Striving for Safe and Efficient Deep Reinforcement Learning

Harshit Sushil Sikchi

CMU-CS-20-136

December 10, 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David Held, Chair
Jeff Schneider

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Keywords: Machine Learning, Robotics, Reinforcement Learning, Safe Reinforcement Learning, Optimization, Model based Reinforcement Learning, Model free Reinforcement Learning, Planning, Trajectory Optimization, Inverse Reinforcement Learning, Imitation Learning

To my family, friends and teachers.

Abstract

Reinforcement Learning has seen tremendous progress in the past few years solving games like Dota and Starcraft, but little attention has been given to the safety of deployed agents. In this thesis, keeping safety in mind, we make progress in different dimensions of Reinforcement learning—Planning, Inverse RL, and Safe Model-Free RL.

Towards the goal of safe and efficient Reinforcement Learning, we propose:

1) A hybrid model-based model-free RL method, "Learning Off-Policy with Online Planning (LOOP)," which effectively combines Online Planning using learned dynamics models with a terminal value function for long-horizon reasoning. This method is favorable for ensuring safe exploration within the planning horizon, and we demonstrate that it achieves state-of-the-art performance competitive with model-based methods.

2) An Inverse Reinforcement Learning method f-IRL that allows specifying preferences using state-marginals or observations only. We derive an analytical gradient to match general f-divergence between agents and experts state marginal. f-IRL achieves more stable convergence than the Adversarial Imitation approaches that rely on min-max optimization. We show that f-IRL outperforms state-of-the-art IRL baselines in sample efficiency. Moreover, we show that the recovered reward function can be used in downstream tasks, and empirically demonstrate its utility on hard-to-explore tasks and for behavior transfer across changes in dynamics.

3) A model-free Safe Reinforcement Learning method, Lyapunov Barrier Policy Optimization (LBPO), that uses a Lyapunov-based barrier function to restrict the policy update to a safe set for each training iteration. Our method also allows the user to control the agent's conservativeness with respect to the constraints in the environment. LBPO significantly outperforms state-of-the-art baselines in terms of the number of constraint violations during training while being competitive in terms of performance.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. David Held, for his extraordinary guidance, patience, and for being a constant source of inspiration with his passion for robotics. I cannot thank him enough for helping me become a better researcher and for his proactive dedication and care in this chaotic time of pandemic and changing international student laws.

I would like to thank my mentor Wenxuan Zhou for helping me learn how to do systematic research and providing insightful advice whenever I hit a roadblock. She is an excellent collaborator to work with and an even better person. I thank Prof. Jeff Schneider for taking the time and being my committee member.

I also want to extend my appreciation to all the amazing people I met at the Robotics Institute - Siddharth Ancha, Yufei Wang, Tianwei Ni, Tejus Gupta, Brian Okorn, Xingyu Lin, Thomas Weng, Sujay Bajracharya, Qiao Gu, Gautham Narsimhan, Himangi Mittal, Benjamin Eysenbach and Lisa Lee. Robotics Institute provided me with a number of opportunities making my research journey quite exciting. I am lucky to be a part of the RPAD group and be amidst such a supportive atmosphere.

At last but most importantly, I would not be here if it were not for the unconditional support of my family, who supported all my decisions every step of the way.

Funding

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092, NSF S&AS grant number: IIS-1849154 and LG Electronics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Challenges

Reinforcement Learning (RL) has seen tremendous progress in the past few years. Games that were thought to be extremely difficult like Dota [13] and Starcraft [87], have been solved using RL. RL has been a leading contender for improving Robot Learning. RL for Robot Learning allows for better generalization and thus lets the roboticist avoid writing a hand-designed controller for each task, incorporate prior knowledge, and also allows for scaling up robotic systems to complex high dimensional state spaces like images. In this large-scale attempt to improve RL, little attention has been given to the safety of the deployed agents. RL agents effectively learn by trial and error, and when such agents are deployed on robots in the real world, they can cause damage to their own hardware and also to the other actors in the world. Safety is especially a concern when these robots are deployed in high stake situations like performing surgeries, house chores, or transport. Humans learn via experience, similar to an RL agent, but they respect notions of safety. For example, when a person is trying to learn to drive a car, he/she will often start driving slowly and gradually learn to improve performance, exploring carefully while avoiding accidents.

The inevitable ubiquity of robot presence in high stake real-life situations requires that researchers have a proper understanding of what safety means. A diverse set of notions for safety exist in previous literature with use-case specific formulation. Some popular notions are 1. Learning a policy that optimizes for reward obtained in the worst $\alpha\%$ of the cases, allowing the robot to be robust to the aleatoric uncertainty of the MDP [81, 43, 80, 14, 19]. 2. Learning a policy that maximizes the expected reward in a way that the expected costs are constrained to be within some threshold [5, 20, 31, 2], also known as the CMDP framework. 3. Other notions include safe exploration [12], avoiding reward hacking [7], safe interruptibility [60], distributional robustness [94]

and minimum side-effects [45].

Despite a wide array of definitions for safety, it is unclear if these are sufficient to ensure safe robot learning. Robot learning, currently, relies on heavy instrumentation [95, 24]. This involves hard-coding safety rules, developing recovery behaviors, or in general an emergency stop button. Despite the progress made, robotics is not at a stage where we can have complex robots in our home or workplace. A key question is what kind of safety definitions will allow for robot learning in the wild without extra instrumentation and how to incorporate them in a single learning framework.

Previous literature tackles a particular notion of safety, and most of them rely on strict assumptions like linear dynamics or quadratic cost functions and restrictive function classes to give provable guarantees on safety and do not scale well when state/action spaces become high-dimensional. An important shortcoming of these methods is that they are limited to simpler environments, often like gridworld. In general, we expect our agents to encounter high dimensional observations and a flexible representation of policy requires us to use function approximators like neural networks. We lack any theoretical guarantees of safety with neural networks and current work only addresses the problem of safety empirically in such settings. Moreover, algorithms like [2, 93], working with neural networks, lack sample efficiency. In this work, we work towards designing safe and efficient algorithms for the Deep RL setting.

Besides learning from scratch, a rich source of information to learn safety rules is via human demonstration which provides a wealth of knowledge, that a robot can exploit to improve its skills quickly by imitation. Understanding human demonstrations require understanding the intention of humans and replicating the behavior precisely. This idea is the motivation behind the subfield of Inverse RL, where we explore a new way to learn skills from demonstrations suitable in risk-averse settings. Ideally in the process of imitation, the robot should query the Human in times of uncertainty such that the process of learning to imitate the human expert does not have unintended consequences. The robot should *know what it knows* and ask questions rather than violate safety rules.

In conclusion, this thesis aims to provide foundational steps to ensure a safe transition of robot learning from simulators into our life. A future where we have robots doing our chores, transporting us to our workplaces, performing surgeries is quite near. An unsafe action can be fatal in these critical settings of close human interaction, and we would like to ensure that these robots rely on methods that have safety guarantees.

1.2 Contributions

In brief the following are contributions of this thesis:

1. **Online Planning with long horizon:** Online Planning lends itself easily to safety by ensuring cautious exploration under a probabilistic world model and also adapts faster to changes in dynamics. In our work, we tackle a fundamental limitation of Online Planning with learned models, which is reasoning over a finite horizon. We provide a new hybrid model-based model-free RL algorithm LOOP: Learning Off-Policy with Online Planning [74], where we extend limited horizon Online Planning to infinite horizon with an efficiently learned terminal value function. .
2. **Inverse RL with state marginals/observations:** Rewards are a popular way to encode preferences in RL but often end up encoding a different preference than the user intended, popularly known as reward hacking [7]. We explore another way to encode preferences - via specifying state marginals. Specifying the state marginal that you want the agent to visit is convenient and also allows encoding preferences for safety trivially, by marking the states that are unsafe with a density of zero. We propose an Inverse Reinforcement Learning method - fIRL [58], which, given the desired state marginal, extracts a reward function and a policy that has a visitation close to the given state marginal. The utility of this method extends far beyond safety, for e.g. in situations where it is hard to get demonstrations for a robot - a typical case when the robot is too high dimensional to control, or when only observations are given instead of the regular observation-actions, this method provides a way to extract portable reward functions that encode the intention of the demonstrator.
3. **Model-free safe RL:** Safety while training the agent is important if we expect our deployed agents to be online learners. To achieve this goal, we work under the Constrained Markov Decision Process (CMDP) framework, which guarantees that the expected cumulative cost incurred will be within a predefined threshold. Lyapunov functions allow us to convert the trajectory-based constraints of the CMDP to state-based constraints. Based on Lyapunov function, we formulate an objective which ensures that the policy update with the Lyapunov constraints remains in the safe set. The constraint is formulated as a log-barrier function and our method Lyapunov Barrier Policy Optimization [75] achieves near-zero constraint violations during training on the OpenAI safety gym benchmark. This work offers a practical algorithm that can be deployed on robots in the real-world with minimal constraint violations.

1.3 Thesis Organization

1. In Chapter 2, we present a new method "Learning Off-Policy with Online Planning". The Chapter is divided into sections: 1) Introduction and Related Work where we discuss the state of the art method in Model-based Reinforcement Learning motivate the need for our method. 2) Learning Off-Policy with Online Planning presents our core method 3) Experiments section shows the results of our method comparing it to previous state of the art. 4) Appendix section contains additional experiments and contains hyperparameters and other information regarding implementation.
2. In Chapter 3, we present a new algorithm for Inverse Reinforcement Learning given Observations only. This chapter is divided into sections: 1) Introduction and Related Work section discusses a number of previous IRL methods and motivates the need of our method. 2) In the next section "What Objective is optimized by previous IL algorithms", we present a novel analysis of previous methods and remark on which objective they truly optimize for in the Imitation Learning Setting. 3) In Preliminaries section, we discuss notations to be used in this chapter. 4) The section "Learning Stationary Rewards via State-Marginal Matching" discussed our proposed method. 5) In Experiments, we present a comparison of our method to state-of-the-art baselines and also its utility in downstream robotics tasks. 6) The appendix section provides proofs for our method and details for experiments performed in this work.
3. In Chapter 4, we present our new method for safety in model-free Reinforcement Learning setting. This chapted is divided into the following sections: 1) Related Work and Background provides a brief introduction to previous methods in Safe Reinforcement Learning and the background required for our method. 2) In the method section, we present our method Lyapunov Barrier Policy Optimization. 3) Experiments section compare our method to the state-of-the-art method in safe reinforcement learning. 4) In Appendix, we present derivations and additional implementation details for the experiments presented in the paper.
4. In Chapter 5, we present a summary of our work and conclusion.

Chapter 2

Learning Off-Policy with Online Planning

2.1 Introduction

Off-policy reinforcement learning is a widely used category of model-free reinforcement learning. It usually aims to learn a value function that encapsulates long-horizon reasoning of the future reward. The policy is obtained by directly taking the action that has the largest action-value [41] in discrete action spaces or by using a parameterized actor [50] in continuous settings. The effectiveness of the value function makes model-free reinforcement learning achieve state-of-the-art performance. However, it usually requires a huge amount of interactions with the environment.

Model-based reinforcement learning provides a mechanism for an agent to learn to perform a task by building a model of the environment through experience. It can scale to highly complex tasks while being orders of magnitude more sample efficient than model-free algorithms [40] [22]. One way to use the learned model is to perform online planning with the model [55] during both training and testing. At each timestep, the agent selects the best action by imagining possible rollouts with its learned model using Model Predictive Control (MPC). However, the number of timesteps it looks into the future is usually fixed to a small number. This is because the required computation for planning grows exponentially with the horizon. At the same time, the accuracy of the learned model usually deteriorates with longer horizons [25]. Thus, this method often suffers from myopic decisions for complex tasks.

To combine the advantages of sample efficiency of model-based reinforcement learning and long-horizon reasoning of model-free reinforcement learning, we propose Learning Off-Policy with Online Planning (LOOP). During online planning, it augments the model-based rollout trajectories with a terminal value function learned using off-policy model-free reinforcement learning. We refer to this policy as the MPC policy. In this way, the agent can select actions by evaluating

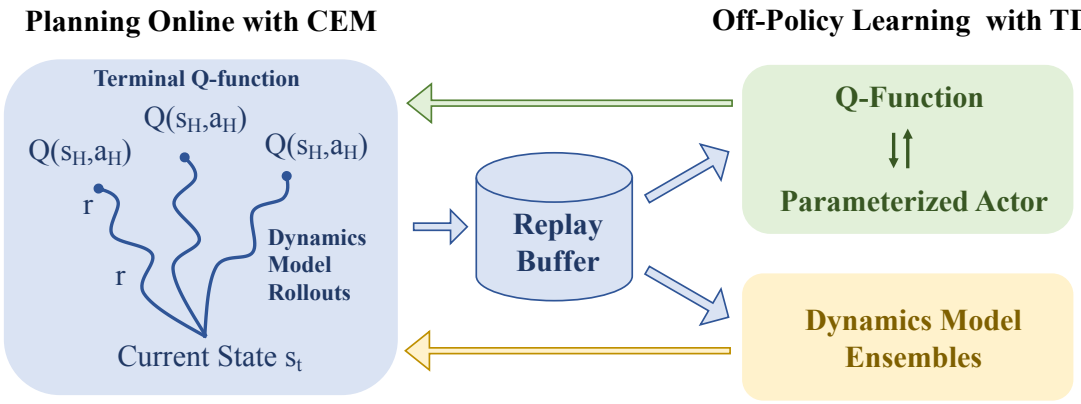


Figure 2.1: Overview: We use online planning with learned dynamic models and a terminal Q-function as the behavior policy. The transitions are saved into the replay buffer to train the Q-function and the dynamics model. The parameterized actor is also used to guide online planning.

short-horizon model rollouts as well as the future consequences after the rollout terminates using the value function. From another perspective, this model-based planning agent can be treated as the behavior policy for the off-policy algorithm that it uses to obtain the value function. Thus, compared to the underlying off-policy algorithm, LOOP can interact with the environment more effectively by planning over the learned model.

This combination introduces an issue in learning the value function. Directly estimate the value function for the MPC policy is computationally inefficient. Instead, if we use the MPC policy as a behavior policy and learn the value-function using an off-policy algorithm, the parameterized actor that is used to update the value function might diverge from the behavior policy and cause the "extrapolation issue" [29] in Q-learning. We identify this divergence to be critical in this combination of model-based and model-free reinforcement learning. We propose actor-guided trajectory optimization which improves learning performance by guiding trajectory optimization using the model-free actor. We evaluate our method on OpenAI Gym MuJoCo environments and demonstrate that the final algorithm is able to learn more efficiently than the underlying model-based and off-policy method.

2.2 Related Work

Model-free reinforcement learning algorithms achieve high performance for a lot of tasks, but these methods are notoriously sample-inefficient. Particularly, on-policy methods like TRPO (Schulman et al. [72]) and PPO (Schulman et al. [73]) require new samples to be collected for every update to

the policy. For instance, learning in-hand dexterous manipulation Andrychowicz et al. [8] required 3 years of simulated experience to learn without domain-randomization. Off-policy methods like SAC [34] and TD3 [30], on the other hand, are more sample-efficient than on-policy methods, as they utilize all the experiences obtained in the past. We use TD3 in our setup due to its simplicity and ease of analysis.

Model based-RL has seen a surge of interest recently, as the benefits involve reducing the sample complexity while maintaining asymptotic performance. Previous work approach model-based reinforcement learning using a learned model and trajectory optimization [22, 55]. These methods can reach asymptotic performance when a large enough planning horizon is used. This method can also scale to tasks like rotating Baoding balls in hand as demonstrated in [55], but has the limitation of not being able to reason for rewards beyond the planning horizon. Increasing the planning horizon increases the number of trajectories that should be sampled, and incurs a heavy computing cost.

Another line of work attempts to get the best of both model-free and model-based reinforcement learning, Feinberg et al. [25] and Buckman et al. [17] uses the model to improve target value estimates and thus accelerates model-free reinforcement learning. Schrittwieser et al. [71] uses Monte-Carlo Tree search with value estimates to constrain the length and uses a policy to constrain the width of the search tree. Their method utilizes on on-policy samples to train the Q-function, making it sample inefficient. It works on discrete action spaces with a latent dynamics structure. Hamrick et al. [36] combines MCTS with Q-learning but work under the setting of known model and discrete space.

The most related work to ours is Lowrey et al. [53] where they use trajectory optimization in the form of Model Predictive Control (MPC) as the behavior policy, and updates the Q function by obtaining a target lookahead value estimate using another instance of trajectory optimization. This method is extremely slow as each batch of sampled data for training the Q-function will require instances of trajectory optimization that scales with batch size. Moreover, they consider access to ground truth dynamics. Our method uses trajectory optimization using a learned model and a terminal value function as an exploratory policy, but the Q function updates are performed entirely off policy.

2.3 Learning Off-Policy with Online Planning

In LOOP, we use trajectory optimization with a terminal value function as the behavior policy that interacts with the environment. This trajectory optimization in its naive form is myopic and in

many cases may not produce optimal policies for the task, since it does fixed horizon planning using a learned model. We address this deficiency by having a value function that reasons for the expected long term rewards under the policy. Different from Lowrey et al. [53], we propose to utilize advances in off-policy learning to make the algorithm computationally efficient. In this section, we start by discussing the methods for learning the value function and the dynamics model. After that, we will further discuss how these two are combined and how we deal with the actor-divergence issue. Going forward, we will use the notation π_{opt} to denote the MPC policy that uses trajectory optimization by forward simulation over learned models in the model-based part and π_ϕ to denote the parameterized actor in the model-free part.

2.3.1 Learning the Value Function with Model-free Actor

To learn a value function, we build our method upon TD3 [30] which is an off-policy algorithm with an actor π_ϕ and a value function Q_θ. Note that other off-policy algorithms can also be used here. We refer to the actor π_ϕ used to update the value function as the "parameterized actor", in the sense that this actor is a parameterized function, in our case a neural network. It will only be used to update the value function and not be used to collect data as standard TD3. The target value is calculated based on the bellman equation:

$$Q^{\text{target}}(s_t, a_t) = r(s_t, a_t) + \gamma Q'_\theta(s_{t+1}, \pi_\phi(s_{t+1})) \tag{2.1}$$

To reduce overestimation error, TD3 calculates the target value by taking the minimum over two value functions Q_{θ_1} and Q_{θ_2}, also called Clipped Double Q learning. Finally the value function is updated by minimizing the mean squared error:

$$\text{MSE} = E_{(s,a)\sim D}[Q^{\text{target}}(s, a) - Q_{\theta_1,\theta_2}(s, a))]^2 \tag{2.2}$$

where s, a are the state action pairs sampled from D, the replay buffer of past experiences. The policy is updated by maximizing $Q_{\theta_1}(s, \pi_\phi(s))$.

2.3.2 Learning the Dynamics Model

Using the transitions collected, we train a dynamics model using supervised learning. Given a state-action pair (s, a), the network is trained to regress the difference δ between the next state and the current state, parameterized as a Gaussian distribution with a diagonal covariance matrix. Following Nagabandi et al. [55], we use probabilistic ensembles of dynamics models that capture the epistemic uncertainty as well as the aleatoric uncertainty in forward predictions [22]. Each

model in the ensemble is initialized with different weights and samples shuffled batches of data during training.

2.3.3 Trajectory Optimization with a Terminal Value Function

Given the dynamics model, the trajectory optimization policy π^{opt} is the MPC-based policy that uses Cross-Entropy Method (CEM) to select actions. Cross-Entropy Method is a strong optimizer and is shown [55] to perform much better than the random-shooting methods. This MPC policy will be used as the behavior policy to collect data in the environment. For each timestep, this policy will sample n action sequences $(a_1, a_2, ...a_H)$, up to a fixed horizon from a sampling distribution, and use the probabilistic dynamics model to unroll the trajectory resulting from the action sequence. The cumulative return for each rollout is calculated by

$$G = \sum_{i=0}^{H-1} (\gamma^i r(s_i, a_i)) + \gamma^H Q_\theta(s_{H+1}, a_{H+1}) \tag{2.3}$$

Note that in previous model-based reinforcement learning methods such as [55][22], the last term $Q(s_H, a_H)$ is often eliminated, resulting in an optimization of the action sequences over a fixed horizon, which might be shortsighted and result in a suboptimal trajectory sequence. The top e highest scoring actions sequences, also called elites, are selected and used to refine the sampling distribution from which the action sequences are sampled from.

$$\begin{aligned} A_i = \{a_0^i, a_1^i, .., a_H^i\}, A_i \sim \mathcal{N}(\mu^m, \Sigma^m) \, \forall i \in n \\ A_{\text{elites}} &= \text{sort}(A_i)[-e:] \\ \mu^{m+1} &= \alpha * \text{mean}(A_{\text{elites}}) + (1-\alpha)\mu^m \\ \Sigma^{m+1} &= \alpha * \text{var}(A_{\text{elites}}) + (1-\alpha)\Sigma^m \end{aligned} \tag{2.4}$$

After N iterations of refinement, we take the mean of the resulting action distribution as the final output. Following MPC, only the first action of the sequence is executed. For each subsequent timestep, replanning is performed. The transitions will be collected into the replay buffer, which will be used to train the models and the Q-function.

2.3.4 Actor-guided Trajectory Optimization (CEM-AG)

Combining trajectory optimization and off-policy learning in the way described above might suffer from the issue of "actor divergence": There is a mismatch between the state-action distribution induced by the model-free actor and the state-action distribution of the MPC-based behavior policy

Algorithm 1: LOOP

Initialize parameterized actor π_ϕ, Q-function Q_θ, predictive model p_ψ, Dataset D, Planning Horizon H
Collect some transitions sampled from a random policy into dataset D.
for $i \leftarrow 1$ **to** $Iter$ **do**
    for *CEM iterations* **do**
        Set the CEM objective to be: $J_{\text{CEM}} = \sum_{t=0}^{H-1}(\gamma^t r(s_t, a_t)) + \gamma^H Q_\theta(s_{H+1}, a_{H+1})$
        Sample state-action trajectories using a mixture distribution of the sampling distribution(gaussian) and the actions suggested by the parameterized actor unrolled with the model.
        Update the mean and the variance of the sampling distribution using Equation 2.4.
    end
    Select the updated mean to perform an action in the environment. Add the transition (s, a, r, s') to D.
    for *M epochs every K timesteps* **do**
        Update model ensemble parameters(ψ) on dataset D using maximum likelihood.
    end
    for *G gradient updates* **do**
        Update policy parameters(ϕ, θ) on dataset D, using any off-policy RL algorithm
    end
end

that collects data. As discussed in Fujimoto et al. [29], the mismatch will lead to extrapolation errors in Q-learning and cause persistent overestimation bias that deteriorates performance. In our experiments, we observe that naively using the method in section C sometimes results in worse performance than the original off-policy method due to this issue.

To mitigate the issue of actor divergence, we propose Actor-guided Trajectory Optimization. At each timestep, we use the model-free actor to propose a sequence of trajectories by rolling out the dynamics model. These trajectories will be included in the batch of samples in each CEM iteration. When these trajectories are selected as elites in CEM, they will guide the optimization and bring the solution closer to the actor distribution. In our experiments, we observe that CEM-AG effectively decreases actor divergence and improves the performance of LOOP. The full algorithm is summarized in Algorithm 3.

One alternative to solve the issue of actor divergence is to incorporate the divergence as an additional cost in CEM using Kullback-Leibler (KL) divergence or L_2 distance. In practice, we found this to be overly conservative and limits the performance benefits that result from trajectory optimization.

2.4 Experimental Results

We benchmark the proposed method LOOP on four OpenAI Gym MuJoCo environments: HalfCheetah, Hopper, Walker, and InvertedPendulum. First, we evaluate the sample efficiency and performance of LOOP comparing to the underlying algorithms that it is built upon. We further analyze the effect of Actor-guided CEM in reducing actor divergence.

2.4.1 Implementation Details

We use a horizon length of 5 for all the environments except Walker (uses horizon=3). We use the author's implementation of TD3 with the original hyperparameters. The dynamics model ensemble has 5 neural networks each consisting of 4 hidden layers, 200 hidden units each. We list all the hyperparameters in Appendix 2.5.1. The CEM optimizer uses 200 particles, sampled from a multivariate Gaussian distribution. Each action sequence is passed through each of the dynamics models and the average return is used as the maximization objective in CEM. The MPC policy will be used for the evaluation of LOOP by default. For CEM-AG we use 1 trajectory from π_ϕ for every 20 trajectories from sampling distribution. We use 5 random seeds to account for variability in training.

