2009.12612*, 2020.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba.

Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5): 469–483, 2009.

Kavosh Asadi, Dipendra Misra, and Michael L Littman. Lipschitz continuity in model-based reinforcement learning. *arXiv preprint arXiv:1804.07193*, 2018.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 263–272. JMLR. org, 2017.

J Andrew Bagnell. An invitation to imitation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 2015.

Michael Bain and Claude Sommut. A framework for behavioural claning. *Machine intelligence*, 15(15):103, 1999.

Peter L Bartlett and Ambuj Tewari. Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 35–42. AUAI Press, 2009.

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

Julian Besag. Comments on "representations of knowledge in complex systems" by u. grenander and mi miller. *J. Roy. Statist. Soc. Ser. B*, 56(591-592):4, 1994.

Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. Conservative safety critics for exploration. *arXiv preprint arXiv:2010.14497*, 2020.

Ross Boczar, Nikolai Matni, and Benjamin Recht. Finite-data performance guarantees for the output-feedback control of an unknown system. *arXiv preprint arXiv:1803.09186*, 2018.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. *ArXive-prints*, July 2018.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

Michael Carter. *Foundations of mathematical economics*. MIT press, 2001.

Jessica Chemali and Alessandro Lazaric. Direct policy iteration with demonstrations. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018a.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018b.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*, 2018.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Christoph Dann, Tor Lattimore, and Emma Brunskill. Unifying pac and regret: Uniform pac bounds for episodic reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5713–5723, 2017.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the sample complexity of the linear quadratic regulator. *CoRR*, abs/1710.01688, 2017. URL `http://arxiv.org/abs/1710.01688`.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. Regret bounds for robust adaptive control of the linear quadratic regulator. *Advances in Neural Information Processing Systems*, 31, 2018.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the sample complexity of the linear quadratic regulator. *Foundations of Computational Mathematics*, 20(4):633–679, 2020.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. 2011.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning. In *International Conference on Machine Learning*, pages 2627–2637. PMLR, 2020.

Priya L Donti, Melrose Roderick, Mahyar Fazlyab, and J Zico Kolter. Enforcing robust control guarantees within neural network policies. *arXiv preprint arXiv:2011.08105*, 2020.

Yilun Du and Karthik Narasimhan. Task-agnostic dynamics priors for deep reinforcement learning. *arXiv preprint arXiv:1905.04819*, 2019.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

Amir-massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494, 2017.

V Feinberg, A Wan, I Stoica, MI Jordan, JE Gonzalez, and S Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018a.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018b.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016a.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016b.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Ronald Ortner. Efficient bias-span-constrained exploration-exploitation in reinforcement learning. *arXiv preprint arXiv:1802.04020*, 2018.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pages 2021–2030, 2019.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018a.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Yang Gao, Ji Lin, Fisher Yu, Sergey Levine, Trevor Darrell, et al. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016a.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016b.

Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

Elad Hazan, Sham Kakade, and Karan Singh. The nonstochastic control problem. In *Algorithmic Learning Theory*, pages 408–421. PMLR, 2020.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Karl Hinderer. Lipschitz continuity of value functions in markovian decision processes. *Mathematical Methods of Operations Research*, 62(1):3–22, 2005.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

K Jetal Hunt, D Sbarbaro, R Żbikowski, and Peter J Gawthrop. Neural networks for control systems—a survey. *Automatica*, 28(6):1083–1112, 1992.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

Nils Jansen, Bettina Könighofer, Sebastian Junges, Alexandru C Serban, and Roderick Bloem. Safe reinforcement learning via probabilistic shields. *arXiv preprint arXiv:1807.06096*, 2018.

Wonseok Jeon, Seokin Seo, and Kee-Eung Kim. A bayesian approach to generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 7429–7439, 2018.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems*, pages 4863–4873, 2018.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *ArXiv*, abs/1903.00374, 2019.

S. Kakade, M. Wang, and L. F. Yang. Variance Reduction Methods for Sublinear Reinforcement Learning. *ArXiv e-prints*, February 2018.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206, 2017.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

S Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5): 943–957, 2011.

Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pages 2859–2867, 2013.

Kee-Eung Kim and Hyun Soo Park. Imitation learning via kernel mean embedding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. 2018.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Kailasam Lakshmanan, Ronald Ortner, and Daniil Ryabko. Improved regret bounds for undiscounted continuous reinforcement learning. In *International Conference on Machine Learning*, pages 524–532, 2015.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

Martin Lawitzky, Jose Ramon Medina, Dongheui Lee, and Sandra Hirche. Feedback motion planning and learning from demonstration in physical robotic assistance: differences and synergies. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3646–3652. IEEE, 2012.

Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1995–2003. JMLR. org, 2017.

Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*, 2018.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1): 1334–1373, 2016.

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

Rudolf Lioutikov, Alexandros Paraschos, Jan Peters, and Gerhard Neumann. Sample-based informationl-theoretic stochastic optimal control. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3896–3902. IEEE, 2014.

Yuping Luo and Tengyu Ma. Learning barrier certificates: Towards safe reinforcement learning with zero training-time violations. *Advances in Neural Information Processing Systems*, 34, 2021.

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019a.

Yuping Luo, Huazhe Xu, and Tengyu Ma. Learning self-correctable policies and value functions from demonstrations with negative sampling. *arXiv preprint arXiv:1907.05634*, 2019b.

Ali Malik, Volodymyr Kuleshov, Jiaming Song, Danny Nemer, Harlan Seymour, and Stefano Ermon. Calibrated model-based deep reinforcement learning. *arXiv preprint arXiv:1906.08312*, 2019.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Teodor Mihai Moldovan, Sergey Levine, Michael I Jordan, and Pieter Abbeel. Optimism-driven exploration for nonlinear systems. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3239–3246. IEEE, 2015.

Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *RoboticsandAutomation(ICRA),2016IEEEInternationalConferenceon*, pages 242–248. IEEE, 2016.

Jun Morimoto and Christopher G Atkeson. Minimax differential dynamic programming: An application to robust biped walking. In *Advances in neural information processing systems*, pages 1563–1570, 2003.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Quan Nguyen and Koushil Sreenath. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *2016 American Control Conference (ACC)*, pages 322–328. IEEE, 2016.

Frank Nielsen and Richard Nock. On the chi square and higher-order chi distances for approximating f-divergences. *IEEE Signal Processing Letters*, 21(1):10–13, 2014.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128, 2017.

Motoya Ohnishi, Li Wang, Gennaro Notomista, and Magnus Egerstedt. Barrier-certified adaptive reinforcement learning with applications to brushbot navigation. *IEEE Transactions on robotics*, 35(5):1186–1205, 2019.

Takayuki Osa, Amir M Ghalamzan Esfahani, Rustam Stolkin, Rudolf Lioutikov, Jan Peters, and Gerhard Neumann. Guiding trajectory optimization by demonstrated distributions. *IEEE Robotics and Automation Letters*, 2(2):819–826, 2017.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

Razvan Pascanu, Guido F Montufar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. 2013.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In

H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018.

Alexandre Piché, Valentin Thomas, Cyril Ibrahim, Yoshua Bengio, and Chris Pal. Probabilistic planning with sequential monte carlo methods. 2018.

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted bellman residual minimization handling expert demonstrations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 549–564. Springer, 2014.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 1394–1402, 2013.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Policy gradient in lipschitz markov decision processes. *Machine Learning*, 100(2-3):255–283, 2015.

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

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018a.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *International Conference on Learning Representations*, 2018b.

Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

Stephen Prajna and Anders Rantzer. On the necessity of barrier certificates. *IFAC Proceedings Volumes*, 38(1):526–531, 2005.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, pages 5690–5701, 2017.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Spencer M. Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems, 2018.

Melrose Roderick, Vaishnavh Nagarajan, and J Zico Kolter. Provably safe pac-mdp exploration using analogies. *arXiv preprint arXiv:2007.03574*, 2020.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. Sample efficient imitation learning for continuous control. 2018.

