LYAPUNOV BARRIER POLICY OPTIMIZATION

Harshit Sikchi
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{hsikchi}@cs.cmu.edu

Wenxuan Zhou, David Held
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{wenxuanz,dheld}@cs.cmu.edu

ABSTRACT

Deploying Reinforcement Learning (RL) agents in the real-world require that the agents satisfy safety constraints. Current RL agents explore the environment without considering these constraints, which can lead to damage to the hardware or even other agents in the environment. We propose a new method, 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 conservativeness of the agent 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. Further, our analysis reveals that baselines like CPO and SDDPG rely mostly on backtracking to ensure safety rather than safe projection, which provides insight into why previous methods might not have effectively limit the number of constraint violations.

1 INTRODUCTION

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 (Silver et al., 2016; Vinyals et al., 2019; Akkaya et al., 2019). 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) (Altman, 1999), 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 (Turchetta et al., 2019; Wachi & Sui, 2020). One popular approach to solve Constrained MDPs in large state and action spaces is to use the Lagrangian method (Altman, 1998; Ray et al., 2019). 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 (Altman, 1998). 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, Chow et al. (2018; 2019) 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.

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) (Altman, 1999), 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:

$$\mathcal{B}_{\pi,h}[V][s] = \sum_{a} \pi(a|s)[h(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V(s')] \tag{1}$$

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:

$$\max_{\pi \in \mathcal{P}}[J_\pi(s_0)] \text{ s.t } D_\pi(s_0) \leq d_0 \tag{2}$$

2.1 Safe Reinforcement Learning using Lyapunov Functions

We will build upon the Lyapunov framework introduced by Chow et al. (2018), 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:

$$\mathcal{L}_{\pi_B}(s_0, d_0) = \{L : S \to R_{\geq 0} : \mathcal{B}_{\pi_B,c}[L](s) \leq L(s), \forall s \in S; L(s_0) \leq d_0\} \tag{3}$$

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:

$$\mathcal{I}_{L_{\pi_B}} = \{\pi \in \mathcal{P} : \mathcal{B}_{\pi,c}[L_{\pi_B}](s) \leq L_{\pi_B}(s), \forall s \in \mathcal{S}\} \tag{4}$$

It can be shown that any policy π in the L_{π_B}-induced policy set is "safe", i.e $D_\pi(s_0) \leq d_0$ (Chow et al., 2018).

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^*. Chow et al. (2018) 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:

$$Q_{L_{\pi_B,\epsilon}}(s,a) = c(s) + \epsilon(s) + \gamma \sum_{s'} P(s'|s,a) L_{\pi_B,\epsilon}(s') \tag{5}$$

It was shown in Chow et al. (2018) 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:

$$\hat{\epsilon}(s) = (1-\gamma)(d_0 - D_{\pi_B}(s_0)) \tag{6}$$

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:

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ s.t \int_{a \in \mathcal{A}} (\pi(a|s) - \pi_B(a|s)) Q_{L_{\pi_B,\hat{\epsilon}}}(s,a) da \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{7}$$

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

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ \text{s.t}\ Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi(s)) - Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi_B(s)) \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{8}$$

We build upon this objective in our work. We include the proof of the Lyapunov approach for completeness in Appendix A.1, and we advise the reader to see previous work (Chow et al., 2018) for a more detailed derivation. Using the Lyapunov function, the trajectory-based constraint of the CMDP is converted to a per-state constraint (Eq. 7), which is often much easier to deal with.

3 Method

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.

3.1.1 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) (Sutton, 1988) 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. 6.

3.1.2 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. 8 :

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0)\ \text{s.t}\ Q^i_{L_{\pi_B,\hat{\epsilon}}}(s,\pi(s)) - Q^i_{L_{\pi_B,\hat{\epsilon}}}(s,\pi_B(s)) \leq \hat{\epsilon}_i(s)\ \forall i \in [1, 2, ...m], \forall s \in \mathcal{S} \tag{9}$$

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 A.1.1). 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.

$$I(Q^{C_i}_{\pi_B}(s, \pi_\theta(s))) = \begin{cases} 0 & Q^{C_i}_{\pi_B}(s, \pi_\theta(s)) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) \le \hat{\epsilon}_i(s) \\ \infty & Q^{C_i}_{\pi_B}(s, \pi_\theta(s)) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) > \hat{\epsilon}_i(s) \end{cases} \tag{10}$$

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

$$\psi(Q^{C_i}_{\pi_B}(s, \boldsymbol{\pi_\theta(s)})) = -\beta \log \left(\hat{\epsilon}(s) - \left(Q^{C_i}_{\pi_B}(s, \boldsymbol{\pi_\theta(s)}) - Q^{C_i}_{\pi_B}(s, \pi_B(s)) \right) \right) \tag{11}$$

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 $\hat{\epsilon}(s) > 0 \, \forall s$.