2.4.2 Improvement over the underlying model-based and model-free algorithm

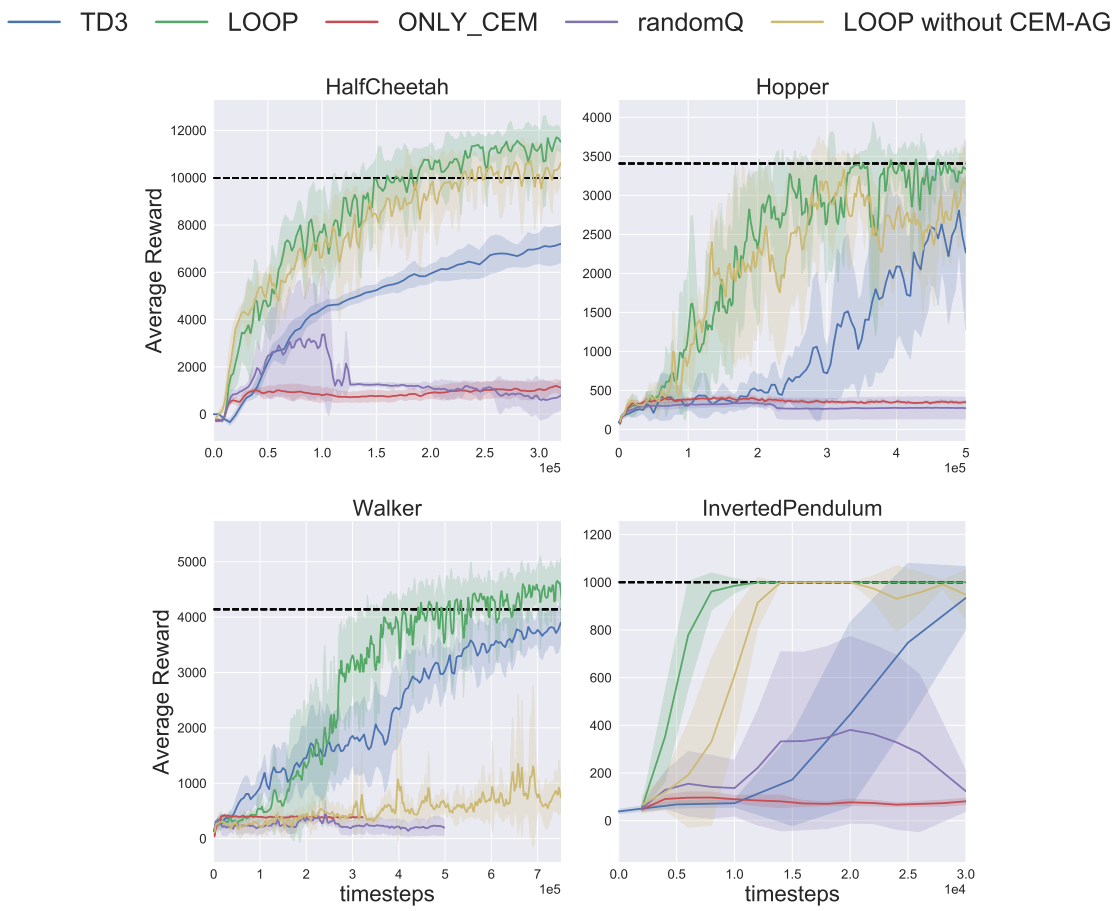


Figure 2.2: Training Performance of LOOP and its baselines on MuJoCo tasks. Green: LOOP. Blue: TD3. Red: CEM without a terminal value function. Violet: The terminal Q function is set to be evaluation of a random policy. Yellow: LOOP without CEM-AG. Dashed line: the performance of TD3 at 1e6 timesteps.

As shown in Figure 2.2, LOOP has significant performance gains over TD3 in all of the four environments due to better exploration. The red curves show the performance of CEM with the same planning horizon as LOOP but without the terminal Q-function, similar to Nagabandi et al. [55]. We observe that fixed-horizon CEM performs poorly in MuJoCo tasks due to the short planning horizon.

2.4.3 Comparison to previous model-based methods

We observe that LOOP when built on top of SAC as the off-policy model-free sub-module is able to achieve comparable results to state of the art model based algorithm MBPO [40]. We use author's implementation without additional hyper-parameter tuning for MBPO.

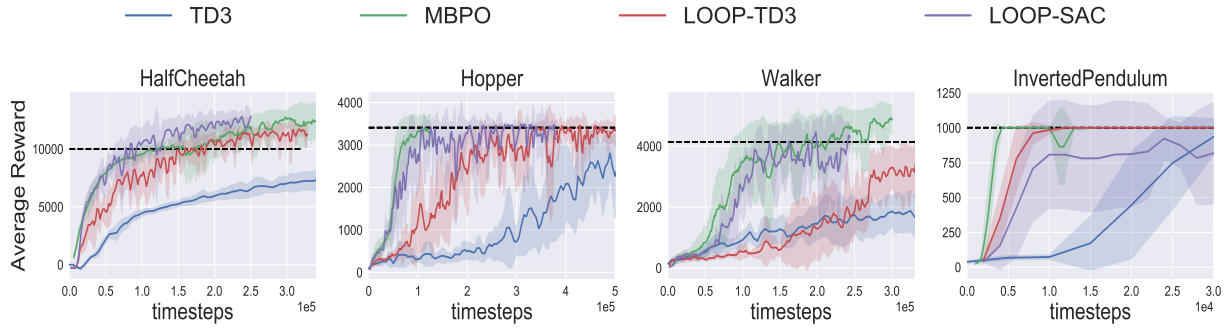


Figure 2.3: Comparison of learning performance with MBPO, a mdoel-based RL algorithm built on top of SAC.

2.4.4 Ablation Studies

To better understand the importance of different components in our method, we do ablation studies on the actor-guided CEM and the parameterized actor.

First, we evaluate the importance of actor-guided CEM comparing to normal CEM. Without using Actor-guided CEM (CEM-AG), Walker completely fails. The performance of other environments also drops and sometimes becomes more unstable. There are two hypotheses of how actor-guided CEM helps. First, the trajectories proposed by the parameterized actor might sometimes provide a better solution to this optimization problem than CEM alone. However, we observe that actor-guided CEM usually achieves similar reward as original CEM (Appendix E). Second, actor-guided CEM biases the solutions of CEM towards the parameterized actor, and thus reduces extrapolation issue for off-policy learning. In Figure 2.4, we compute the L_2 distance between the action proposed by the MPC policy (final output from CEM) and the TD3 policy. We observe that actor-guided CEM in LOOP is indeed able to reduce the actor-divergence compared to normal CEM. In Appendix C, we also include the plot for "actor usage" in CEM by measuring the fraction of time when the trajectories suggested by the parameterized actor are selected to be the elites in CEM. It further demonstrates that the actor-proposed trajectories are biasing the sampling distribution of CEM.

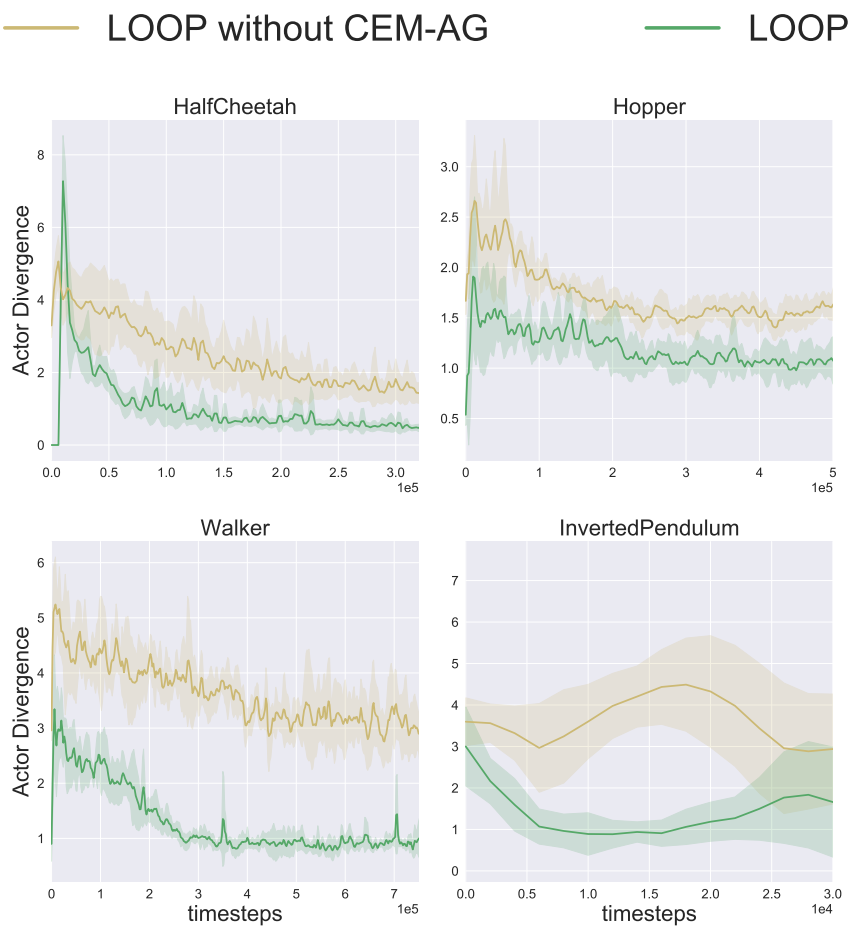


Figure 2.4: Actor-guided CEM reduces actor-divergence between the MPC policy and the model-free actor. Green: LOOP. Yellow: LOOP without actor-guided CEM (CEM-AG)

Next, instead of training the Q-function over the parameterized actor in off-policy learning, we train the Q-function over a randomly initialized policy. The motivation behind this experiment is that the parameterized actor in LOOP usually results in a bad evaluation performance comparing to the MPC policy. Thus, we want to verify whether training the parameterized actor is critical to obtain an informative Q-function for the MPC policy. In this experiment, we observe that the performance drastically drops if we replace the actor by a random policy. This indicates that the parameterized actor does provide meaningful information for the MPC policy to reason for the cumulative return beyond the planning horizon.

2.5 Appendix

2.5.1 Implementation Details

We benchmark the proposed method LOOP on four OpenAI Gym MuJoCo tasks: HalfCheetah-v2, Hopper-v2, Walker2d-v2, and InvertedPendulum-v2. Figure 2.5 show what the environments look like.

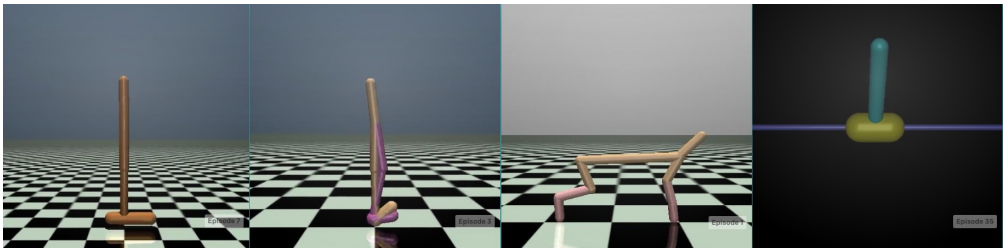


Figure 2.5: Left to Right: Hopperv2, Walker2d-V2, HalfCheetah-v2,InvertedPendulum-v2

Our probabilisitc ensemble of dynamics model resembles the one used in Chua et al. [22]. We use the TS-∞ sampling, and aggregate the total return by taking an return average across trajectory samples similar to [22] (Section 5.1). The hyperparameters used for the dynamics model, TD3 learning and the CEM controller are shown below:

2.5.2 Effect of horizon on LOOP

Horizon is a hyperparameter in LOOP and can be tuned to get the best performance. We test of horizon of 3, 5 and 7. Figure 2.6 shows the performance of LOOP with different planning horizon. We observe that horizon of 5 works best in HalfCheetah, Walker, Hopper and horizon of 3 for Walker.

2.5.3 Actor-usage in CEM-AG

To further verify if the actor-suggested actions are any good and influence the sampling distribution for CEM, we plot the fraction of actor-suggested samples that make it to the elites of CEM each timestep. The fraction is averaged over the number of iterations of CEM.

Table 2.1: LOOP Hyperparameters

Environments	**HalfCheetah**	**Hopper**	**Walker2d**	**InvertedPendulum**
Total Timesteps	1×10^6	1×10^6	1×10^6	3×10^4
Environment Steps per episode	1000			
Model Update frequency (K)	250			
Model Update epochs (M)	5			
Ensemble Size	5			
Network Architecture	MLP with 4 hidden layers of size 200			
Model Horizon (H)	5	5	3	5
Model Learning rate	0.001			
Policy update per environment step (G)	1			
Replay Buffer Size	1e6			

Table 2.2: CEM Hyperparameters

Hyperparamater	**Value**
Planning Horizon	5
Population Size (n)	100
Elites (e)	20
Number of Particles	4
Alpha α	0.25
Iterations (N)	5
Actor Mix	5 %

2.5.4 Replay Buffer Augmentation

It is observed that the performance of the parameterized actor(π_ϕ) during evaluation is quite poor comparing to the model-based policy. Wang and Ba [91] uses Behavior Cloning to distill the search policy into π_ϕ but still observe poor performance. π_ϕ suffers from covariate shift issue since it is never explicitly trained on the state-distribution that it encounters when it is deployed (since it is

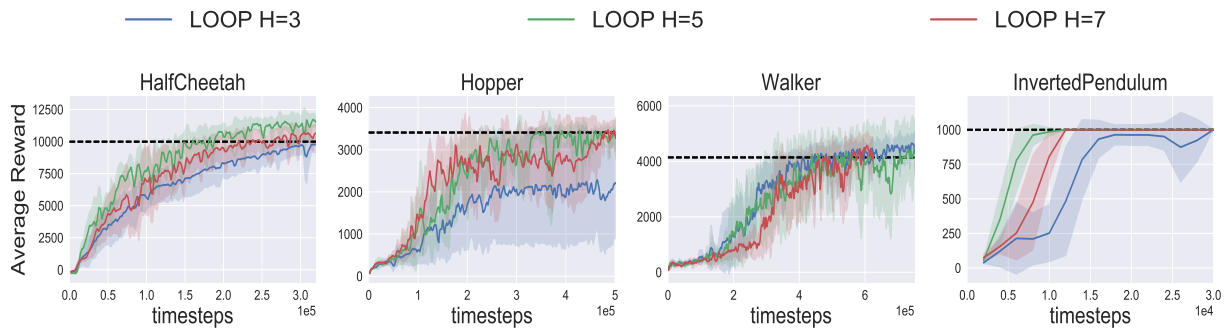


Figure 2.6: LOOP Plan horizon comparison

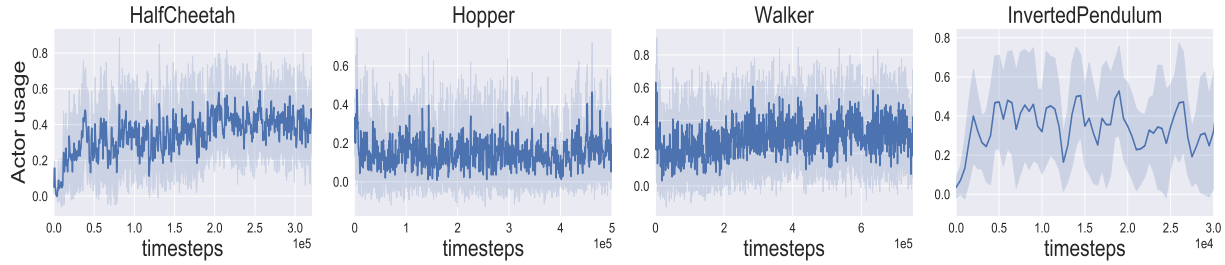


Figure 2.7: Actor-usage in LOOP while using CEM-AG: Fraction of actor suggested trajectories that make it to the elites in CEM.

not the behavior policy). For instance, if the policy π_ϕ were to make a small mistake and land in a state that is out of the distribution of states on which Q-function is trained on, the Q value might be arbitrarily bad. We can further modify LOOP to improve π_ϕ performance by training it on a mixture of simulated transitions that are observed during CEM and the real transitions, thus increasing the support of state-action marginal on which Q-function is trained to make π_ϕ more robust. We randomly sample a set of transitions encountered during CEM search procedure, and put them in a separate replay buffer($\mathcal{M}$). This replay buffer has a smaller size than the regular one to ensures that it only contains the samples generated by the latest dynamics model. In our experiments, we sample real transitions with 0.7 probability and transitions from $\mathcal{M}$ with 0.3 for off-policy learning. The size of $\mathcal{M}$ is set to be 12,000. We observe an improvement in the performance of π_ϕ as shown in Figure 2.9.

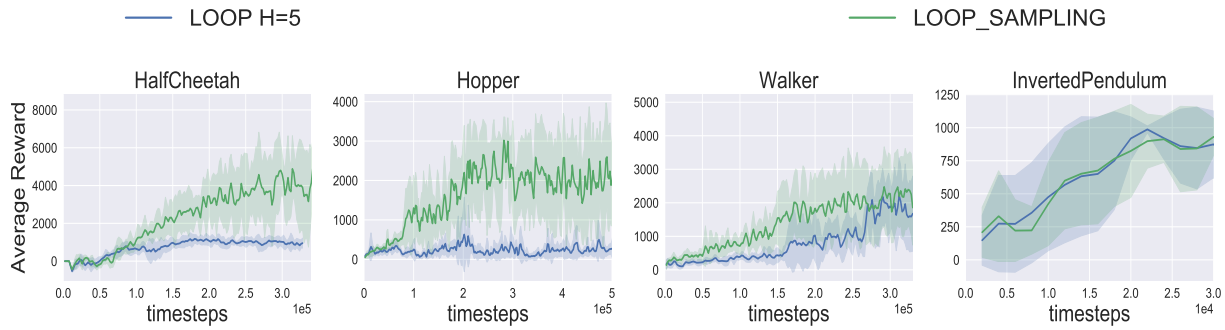


Figure 2.8: Replay buffer augmentation allows improves parameterized actor's performance.

2.5.5 Is CEM-AG a better optimizer than CEM?

We do an experiment to check if CEM-AG is able to find a better solution in trajectory space (with higher return) than CEM. To do this, we run LOOP, but additionally at each timestep in the environment we run CEM-AG and CEM separately and plot the difference in the cumulative discounted return ($\sum_{i=1}^{H-1}(\gamma^{i-1}r(s_i,a_i)) + \gamma^{H-1}Q_\theta(s_H,a_H)$) of their optimized solution. In Figure **??** we see that this is not the case, both optimized solution from both CEM and CEM-AG have similar reward. CEM-AG does offer improvement over the trajectories suggested by the parameterized actor, that are used to guide the CEM.

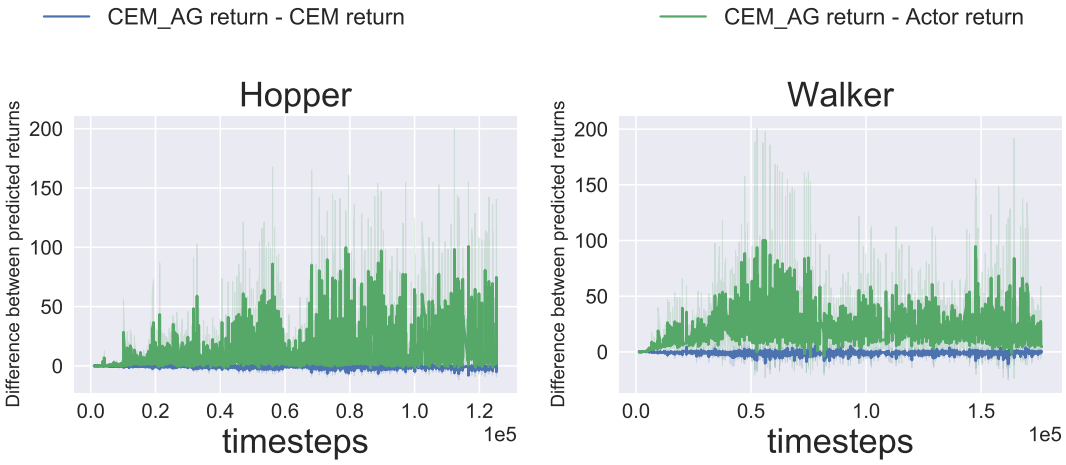


Figure 2.9: Blue Curve shows the difference between optimized return of CEM-AG and CEM. Green curve shows the difference between optimized return of CEM-AG and the mean return of the raw trajectories suggested by the parameterized actor.

Chapter 3

f-IRL: Inverse Reinforcement Learning via State Marginal Matching

3.1 Introduction

Imitation learning (IL) is a powerful tool to design autonomous behaviors in robotic systems. Although reinforcement learning methods promise to learn such behaviors automatically, they have been most successful in tasks with a clear definition of the reward function. Reward design remains difficult in many robotic tasks such as driving a car [64], tying a knot [61], and human-robot cooperation [35]. Imitation learning is a popular approach to such tasks, since it is easier for an expert teacher to demonstrate the desired behavior rather than specify the reward [10, 37, 39].

Methods in IL frameworks are generally split into behavior cloning (BC) [11] and inverse reinforcement learning (IRL) [70, 57, 1]. BC is typically based on supervised learning to regress expert actions from expert observations without the need for further interaction with the environment, but suffers from the covariate shift problem [69]. On the other hand, IRL methods aim to learn the reward function from expert demonstrations, and use it to train the agent policy. Within IRL, adversarial imitation learning (AIL) methods (GAIL [38], AIRL [28], f-MAX [32], SMM [47]) train a discriminator to guide the policy to match the expert's state-action distribution.

AIL methods learn a *non-stationary* reward by iteratively training a discriminator and taking a single policy update step using the reward derived from the discriminator. After convergence, the learned AIL reward cannot be used for training a new policy from scratch, and is thus discarded. In contrast, IRL methods such as ours learn a **stationary reward** such that, if the policy is trained from scratch using the reward function until convergence, then the policy will match the expert behavior. We argue that learning a stationary reward function can be useful for solving downstream

IL Method	Space	f-Divergence	Recover Reward?
MaxEntIRL [97], GCL [27]	τ	Forward Kullback-Leibler	✓
GAN-GCL [26]	τ	Forward Kullback-Leibler*	✓
AIRL [28], EAIRL [65]	τ	Forward Kullback-Leibler*	✓
EBIL [52]	τ	Reverse Kullback-Leibler	✓
GAIL [38]	s, a	Jensen-Shannon	×
f-MAX [32]	s, a	f-divergence	×
SMM [47]	s	Reverse Kullback-Leibler	×
f-IRL (Our method)	s	f-divergence	✓

Table 3.1: IL methods vary in the domain of the expert distribution that they model ("space"), the choice of f-divergence, and whether they recover a stationary reward function. *GAN-GCL and AIRL use biased IS weights to approximate FKL (see Appendix 3.3).

tasks and transferring behavior across different dynamics.

Traditionally, IL methods assume access to expert demonstrations and minimize some divergence between policy and expert's trajectory distribution. However, in many cases, it may be easier to directly specify the state distribution of the desired behavior rather than to provide fully-specified demonstrations of the desired behavior [47]. For example, in a safety-critical application, it may be easier to specify that the expert never visits some unsafe states, instead of tweaking reward to penalize safety violations [23]. Similarly, we can specify a uniform density over the whole state space for exploration tasks, or a Gaussian centered at the goal for goal-reaching tasks. Reverse KL instantiation for f-divergence in f-IRL allows for unnormalized density specification, which further allows for easier preference encoding.

In this work, we propose a new method, f-IRL, that learns a stationary reward function from the expert density via gradient descent. To do so, we derive an analytic gradient of any arbitrary f-divergence between the agent and the expert state distribution w.r.t. reward parameters. We demonstrate that f-IRL is especially useful in the limited data regime, exhibiting better sample efficiency than prior work in terms of the number of environment interactions and expert trajectories required to learn the MuJoCo benchmark tasks. We also demonstrate that the reward functions recovered by f-IRL can accelerate the learning of hard-to-explore tasks with sparse rewards, and these same reward functions can be used to transfer behaviors across changes in dynamics.

3.2 Related Work

IRL methods [70, 57, 1] obtain a policy by learning a reward function from sampled trajectories of an expert policy. MaxEntIRL [97] learns a stationary reward by maximizing the likelihood of expert trajectories, i.e., it minimizes forward KL divergence in trajectory space under the maximum entropy RL framework. Similar to MaxEntIRL, Deep MaxEntIRL [92] and GCL [27] optimize the forward KL divergence in trajectory space. A recent work, EBIL [52], optimizes the reverse KL divergence in the trajectory space by treating the expert state-action marginal as an energy-based model. Another recent method, RED [90], uses support estimation on the expert data to extract a fixed reward, instead of trying to minimize a f-divergence between the agent and expert distribution.