Igal Sason and Sergio Verdú. $f$-divergence inequalities. *IEEE Transactions on Information Theory*, 62(11):5973–6006, 2016.

Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

Yannick Schroecker and Charles L Isbell. State aware imitation learning. In *Advances in Neural Information Processing Systems*, pages 2911–2920, 2017.

Yannick Schroecker, Mel Vecerik, and Jon Scholz. Generative predecessor models for sample-efficient imitation learning. 2018.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Iulian Vlad Serban, Chinnadhurai Sankar, Michael Pieper, Joelle Pineau, and Yoshua Bengio. The bottleneck simulator: A model-based deep reinforcement learning approach. *arXiv preprint arXiv:1807.04723*, 2018.

Harshit Sikchi, Wenxuan Zhou, and David Held. Lyapunov barrier policy optimization. *arXiv preprint arXiv:2103.09230*, 2021.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3191–3199. JMLR. org, 2017b.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404.

Max Simchowitz, Horia Mania, Stephen Tu, Michael I Jordan, and Benjamin Recht. Learning without mixing: Towards a sharp analysis of linear system identification. *arXiv preprint arXiv:1802.08334*, 2018.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

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.

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3309–3318. JMLR. org, 2017.

Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018a.

Wen Sun, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Dual policy iteration. *arXiv preprint arXiv:1805.10755*, 2018b.

Wen Sun, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *Conference on Learning Theory*, pages 2898–2933, 2019.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2:160–163, 1990a.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990b.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*, 2012.

István Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1031–1038, 2010.

Erik Talvitie. Model regularization for stable sample rollouts. In *UAI*, pages 780–789, 2014.

Aviv Tamar, Dotan Di Castro, and Ron Meir. Integrating a partial model into model free reinforcement learning. *Journal of Machine Learning Research*, 13(Jun): 1927–1966, 2012.

Voot Tangkaratt, Syogo Mori, Tingting Zhao, Jun Morimoto, and Masashi Sugiyama. Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. *Neural networks*, 57:128–140, 2014.

Andrew J Taylor, Victor D Dorobantu, Hoang M Le, Yisong Yue, and Aaron D Ames. Episodic learning with control lyapunov functions for uncertain robotic systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6878–6884. IEEE, 2019.

Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012a.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012b.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Stephen Tu and Benjamin Recht. The gap between model-based and model-free methods on the linear quadratic regulator: An asymptotic viewpoint. *arXiv preprint arXiv:1812.03565*, 2018.

Matteo Turchetta, Andrey Kolobov, Shital Shah, Andreas Krause, and Alekh Agarwal. Safe reinforcement learning via curriculum induction. *arXiv preprint arXiv:2006.12136*, 2020.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329, 2017.

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

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Chris Xie, Sachin Patil, Teodor Moldovan, Sergey Levine, and Pieter Abbeel. Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 504–511. IEEE, 2016.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Accelerating safe reinforcement learning with constraint-mismatched policies. *arXiv preprint arXiv:2006.11645*, 2020.

Gu Ye and Ron Alterovitz. guided motion planning. In *Robotics research*, pages 291–307. Springer, 2017.

Michael C Yip and David B Camarillo. Model-less feedback control of continuum manipulators in constrained environments. *IEEE Transactions on Robotics*, 30(4): 880–889, 2014.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

Andrea Zanette and Emma Brunskill. Tighter problem-dependent regret bounds in reinforcement learning without domain knowledge using value function bounds. In *International Conference on Machine Learning*, pages 7304–7312, 2019.

Moritz A Zanger, Karam Daaboul, and J Marius Zöllner. Safe continuous control with constrained model-based policy optimization. *arXiv preprint arXiv:2104.06922*, 2021.

Jun Zeng, Bike Zhang, and Koushil Sreenath. Safety-critical model predictive control with discrete-time control barrier function. *arXiv preprint arXiv:2007.11718*, 2020.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

Shengjia Zhao, Tengyu Ma, and Stefano Ermon. Individual calibration with randomized forecasting. *arXiv preprint arXiv:2006.10288*, 2020.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.