One branch of IRL methods train a GAN [33] with a special structure in the discriminator to learn the reward. This is first justified by Finn et al. [26] to connect GCL [27] with GAN, and several methods [26, 28, 65] follow this direction. Our analysis in Appendix 3.3 suggests that the importance-sampling weights used in these prior methods may be biased. We show that AIRL does not minimize the reverse RL in state-marginal space (as argued by [32]). Moreover, AIRL [28] uses expert state-action-next state transitions, while our method can work in a setting where only expert states are provided.

A set of IL methods [38, 32] use a discriminator to address the issue of running RL in the *inner* loop as classical IRL methods. Instead, these methods directly optimize the policy in the *outer* loop using adversarial training. These methods can be shown to optimize the Jensen-Shannon, and a general f-divergence respectively, but do not learn a reward function. SMM [47] optimizes the reverse KL divergence between the expert and policy state marginals but also does not recover a reward function due to its fictitious play approach. SQIL [67] and DRIL [15] utilize regularized behavior cloning for imitation without recovering a reward function. Unlike these prior methods, f-IRL can optimize any f-divergence between the state-marginal of the expert and the agent, while also recovering a stationary reward function. Table 3.1 summarizes the comparison among imitation learning methods.

3.3 What Objective is Optimized by Previous IL Algorithms?

In this section, we discuss previous IL methods and analyze which objectives they may truly optimize. Our analysis shows that AIRL and GAN-GCL methods possibly optimize for a different objective than they claim, due to their usage of biased importance sampling weights.

3.3.1 MaxEntIRL [97], Deep MaxEntIRL [92], GCL [27]

Classical IRL methods [70, 57] obtain a policy by learning a reward function from sampled trajectories of an expert policy. MaxEntIRL [97] learns a stationary reward by maximizing likelihood on expert trajectories, i.e., it minimizes forward KL divergence in trajectory space under the maximum entropy RL framework. A trajectory is a temporal collection of state-action pairs, and this makes the trajectory distribution different from state-action marginal or state marginal distribution. Each objective - minimizing divergence in trajectory space τ, in state-action marginal space (s, a) and state marginal s are different IL methods in their own sense.

MaxEntIRL derives a surrogate objective w.r.t. reward parameter as the difference in cumulative rewards of the trajectories between the expert and the soft-optimal policy under current reward function. To train the soft-optimal policy, it requires running MaxEnt RL in an inner loop after every reward update. This algorithm has been successfully applied for predicting behaviors of taxi drivers with a linear parameterization of reward. Wulfmeier et al. [92] shows that MaxEntIRL reward function can be parameterized as deep neural networks as well.

Guided cost learning (GCL) [27] is one of the first methods to train rewards using neural network directly through experiences from real robots. They achieve this result by leveraging guided policy search for policy optimization, employing importance sampling to correct for distribution shift when the policy has not converged, and using novel regularizations in reward network. GCL optimizes for the same objective as MaxEntIRL and Deep MaxEntIRL. To summarize these three works, we have the following observation:

Observation 3.3.0.1. *MaxEntIRL, Deep MaxEntIRL, GCL all optimize for the forward KL divergence in trajectory space, i.e.* $D_{\mathrm{KL}}(\rho_E(\tau) \,||\, \rho_\theta(\tau))$.

3.3.2 GAN-GCL [26], AIRL [28], EAIRL [65]

Finn et al. [26] shows that GCL is equivalent to training GANs with a special structure in the discriminator (GAN-GCL). Note that this result uses an approximation in importance sampling, and hence the gradient estimator is *biased*. Fu et al. [28] shows that GAN-GCL does not perform well in practice since its discriminator models density ratio over trajectories which leads to high variance. They propose an algorithm AIRL in which the discriminator estimates the density ratio of state-action marginal, and shows that AIRL empirically performs better than GAN-GCL. AIRL also uses approximate importance sampling in its derivation, and therefore its gradient is also *biased*. GAN-GCL and AIRL claim to be able to recover a reward function due to the special structure in the discriminator. EAIRL [65] uses empowerment regularization on policy objective

based on AIRL.

All the above algorithm intend to optimize for same objective as MaxEntIRL. However, there is an approximation involved in the procedure and let us analyze what that is, by going through the derivation for *equivalence of AIRL to MaxEntIRL* as shown in Fu et al. [28] (Appendix A of that paper).

The authors start from writing down the objective for MaxEntIRL: $\max_\theta L_{\text{MaxEntIRL}}(\theta) = \mathbb{E}_{\tau\sim D}[\log p_\theta(\tau)]$, where D is the collection of expert demonstrations, and reward function is parameterized by θ.

When the trajectory distribution is induced by the soft-optimal policy under reward r_θ, it can be parameterized as $p_\theta(\tau) \propto p(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t,a_t)e^{r_\theta(s_t,a_t)}$, then its gradient is derived as follows:

$$\begin{aligned}\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) &= \mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \frac{d}{d\theta}\log(Z_\theta) \\ &= \mathbb{E}_D\left[\sum_{t=1}^{T}\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{p_\theta}\left[\sum_{t=1}^{T}\frac{d}{d\theta}r_\theta(s_t,a_t)\right] \\ &= \sum_{t=1}^{T}\mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{p_{\theta,t}}\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right]\end{aligned} \tag{3.1}$$

where Z_θ is the normalizing factor of $p_\theta(\tau)$, and $p_{\theta,t}(s_t,a_t) = \int_{s_{t'!=t},a_{t'!=t}} p_\theta(\tau)$ denote the state action marginal at time t.

As it is difficult to draw samples from p_θ, the authors instead train a separate importance sampling distribution $\mu(\tau)$. For the choice of distribution they follow [27] and use a mixture policy $\mu(a|s) = 0.5\pi(a|s) + 0.5\hat{q}(a|s)$ where $\hat{q}(a|s)$ is the rough density estimate trained on the demonstrations. This is justified as reducing the variance of the importance sampling distribution. Thus the new gradient becomes:

$$\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) = \sum_{t=1}^{T}\mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{\mu_t}\left[\frac{p_{\theta,t}(s_t,a_t)}{\mu_t(s_t,a_t)}\frac{d}{d\theta}r_\theta(s_t,a_t)\right] \tag{3.2}$$

We emphasize here $\hat{q}(a|s)$ is the density estimate trained on the demonstrations.

They additionally aim to adapt the importance sampling distribution to reduce variance by minimizing $D_{\text{KL}}(\pi(\tau)\ ||\ p_\theta(\tau))$, and this KL objective can be simplified to the following MaxEnt RL objective:

$$\max_\pi \mathbb{E}_\pi\left[\sum_{t=1}^{T} r_\theta(s_t,a_t) - \log\pi(a_t|s_t)\right] \tag{3.3}$$

This ends the derivation of gradient of MaxEntIRL. Now, AIRL tries to show that the gradient of AIRL matches the gradient for MaxEntIRL objective shown above, i.e. $\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) = \frac{d}{d\theta}L_{\text{AIRL}}(\theta)$, then AIRL is equivalent to MaxEntIRL to a constant, i.e. $L_{\text{MaxEntIRL}}(\theta) = L_{\text{AIRL}}(\theta) + C$.

In AIRL, the cost learning objective is replaced by training a discriminator of the following form:

$$D_\theta(s,a) = \frac{e^{f_\theta(s,a)}}{e^{f_\theta(s,a)} + \pi(a|s)} \tag{3.4}$$

The objective of the discriminator is to maximize the cross-entropy between the expert demonstrations and the generated samples:

$$\begin{aligned}
\max_\theta L_{\text{AIRL}}(\theta) &= \sum_{t=1}^{T} \mathbb{E}_D[\log D_\theta(s_t,a_t)] + \mathbb{E}_{\pi_t}[\log(1 - D_\theta(s_t,a_t))] \\
&= \sum_{t=1}^{T} \mathbb{E}_D\left[\log \frac{e^{f_\theta(s_t,a_t)}}{e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t)}\right] + \mathbb{E}_{\pi_t}\left[\log \frac{\pi(a_t|s_t)}{\pi(a_t|s_t) + e^{f_\theta(s_t,a_t)}}\right] \\
&= \sum_{t=1}^{T} \mathbb{E}_D[f_\theta(s_t,a_t)] + \mathbb{E}_{\pi_t}[\log \pi(a_t|s_t)] - 2\mathbb{E}_{\mu_t}\left[\log(\pi(a_t|s_t)) + e^{f_\theta(s_t,a_t)}\right]
\end{aligned} \tag{3.5}$$

where μ_t is the mixture of state-action marginal from expert demonstrations and from state-action marginal induced by current policy π at time t.

In AIRL, the policy π is optimized with the following reward:

$$\begin{aligned}
\hat{r}(s,a) &= \log(D_\theta(s,a)) - \log(1 - D_\theta(s,a)) \\
&= f_\theta(s,a) - \log \pi(a|s)
\end{aligned} \tag{3.6}$$

Taking the derivative with respect to θ,

$$\frac{d}{d\theta}L_{\text{AIRL}}(\theta) = \sum_{t=1}^{T} \mathbb{E}_D\left[\frac{d}{d\theta} f_\theta(s_t,a_t)\right] - \mathbb{E}_{\mu_t}\left[\frac{e^{f_\theta(s_t,a_t)}}{(e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))/2}\frac{d}{d\theta} f_\theta(s_t,a_t)\right] \tag{3.7}$$

The authors multiply state marginal $\pi(s_t) = \int_a \pi_t(s_t,a_t)$ to the fraction term in the second expectation, and denote that $\hat{p}_{\theta,t}(s_t,a_t) \triangleq e^{f_\theta(s_t,a_t)}\pi(s_t)$ and $\hat{\mu}_t(s_t,a_t) \triangleq (e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))\pi(s_t)/2$.

Thus the gradient of the discriminator becomes:

$$\frac{d}{d\theta}L_{\text{AIRL}}(\theta) = \sum_{t=1}^{T} \mathbb{E}_D\left[\frac{d}{d\theta} f_\theta(s_t,a_t)\right] - \mathbb{E}_{\mu_t}\left[\frac{\hat{p}_{\theta,t}(s_t,a_t)}{\hat{\mu}_t(s_t,a_t)}\frac{d}{d\theta} f_\theta(s_t,a_t)\right] \tag{3.8}$$

AIRL is Not Equivalent to MaxEntIRL

The issues occurs when AIRL tried to match Eq. 3.8 with Eq. 3.42, i.e. $\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) \stackrel{?}{=} \frac{d}{d\theta}L_{\text{AIRL}}(\theta)$ with same reward parameterization $f_\theta = r_\theta$.

If they are equivalent, then we have the importance weights equality:

$$\boxed{\hat{p}_{\theta,t}(s_t, a_t) = p_{\theta,t}(s_t, a_t), \hat{\mu}_t(s_t, a_t) = \mu_t(s_t, a_t)} \tag{3.9}$$

Then given the definitions, we have:

$$\begin{aligned}\hat{p}_{\theta,t}(s_t, a_t) &\triangleq e^{f_\theta(s_t,a_t)}\pi(s_t) = \pi^*_\theta(s_t)\pi^*_\theta(a_t|s_t) \triangleq p_{\theta,t}(s, a) \\ \hat{\mu}_t(s_t, a_t) &\triangleq (e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))\pi(s_t)/2 = (\pi(a_t|s_t) + \hat{q}(a_t|s_t))\pi(s_t)/2 \triangleq \mu_t(s_t, a_t)\end{aligned} \tag{3.10}$$

where π^*_θ is soft-optimal policy under reward $r_\theta = f_\theta$ (assumption), thus $\log \pi^*_\theta(a_t|s_t) = f_\theta(s_t, a_t)$. Then equivalently,

$$\boxed{e^{f_\theta(s_t,a_t)} = \hat{q}(a_t|s_t) = \pi^*_\theta(a_t|s_t) = \pi(a_t|s_t)} \tag{3.11}$$

Unfortunately these equivalences hold only at the global optimum of the algorithm, when the policy π reaches the expert policy $\pi_E \approx \hat{q}$ and the discriminator is also optimal. This issue also applies to GAN-GCL and EAIRL. Therefore, we have the following conclusion:

Observation 3.3.0.2. *GAN-GCL, AIRL, EAIRL are **not** equivalent to MaxEntIRL, i.e. **not** minimizing forward KL in trajectory space and possibly optimizing a biased objective.*

AIRL is Not Optimizing Reverse KL in State-Action Marginal

f-MAX [32] (refer to their Appendix C) states that AIRL is equivalent to f-MAX with $f = -\log u$. This would imply that AIRL minimizes reverse KL in state-action marginal space.

However, there are some differences in AIRL algorithm and the f-MAX algorithm with reverse KL divergence. f-MAX[32] considers a vanilla discriminator. This is different than the original AIRL [28], which uses a specially parameterized discriminator. To highlight this difference we refer to f-MAX with $f = -logu$ (called AIRL in their paper) as f-MAX-RKL in this work, since it aims to minimize reverse-KL between state-action marginal. We see below that using f-MAX method with special discriminator(instead of vanilla) might not correspond to reverse KL minimization in state-action marginal which shows that AIRL does not truly minimize reverse KL divergence.

To show the equivalence of AIRL to reverse KL matching objective, Ghasemipour et al. [32] considers that the AIRL discriminator can be trained till convergence. With the special discriminator of AIRL, at convergence the following equality holds:

$$\frac{e^{f_\theta(s,a)}}{e^{f_\theta(s,a)} + \pi(a|s)} \equiv \frac{\rho_E(s,a)}{\rho_E(s,a) + \rho_\theta(s,a)} \text{ (at convergence)} \tag{3.12}$$

but if this is true then $f_\theta(s,a)$ can no longer be interpreted as the stationary reward function as it is a function of current policy:

$$f_\theta(s,a) = \frac{\rho_E(s,a)}{\rho_\theta(s,a)}\pi(a|s) \tag{3.13}$$

Observation 3.3.0.3. *AIRL is **not** optimizing reverse KL in state-action marginal space.*

3.3.3 GAIL [38], FAIRL, f-MAX-RKL [32]

Generative Adversarial Imitation Learning (GAIL) [38] addresses the issue of running RL in an inner step by adversarial training [33]. A discriminator learns to differentiate over state-action marginal and a policy learns to maximize the rewards acquired from the discriminator in an alternating fashion. It can be further shown that the GAIL is minimizing the Jensen-Shannon divergence over state-action marginal given optimal discriminator.

Recently the idea of minimizing the divergence between expert and policy's marginal distribution is further comprehensively studied and summarized in Ke et al. [42] and Ghasemipour et al. [32], where the authors show that any f-divergence can be minimized for imitation through f-GAN framework [59]. f-MAX proposes several instantiations of f-divergence: forward KL for f-MAX-FKL (FAIRL), reverse KL for f-MAX-RKL, and Jensen-Shannon for original GAIL. Their objectives are summarized as below, where θ is policy parameter, f^* is the convex conjugate of f and T_ω is the discriminator.

$$\min_\theta D_f(\rho_E(s,a) \,||\, \rho_\theta(s,a)) = \min_\theta \max_\omega \mathbb{E}_{(s,a)\sim\rho_E(s,a)}[T_\omega(s,a)] - \mathbb{E}_{(s,a)\sim\rho_\theta(s,a)}[f^*(T_\omega(s,a))] \tag{3.14}$$

These adversarial IRL methods cannot recover a reward function because they do minimax optimization with discriminator in the inner-loop (when optimal, the discriminator predicts $\frac{1}{2}$ everywhere), and have poorer convergence guarantees opposed to using an analytical gradient.

Observation 3.3.0.4. *GAIL, FAIRL, f-MAX-RKL are optimizing JS, forward KL, and reverse KL in state-action marginal space, respectively without recovering a reward function.*

3.3.4 SMM [47]

Lee et al. [47] presents state marginal matching (SMM) for efficient exploration by minimizing reverse KL between expert and policy's state marginals (Eq 3.15). However, their method cannot recover the stationary reward function because it uses fictitious play between policy π_θ and variational density q, and requires storing a historical average of policies and densities over previous iterations.

$$\max_\theta -D_{\mathrm{KL}}(\rho_\theta(s) \,||\, \rho_E(s)) = \max_\theta \mathbb{E}_{\rho_\theta(s)}\left[\log \frac{\rho_E(s)}{\rho_\theta(s)}\right] = \max_\theta \min_q \mathbb{E}_{\rho_\theta(s)}\left[\log \frac{\rho_E(s)}{q(s)}\right] \tag{3.15}$$

3.3.5 Summary of IL/IRL Methods: Two Classes of Bilevel Optimization

Now we generalize the related works including our method into *reward-dependent* and *policy-dependent* classes from the viewpoint of optimization objective.

For the **reward-dependent** (IRL) methods such as MaxEntIRL, AIRL, and our method, the objective of reward/discriminator r_θ and policy π_ϕ can be viewed as a bilevel optimization:

$$\begin{aligned} &\min_{\theta,\phi} L(r_\theta, \pi_\phi) \\ &\text{s.t. } \phi \in \arg\max_\phi g(r_\theta, \pi_\phi) \end{aligned} \tag{3.16}$$

where $L(\cdot,\cdot)$ is the joint loss function of reward and policy, and $g(r,\cdot)$ is the objective of policy given reward r. Thus the optimal policy is dependent on current reward, and training on the final reward does produce optimal policy, i.e. recovering the reward.

For the **policy-dependent** (IL) method such as f-MAX, GAIL, and SMM, the objective of reward/discriminator r_θ and policy π_ϕ can be viewed as:

$$\max_\phi \min_\theta L(r_\theta, \pi_\phi) \tag{3.17}$$

This is a special case of bilevel optimization, minimax game. The optimal reward is dependent on current policy as the inner objective is on reward, thus it is non-stationary and cannot guarantee to recover the reward.

3.4 Preliminaries

In this section, we review notation on maximum entropy (MaxEnt) RL [48] and state marginal matching (SMM) [47] that we build upon in this work.

MaxEnt RL. Consider a Markov Decision Process (MDP) represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, T)$ with state-space $\mathcal{S}$, action-space $\mathcal{A}$, dynamics $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $r(s, a)$, initial state distribution ρ_0, and horizon T. The optimal policy π under the maximum entropy framework [96] maximizes the objective $\sum_{t=1}^{T} \mathbb{E}_{\rho_{\pi,t}(s_t,a_t)}[r(s_t, a_t) + \alpha H(\cdot|s_t)]$. Here $\rho_{\pi,t}$ is the state-action marginal distribution of policy π at timestamp t, and $\alpha > 0$ is the entropy temperature.

Let $r_\theta(s)$ be a parameterized differentiable reward function only dependent on state. Let trajectory τ be a time series of visited states $\tau = (s_0, s_1, \ldots, s_T)$. The optimal MaxEnt trajectory distribution $\rho_\theta(\tau)$ under reward r_θ can be computed as $\rho_\theta(\tau) = \frac{1}{Z} p(\tau) e^{r_\theta(\tau)/\alpha}$, where

$$p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) \ , \quad r_\theta(\tau) = \sum_{t=1}^{T} r_\theta(s_t), \quad Z = \int p(\tau) e^{r_\theta(\tau)/\alpha} d\tau.$$

Slightly overloading the notation, the optimal MaxEnt state marginal distribution $\rho_\theta(s)$ under reward r_θ is obtained by marginalization:

$$\rho_\theta(s) \propto \int p(\tau) e^{r_\theta(\tau)/\alpha} \eta_\tau(s) d\tau \tag{3.18}$$

where $\eta_\tau(s) \triangleq \sum_{t=1}^{T} \mathbb{1}(s_t = s)$ is the visitation count of a state s in a particular trajectory τ.

State Marginal Matching. Given the expert state density $p_E(s)$, one can train a policy to match the expert behavior by minimizing the following f-divergence objective:

$$L_f(\theta) = D_f(\rho_E(s) \ || \ \rho_\theta(s)) \tag{3.19}$$

where common choices for the f-divergence D_f [4, 32] include forward KL divergence, reverse KL divergence, and Jensen-Shannon divergence. Our proposed f-IRL algorithm will compute the analytical gradient of Eq. 3.19 w.r.t. θ and use it to optimize the reward function via gradient descent.

3.5 Learning Stationary Rewards via State-Marginal Matching

In this section, we describe our algorithm f-IRL, which takes the expert state density as input, and optimizes the f-divergence objective (Eq. 3.19) via gradient descent. Our algorithm trains a policy whose state marginal is close to that of the expert, and a corresponding stationary reward function that would produce the same policy if the policy were trained with MaxEnt RL from scratch.

Name	f-divergence $D_f(P \mid\mid Q)$	Generator $f(u)$	$h_f(u)$
FKL	$\int p(x) \log \frac{p(x)}{q(x)} dx$	$u \log u$	$-u$
RKL	$\int q(x) \log \frac{q(x)}{p(x)} dx$	$-\log u$	$1 - \log u$
JS	$\frac{1}{2} \int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} dx$	$u \log u - (1+u) \log \frac{1+u}{2}$	$-\log(1+u)$

Table 3.2: Selected list of f-divergences $D_f(P \mid\mid Q)$ with generator functions f and h_f defined in Theorem 3.5.1, where f is convex, lower-semicontinuous and $f(1) = 0$.

3.5.1 Analytic Gradient for State Marginal Matching in f-divergence

One of our main contributions is the exact gradient of the f-divergence objective (Eq. 3.19) w.r.t. the reward parameters θ. This gradient will be used by f-IRL to optimize Eq. 3.19 via gradient descent. The proof is provided in Appendix 3.7.1.

Theorem 3.5.1 (**f-divergence analytic gradient**). *The analytic gradient of the f-divergence $L_f(\theta)$ between state marginals of the expert (ρ_E) and the soft-optimal agent w.r.t. the reward parameters θ is given by:*

$$\nabla_\theta L_f(\theta) = \frac{1}{\alpha T} \mathrm{cov}_{\tau \sim \rho_\theta(\tau)} \left(\sum_{t=1}^{T} h_f \left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)} \right), \sum_{t=1}^{T} \nabla_\theta r_\theta(s_t) \right) \tag{3.20}$$

where $h_f(u) \triangleq f(u) - f'(u)u$, $\rho_E(s)$ is the expert state marginal and $\rho_\theta(s)$ is the state marginal of the soft-optimal agent under the reward function r_θ, and the covariance is taken under the agent's trajectory distribution $\rho_\theta(\tau)$.[1]

Choosing the f-divergence to be Forward Kullback-Leibler (FKL), Reverse Kullback-Leibler (RKL), or Jensen-Shannon (JS) instantiates h_f (see Table 3.2). Note that the gradient of the RKL objective has a special property in that we can specify the expert as an *unnormalized* log-density (i.e. energy), since in $h_{\mathrm{RKL}}(\frac{\rho_E(s)}{\rho_\theta(s)}) = 1 - \log \rho_E(s) + \log \rho_\theta(s)$, the normalizing factor of $\rho_E(s)$ does not change the gradient (by linearity of covariance). This makes density specification much easier in a number of scenarios. Intuitively, since h_f is a monotonically decreasing function ($h'_f(u) = -f''(u)u < 0$) over $\mathbb{R}^+$, the gradient descent tells the reward function to increase the rewards of those state trajectories that have higher sum of density ratios $\sum_{t=1}^{T} \frac{\rho_E(s_t)}{\rho_\theta(s_t)}$ so as to minimize the objective.

[1]Here we assume f is differentiable, which is often the case for common f-divergence (e.g. KL divergence).

3.5.2 Learning a Stationary Reward by Gradient Descent

We now build upon Theorem 3.5.1 to design a practical algorithm for learning the reward function r_θ (Algorithm. 2). Given expert information (state density or observation samples) and an arbitrary f-divergence, the algorithm alternates between using MaxEnt RL with the current reward, and updating the reward parameter using gradient descent based on the analytic gradient.

If the provided expert data is in the form of expert state density $\rho_E(s)$, we can fit a density model $\hat{\rho}_\theta(s)$ to estimate agent state density $\rho_\theta(s)$ and thus estimate the density ratio required in gradient. If we are given samples from expert observations s_E, we can fit a discriminator $D_\omega(s)$ in each iteration to estimate the density ratio by optimizing the binary cross-entropy loss:

$$\max_{\omega} \mathbb{E}_{s\sim s_E}[\log D_\omega(s)] + \mathbb{E}_{s\sim\rho_\theta(s)}[\log(1 - D_\omega(s)] \tag{3.21}$$

where the optimal discriminator satisfies $D^*_\omega(s) = \frac{\rho_E(s)}{\rho_E(s)+\rho_\theta(s)}$ [33], thus the density ratio can be estimated by $\frac{\rho_E(s)}{\rho_\theta(s)} \approx \frac{D_\omega(s)}{1-D_\omega(s)}$, which is the input to h_f.

Algorithm 2: Inverse RL via State Marginal Matching (f-IRL)

Input : Expert state density $\rho_E(s)$ or expert observations s_E , f-divergence

Output: Learned reward r_θ, Policy π_θ

Initialize r_θ, and density estimation model (provided $\rho_E(s)$) or disciminator D_ω (provided s_E)

for $i \leftarrow 1$ **to** $Iter$ **do**

  $\pi_\theta \leftarrow$ MaxEntRL(r_θ) and collect agent trajectories τ_θ

  if *provided* $\rho_E(s)$ **then**

    Fit the density model $\hat{\rho}_\theta(s)$ to the state samples from τ_θ

  end

  else

    `// provided` s_E

    Fit the discriminator D_ω by Eq. 3.21 using expert and agent state samples from s_E and τ_θ

  end

  Compute sample gradient $\hat{\nabla}_\theta L_f(\theta)$ for Eq. 3.20 over τ_θ

  $\theta \leftarrow \theta - \lambda\hat{\nabla}_\theta L_f(\theta)$

end

3.5.3 Robust Reward Recovery under State-only Ground-truth Reward

IRL methods are different from IL methods in that they recover a reward function in addition to the policy. A hurdle in this process is often the reward ambiguity problem, explored in [56, 28]. This ambiguity arises due to the fact that the optimal policy remains unchanged under the following reward transformation [56]:

$$\hat{r}(s, a, s') = r_{\text{gt}}(s, a, s') + \gamma\Phi(s') - \Phi(s) \tag{3.22}$$

for any function Φ. In the case where the ground-truth reward is a function over states only (i.e., $r_{\text{gt}}(s)$), f-IRL is able to recover the *disentangled* reward function (r_{IRL}) that matches the ground truth reward r_{gt} up to a constant. The obtained reward function is robust to different dynamics – for any underlying dynamics, r_{IRL} will produce the same optimal policy as r_{gt}. We formalize this claim in Appendix 3.7.1 (based on Theorem 5.1 of AIRL [28]).

AIRL uses a special parameterization of the discriminator to learn state-only rewards. A disadvantage of their approach is that AIRL needs to approximate a separate reward-shaping network apart from the reward network. In contrast, our method naturally recovers a state-only reward function.

3.5.4 Practical Modification in the Exact Gradient

In practice with high-dimensional observations, when the agent's current trajectory distribution is far off from the expert trajectory distribution, we find that there is little supervision available through our derived gradient, leading to slow learning. Therefore, when expert trajectories are provided, we bias the gradient (Eq. 3.20) using a mixture of agent and expert trajectories inspired by GCL [27], which allows for richer supervision and faster convergence. Note that at convergence, the gradient becomes unbiased as the agent's and expert's trajectory distribution matches.

$$\tilde{\nabla}_\theta L_f(\theta) := \frac{1}{\alpha T} \text{cov}_{\tau \sim \frac{1}{2}(\rho_\theta(\tau) + \rho_E(\tau))} \left(\sum_{t=1}^{T} h_f \left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)} \right), \sum_{t=1}^{T} \nabla_\theta r_\theta(s_t) \right) \tag{3.23}$$

where the expert trajectory distribution $\rho_E(\tau)$ is uniform over samples τ_E.

3.6 Experiments

In our experiments, we seek answers to the following questions:

1. Can f-IRL learn a policy that matches the given expert state density?
2. Can f-IRL learn good policies on high-dimensional continuous control tasks in a sample-efficient manner?
3. Can f-IRL learn a reward function that induces the expert policy?
4. How can learning a stationary reward function help solve downstream tasks?

Comparisons. To answer these questions, we compare f-IRL against two classes of existing imitation learning algorithms: (1) those that learn only the policy, including Behavior Cloning (BC), GAIL [38], and f-MAX-RKL[2] [32]; and (2) IRL methods that learn both a reward and a policy simultaneously, including MaxEnt IRL [97] and AIRL [28]. The rewards/discriminators of the baselines are parameterized to be state-only. We use SAC [34] as the base MaxEnt RL algorithm. Since the original AIRL uses TRPO [72], we re-implement a version of AIRL that uses SAC as the underlying RL algorithm for fair comparison. For our method (f-IRL), MaxEnt IRL, and AIRL, we use a MLP for reward parameterization.

Tasks. We evaluate the algorithms on several tasks:

- **Matching Expert State Density**: In Section 3.6.1, the task is to learn a policy that matches the given expert state density.
- **Inverse Reinforcement Learning Benchmarks**: In Section 3.6.2, the task is to learn a reward function and a policy from expert trajectory samples. We collected expert trajectories by training SAC [34] to convergence on each environment. We trained all the methods using varying numbers of expert trajectories $\{1, 4, 16\}$ to test the robustness of each method to the amount of available expert data.
- **Using the Learned Reward for Downstream Tasks**: In Section 3.6.3, we first train each algorithm to convergence, then use the learned reward function to train a new policy on a related downstream task. We measure the performance on downstream tasks for evaluation.

We use five MuJoCo continuous control locomotion environments [83, 16] with joint torque actions, illustrated in Figure 3.1. Further details about the environment, expert information (samples or density specification), and hyperparameter choices can be found in Appendix 3.7.2.

3.6.1 Matching the Specified Expert State Density

First, we check whether f-IRL can learn a policy that matches the given expert state density of the fingertip of the robotic arm in the 2-DOF Reacher environment. We evaluate the algorithms using

[2]A variant of AIRL [28] proposed in [32] only learns a policy and does not learn a reward.

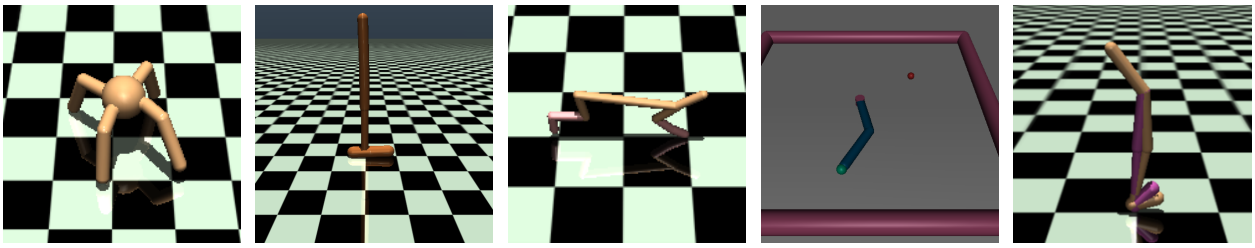
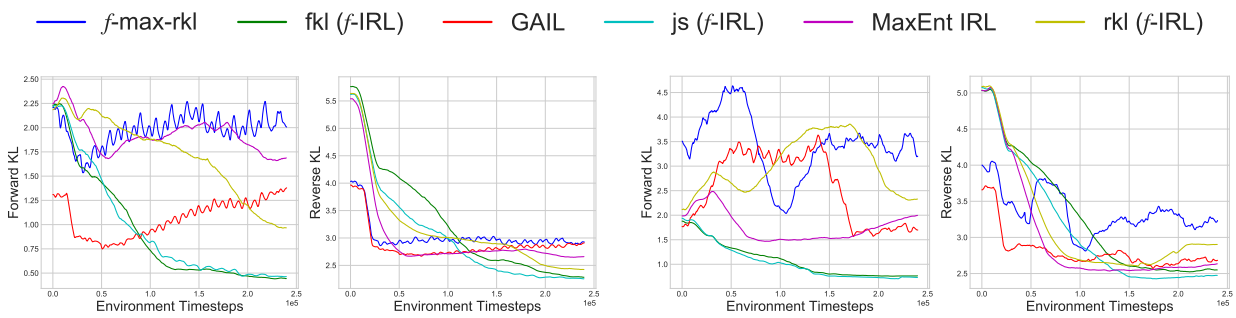


Figure 3.1: **Environments**: (left to right) Ant-v2, Hopper-v2, HalfCheetah-v2, Reacher-v2, and Walker2d-v2.



(a) Expert Density: Gaussian

(b) Expert Density: Mixture of two Gaussians

Figure 3.2: Forward (left) and Reverse (right) KL curves in the Reacher environment for different expert densities of all methods. Curves are smoothed in a window of 120 evaluations.

two different expert state marginals: (1) a Gaussian distribution centered at the goal for single goal-reaching, and (2) a mixture of two Gaussians, each centered at one goal. Since this problem setting assumes access to the expert density only, we use importance sampling to generate expert samples required by the baselines.

In Figure 3.2, we report the estimated forward and reverse KL divergences in state marginals between the expert and the learned policy. For f-IRL and MaxEnt IRL, we use Kernel Density Estimation (KDE) to estimate the agent's state marginal. We observe that the baselines demonstrate unstable convergence, which might be because those methods optimize the f-divergence approximately. Our method $\{\text{FKL}, \text{JS}\}$ f-IRL outperforms the baselines in the forward KL and the reverse KL metric, respectively.

3.6.2 Inverse Reinforcement Learning Benchmarks

Next, we compare f-IRL and the baselines on IRL benchmarks, where the task is to learn a reward function and a policy from expert trajectory samples. We use the modification proposed in

Section 3.5.4 to alleviate the difficulty in optimizing the f-IRL objective with high-dimensional states.

Policy Performance. We check whether f-IRL can learn good policies on high-dimensional continuous control tasks in a sample-efficient manner from expert trajectories. Figure 3.3 shows the learning curves of each method in the four environments with *one* expert trajectory provided. f-IRL and MaxEnt IRL demonstrate much faster convergence in most of the tasks than f-MAX-RKL. Table 3.3 shows the final performance of each method in the four tasks, measured by the ratio of agent returns (evaluated using the ground-truth reward) to expert returns.[3] While MaxEnt IRL provides a strong baseline, f-IRL outperforms all baselines on most tasks especially in Ant, where the FKL (f-IRL) has much higher final performance and is less sensitive to the number of expert trajectories compared to the baselines. In contrast, we found the original implementation of f-MAX-RKL to be extremely sensitive to hyperparameter settings. We also found that AIRL performs poorly even after tremendous tuning, similar to the findings in [51, 52].

Recovering the Stationary Reward Function. We also evaluate whether f-IRL can recover a stationary reward function that induces the expert policy. To do so, we train a SAC agent from scratch to convergence using the reward model obtained from each IRL method. We then evaluate the trained agents using the ground-truth reward to test whether the learned reward functions are good at inducing the expert policies.

Table 3.4 shows the ratio of the final returns of policy trained from scratch using the rewards learned from different IRL methods with one expert trajectory provided, to expert returns. Our results show that MaxEnt IRL and f-IRL are able to learn *stationary* rewards that can induce a policy close to the optimal expert policy.

3.6.3 Using the Learned Stationary Reward for Downstream Tasks

Finally, we investigate how the learned stationary reward can be used to learn related, downstream tasks.

Reward prior for downstream hard-exploration tasks. We first demonstrate the utility of the learned stationary reward by using it as a prior reward for the downstream task. Specifically, we construct a didactic point mass environment that operates under linear dynamics in a 2D 6×6 room, and actions are restricted to $[-1, 1]$. The prior reward is obtained from a *uniform* expert density over the whole state space, and is used to ease the learning in the hard-exploration task, where we design a difficult goal to reach with distraction rewards (full details in appendix 3.7.2).

[3]The unnormalized agent and expert returns are reported in Appendix 3.7.5.

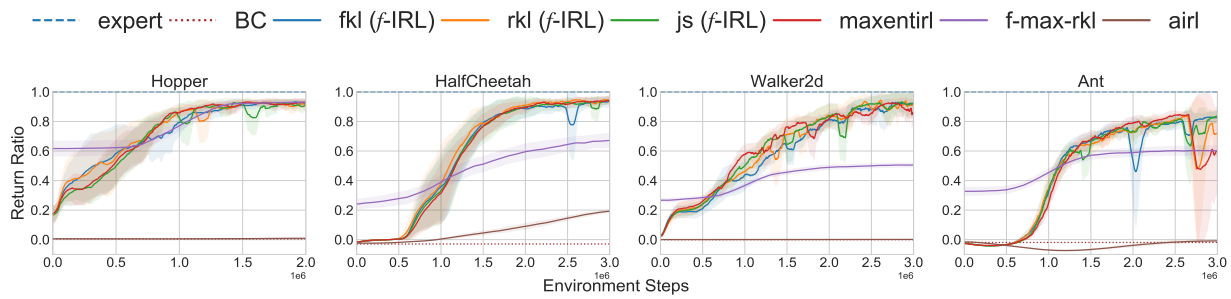


Figure 3.3: Training curves for f-IRL and 4 other baselines - BC, MaxEnt IRL, f-MAX-RKL and AIRL with one expert demonstration. Solid curves depict the mean of 3 trials and the shaded area shows the standard deviation. The dashed blue line represents the expert performance and the dashed red line shows the performance of a BC agent at convergence.

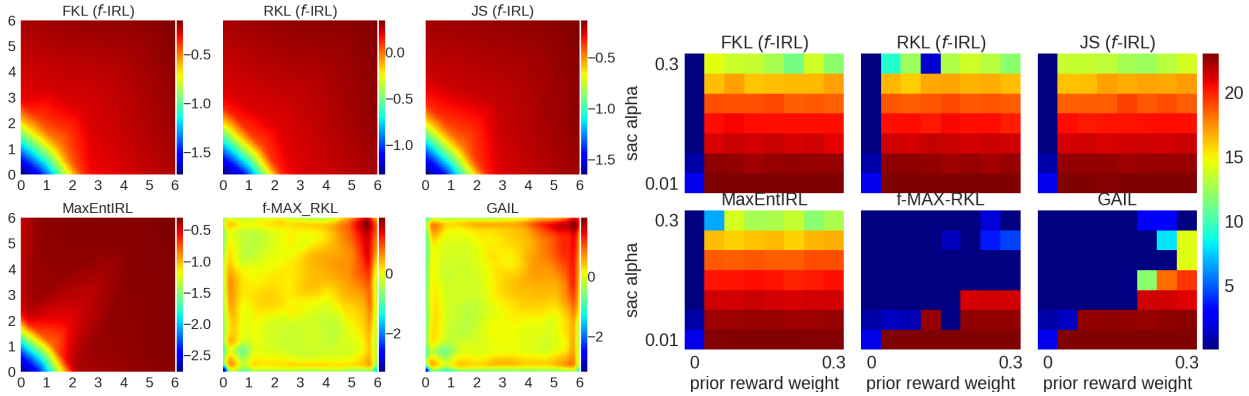


Figure 3.4: Left: Extracted final reward of all compared methods for the uniform expert density in the point environment. Right: The task return (in terms of r_{task}) with different α and prior reward weight λ. The performance of vanilla SAC is shown in the leftmost column with $\lambda = 0$ in each subplot.

We use the learned prior reward r_{prior} to augment the task reward r_{task} as follows: $r(s) = r_{\text{task}}(s) + \lambda(\gamma r_{\text{prior}}(s') - r_{\text{prior}}(s))$. The main theoretical result of [56] dictates that adding a potential-based reward in this form will not change the optimal policy. GAIL and f-MAX-RKL do not extract a reward function but rather a discriminator, so we derive a prior reward from the discriminator in the same way as [32, 38].

Figure 3.4 illustrates that the reward recovered by {FKL, RKL, JS} f-IRL and the baseline MaxEnt IRL are similar: the reward increases as the distance to the agent's start position, the bottom left corner, increases. This is intuitive for achieving the target uniform density: states

Method	Hopper			Walker2d			HalfCheetah			Ant		
Expert return	3592.63 $\pm$ 19.21			5344.21 $\pm$ 84.45			12427.49 $\pm$ 486.38			5926.18 $\pm$ 124.56		
# Expert traj	1	4	16	1	4	16	1	4	16	1	4	16
BC	0.00	0.13	0.16	0.00	0.05	0.08	0.00	0.01	0.02	0.00	0.22	0.47
MaxEnt IRL	0.93	0.92	**0.94**	0.88	0.88	**0.91**	**0.95**	**0.98**	0.91	0.54	0.71	0.81
f-MAX-RKL	**0.94**	**0.93**	0.91	0.49	0.49	0.47	0.71	0.41	0.65	0.60	0.65	0.62
AIRL	0.01	0.01	0.01	0.00	0.00	0.00	0.19	0.19	0.19	0.00	0.00	0.00
FKL (f-IRL)	0.93	0.90	0.93	**0.90**	0.90	0.90	0.94	0.97	0.94	**0.82**	**0.83**	**0.84**
RKL (f-IRL)	0.93	0.92	0.93	0.89	0.90	0.85	**0.95**	0.97	**0.96**	0.63	0.82	0.81
JS (f-IRL)	0.92	**0.93**	**0.94**	0.89	**0.92**	0.88	0.93	**0.98**	0.94	0.77	0.81	0.73

Table 3.3: We report the ratio between the average return of the trained (stochastic) policy vs. that of the expert policy for different IRL algorithms using 1, 4 and 16 expert trajectories. All results are averaged across 3 seeds. Negative ratios are clipped to zero.

Method	Hopper	Walker2d	HalfCheetah	Ant
AIRL	-	-	-0.03	-
MaxEntIRL	**0.93**	**0.92**	0.96	0.79
f-IRL	**0.93**	0.88	**1.02**	**0.82**

Table 3.4: The ratios of final return of the obtained policy against expert return across IRL methods. We average f-IRL over FKL, RKL, and JS. '-' indicates that we do not test learned rewards since AIRL does poorly at these tasks in Table 3.3.

Policy Transfer using GAIL	AIRL	MaxEntIRL	f-IRL	Ground-truth Reward
-29.9	130.3	**145.5**	141.1	315.5

Table 3.5: Returns obtained after transferring the policy/reward on modified Ant environment using different IL methods.

farther away should have higher rewards. f-MAX-RKL and GAIL's discriminator demonstrate a different pattern which does not induce a uniform state distribution. The leftmost column in the Figure 3.4 (Right) shows the poor performance of SAC training without reward augmentation ($\lambda = 0$). This verifies the difficulty in exploration for solving the task. We vary λ in the x-axis, and α in SAC in the y-axis, and plot the final task return (in terms of r_{task}) as a heatmap in the figure. The presence of larger red region in the heatmap shows that our method can extract a prior reward that is more robust and effective in helping the downstream task attain better final performance with its original reward.

Reward transfer across changing dynamics. Lastly, we evaluate the algorithms on transfer learning across different environment dynamics, following the setup from [28]. In this setup, IL algorithms are provided expert trajectories from a quadrupedal ant agent which runs forward. The algorithms are tested on an ant with two of its legs being disabled and shrunk. This requires the ant to significantly change its gait to adapt to the disabled legs for running forward.

We found that a forward-running policy obtained by GAIL fails to transfer to the disabled ant. In contrast, IRL algorithms such as f-IRL are successfully able to learn the expert's reward function using expert demonstrations from the quadrupedal ant, and use the reward to train a policy on the disabled ant. The results in Table 3.5 show that the reward learned by f-IRL is robust and enables the agent to learn to move forward with just the remaining two legs.

3.7 Appendix

3.7.1 Derivation and Proof

This section provides the derivations and proofs for the main ideas in this work. Section 3.7.1 and 3.7.1 provide the derivation of Theorem 3.5.1, and section 3.7.1 provides the details about section 3.5.3.

Analytical Gradient of State Marginal Distribution

In this subsection, we start by deriving a general result - gradient of state marginal distribution w.r.t. parameters of the reward function. We will use this gradient in the next subsection 3.7.1 where we derive the gradient of f-divergence objective.

Based on the notation introduced in section 3.4, we start by writing the probability of trajectory $\tau = (s_0, s_1, \ldots, s_T)$ of fixed horizon T under the optimal MaxEnt trajectory distribution for

$r_\theta(s)$ [96].

$$\rho_\theta(\tau) \propto \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \tag{3.24}$$

Let $p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)$, which is the probability of the trajectory under the dynamics of the environment.

Explicitly computing the normalizing factor, we can write the distribution over trajectories as follows:

$$\rho_\theta(\tau) = \frac{p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha}}{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} d\tau} \tag{3.25}$$

Let $\eta_\tau(s)$ denote the number of times a state occurs in a trajectory τ. We now compute the marginal distribution of all states in the trajectory:

$$\rho_\theta(s) \propto \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \tag{3.26}$$

where

$$\eta_\tau(s) = \sum_{t=1}^{T} \mathbb{1}(s_t = s) \tag{3.27}$$

is the empirical frequency of state s in trajectory τ (omitting the starting state s_0 as the policy cannot control the initial state distribution).

The marginal distribution over states can now be written as:

$$\rho_\theta(s) \propto \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \tag{3.28}$$

In the following derivation, we will use s_t to denote states in trajectory τ and s'_t to denote states from trajectory τ'. Explicitly computing the normalizing factor, the marginal distribution can be written as follows:

$$\begin{aligned} \rho_\theta(s) &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{\int \int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} \eta_{\tau'}(s') d\tau' ds'} \\ &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{\int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} \int \eta_{\tau'}(s') ds' d\tau'} \\ &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{T \int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} d\tau'} \end{aligned} \tag{3.29}$$

In the second step we swap the order of integration in the denominator. The last line follows because only the T states in τ satisfy $s \in \tau$. Finally, we define $f(s)$ and Z to denote the numerator (dependent on s) and denominator (normalizing constant), to simplify notation in further

calculations.

$$
\begin{aligned}
f(s) &= \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \\
Z &= T \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} d\tau \\
\rho_\theta(s) &= \frac{f(s)}{Z}
\end{aligned} \tag{3.30}
$$

As an initial step, we compute the derivatives of $f(s)$ and Z w.r.t reward function at some state $r_\theta(s^*)$.

$$
\frac{df(s)}{dr_\theta(s^*)} = \frac{1}{\alpha} \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) d\tau \tag{3.31}
$$

$$
\frac{dZ}{dr_\theta(s^*)} = \frac{T}{\alpha} \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s^*) d\tau = \frac{T}{\alpha} f(s^*) \tag{3.32}
$$

We can then apply the quotient rule to compute the derivative of policy marginal distribution w.r.t. the reward function.

$$
\begin{aligned}
\frac{d\rho_\theta(s)}{dr_\theta(s^*)} &= \frac{Z \frac{df(s)}{dr_\theta(s^*)} - f(s) \frac{dZ}{dr_\theta(s^*)}}{Z^2} \\
&= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) d\tau}{\alpha Z} - \frac{f(s)}{Z} \frac{T f(s^*)}{\alpha Z} \\
&= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) d\tau}{\alpha Z} - \frac{T}{\alpha} \rho_\theta(s) \rho_\theta(s^*)
\end{aligned} \tag{3.33}
$$

Now we have all the tools needed to get the derivative of ρ_θ w.r.t. θ by the chain rule.

$$
\begin{aligned}
\frac{d\rho_\theta(s)}{d\theta} &= \int \frac{d\rho_\theta(s)}{dr_\theta(s^*)} \frac{dr_\theta(s^*)}{d\theta} ds^* \\
&= \frac{1}{\alpha} \int \left(\frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) d\tau}{Z} - T \rho_\theta(s) \rho_\theta(s^*) \right) \frac{dr_\theta(s^*)}{d\theta} ds^* \\
&= \frac{1}{\alpha Z} \int \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^* d\tau - \frac{T}{\alpha} \rho_\theta(s) \int \rho_\theta(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^* \\
&= \frac{1}{\alpha Z} \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \sum_{t=1}^{T} \frac{dr_\theta(s_t)}{d\theta} d\tau - \frac{T}{\alpha} \rho_\theta(s) \int \rho_\theta(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^*
\end{aligned} \tag{3.34}
$$

Analytical Gradient of f-divergence objective

f-divergence [4] is a family of divergence, which generalizes forward/reverse KL divergence. Formally, let P and Q be two probability distributions over a space Ω, then for a convex and lower-

semicontinuous function f such that $f(1) = 0$, the f-divergence of P from Q is defined as:

$$D_f(P \,||\, Q) := \int_\Omega f\left(\frac{dP}{dQ}\right) dQ \tag{3.35}$$

Applied to state marginal matching between expert density $\rho_E(s)$ and agent density $\rho_\theta(s)$ over state space $\mathcal{S}$, the f-divergence objective is:

$$\min_\theta L_f(\theta) := D_f(\rho_E \,||\, \rho_\theta) = \int_{\mathcal{S}} f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \rho_\theta(s) ds \tag{3.36}$$

Now we show the proof of **Theorem 3.5.1** on the gradient of f-divergence objective:

Proof. The gradient of the f-divergence objective can be derived by chain rule:

$$\begin{aligned}
\nabla_\theta L_f(\theta) &= \int \nabla_\theta \left(f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \rho_\theta(s) \right) ds \\
&= \int \left(f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) - f'\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \frac{\rho_E(s)}{\rho_\theta(s)} \right) \frac{d\rho_\theta(s)}{d\theta} ds \\
&\triangleq \int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \frac{d\rho_\theta(s)}{d\theta} ds
\end{aligned} \tag{3.37}$$

where we denote $h_f(u) \triangleq f(u) - f'(u)u.$ for convenience.[4]

[4]Note that if $f(u)$ is non-differentiable at some points, such as $f(u) = |u-1|/2$ at $u = 1$ for Total Variation distance, we take one of its subderivatives.

Substituting the gradient of state marginal distribution w.r.t θ in Eq. 3.34, we have:

$$
\begin{aligned}
&\nabla_\theta L_f(\theta) \\
&= \int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\left(\frac{1}{\alpha Z}\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta}d\tau - \frac{T}{\alpha}\rho_\theta(s)\int \rho_\theta(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^*\right)ds \\
&= \frac{1}{\alpha Z}\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right)\sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta}d\tau \\
&\quad - \frac{T}{\alpha}\int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\rho_\theta(s)\left(\int \rho_\theta(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^*\right)ds \\
&= \frac{1}{\alpha T}\int \rho_\theta(\tau)\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right)\sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta}d\tau \\
&\quad - \frac{T}{\alpha}\left(\int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\rho_\theta(s)ds\right)\left(\int \rho_\theta(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^*\right) \\
&= \frac{1}{\alpha T}\mathbb{E}_{\tau\sim\rho_\theta(\tau)}\left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right)\sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta}\right] - \frac{T}{\alpha}\mathbb{E}_{s\sim\rho_\theta(s)}\left[h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\right]\mathbb{E}_{s\sim\rho_\theta(s)}\left[\frac{dr_\theta(s)}{d\theta}\right]
\end{aligned}
\tag{3.38}
$$

To gain more intuition about this equation, we can convert all the expectations to be over the trajectories:

$$
\begin{aligned}
&\nabla_\theta L_f(\theta) \\
&= \frac{1}{\alpha T}\left(\mathbb{E}_{\rho_\theta(\tau)}\left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right)\sum_{t=1}^T \nabla_\theta r_\theta(s_t)\right] - \mathbb{E}_{\rho_\theta(\tau)}\left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right)\right]\mathbb{E}_{\rho_\theta(\tau)}\left[\sum_{t=1}^T \nabla_\theta r_\theta(s_t)\right]\right) \\
&= \frac{1}{\alpha T}\mathrm{COV}_{\tau\sim\rho_\theta(\tau)}\left(\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right), \sum_{t=1}^T \nabla_\theta r_\theta(s_t)\right)
\end{aligned}
\tag{3.39}
$$

Thus we have derived the analytic gradient of f-divergence for state-marginal matching as shown in Theorem 3.5.1. □

Extension to Integral Probability Metrics in f-IRL

Integral Probability Metrics (IPM) [54] is another class of divergence based on dual norm, examples of which include Wasserstein distance [9] and MMD [49]. We can use Kantorovich-Rubinstein duality [86] to rewrite the IPM-based state marginal matching as:

$$
L_{\mathrm{B}}(\theta) = \|\rho_E(s) - \rho_\theta(s)\|_B := \max_{D_\omega \in B} \mathbb{E}_{\rho_E(s)}[D_\omega(s)] - \mathbb{E}_{\rho_\theta(s)}[D_\omega(s)] \tag{3.40}
$$

where B is a symmetric convex set of functions and D_ω is the critic function in [9].

Then the analytical gradient of the objective $L_{\mathrm{B}}(\theta)$ can be derived to be:

$$\nabla_\theta L_{\mathrm{B}}(\theta) = -\frac{1}{\alpha T}\mathrm{cov}_{\tau\sim\rho_\theta(\tau)}\left(\sum_{t=1}^{T} D_\omega\left(s_t\right), \sum_{t=1}^{T}\nabla_\theta r_\theta(s_t)\right) \tag{3.41}$$

where the derivation directly follows the proof of Theorem 3.5.1.

f-IRL Learns Disentangled Rewards w.r.t. Dynamics

We follow the derivation and definitions as given in Fu et al. [28] to show that f-IRL learns disentangled rewards. We show the definitions and theorem here for completeness. For more information, please refer to Fu et al. [28].

We first redefine the notion of "disentangled rewards".

Definition 1 (Disentangled rewards)**.** *A reward function $r'(s,a,s')$ is (perfectly) disentangled with respect to ground truth reward $r_{gt}(s,a,s')$ and a set of dynamics $\mathcal{T}$ such that under all dynamics in $T \in \mathcal{T}$, the optimal policy is the same: $\pi^*_{r',T}(a|s) = \pi^*_{r_{gt},T}(a|s)$*

Disentangled rewards can be loosely understood as learning a reward function which will produce the same optimal policy as the ground truth reward for the environment, on any underlying dynamics.

To show how f-IRL recovers a disentangled reward function, we need go through the definition of "Decomposability condition"

Definition 2 (Decomposability condition)**.** *Two states s_1,s_2 are defined as "1-step linked" under a dyanamics or transition distribution $T(s'|a,s)$, if there exists a state that can reach s_1 and s_2 with positive probability in one timestep. Also, this relationship can transfer through transitivity: if s_1 and s_2 are linked, and s_2 and s_3 are linked then we can consider s_1 and s_3 to be linked.*
A transition distribution T satisfies the decomposibility condition if all states in the MDP are linked with all other states.

This condition is mild and can be satisfied by any of the environments used in our experiments.

Theorem 3.7.1 and 3.7.2 formalize the claim that f-IRL recovers disentangled reward functions with respect to the dynamics. The notation $Q^*_{r,T}$ denotes the optimal Q function under reward function r and dynamics T, and similarly $\pi^*_{r,T}$ is the optimal policy under reward function r and dynamics T.

Theorem 3.7.1. *Let $r_{gt}(s)$ be the expert reward, and T be a dynamics satisfying the decomposability condition as defined in [28]. Suppose f-IRL learns a reward r_{IRL} such that it produces an*

optimal policy in T: $Q^*_{r_{IRL},T}(s,a) = Q^*_{r_{gt},T}(s,a) - f(s)$,where $f(s)$ is an arbitrary function of the state. Then we have:

$r_{IRL}(s) = r_{gt}(s) + C$ for some constant C, and thus $r_{IRL}(s)$ is robust to all dynamics.

Proof. Refer to Theorem 5.1 of AIRL [28]. □

Theorem 3.7.2. *If a reward function $r'(s,a,s')$ is disentangled with respect to all dynamics functions, then it must be state-only.*

Proof. Refer to Theorem 5.2 of AIRL [28]. □

3.7.2 Implementation Details

Matching the Specified Expert State Density on Reacher (Sec 3.6.1)

Environment: The OpenAI gym `Reacher-v2` environment [16] has a robotic arm with 2 DOF on a 2D arena. The state space is 8-dimensional: sine and cosine of both joint angles, and the position and velocity of the arm fingertip in x and y direction. The action controls the torques for both joints. The lengths of two bodies are $r_1 = 0.1, r_2 = 0.11$, thus the trace space of the fingertip is an annulus with $R = r_1 + r_2 = 0.21$ and $r = r_2 - r_1 = 0.01$. Since r is very small, it can be approximated as a disc with radius $R = 0.21$. The time horizon is $T = 30$. We remove the object in original reacher environment as we only focus on the fingertip trajectories.

Expert State Density: The domain is x-y coordinate of fingertip position. We experiment with the following expert densities:

- *Single Gaussian:* $\mu = (-R, 0) = (-0.21, 0), \sigma = 0.05$.
- *Mixture of two equally-weighted Gaussians:* $\mu_1 = (-R/\sqrt{2}, -R/\sqrt{2}), \mu_2 = (-R/\sqrt{2}, R/\sqrt{2}), \sigma_1 = \sigma_2 = 0.05$

Training Details: We use SAC as the underlying RL algorithm for all compared methods. The policy network is a tanh squashed Gaussian, where the mean and std is parameterized by a (64, 64) ReLU MLP with two output heads. The Q-network is a (64, 64) ReLU MLP. We use Adam to optmize both the policy and the Q-network with a learning rate of 0.003. The temperature parameter α is fixed to be 1. The replay buffer has a size of 12000, and we use a batch size of 256.

For f-IRL and MaxEntIRL, the reward function is a (64, 64) ReLU MLP. We clamp the output of the network to be within the range [-10, 10]. We also use Adam to optimize the reward network with a learning rate of 0.001.

For other baselines including AIRL, f-MAX-RKL, GAIL, we refer to the f-MAX [32] authors' official implementation[5]. We use the default discriminator architecture as in [32]. In detail, first the input is linearly embedded into a 128-dim vector. This hidden state then passes through 6 Resnet blocks of 128-dimensions; the residual path uses batch normalization and tanh activation. The last hidden state is then linearly embedded into a single-dimensional output, which is the logits of the discriminator. The logit is clipped to be within the range $[-10, 10]$. The discriminator is optimized using Adam with a learning rate of 0.0003 and a batch size of 128.

At each epoch, for all methods, we train SAC for 10 episodes using the current reward/discriminator. We warm-start SAC policy and critic networks from networks trained at previous iteration. We do not empty the replay buffer, and leverage data collected in earlier iterations for training SAC. We found this to be effective empirically, while saving lots of computation time for the bilevel optimization.

For f-IRL and MaxEntIRL, we update the reward for 2 gradient steps in each iteration. For AIRL, f-MAX-RKL and GAIL, the discriminator takes 60 gradient steps per epoch. We train all methods for 800 epochs.

f-IRL and MaxEntIRL require an estimation of the agent state density. We use kernel density estimation to fit the agent's density, using epanechnikov kernel with a bandwidth of 0.2 for point-mass, and a bandwidth of 0.02 for Reacher. At each epoch, we sample 1000 trajectories (30000 states) from the trained SAC to fit the kernel density model.

Baselines: Since we assume only access to expert density instead of expert trajectories in traditional IL framework, we use *importance sampling* for the expert term in the objectives of baselines.

- *For MaxEntIRL*: Given the reward is only dependent on state, its reward gradient can be transformed into covariance in state marginal space using importance sampling from agent states:

$$\begin{aligned}\nabla_\theta L_{\text{MaxEntIRL}}(\theta) &= \frac{1}{\alpha}\sum_{t=1}^{T}\left(\mathbb{E}_{s_t\sim\rho_{E,t}}[\nabla r_\theta(s_t)] - \mathbb{E}_{s_t\sim\rho_{\theta,t}}[\nabla r_\theta(s_t)]\right)\\ &= \frac{T}{\alpha}\left(\mathbb{E}_{s\sim\rho_E}[\nabla r_\theta(s)] - \mathbb{E}_{s\sim\rho_\theta}[\nabla r_\theta(s)]\right)\\ &= \frac{T}{\alpha}\left(\mathbb{E}_{s\sim\rho_\theta}\left[\frac{\rho_E(s)}{\hat{\rho}_\theta(s)}\nabla r_\theta(s)\right] - \mathbb{E}_{s\sim\rho_\theta}[\nabla r_\theta(s)]\right)\end{aligned} \tag{3.42}$$

  where $\rho_t(s)$ is state marginal at timestamp t, and $\rho(s) = \sum_{t=1}^{T}\rho_t(s)/T$ is state marginal averaged over all timestamps, and we fit a density model to the agent distribution as $\hat{\rho}_\theta$.

[5] https://github.com/KamyarGh/rl_swiss

- *For GAIL, AIRL, f-MAX-RKL*: Original discriminator needs to be trained using expert samples, thus we use the same density model as described above, and then use importance sampling to compute the discriminator objective:

$$\max_D L(D) = \mathbb{E}_{s\sim\rho_\theta}\left[\frac{\rho_E(s)}{\hat{\rho}_\theta(s)}\log D(s)\right] + \mathbb{E}_{s\sim\rho_\theta}[\log(1-D(s))] \tag{3.43}$$

Evaluation: For the approximation of both forward and reverse KL divergence, we use non-parametric Kozachenko-Leonenko estimator [44, 46] with lower error [78] compared to plug-in estimators using density models. Suggested by [85][6], we choose $k = 3$ in k-nearest neighbor for Kozachenko-Leonenko estimator. Thus for each evaluation, we need to collect agent state samples and expert samples for computing the estimators.

In our experiments, before training we sample $M = 10000$ expert samples and keep the valid ones within observation space. For agent, we collect 1000 trajectories of $N = 1000 * T = 30000$ state samples. Then we use these two batches of samples to estimate KL divergence for every epoch during training.

Inverse Reinforcement Learning Benchmarks (Sec 3.6.2)

Environment: We use the `Hopper-v2`, `Ant-v2`, `HalfCheetah-v2`, `Walker2d-v2` environments from OpenAI Gym.

Expert Samples: We use SAC to train expert policies for each environment. SAC uses the same policy and critic networks, and the learning rate as section 3.7.2. We train using a batch size of 100, a replay buffer of size 1 million, and set the temperature parameter α to be 0.2. The policy is trained for 1 million timesteps on Hopper, and for 3 million timesteps on the other environments. All algorithms are tested on 1, 4, and 16 trajectories collected from the expert stochastic policy.

Training Details: We train f-IRL, Behavior Cloning (BC), MaxEntIRL, AIRL, and f-MAX-RKL to imitate the expert using the provided expert trajectories.

We train f-IRL using Algorithm 2. Since we have access to expert samples, we use the practical modification described in section 3.5.4 for training f-IRL, where we feed a mixture of 10 agent and 10 expert trajectories (resampled with replacement from provided expert trajectories) into the reward objective.

SAC uses the same hyperparameters used for training expert policies. Similar to the previous section, we warm-start the SAC policy and critic using trained networks from previous iterations, and train them for 10 episodes. At each iteration, we update the reward parameters once using

[6] https://github.com/gregversteeg/NPEET

Adam optimizer. For the reward network of f-IRL and MaxEntIRL, we use the same reward structure as section 3.7.2 with the learning rate of 0.0001, and ℓ_2 weight decay of 0.001. We take one gradient step for the reward update.

MaxEntIRL is trained in the standard manner, where the expert samples are used for estimating reward gradient.

For Behavior cloning, we use the expert state-action pairs to learn a stochastic policy that maximizes the likelihood on expert data. The policy network is same as the one used in SAC for training expert policies.

For f-MAX-RKL and AIRL, we tuned the hyperparameters based on the code provided by f-MAX that is used for *state-action* marginal matching in Mujoco benchmarks. For f-MAX-RKL, we fix SAC temperature $\alpha = 0.2$, and tuned reward scale c and gradient penalty coefficient λ suggested by the authors, and found that $c = 0.2, \lambda = 4.0$ worked for $\{$Ant, Hopper, Walker2d$\}$ *with* the normalization in each dimension of states and with a replay buffer of size 200000. However, for HalfCheetah, we found it only worked with $c = 2.0, \lambda = 2.0$ *without* normalization in states and with a replay buffer of size 20000. For the other hyperparameters and training schedule, we keep them same as f-MAX original code: e.g. the discriminator is parameterized as a two-layer MLP of hidden size 128 with tanh activation and the output clipped within [-10,10]; the discriminator and policy are alternatively trained once for 100 iterations per 1000 environment timesteps.

For AIRL, we re-implement a version that uses SAC as the underlying RL algorithm for a fair comparison, whereas the original paper uses TRPO. Both the reward and the value model are parameterized as a two-layer MLP of hidden size 256 and use ReLU as the activation function. For SAC training, we tune the learning rates and replay buffer sizes for different environments, but find it cannot work on all environments other than HalfCheetah even after tremendous tuning. For reward and value model training, we tune the learning rate for different environments. These hyper-parameters are summarized in table 3.6. We set $\alpha = 1$ in SAC for all environments. For every 1000 environment steps, we alternatively train the policy and the reward/value model once, using a batch size of 100 and 256.

Hyper-parameter	Ant	Hopper	Walker	HalfCheetah
SAC learning rate	$3e-4$	$1e-5$	$1e-5$	$3e-4$
SAC replay buffer size	1000000	1000000	1000000	10000
Reward/Value model learning rate	$1e-4$	$1e-5$	$1e-5$	$1e-4$

Table 3.6: AIRL IRL benchmarks task-specific hyper-parameters.

Evaluation: We compare the trained policies by f-IRL, BC, MaxEntIRL, AIRL, and f-MAX-RKL by computing their returns according to the ground truth return on each environment. We report the mean of their performance across 3 seeds.

For the IRL methods, f-IRL, MaxEntIRL, and AIRL, we also evaluate the learned reward functions. We train SAC on the learned rewards, and evaluate the performance of learned policies according to ground-truth rewards.

3.7.3 Reward Prior for Downstream Hard-exploration Tasks (Sec 3.6.3.1)

Environment: The pointmass environment has 2D square state space with range $[0, 6]^2$, and 2D actions that control the delta movement of the agent in each dimension. The agent starts from the bottom left corner at coordinate $(0, 0)$.

Task Details: We designed a hard-to-explore task for the pointmass. The grid size is 6×6, the agent is always born at $[0, 0]$, and the goal is to reach the region $[5.95, 6] \times [5.95, 6]$. The time horizon is $T = 30$. The agent only receives a reward of 1 if it reaches the goal region. To make the task more difficult, we add two distraction goals: one is at $[5.95, 6] \times [0, 0.05]$, and the other at $[0, 0.05] \times [5.95, 6]$. The agent receives a reward of 0.1 if it reaches one of these distraction goals. Vanilla SAC always converges to reaching one of the distraction goals instead of the real goal.

Training Details: We use SAC as the RL algorithm. We train SAC for 270 episodes, with a batch size of 256, a learning rate of 0.003, and a replay buffer size of 12000. To encourage the exploration of SAC, we use a random policy for the first 100 episodes.

3.7.4 Reward Transfer across Changing Dynamics (Sec 3.6.3.2)

Environment: In this experiment, we use Mujoco to simulate a healthy Ant, and a disabled Ant with two broken legs (Figure 3.5). We use the code provided by Fu et al. [28]. Note that this Ant environment is a slightly modified version of the `Ant-v2` available in OpenAI gym.

Expert Samples: We use SAC to obtain a forward-running policy for the Ant. We use the same network structure and training parameters as section 3.7.2 for training this policy. We use 16 trajectories from this policy as expert demonstrations for the task.

Training Details: We train f-IRL and MaxEntIRL using the same network structure and training parameters as section 3.7.2. We also run AIRL, but couldn't match the performance reported in Fu et al. [28].

Evaluation: We evaluate f-IRL and MaxEntIRL by training a policy on their learned rewards using SAC. We report the return of this policy on the disabled Ant environment according to

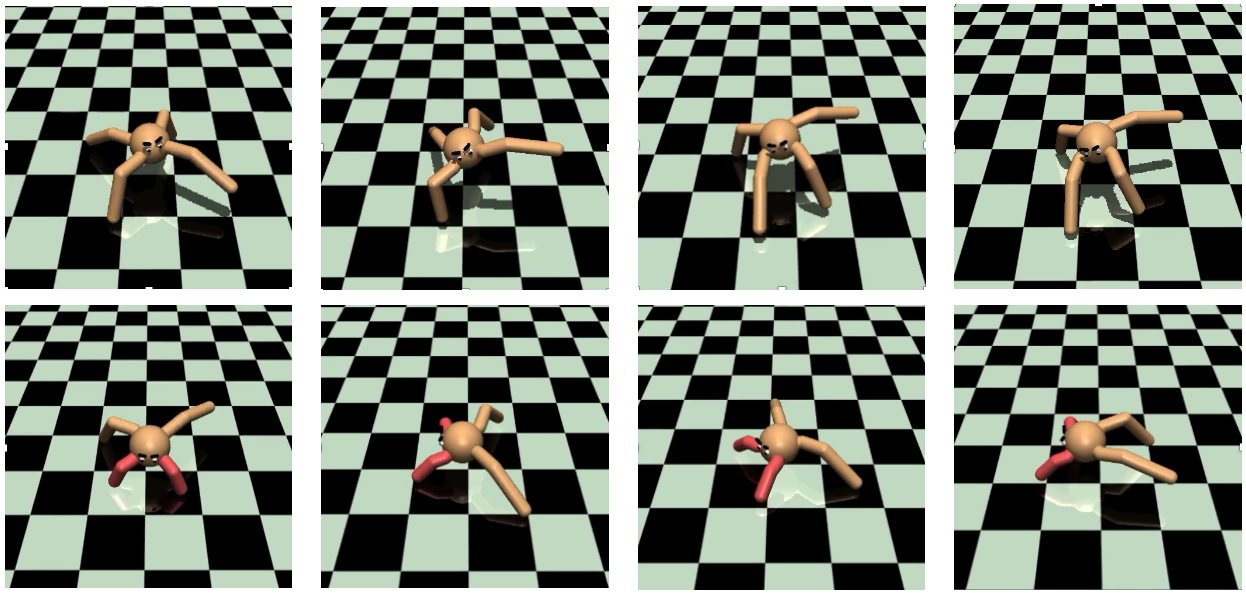


Figure 3.5: **Top row**: A healthy Ant executing a forward walk. **Bottom row**: A successful transfer of walking behavior to disabled Ant with 2 legs active. The disabled Ant learns to use the two disabled legs as support and crawl forward, executing a very different gait than previously seen in healthy Ant.

the ground-truth reward for forward-running task. Note that we directly report results for policy transfer using GAIL, and AIRL from Fu et al. [28].

3.7.5 Additional Experiment Results

Inverse RL Benchmark Unnormalized Performance

In this section, we report the unnormalized return of the agent stochastic policy for ease of comparison to expert in Table 3.7. We analyze situations when we are provided with 1,4 and 16 expert trajectories respectively. For IL/IRL methods, all the results are averaged across three random seeds to show the mean and standard deviation in the last 10% training iterations.

Note that for the row of "Expert return", we compute the mean and std among the expert trajectories (by stochastic policy) we collected, so for one expert trajectory, it does not have std. Moreover, since we pick the best expert trajectories for training IL/IRL algorithms, the std of "Expert return" is often lower than that of IL/IRL.

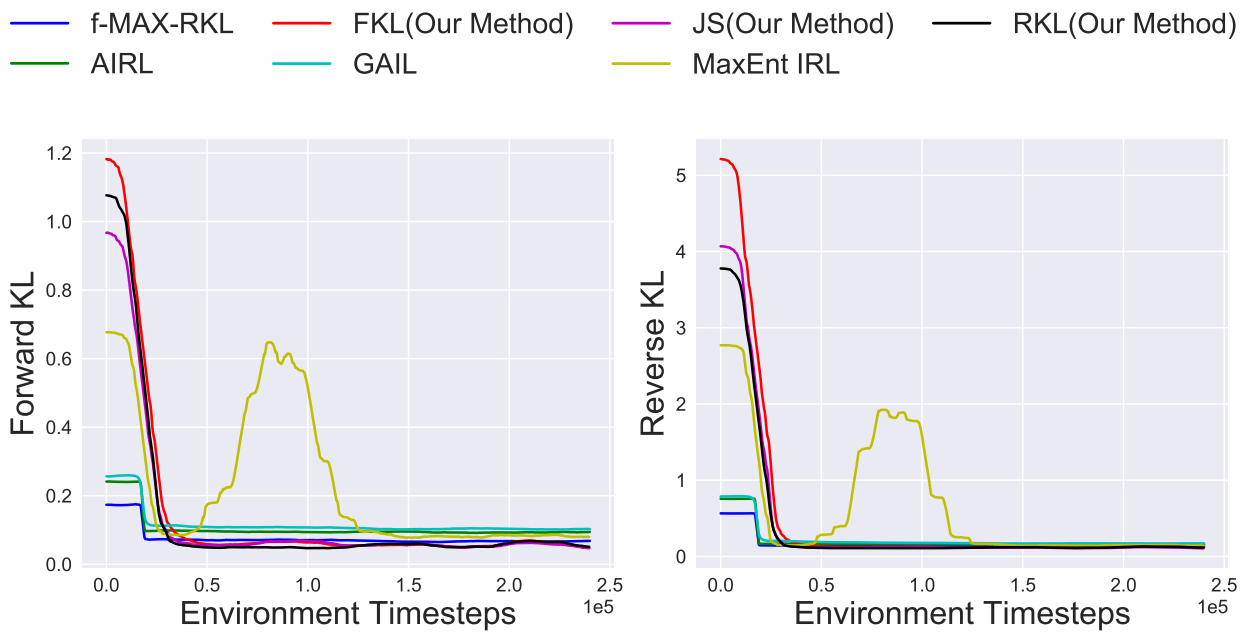


Figure 3.6: Forward and Reverse KL curves in pointmass environment for Gaussian target density for all the methods during training. Smoothed in a window of 120 evaluations.

Additional Result of Reward Transfer across Changing Dynamics

In section 3.6.3 ("Reward transfer across changing dynamics") we show the result of the setting with 32 expert trajectories provided. We follow AIRL paper [28] setting for this experiment, but the number of experiment trajectories used in their experiment is *unknown*. We use a maximum of 50 expert trajectories and show the best seed performance in Table 3.9. Note that this table has same values as Table 3.5 except for our method. We see that with more expert trajectories f-IRL is able to outperform baselines with a large margin. The disabled Ant agent is able to learn a behavior to walk while taking support from both of its disabled legs.

Matching the Specified Expert State Density on PointMass

We also conducted the experiment described in section 3.7.2 on the pointmass environment similar to that in section 3.7.3. This environment has size $[4, 4]$, and the target density is a unimodal Gaussian with $\mu = (2, 2), \sigma = 0.5$ for goal-reaching task.

This experiment is didactic in purpose. In Figure 3.7, we observe that all methods converge (MaxEntIRL is slightly unstable) and are able to reduce the FKL and RKL to near zero.

In Figure 3.6, we observe that rewards learned by f-IRL using Forward KL and Reverse KL

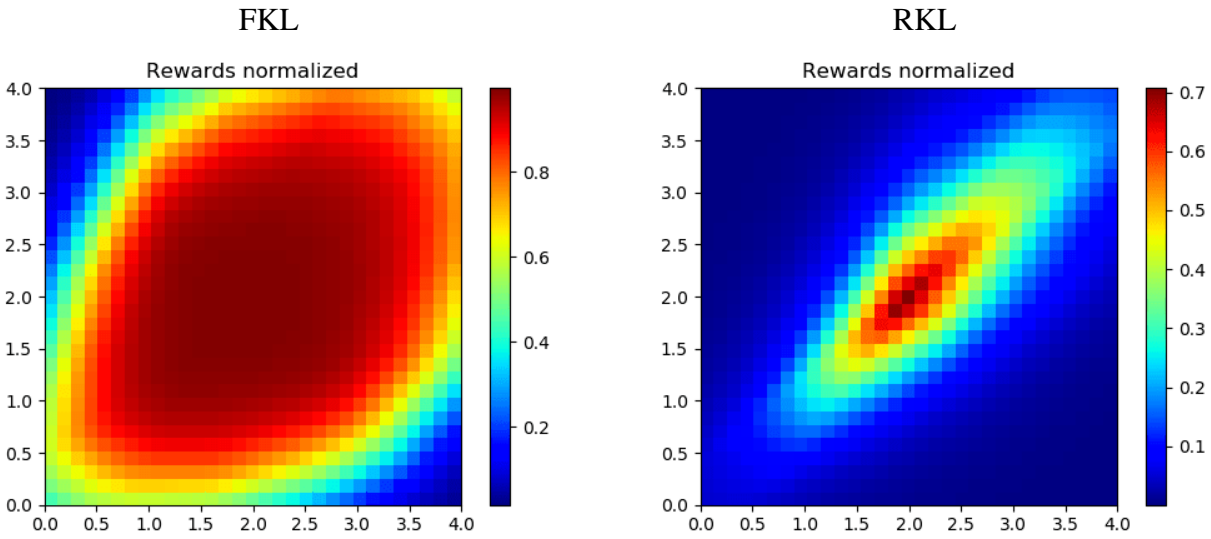


Figure 3.7: Forward and Reverse KL curves in pointmass environment for Gaussian target density for all the methods during training. Smoothed in a window of 120 evaluations.

divergence objective demonstrate the expected *mode-covering* and *mode-seeking* behavior, respectively.

Method	Hopper		
# Expert traj	1	4	16
Expert return	3570.87	3585.59 $\pm$ 12.39	3496.62 $\pm$ 10.13
BC	17.39 $\pm$ 5.99	468.49 $\pm$ 83.94	553.56 $\pm$ 46.70
MaxEntIRL	3309.72 $\pm$ 171.28	3300.81 $\pm$ 229.84	**3298.50** $\pm$ 255.35
f-MAX-RKL	**3349.62** $\pm$ 68.89	3326.83 $\pm$ 85.42	3165.51 $\pm$ 102.83
AIRL	49.12 $\pm$ 2.58	49.33 $\pm$ 3.93	48.63 $\pm$ 5.88
FKL (f-IRL)	3329.94 $\pm$ 152.33	3243.83 $\pm$ 312.44	3260.35 $\pm$ 175.58
RKL (f-IRL)	3276.55 $\pm$ 221.27	3303.44 $\pm$ 286.26	3250.74 $\pm$ 161.89
JS (f-IRL)	3282.37 $\pm$ 202.30	**3351.99** $\pm$ 172.70	3269.49 $\pm$ 160.99

Method	Walker2d		
# Expert traj	1	4	16
Expert return	5468.36	5337.85 $\pm$ 92.46	5368.01 $\pm$ 78.99
BC	-2.03 $\pm$ 1.05	303.24 $\pm$ 6.95	431.60 $\pm$ 63.68
MaxEntIRL	4823.82 $\pm$ 512.58	4697.11 $\pm$ 852.19	**4884.30** $\pm$ 467.16
f-MAX-RKL	2683.11 $\pm$ 128.14	2628.10 $\pm$ 548.93	2498.78 $\pm$ 824.26
AIRL	9.8 $\pm$ 1.82	9.24 $\pm$ 2.28	8.45 $\pm$ 1.56
FKL (f-IRL)	**4927.02** $\pm$ 615.34	4809.80 $\pm$ 750.05	4851.81 $\pm$ 547.12
RKL (f-IRL)	4847.12 $\pm$ 806.61	4806.72 $\pm$ 433.02	4578.39 $\pm$ 564.17
JS (f-IRL)	4888.09 $\pm$ 664.86	**4935.42** $\pm$ 384.15	4725.78 $\pm$ 613.45

Table 3.7: Benchmark of Mujoco Environment, from top to bottom, Hopper-v2, Walker2d-v2.

Method	HalfCheetah		
# Expert traj	1	4	16
Expert return	12258.71	11944.45 ± 985.08	12406.29 ± 614.02
BC	-367.56 ± 23.57	209.59 ± 178.71	287.05 ± 109.32
MaxEntIRL	**11637.41** ± 438.16	11685.92 ± 478.95	11228.32 ± 1752.32
f-MAX-RKL	8688.37 ± 633.58	4920.66 ± 2996.15	8108.81 ± 1186.77
AIRL	2366.84 ± 175.51	2343.17 ± 103.51	2267.68 ± 83.59
FKL (f-IRL)	11556.23 ± 539.83	11556.51 ± 673.13	11642.72 ± 629.29
RKL (f-IRL)	11612.46 ± 703.25	11644.19 ± 488.79	**11899.50** ± 605.43
JS (f-IRL)	11413.47 ± 1227.89	**11686.09** ± 748.30	11711.77 ± 1091.74

Method	Ant		
# Expert traj	1	4	16
Expert return	5926.18	5859.09 ± 88.72	5928.87 ± 136.44
BC	-113.60 ± 12.86	1321.69 ± 172.93	2799.34 ± 298.93
MaxEntIRL	3179.23 ± 2720.63	4171.28 ± 1911.67	4784.78 ± 482.01
f-MAX-RKL	3585.03 ± 255.91	3810.56 ± 252.57	3653.53 ± 403.73
AIRL	-54.7 ± 28.5	-14.15 ± 31.65	-49.68 ± 41.32
FKL (f-IRL)	**4859.86** ± 302.94	**4861.91** ± 452.38	**4971.11** ± 286.81
RKL (f-IRL)	3707.32 ± 2277.74	4814.58 ± 376.13	4813.80 ± 361.93
JS (f-IRL)	4590.11 ± 1091.22	4745.11 ± 348.97	4342.39 ± 1296.93

Table 3.8: Benchmark of Mujoco Environment, from top to bottom HalfCheetah-v2, Ant-v2.

Policy Transfer using GAIL	AIRL	MaxEntIRL	f-IRL	Ground-truth Reward
-29.9	130.3	145.5	**232.5**	315.5

Table 3.9: Returns obtained after transferring the policy/reward on modified Ant environment using different IL methods. In this case, we report the performance of best seed with a maximum of 50 expert trajectories.

Chapter 4

Lyapunov Barrier Policy Optimization

Current reinforcement learning methods are trained without any notion of safe behavior. As a result, these methods might cause damage to themselves, their environment, or even harm other agents in the scene. Ideally, an agent in a real-world setting should start with a conservative policy and iteratively refine it while maintaining safety constraints. For example, an agent that is learning to drive around other agents should start driving slowly and gradually learn to improve its performance by exploring carefully while avoiding accidents. In contrast, most deep reinforcement learning methods learn by trial and error without taking into consideration the safety-related consequences of their actions [77, 88, 3]. In this work, we address the problem of learning control policies that optimize a reward function while satisfying some predefined constraints throughout the learning process.

As in previous work for safe reinforcement learning, we use human-defined constraints to specify safe behavior. A classical model for RL with constraints is the constrained Markov Decision Process (CMDP) [6], where an agent tries to maximize the standard RL objective of expected returns while satisfying constraints on expected costs. A number of previous works on CMDPs mainly focus on environments that have low dimensional action spaces, and they are difficult to scale up to more complex environments [84, 89]. One popular approach to solve Constrained MDPs in large state and action spaces is to use the Lagrangian method [5, 66]. This method augments the original RL objective with a penalty on constraint violations and computes the saddle point of the constrained policy optimization via primal-dual methods [5]. While safety is ensured asymptotically, no guarantees are made about safety during training. As we show, other methods which claim to maintain safety during training also lead to many safety violations during training in practice. In other recent work, [20, 21] use Lyapunov functions to explicitly model constraints in the CMDP framework to guarantee safe policy updates. We build on this idea, where the Lyapunov

function allow us to convert trajectory-based constraints in the CMDP framework to state-based constraints which are much easier to deal with.

In this work, we present a new method called LBPO (Lyapunov Barrier Policy Optimization) for safe reinforcement learning in the CMDP framework. We formulate the policy update as an unconstrained update augmented by a barrier function which ensures that the policy lies in the set of policies induced by the Lyapunov function, thereby guaranteeing safety. We show that LBPO allows us to control the amount of risk-aversion of the agent by adjusting the barrier. We also analyze previous baselines that use a backtracking recovery rule and empirically show that their near-constraint satisfaction can be explained by their recovery rule; this approach leads to many constraint violations in practice. Finally, we demonstrate that LBPO outperforms state-of-the-art CMDP baselines in terms of the number of constraint violations while being competitive in performance.

4.1 Related Work

Constrained Markov Decision Process CMDP's [5] have been a popular framework for incorporating safety in the form of constraints. In CMDP's the agent tries to maximize expected returns by satisfying constraints on expectation of costs. [6] demontrated that for finite MDP with known models, CMDP's can be solved by solving the dual LP program. For large state dimensions (or continuous), solving the LP becomes intractable. A common way to solve CMDP in large spaces is to use the Lagrangian Method [6, 31, 19]. These methods augment the original RL objective with a penalty on constraint violation and computes the saddle point of the constrained policy optimization via primal-dual methods. These methods give no guarantees of safety during training and are only guaranteed asymptotically at convergence. CPO [2] is another method for solving CMDP's that derives an update rule in the trust region which guarantees monotonic policy improvement under constraint satisfaction, similar to TRPO [72]. [20, 21] presents another class of method that formulates safe policy update under a Lyapunov constraint. [62] explored the relevance of Lyapunov functions in control and [12] used Lyapunov functions in RL to guarantee exploration such that the agent can return to a "region of attraction" in the model-based regime. In our work, we show that previous baselines rely on a backtracking recovery rule to ensure near constraint satisfaction and are sensitive to Q-function errors; we present a new method that uses a Lyapunov constraint with a barrier function to ensure a conservative policy update.

Other notions of safety. Recent works [63, 23] use a safety layer along with the policy, which ensures that all the unsafe actions suggested by the policy are projected in the safe set. [23] sat-

isfies state-based costs rather than trajectory-based costs. [82] utilizes demonstrations to ensure safety in the model based framework, and [94] learns the epistemic uncertainty of the environment by training the model in simulation for a distribution of environments, which is then used to cautiously adapt the policy while deploying on a new test environment. Another line of work focuses on optimizing policies that minimize an agent's conditional value at risk (cVAR). cVAR [68] is commonly used in quantitative finance, which aims to maximize returns in the worst $\alpha\%$ of cases. This allows the agent to ensure that it learns safe policies for deployment that achieve high reward under the aleatoric uncertainty of the MDP [81, 43, 80, 41, 14, 18].

4.2 Background

We consider the Reinforcement Learning setting where an agent's interaction with the environment is modeled as a Markov Decision Process (MDP). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r, s_0)$ with state-space $\mathcal{S}$, action-space $\mathcal{A}$, dynamics $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and initial state s_0. $P(.|s, a)$ is the transition probability distribution and $r(s, a) \in [0, R_{\text{max}}]$. We focus on the special case of constrained Markov decision processes (CMDP) [6], which is an augmented version of MDP with additional costs and trajectory-based constraints. A CMDP is represented as a tuple $(\mathcal{S}, \mathcal{A}, P, r, c, s_0, d_0)$. The terms $\mathcal{S}, \mathcal{A}, P, r, s_0$ are the same as in the unconstrained MDP; the additional terms $c(s)$ is the immediate cost and $d_0 \in \mathcal{R}_{\geq 0}$ is the maximum allowed value for the expected cumulative cost of the policy. We define a generic version of the Bellman operator w.r.t policy π and a function $h(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as follows:

$$\mathcal{B}_{\pi,h}[V][s] = \sum_a \pi(a|s)[h(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V(s')] \tag{4.1}$$

The function $h(s, a)$ can be instantiated to be the reward function or the cost function. When it is the reward function, this becomes the normal Bellman operator in RL. When $h(s, a)$ is replaced by the cost function $c(s)$, it becomes the Bellman operator over the cost objective, which will be used later in designing the Lyapunov function. We further define $J_\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0, \pi]$ to be the performance of the policy π, $D_\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)|s_0, \pi]$ to be the expected cumulative cost of the policy π, where π belongs to the set of stationary policies $\mathcal{P}$. Given a CMDP, we are interested in finding a solution to the following constrained optimization problem:

$$\max_{\pi \in \mathcal{P}}[J_\pi(s_0)] \text{ s.t } D_\pi(s_0) \leq d_0 \tag{4.2}$$

4.2.1 Safe Reinforcement Learning using Lyapunov Functions

We will build upon the Lyapunov framework introduced by [20], also known as SDDPG. It proposes to use Lyapunov functions to derive a policy improvement procedure with safety guarantees. The basic idea is that, given a safe baseline policy π_B, it finds a set of safe policies based on π_B using Lyapunov functions. For each policy improvement step, it will then choose the policy with the best performance within this set.

The method works by first designing a Lyapunov function for a safe update around the current safe baseline policy π_B. A set of Lyapunov functions is defined as follows:

$$\mathcal{L}_{\pi_B}(s_0, d_0) = \{L : S \to R_{\geq 0} : \mathcal{B}_{\pi_B,c}[L](s) \leq L(s), \forall s \in S; L(s_0) \leq d_0\} \tag{4.3}$$

The Lyapunov functions in this set are designed in a way to construct provably safe policy updates. Given any Lyapunov function within this set $L_{\pi_B} \in \mathcal{L}_{\pi_B}(s_0, d_0)$, we define the set of policies that are consistent with it to be the L_{π_B}-induced policies:

$$\mathcal{I}_{L_{\pi_B}} = \{\pi \in \mathcal{P} : \mathcal{B}_{\pi,c}[L_{\pi_B}](s) \leq L_{\pi_B}(s), \forall s \in \mathcal{S}\} \tag{4.4}$$

It can be shown that any policy π in the L_{π_B}-induced policy set is "safe", i.e $D_\pi(s_0) \leq d_0$ [20].

The choice of L_{π_B} affects the L_{π_B}-induced policy set. We need to construct L_{π_B} such that the L_{π_B}-induced policy set contains the optimal policy π^*. [20] show that one such Lyapunov function is $L_{\pi_B,\epsilon}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t (c(s_t) + \epsilon(s_t))|\pi_B, s]$, where $\epsilon(s_t) \geq 0$. The function $L_{\pi_B,\epsilon}(s)$ can be thought of as a cost-value function for policy π_B augmented by an additional per-step cost $\epsilon(s_t)$. Accordingly, we can define the following state-action value function:

$$Q_{L_{\pi_B,\epsilon}}(s, a) = c(s) + \epsilon(s) + \gamma \sum_{s'} P(s'|s, a) L_{\pi_B,\epsilon}(s') \tag{4.5}$$

It was shown in [20] that finding a state dependent function ϵ such that the the optimal policy is inside the corresponding $L_{\pi_B,\epsilon}$-induced set is generally not possible and requires knowing the optimal policy. As an approximation, they suggest to create the Lyapunov function with the largest auxiliary cost $\hat{\epsilon}$, such that $L_{\pi_B,\hat{\epsilon}}(s) \geq \mathcal{B}_{\pi_B,c}[L_{\pi_B,\hat{\epsilon}}](s)$ and $L_{\pi_B,\hat{\epsilon}}(s_0) \leq d_0$. A larger auxiliary cost ϵ per state ensures that we have a larger set of L-induced policies, making it more likely to include the optimal policy in the set. The authors show that the following $\hat{\epsilon}(s)$ in the form of a constant function satisfies the conditions described:

$$\hat{\epsilon}(s) = (1 - \gamma)(d_0 - D_{\pi_B}(s_0)) \tag{4.6}$$

Plugging this function $\hat{\epsilon}(s)$ and the definition of $Q_{L_{\pi_B,\epsilon}}(s,a)$ into the CMDP objective, the policy update under the set of policies that lie in the $L_{\pi_B,\hat{\epsilon}}$-induced policy set, or equivalently the policies that are safe, is given by:

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ s.t \int_{a \in \mathcal{A}} (\pi(a|s) - \pi_B(a|s)) Q_{L_{\pi_B,\hat{\epsilon}}}(s,a) da \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{4.7}$$

In the case of a deterministic policy, the policy update becomes:

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ \text{s.t}\ Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi(s)) - Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi_B(s)) \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{4.8}$$

We build upon this objective in our work. We include the proof of the Lyapunov approach for completeness in Appendix 4.5.1, and we advise the reader to see previous work [20] for a more detailed derivation. Using the Lyapunov function, the trajectory-based constraint of the CMDP is converted to a per-state constraint (Eq. 4.7), which is often much easier to deal with.

4.3 Method

4.3.1 Barrier function for Lyapunov Constraint

We present Lyapunov Barrier Policy Optimization (LBPO) that aims to update policies inside the $L_{\pi_B,\hat{\epsilon}}$-induced policy set. We work under the standard policy iteration framework which contains two steps: Q-value Evaluation and Safe Policy Improvement. We initialize LBPO with a safe baseline policy π_B. In practice, we can obtain safe initial policies using a simple (usually poorly performing) hand-designed control policy; in our experiments, we simplify this process and achieve safe initial policies by training on the safety objective. We assume that we have m different constraints, as LBPO naturally generalizes to more than one constraint.

Q Evaluation

We use on-policy samples to evaluate the current policy. We compute a reward Q function Q^R, and cost Q functions Q^{C_i} corresponding to each cost constraint $i \in [1, 2...m]$. Each Q function is updated using TD(λ) [79] which helps us more accurately estimate the Q functions. Furthermore, we use the on-policy samples to get the cumulative discounted cost $D^i_\pi(s_0)$ of the current policy, which allows us to set up the constraint budget for each constraint given by $\epsilon_i = (1-\gamma)(d^i_0 - D^i_\pi(s_0))$ as shown in Eq. 4.6.

Regularized Safe Policy Update

In this work, we focus on deterministic policies, where we have the following policy update under the L-induced set for each constraint as given in Eq. 4.8 :

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0) \text{ s.t } Q^i_{L_{\pi_B,\hat{\epsilon}}}(s, \pi(s)) - Q^i_{L_{\pi_B,\hat{\epsilon}}}(s, \pi_B(s)) \leq \hat{\epsilon}_i(s) \; \forall i \in [1, 2, ...m], \forall s \in \mathcal{S} \tag{4.9}$$

We can simplify this equation further by replacing $Q^i_{L_{\pi_B,\hat{\epsilon}}}$ with $Q^{C_i}_{\pi_B}$ which is the i^{th} cost Q-function under the policy π_B, when ϵ is a constant function (see Appendix 4.5.1). To ensure that the Lyapunov constraints are satisfied, we construct an unconstrained objective using an indicator penalty $I(Q^{C_i}_{\pi_B}(s, \pi_\theta(s)))$ for each constraint.

$$I(Q^{C_i}_{\pi_B}(s, \pi_\theta(s))) = \begin{cases} 0 & Q^{C_i}_{\pi_B}(s, \pi_\theta(s)) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) \leq \hat{\epsilon}_i(s) \\ \infty & Q^{C_i}_{\pi_B}(s, \pi_\theta(s)) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) > \hat{\epsilon}_i(s) \end{cases} \tag{4.10}$$

We will use a differentiable version of the indicator penalty called the logarithmic barrier function:

$$\psi(Q^{C_i}_{\pi_B}(s, \boldsymbol{\pi_\theta(s)})) = -\beta \log \left(\hat{\epsilon}(s) - (Q^{C_i}_{\pi_B}(s, \boldsymbol{\pi_\theta(s)}) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) \right) \tag{4.11}$$

The function ψ is parameterized by θ and $Q^{C_i}_{\pi_B}(s, \pi_B(s))$ is a constant. Our policy update will use the gradient at $\pi_\theta = \pi_B$, ensuring that the logarithmic barrier function is well defined, since $\hat{\epsilon}(s) > 0 \; \forall s$.

Figure 4.1 captures the behavior of the function ψ for different β. The parameter β captures the amount of risk-aversion we desire from the agent. A high β will help avoid constraint violations occurring due to approximation errors in our learned Q-functions. We verify this empirically in Section 4.4.

We use the Deterministic Policy Gradient Theorem [76] for the policy update. For updating a θ-parameterized policy with respect to the expected return, the objective can be written as: $\text{argmin}_\theta \, \mathbb{E}_{s \sim \rho^{\pi_B}} \left[(-Q^R_{\pi_B}(s, \pi_\theta(s))) \right]$, where ρ^{π_B} is the on-policy state distribution.

Since we rely on on-policy samples for Q-function evaluation, the Q-function estimation outside the on-policy state distribution can be arbitrarily bad. Similar to [72], we constrain our policy update using a hard KL constraint (i.e. a trust region) between the current policy and the updated policy under the presence of stochastic exploration noise. The trust region also allows us to make sure that our policy change is bounded, which allows us to ensure that with a small enough trust region, our first order approximation of the objective is valid.

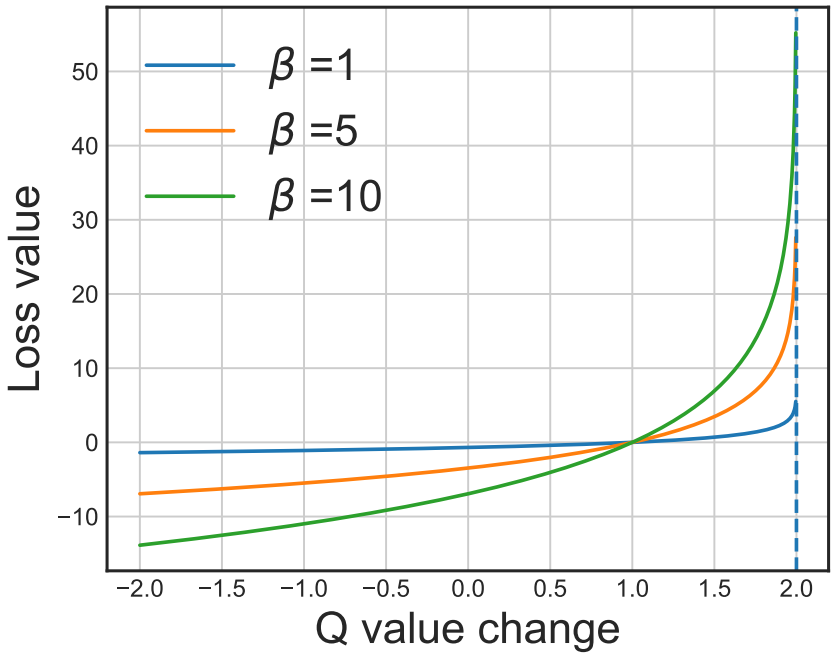


Figure 4.1: As the difference between Q values for action a and the baseline action reaches ϵ (in this case $\epsilon = 2$), the loss increases to ∞.

Augmenting the on-policy update with the Lyapunov barrier and a KL regularization, we have the following LBPO policy update:

$$\theta_{k+1} = \text{argmin}_\theta \, \mathbb{E}_{s\sim\rho^{\pi_B}} \left[-Q^R_{\pi_B}(s, \pi_\theta(s)) + \sum_{i=1}^{m} \psi(Q^{C_i}_{\pi_B}(s, \pi_\theta(s))) \right] \tag{4.12}$$

$$\text{subject to } \mathbb{E}_{s\sim\rho^{\pi_B}} [D_{\text{KL}}(\pi_\theta[.|s] + \mathcal{N}(0, \delta) \parallel \pi_B[.|s] + \mathcal{N}(0, \delta))] < \mu \tag{4.13}$$

where δ is the exploration noise, ρ^{π_B} is the state distribution induced by the current policy, μ is the expected KL constraint threshold and we set π_B to the safe policy at iteration k, as the update guarantees safe policies at each iteration. In practice, we expand our objective using a Taylor series expansion and solve to a leading order approximation around θ_k. Letting the gradient of the objective in Eq 4.12 be denoted by g and the Hessian of the KL divergence by H, our objective becomes:

$$\theta_{k+1} = \text{argmin}_\theta \, g^T(\theta - \theta_k), \quad \text{subject to } \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \mu \tag{4.14}$$

We solve this constrained optimization using the Fisher vector product with Conjugate gradient method similar to [72].

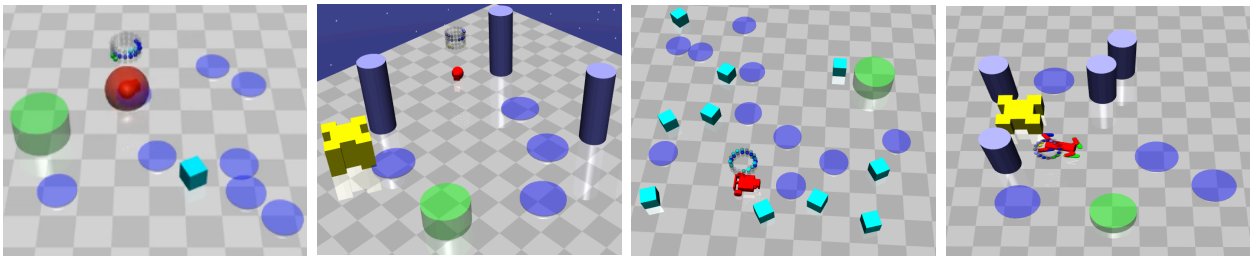


Figure 4.2: OpenAI Safety Environments: PointGoal1, PointPush2, CarGoal2, DoggoPush2

4.4 Experiments

In this section, we evaluate LBPO and compare it to prior work. First, we benchmark our method against previous baselines to show that LBPO can achieve better constraint satisfaction while being competitive in performance. Second, we give empirical evidence that previous methods near constraint satisfaction can be explained by backtracking. Third, we show by a didactic example that LBPO is more robust than CPO and SDDPG to Q-function errors, hence making it a preferable alternative, especially when function approximation is used. Finally, we show that LBPO allows flexible tuning of the amount of risk-aversion for the agent.

Comparisons. For our experiments, we compare LBPO against a variety of baselines: PPO, PPO-lagrangian, SDDPG, CPO and BACKTRACK. PPO [73] is an on-policy RL algorithm that updates in an approximate trust-region without considering any constraints. PPO-lagrangian belongs to a class of Lagrangian methods [5] for safe Reinforcement Learning, which transforms the constrained optimization problem to a penalty form

$$\max_{\pi\in\mathcal{P}}\min_{\lambda\geq 0}\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)+\lambda(\sum_{t=0}^{\infty}\gamma^t c(s_t)-d_0)|\pi,s_0]$$

π and λ are jointly optimized to find a saddle point of the penalized objective. SDDPG [20, 21] introduces the Lyapunov framework for safe-RL and proposes an action-projection method which in theory guarantees the update of the policy within a safe set. We evaluate the α-projection version of SDDPG [21]. Since the original implementation for the method is unavailable, we re-implemented the method to the best of our abilities. CPO [2] derives a trust-region update rule which guarantees the monotonic improvement of the policy while satisfying constraints. CPO also uses a backtracking recovery rule. We elaborate on the BACKTRACK baseline in Section 4.4.2.

Tasks. We evaluate these methods using the OpenAI Safety Gym [66], which consists of 12 continuous control MuJoCo tasks [83]. These tasks use 3 robots: Point, Car, and Doggo. Point is the simplest of three which can be commanded to move forward/backward or to turn.

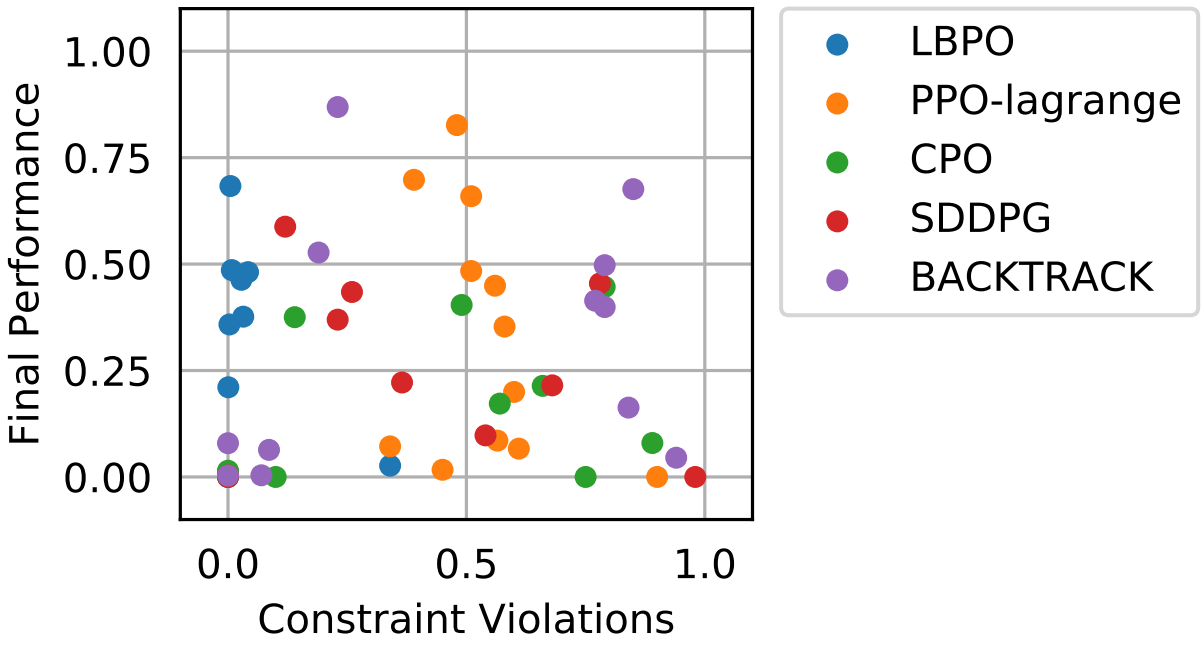


Figure 4.3: Each point corresponds to a particular safety method applied to a certain safety environment. The x-axis shows the fraction of constraint violations encountered by the behavior policy during training and y-axis shows the policy performance normalized by the corresponding environment's PPO return.

Car has two driven wheels which needs to be controlled together to obtain forward/backward and turning behavior. Doggo is a quadrupedal robot whose joint angles at hip, knee and torso can be commanded to obtain similar behavior. Each robot has 2 types of tasks (Goal, Push) with 2 difficulty levels (1, 2). In Goal tasks, the robot has to move to a series of goal locations, and in Push tasks, the robot has to push a box to a series of goal locations. There are mobile and immobile obstacles made up of a hazard region, vases and pillars which generate a cumulative penalty for the agent. Point has an observation space of 60 dimensions, Car has 72 dimensions, and Doggo has 104 dimensions, which constitute sensor readings, joint angles, and velocities. The environments are shown in Figure 4.2.

4.4.1 Safe Reinforcement Learning Benchmarks

We summarize the comparison of LBPO to all of the baselines (PPO, PPO-lagrangian, BACK-TRACK, SDDPG and CPO) on the OpenAI safety benchmarks in Figure 4.3 and Tables 4.1 and 4.2. Additional training plots for policy return and policy cost can be found in Appendix 4.5.2.

Method	PPO	PPO-lagrangian	CPO	SDDPG	BACKTRACK	LBPO
PointGoal1	1.00	0.48	0.79	0.78	0.85	**0.04**
PointGoal2	0.98	0.60	0.75	0.98	0.94	**0.34**
PointPush1	1.00	0.51	0.14	0.12	0.19	**0.00**
PointPush2	1.00	0.51	0.66	0.36	0.77	**0.00**
CarGoal1	1.00	0.56	0.89	0.54	0.79	**0.03**
CarGoal2	1.00	0.61	0.57	0.68	0.84	**0.00**
CarPush1	0.99	0.39	0.10	0.26	0.23	**0.01**
CarPush2	1.00	0.58	0.49	0.23	0.79	**0.03**
DoggoGoal1	1.00	0.90	**0.00**	**0.00**	0.07	**0.00**
DoggoGoal2	1.00	0.45	**0.00**	**0.00**	**0.00**	**0.00**
DoggoPush1	0.98	0.56	**0.00**	**0.00**	**0.00**	**0.00**
DoggoPush2	1.00	0.34	**0.00**	**0.00**	0.09	**0.00**

Table 4.1: We report the fraction of unsafe behavior policies encountered during training across different OpenAI safety environments for the policy updates across $2e^7$ training timesteps. LBPO obtains fewer constraint violations consistently across all environments.

Constraint Satisfaction. Table 4.1 shows that in all the environments, LBPO actively avoids constraint violations, staying below the threshold in most cases. In the PointGoal2 environment, no

Method	PPO	PPO-lagrangian	CPO	SDDPG	BACKTRACK	LBPO
PointGoal1	1.00	**0.826**	0.450	0.451	0.670	0.480
PointGoal2	1.00	**0.200**	0.000	0.000	0.045	0.026
PointPush1	1.00	0.659	0.375	0.587	0.527	**0.683**
PointPush2	1.00	**0.483**	0.213	0.221	0.413	0.358
CarGoal1	1.00	0.449	0.079	0.097	**0.497**	0.376
CarGoal2	1.00	0.066	0.172	**0.215**	0.162	0.210
CarPush1	1.00	0.697	0.000	0.434	**0.868**	0.485
CarPush2	1.00	0.353	0.403	0.369	0.399	**0.430**
DoggoGoal1	1.00	0.000	0.003	0.002	0.003	**0.007**
DoggoGoal2	1.00	**0.016**	0.002	0.003	0.003	0.003
DoggoPush1	1.00	**0.080**	0.014	0.001	0.079	0.012
DoggoPush2	1.00	**0.071**	0.000	0.000	0.063	0.000

Table 4.2: Cumulative return of the converged policy for each safety algorithm normalized by PPO's return. Negative returns are clipped to zero. LBPO tradeoffs return for better constraint satisfaction. Bold numbers show the best performance obtained by a safety algorithm (thus excluding PPO).

method can achieve good constraint satisfaction which we attribute to the nature of the environment as it was found that safe policies were not obtained even when training only on the cost objective. In all the other cases we note that LBPO achieves near-zero constraint violations.

Like our method, SDDPG also builds upon the optimization problem from Equation 4.8 but solves this optimization using a projection onto a safe set instead of using a barrier function. We noticed the following practical issues with this approach: First, in SDDPG, each safe policy is composed of a projection layer, which itself relies on previous safe policies. This requires us to maintain all of the previous policies and thus the memory requirement grows linearly with the number of iterations. SDDG circumvents this issue by using a policy distillation scheme [20], which behavior clones the safe policy into a parameterized policy not requiring a projection layer. However, behavior cloning introduces errors in the policy leading to frequent constraint violations. Second, we will show in section 4.4.3 that SDDPG is more sensitive to Q-function errors. PPO-lagrangian produces policies that are only safe *asymptotically* and makes no guarantee of the safety of the behavior policy during each training iteration. In practice, we observe that it often violates constraints during training.

Behavior Policy Performance. OpenAI safety gym environment provide a natural tradeoff between reward and constraint. A better constraint satisfaction often necessitates a lower perfor-

mance. We observe in Table 4.2 that LBPO achieves performance competitive to the baselines.

4.4.2 Backtracking Baseline

CPO [2] and SDDPG [21] both use a recovery rule once the policy becomes unsafe, which is to train on the safety objective to minimize the cumulative cost until the policy becomes safe again. In this section, we test the hypothesis that CPO and SDDPG are unable to actively avoid constraint violation but their near constraint satisfaction behavior can be explained by the recovery rule. To this end, we introduce a simple baseline, BACKTRACK, which uses the following objective for policy optimization under a trust region (we use the same trust region as in LBPO):

$$\pi = \begin{cases} \max_{\pi \in \mathcal{P}} \mathbb{E}_{s \sim \rho^{\pi_B}} \left[Q^R_{\pi_B}(s, \pi(s)) \right] & \text{if } \pi_B \text{ is safe} \\ \min_{\pi \in \mathcal{P}} \mathbb{E}_{s \sim \rho^{\pi_B}} \left[Q^C_{\pi_B}(s, \pi(s)) \right] & \text{if } \pi_B \text{ is unsafe} \end{cases} \tag{4.15}$$

Thus, if the most recent policy π_B is evaluated to be safe, BACKTRACK exclusively optimizes the reward; however, if the most recent policy π_B is evaluated to be unsafe, BACKTRACK exclusively optimizes the safety constraint. Effectively, BACKTRACK relies only on the recovery behavior that is used in CPO and SDDPG, without incorporating their mechanisms for constrained policy updates. In Tables 4.1 and 4.2, we see that BACKTRACK is competitive to both CPO and SDDPG in terms of both constraint satisfaction and performance (maximizing reward), suggesting that the recovery behavior is itself sufficient to explain their performance. In Appendix 4.5.2, we compare the number of backtracks performed by CPO, SDDPG and BACKTRACK.

4.4.3 Robustness to finite sample sizes

We generally work in the function approximation setting to accommodate high dimensional observations and actions, and this makes it necessary to rely on safety methods that are robust to Q-function errors. To analyze how robust different methods are to such errors, we define a simple reinforcement learning problem: Consider an MDP with two dimensional state space given by $(x, y) \in \mathbb{R}$. The initial state is (0,0). Actions are two dimensional, given by (a_1, a_2) : $a_1, a_2 \in [-0.2, 0.2]$. The horizon is 10 and the transition probability distribution is $(x', y') = (x, y) + (a_1, a_2) + \mathcal{N}(0, 0.1)$. The reward function is $r(x, y) = \sqrt{x^2 + y^2}$. The cost function is equal to the reward function for all states, and the constraint threshold is set to 2. We plot in Figure 4.4 the total constraint violations during 100 epochs of training with varying number of samples used to estimate the cost Q-function. We find that LBPO is more robust to Q-function errors due

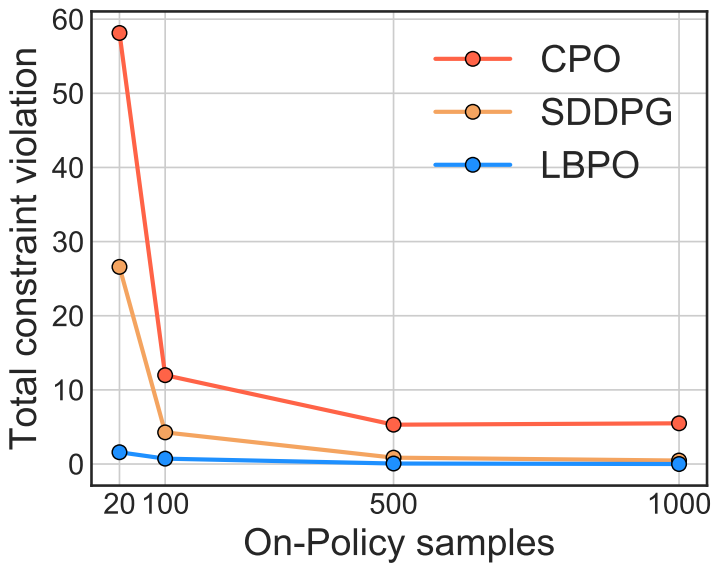


Figure 4.4: An analysis of the robustness of safe RL algorithms CPO, SDDPG, and LBPO to finite sample Q function errors for a simple didactic environment. Constraint violations in CPO and SDDPG increase quickly as the number of on-policy samples used to estimate the Q function decreases. Results are averaged over 5 seeds.

to limited data compared to CPO and SDDPG. In this experiment we use $\beta = 0.005$, similar to the value used for the benchmark experiments.

4.4.4 Tuning conservativeness with the barrier

A strength of LBPO is the ability to tune the barrier to adjust the amount of risk-aversion of the agent. Specifically, β in Equation 4.11 can be tuned; a larger β leads to more conservative policies. In Figure 4.5, we empirically demonstrate the sensitivity of β to the conservativeness of the policy update. For our benchmark results, we do a hyperparameter search for β in the set (0.005, 0.008, 0.01, 0.02) and found that 0.005 works well across most environments.

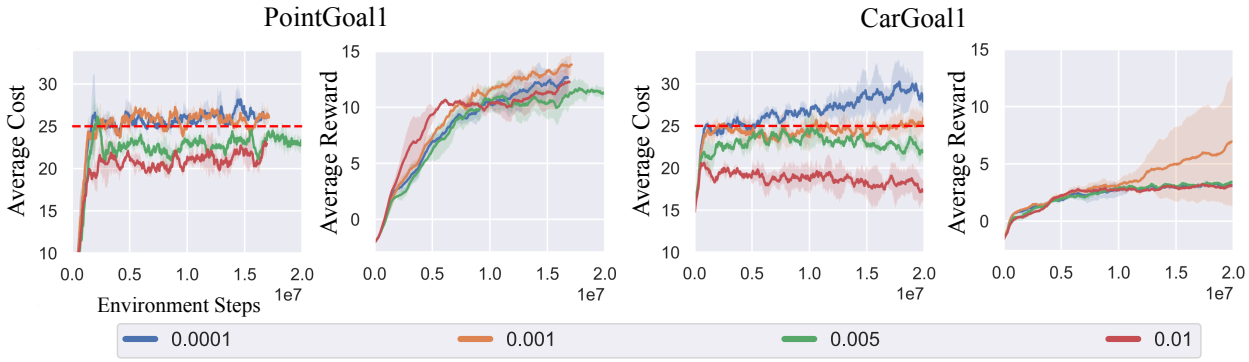


Figure 4.5: Increasing β parameter for the barrier increases the risk aversion of the agent as can be seen in the plots above.

4.5 Appendix

4.5.1 Safe policy update under the Lyapunov constraint

Let the safe initial (baseline) policy be given by π_B and the Lyapunov function be defined as follows:

$$\mathcal{L}_{\pi_B}(s_0, d_0) = \{L : S \to R_{\geq 0} : \mathcal{B}_{\pi_B,c}[L](s) \leq L(s), \forall s \in S; L(s_0) \leq d_0\} \tag{4.16}$$

where c is the immediate cost function. Lyapunov functions depends on the safe baseline policy, an initial state and the cost constraint, and have the property that a one-step Bellman operator produces a value that is less that the value of the function at each state. We also know that the cost value function belongs to the set and hence the set is non-empty. Consider any Lyapunov function $L_{\pi_B} \in \mathcal{L}_{\pi_B}(s_0, d_0)$ and define:

$$\mathcal{I}_{L_{\pi_B}} = \{\pi(.|s) \in \mathcal{P} : \mathcal{B}_{\pi,c}[L_{\pi_B}](s) \leq L_{\pi_B}(s) \forall s\} \tag{4.17}$$

to be set of policies consistent with the Lyapunov function L_{π_B}, called L_{π_B}-induced policies. These are the set of policies ($\pi \in \mathcal{I}_{L_{\pi_B}}$) for which a Bellman Operator $\mathcal{B}_{\pi,c}$ on a state s produces a value that is less than the value of function at that state $L_{\pi_B}(s)$

Note that $\mathcal{B}_{\pi,c}$ is a contraction mapping, so we have

$$V_\pi^c(s) = \lim_{k \to \infty} \mathcal{B}_{\pi,c}^k[L_{\pi_B}](s) \leq L_{\pi_B}(s) \; \forall s \in \mathcal{S} \tag{4.18}$$

From the definition of Lyapunov function, we also have that $L_{\pi_B}(s_0) \leq d_0$. This implies that any policy induced by the Lyapunov function, i.e. policies in the L_{π_B}-induced policy set, are "safe" i.e $V_\pi^c(s_0) = D_\pi(s_0) < d_0$. The method for safe reinforcement learning then searches for

the highest performing policy within the safe policies defined by the set of L_{π_B}-induced policies. The objective here then is to design a Lyapunov function which contains the optimal policy, i.e optimal policy belongs to the set of L_{π_B}-induced policies so that the optimization restricted in this set indeed results in the solution of Eq. 4.2.

In general, the optimal policy π^* does not belong the policies induced by the Lyapunov functions. [20] show that without loss of optimality, the Lyapunov function that contains the optimal policy in its L_{π_B}-induced policy set can be expressed as $L_{\pi_B,\epsilon}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t (c(s_t) + \epsilon(s_t))|\pi_B, s]$, where $\epsilon(s_t) \geq 0$. The function $L_{\pi_B,\epsilon}(s)$ can be thought of as a cost-value function for policy π_B augmented by an additional per-step cost $\epsilon(s_t)$. First, it can be verified that π_B is indeed, in the set of L_ϵ-induced policies:

$$L_{\pi_B,\epsilon}(x) = \mathcal{B}_{\pi_B,c+\epsilon}[L_{\pi_B,\epsilon}](s) \geq \mathcal{B}_{\pi_B,c}[L_{\pi_B,\epsilon}](s) \quad (\epsilon(s_t) > 0 \; \forall s_t). \tag{4.19}$$

It was shown in [20] that finding a state dependent function ϵ such that the the optimal policy is inside the corresponding $L_{\pi_B,\epsilon}$-induced set is generally not possible and requires knowing the optimal policy. As an approximation, they suggest to create the Lyapunov function with the largest auxiliary cost $\hat{\epsilon}$, such that $L_{\pi_B,\hat{\epsilon}}(s) \geq \mathcal{B}_{\pi_B,c}[L_{\pi_B,\hat{\epsilon}}](s)$ and $L_{\pi_B,\hat{\epsilon}}(s_0) \leq d_0$. The first condition is satisfied as shown in Eq. 4.19 when $\hat{\epsilon}(s) \geq 0 \; \forall s$ and the second condition can be satisfied by the following derivation. Bold letters are used to denote vectors and $\boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}}$ is the transition probability matrix from state s to s' under policy π_B. The vectors contain the function value at each state.

$$\boldsymbol{L}_{\boldsymbol{\pi_B},\hat{\boldsymbol{\epsilon}}} = \boldsymbol{d} + \hat{\boldsymbol{\epsilon}} + \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}} \boldsymbol{L}_{\boldsymbol{\pi_B},\hat{\boldsymbol{\epsilon}}} \tag{4.20}$$

$$\boldsymbol{L}_{\boldsymbol{\pi_B},\hat{\boldsymbol{\epsilon}}} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}})^{-1} \boldsymbol{d} + (\boldsymbol{I} - \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}})^{-1} \hat{\boldsymbol{\epsilon}} \tag{4.21}$$

$$\mathbf{1}(\boldsymbol{s})^{\boldsymbol{T}} \boldsymbol{L}_{\pi_B,\hat{\epsilon}} = \mathbf{1}(\boldsymbol{s})^{\boldsymbol{T}} \boldsymbol{D}_{\pi_B} + \mathbf{1}(s)^T (\boldsymbol{I} - \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}})^{-1} \hat{\boldsymbol{\epsilon}} \tag{4.22}$$

where $\mathbf{1}(\boldsymbol{s})^{\boldsymbol{T}}$ is a one-hot vector in which the non-zero unit element is present at s. To ensure that the cumulative cost at the starting state is less than the constraint threshold, using Eq 4.22 we have:

$$L_{\pi_B,\hat{\epsilon}}(s_0) \leq d_0$$
$$D_{\pi_B}(s_0) + \mathbf{1}(s_0)^T (I - \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}})^{-1} \hat{\boldsymbol{\epsilon}} \leq d_0$$

Notice that $\mathbf{1}(s_0)^T (I - \gamma \boldsymbol{P}^{\boldsymbol{\pi_B}}_{\boldsymbol{s,s'}})^{-1} \mathbf{1}(s)$ represents the total discounted visiting probability $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}(s_t = s)|s_0, \pi$ of any state s from the initial state s_0. Restricting $\hat{\epsilon}$ to be a constant function w.r.t state for simplicity, the value of $\hat{\epsilon}$ can be upper-bounded as:

$$\hat{\epsilon}(s) \leq (d_0 - D_{\pi_B}(s_0)) / \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t\right] = (1 - \gamma)(d_0 - D_{\pi_B}(s_0)) \tag{4.23}$$

In summary, a Lyapunov function is obtained such that optimizing policies in the $L_{\pi_B,\hat{\epsilon}}$-induced set of policies, safety in ensured. For any policy π to lie in the $L_{\pi_B,\hat{\epsilon}}$-induced set the following condition needs to hold $\forall\ s \in \mathcal{S}$:

$$L_{\pi_B,\epsilon}(s) \geq T_{\pi,d}[L_{\pi_B,\epsilon}(s)]$$
$$d(s) + \hat{\epsilon}(s) + \gamma \sum_a \pi_B(a|s)(\sum_{s'} P(s'|s,a)L_{\pi_B,\epsilon}(s')) \geq d(s) + \gamma \sum_a \pi(a|s)(\sum_{s'} P(s'|s,a)L_{\pi_B,\epsilon}(s'))$$

We can simplify further to get:

$$\hat{\epsilon}(s) \geq (\sum_a (\pi(a|s) - \pi_B(a|s)) \left[\gamma \sum_{s'} P(s'|s,a)L_{\pi_B,\epsilon}(s')\right]$$
$$\hat{\epsilon}(s) \geq (\sum_a (\pi(a|s) - \pi_B(a|s)) \left[\gamma \sum_{s'} P(s'|s,a)L_{\pi_B,\epsilon}(s') + d(s) + \hat{\epsilon}(s)\right]$$
$$\hat{\epsilon}(s) \geq \left[\sum_a (\pi(a|s) - \pi_B(a|s))Q_{L_{\pi_B,\epsilon}}(s,a)\right]$$

where

$$Q_{L_{\pi_B,\epsilon}}(s,a) = d(s) + \hat{\epsilon}(s) + \gamma \sum_{s'} P(s'|s,a)L_{\hat{\epsilon}}^{\pi_B,\epsilon}(s') \tag{4.24}$$

This can be extended to continuous action spaces to get the following objective:

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ s.t \int_{a \in \mathcal{A}} (\pi(a|s) - \pi_B(a|s))Q_{L_{\pi_B,\hat{\epsilon}}}(s,a)da \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{4.25}$$

Using the Lyapunov function, the trajectory-based constraints of CMDP are converted to a per-state constraint (Eq.4.25), which are often much easier to deal with.

In the case of deterministic policy, the policy update becomes:

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ \text{s.t}\ Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi(s)) - Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi_B(s)) \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{4.26}$$

An intuitive way to understand the constraint in deterministic policies is to see that at every timestep we are willing to tolerate an additional constant cost of ϵ compared to the baseline safe policy. At the start state, the maximum increase in expected cost will be $\sum_{t=0}^{\infty} \gamma^t \epsilon = \frac{\epsilon}{1-\gamma}$. We want that the new expected cost by less than the threshold, i.e $D_\pi(s_0) + \frac{\epsilon}{1-\gamma} \leq d_0$ which gives us the Lyapunov constraint equation.

From Lyapunov Functions to cost Q Functions

Using the definition of $Q_{L_{\pi_B,\hat{\epsilon}}}(s,a)$ from Eq. 4.5 and when $\hat{\epsilon}(s)$ is a constant function (denote by $\hat{\epsilon}$), we can replace $Q_{L_{\pi_B,\hat{\epsilon}}}$ by $Q^C_{\pi_B}$,

$$\begin{aligned}
Q_{L_{\pi_B,\hat{\epsilon}}}(s,a) &= c(s) + \hat{\epsilon} + \gamma \sum_{s'} P(s'|s,a) L_{\pi_B,\hat{\epsilon}}(s') \\
&= c(s) + \hat{\epsilon} + \left[\gamma \sum_{s'} P(s'|s,a)[c(s') + \hat{\epsilon} + \sum_{s''} P^{\pi_B}_{(s''|s')}(L_{\pi_B,\hat{\epsilon}}(s''))] \right] \\
&= \sum_{t=0}^{\infty} \gamma^t \hat{\epsilon} + \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t) | \pi_B, a_0 = a, s_0 = s \right] \\
&= \sum_{t=0}^{\infty} \gamma^t \hat{\epsilon} + Q^C_{\pi_B}(s,a)
\end{aligned}$$

which is the cost Q function, since the Lyapunov function $Q_{L\pi_B,\hat{\epsilon}}(s,a)$ and the cost-Q function $Q^C_{\pi_B}(s,a)$ only differ by a constant ($\sum_{t=0}^{\infty} \gamma^t \hat{\epsilon}$).

4.5.2 Additional Results

Benchmarks on OpenAI safety gym

In this section, we present the training curves for all the OpenAI safety gym environments with Point and Car robot. Figure 4.5.2 shows the Average Cost and Average return for these environments. The dotted red line indicates the constraint threshold which is kept to be 25 across all environments. We observe that LBPO rarely violates constraint during training. Table 4.3 shows the raw cumulative returns of the converged policy for different methods on the safety environments. We average all results over 3 random seeds.

We observe that in tasks with Doggo robot, none of the methods are able to obtain good performing policy. We attribute this to be the difficulty of Doggo environments, involving an inherent tradeoff of reward with cost. In the environment PointGoal2, we are unable to obtain safe policies even when training an RL agent solely on the cost objective. LBPO still outperforms baselines for constraint satisfaction on this environment.

Method	PPO	PPO-lagrangian	CPO	SDDPG	BACKTRACK	LBPO
PointGoal1	22.99	**19.00**	10.26	10.45	15.54	11.06
PointGoal2	23.04	**4.60**	-0.37	-0.08	1.04	0.61
PointPush1	4.61	3.04	1.73	2.71	2.43	**3.15**
PointPush2	2.15	**1.04**	0.46	0.48	0.89	0.77
CarGoal1	34.62	15.55	2.76	3.38	**17.22**	13.03
CarGoal2	26.70	1.78	4.60	**5.74**	4.35	5.62
CarPush1	3.89	2.72	-3.13	1.69	**3.38**	1.89
CarPush2	2.03	0.72	0.82	0.75	0.81	**0.94**
DoggoGoal1	38.76	-0.65	0.14	0.10	0.15	**0.28**
DoggoGoal2	18.38	**0.31**	0.04	0.06	0.06	0.06
DoggoPush1	0.82	**0.07**	0.01	0.00	0.06	0.01
DoggoPush2	1.10	**0.08**	-0.00	-0.00	0.07	-0.01

Table 4.3: Cumulative unnormalized return of the converged policy for each safety algorithm. LBPO tradeoffs return for better constraint satisfaction. Bold numbers show the best performance obtained by a safety algorithm (thus excluding PPO).

Backtracks in CPO and SDDPG

Figure 4.7 shows the cumulative number of backtracks performed by each method CPO, SDDPG, BACKTRACK during the first 400 policy update steps. We see the CPO and SDDPG performs a high number of backtracks, often comparable to the method BACKTRACK which relies explicitly on backtracking for safety.

4.5.3 Implementation Details

In LBPO, Q-functions (both reward and cost) have network architecture comprising of two hidden layers of 64-hidden size each. The policy is also a multilayer neural network comprising of three hidden layers of 256 units each. LBPO policies are deterministic and have a fixed exploration noise in the action space given by $\mathcal{N}(0, 0.05)$. Our trust region update for the policy takes into account the exploration noise which makes our behavior deployment policy stochastic. We use $N = 30$ trajectories each of 1000 horizon length for generating our on-policy samples. These samples are used for estimation of $\hat{\epsilon}$ and evaluating the Q functions. We update the policy under a trust region followed by a line search with exponential decay which ensures that the resulting update is indeed

satisfying the KL constraint as well the safety Lyapunov constraint. We do a hyperparameter search for β in the set [0.005, 0.008, 0.01, 0.02] to find the best tradeoff between cost and reward and observe that a value of 0.005 works well across most environments. PointGoal1, PointPush1, PointPush2, CarGoal1, CarPush1, CarPush2, DoggoGoal1, DoggoPush1, DoggoPush2 use beta value of 0.005. CarGoal2 and DoggoGoal2 uses value of 0.008 and Pointgoal2 uses beta value of 0.01. We ignore the barrier loss if β is sufficiently low. We call this parameter β-thres and it is set to 0.05 across all environments.

Algorithm 3: LBPO

Initialize parameterized actor π_ϕ with a safe initial policy, reward Q-function Q^R_θ and a cost Q function Q^C_θ

for $i \leftarrow 1$ **to** *Iter* **do**

Step 1: Collect N trajectories $\{\tau\}_{j=1}^N$ using the safe policy $\pi_{\phi,i-1}$ from previous iteration $i-1$.

Step 2: Using the on-policy trajectories, evaluate the reward Q-function and the cost-function, by minimizing the respective bellman residual of the TD-(λ) estimate.

Step 3: Update the policy parameters by minimizing the objective in Eq 4.12.

$$\min_\phi \mathbb{E}_{s\sim\mathcal{R}}\Big[-Q^R_{\pi_{\phi,i-1}}(s,\pi_\phi(s)) + \psi(Q^C_{\pi_{\phi,i-1}}(s,\pi_\phi(s)))\Big]$$
$$\text{s.t } \mathrm{D}\text{KL}(\pi_\phi + \mathcal{N}(0,\delta)\|\pi_{\phi,i-1} + \mathcal{N}(0,\delta)) < \mu$$

Step 4: Set $\pi_{\phi,i}$ to be the safe policy resulting from the update in Step 3 π_ϕ.

end

We obtain safe initial policies for benchmarking by pretraining the policy using standard RL methods to minimize the cumulative cost. Although this strategy is usually not suitable for deployment in real-world as the pretraining might itself violate safety constraints, we can use simple hand-designed safe policy for initializing the method in real-world experiments.

To ensure fair comparison across methods, we use the same safe initial policy for each of the safety methods. Note that, our results for CPO [2] significantly differ from the benchmarks shown in [66] due to the fact that we initialize CPO from safe policy contrary to their approach. We also keep the same policy architecture across methods although CPO, PPO and PPO-lagrangian uses policies with learned variance so as to replicate the original behavior of these methods.

[1] β is set to 0.005 for most environments. Appendix 4.5.3 describes specific value of β for each environment.

Table 4.4: LBPO Hyperparameters

Hyperparamater	**Value**
N	30
β	0.005^1
β-thres	0.05
Policy learning rate	3×10^{-4}
Q-function learning rate	1×10^{-3}
Trust region (μ)	0.012
λ	0.97
δ	0.05
Horizon	1000

We implement our version of SDDPG which uses the α-projection technique as shown in [21]. A brief discussion of practical issue faced in the implementation is present in Section 4.4. We use behavior cloning to distill the policy with the projection layer into a parameterized multilayer perceptron policy. We run 100 iterations of behavior cloning with learning rate of 0.001. We implement a line search with exponential decay in parameter space to ensure that the resulting update do not violate the Lyapunov constraints to incorporate additional safety. We use similar policy architecture as LBPO for α-SDDPG.

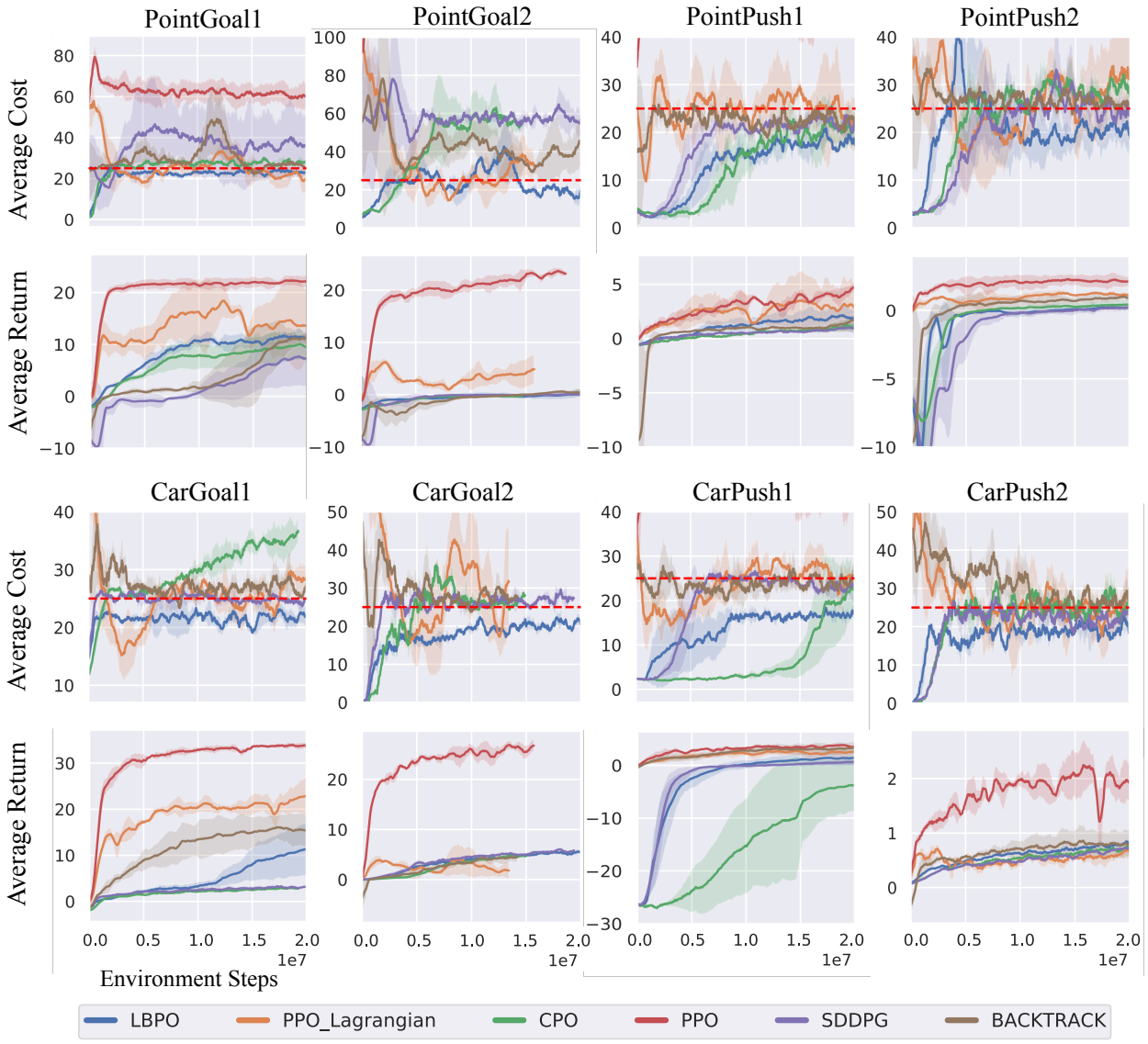


Figure 4.6: Training curved for LBPO in comparison to baselines: PPO, PPO-lagrangian, CPO, SDDPG. We also compare against our simple baseline BACKTRACK here. For each environment, the top row shows the Average undiscounted cumulative cost during training, and bottom row shows the Average undiscounted return. PPO often has large constraint violations and is clipped from some plots, when its constraint violations are high. Red dashed line in Average Cost plots shows the constraint limit which is 25 in all environments.

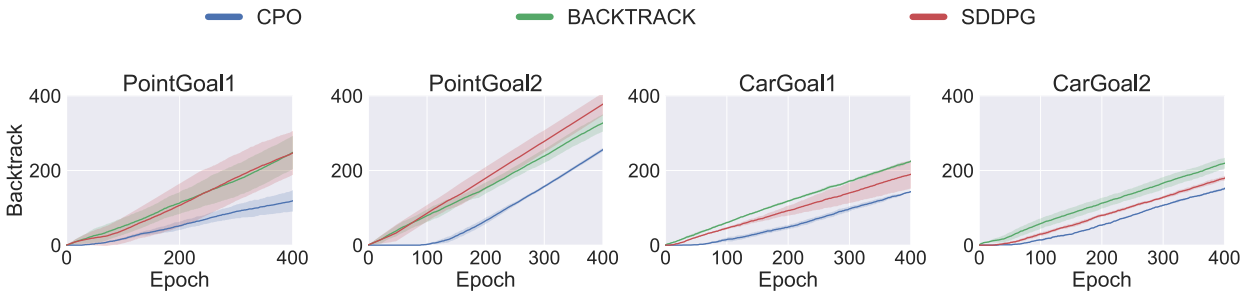


Figure 4.7: We compare the cumulative number of backtracking steps taken by CPO and SDDPG to BACKTRACK method for the first 400 epochs/policy updates.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we present methods that are amenable to safety in different dimensions of Reinforcement Learning.

1. In Chapter 2 [74], we present a novel method, LOOP, that combines model-free and model-based reinforcement learning. It allows model-based online planning to reason about long-horizon cumulative returns. From another perspective, it improves upon the model-free algorithm by using a more efficient behavior policy. We highlight the issues present in applying a terminal Q-function to the online planning methods and present the actor-guided solution. From the experiments, we demonstrate that LOOP improves the performance and sample-efficiency over the underlying model-based and model-free method.
2. In Chapter 3 [58], we propose f-IRL, a practical IRL algorithm that distills an expert's state distribution into a stationary reward function. Our f-IRL algorithm can learn from either expert samples (as in traditional IRL), or a specified expert density (as in SMM [47]), which opens the door to supervising IRL with different types of data. These types of supervision can assist agents in solving tasks faster, encode preferences for how tasks are performed, and indicate which states are unsafe and should be avoided. Our experiments demonstrate that f-IRL is more sample efficient in the number of expert trajectories and environment timesteps as demonstrated on MuJoCo benchmarks.
3. In Chapter 4 [75], we present a new method, LBPO, that formulates a safe policy update as an unconstrained policy optimization augmented by a barrier function derived from Lyapunov-based constraints. LBPO allows the agent to control the risk aversion of the RL agent and

is empirically observed to be more robust to Q-function errors. We also present a simple baseline BACKTRACK to provide insight into previous methods' reliance on backtracking recovery behavior to achieve near constraint satisfaction. LBPO achieves fewer constraint violations, in most cases close to zero, on a number of challenging continuous control tasks and outperforms state-of-the-art safe RL baselines.

5.2 Future Directions

Safety is a necessary component precluding the deployment of robots in the real world. First, we require a definition of safety that actually allows the deployment of the robots without any fear of damage to the robot or to the outside environment. Second, we require methods that follow these definitions and have provable guarantees. A promising direction is to explore if we can learn notions of safety by observing human agents. In this thesis, we have approached safety using the definition of expected cumulative safety but we look forward to exploring other notions that are more aligned to human intent.

Current safety methods have provable guarantees under a narrow function class definition such as linear and do not extend to expressive classes like neural networks. Neural network policies have been very successful in solving complex tasks in a complex observation space and we look forward to developing guarantees for such settings. Calibrated uncertainty estimation and constrained optimization are core to this problem in order to get meaningful guarantees. We explored one way of optimization for safety that works well empirically in this thesis and we aim to develop further analysis to understand its performance theoretically.

Human demonstrations are a rich source of information to bootstrap from as prior knowledge. We explored a new way to encode preferences in this thesis - via state marginals. This method opens up new ways to encode preferences and serves as a theoretical building block for safe Imitation Learning methods, which we intend to explore in the future.

Finally, we require RL algorithms to be safe and sample efficient. Current safety methods have poor sample efficiency but safety should not necessarily involve a huge trade-off with efficiency. Model-based approaches are a promising solution as they are sample efficient and amenable to safety. Solving these challenges is a major step that would enable RL to be deployed in real-world applications like healthcare, transport, education, household robots among others.

Bibliography

[1] 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, 2004.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017.

[3] 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.

[4] Syed Mumtaz Ali and Samuel D Silvey. A general class of coefficients of divergence of one distribution from another. *Journal of the Royal Statistical Society: Series B (Methodological)*, 28(1):131–142, 1966.

[5] Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3): 387–417, 1998.

[6] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[7] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[8] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[10] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*,

volume 97, pages 12–20. Citeseer, 1997.

[11] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[12] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[13] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[14] Vivek Borkar and Rahul Jain. Risk-constrained markov decision processes. In *49th IEEE Conference on Decision and Control (CDC)*, pages 2664–2669. IEEE, 2010.

[15] Kianté Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[17] 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.

[18] Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for cvar optimization in mdps. In *Advances in neural information processing systems*, pages 3509–3517, 2014.

[19] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.

[20] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101, 2018.

[21] 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.

[22] 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, 2018.

[23] 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.

[24] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[25] 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*, 2018.

[26] 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*, 2016.

[27] 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, 2016.

[28] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[29] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[30] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[31] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

[32] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. *arXiv preprint arXiv:1911.02256*, 2019.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[35] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016.

[36] Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W Battaglia. Combining q-learning and search with amortized value estimates. *arXiv preprint arXiv:1912.02807*, 2019.

[37] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[38] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[39] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[40] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2019.

[41] 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.

[42] Liyiming Ke, Matt Barnes, Wen Sun, Gilwoo Lee, Sanjiban Choudhury, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. *arXiv preprint arXiv:1905.12888*, 2019.

[43] Ramtin Keramati, Christoph Dann, Alex Tamkin, and Emma Brunskill. Being optimistic to be conservative: Quickly learning a cvar policy. *arXiv preprint arXiv:1911.01546*, 2019.

[44] LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

[45] Victoria Krakovna, Laurent Orseau, Ramana Kumar, Miljan Martic, and Shane Legg. Penalizing side effects using stepwise relative reachability. *arXiv preprint arXiv:1806.01186*, 2018.

[46] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[47] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

[48] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[49] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017.

[50] 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, 2015.

[51] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019.

[52] Minghuan Liu, Tairan He, Minkai Xu, and Weinan Zhang. Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*, 2020.

[53] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[54] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, pages 429–443, 1997.

[55] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

[56] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[57] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670, 2000.

[58] Tianwei Ni, Harshit Sikchi, Yufei Wang, Tejus Gupta, Lisa Lee, and Benjamin Eysenbach. f-irl: Inverse reinforcement learning via state marginal matching. *arXiv preprint arXiv:2011.04709*, 2020.

[59] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neu-

ral samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016.

[60] Laurent Orseau and MS Armstrong. Safely interruptible agents. 2016.

[61] Takayuki Osa, Naohiko Sugita, and Mamoru Mitsuishi. Online trajectory planning and force control for automation of surgical tasks. *IEEE Transactions on Automation Science and Engineering*, 15(2):675–691, 2017.

[62] Theodore J Perkins and Andrew G Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3(Dec):803–832, 2002.

[63] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.

[64] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[65] Ahmed H Qureshi, Byron Boots, and Michael C Yip. Adversarial imitation via variational inverse reinforcement learning. *arXiv preprint arXiv:1809.06404*, 2018.

[66] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019.

[67] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

[68] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[69] 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.

[70] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[71] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[72] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust

region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[73] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[74] Harshit Sikchi, Wenxuan Zhou, and David Held. Learning off-policy with online planning, 2020.

[75] Harshit Sikchi, Wenxuan Zhou, and David Held. Lyapunov barrier policy optimization, 2021. URL https://openreview.net/forum?id=qUs18ed9oe.

[76] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[77] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484–489, 2016.

[78] Shashank Singh and Barnabás Póczos. Analysis of k-nearest neighbor distances with application to entropy estimation. *arXiv preprint arXiv:1603.08578*, 2016.

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

[80] Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling. *arXiv preprint arXiv:1404.3862*, 2014.

[81] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. Worst cases policy gradients. *arXiv preprint arXiv:1911.03618*, 2019.

[82] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. *IEEE Robotics and Automation Letters*, 5(2):3612–3619, 2020.

[83] 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, 2012.

[84] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *Advances in Neural Information Processing Systems*, pages 2891–2901,

2019.

[85] Greg Ver Steeg. Non-parametric entropy estimation toolbox (npeet). 2000.

[86] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[87] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[88] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[89] Akifumi Wachi and Yanan Sui. Safe reinforcement learning in constrained markov decision processes. *arXiv preprint arXiv:2008.06626*, 2020.

[90] Ruohan Wang, Carlo Ciliberto, Pierluigi Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. *arXiv preprint arXiv:1905.06750*, 2019.

[91] Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[92] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[93] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-based constrained policy optimization. In *ICLR*, 2020.

[94] Jesse Zhang, Brian Cheung, Chelsea Finn, Sergey Levine, and Dinesh Jayaraman. Cautious adaptation for reinforcement learning in safety-critical settings. *arXiv preprint arXiv:2008.06622*, 2020.

[95] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

[96] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

[97] 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.