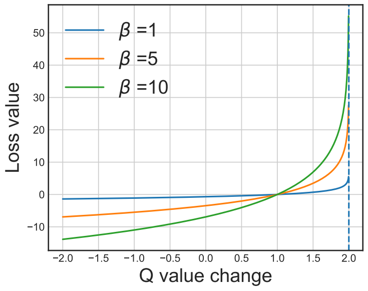

Figure 1: As the difference between Q values for action a and the baseline action reaches ϵ (in this case $\epsilon = 2$), the loss increases to ∞.

Figure 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.

We use the Deterministic Policy Gradient Theorem (Silver et al., 2014) 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 Schulman et al. (2015), 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.

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

$$\theta_{k+1} = \text{argmin}_\theta \; \mathbb{E}_{s\sim\rho^{\pi_B}} \left[-Q^R_{\pi_B}(s, \pi_\theta(s)) + \sum_{i=1}^{m} \psi(Q^{C_i}_{\pi_B}(s, \pi_\theta(s))) \right] \tag{12}$$

$$\text{subject to } \mathbb{E}_{s\sim\rho^{\pi_B}} \left[D_{\text{KL}}(\pi_\theta[.|s] + \mathcal{N}(0,\delta) \parallel \pi_B[.|s] + \mathcal{N}(0,\delta)) \right] < \mu \tag{13}$$

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 12 be denoted by g and the Hessian of the KL divergence by H, our objective becomes:

$$\theta_{k+1} = \text{argmin}_\theta \; g^T(\theta - \theta_k), \quad \text{subject to } \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \le \mu \tag{14}$$

We solve this constrained optimization using the Fisher vector product with Conjugate gradient method similar to Schulman et al. (2015).

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

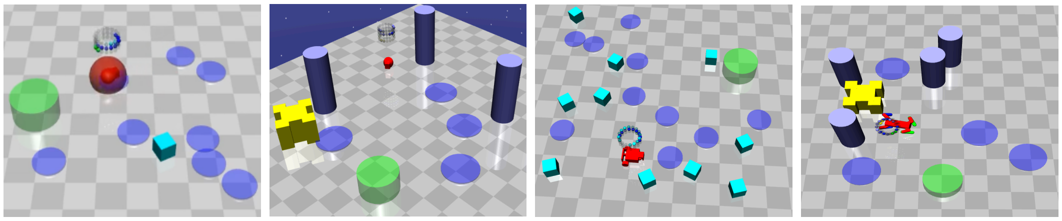

Figure 2: OpenAI Safety Environments: PointGoal1, PointPush2, CarGoal2, DoggoPush2

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 (Schulman et al., 2017) 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 (Altman, 1998) 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 (Chow et al., 2018; 2019) 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 (Chow et al., 2019). Since the original implementation for the method is unavailable, we re-implemented the method to the best of our abilities. CPO (Achiam et al., 2017) 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 4.2.

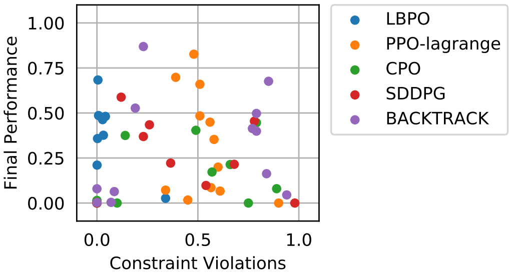

Figure 3: Each point corresponds to a particular safety method applied to a certain safety environment. The x-axis shows the fraction of constraint violations encountered by the behavior policy during training and y-axis shows the policy performance normalized by the corresponding environment's PPO return.

Tasks. We evaluate these methods using the OpenAI Safety Gym (Ray et al., 2019), which consists of 12 continuous control MuJoCo tasks (Todorov et al., 2012). 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. 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 2.

4.1 Safe Reinforcement Learning Benchmarks

We summarize the comparison of LBPO to all of the baselines (PPO, PPO-lagrangian, BACKTRACK, SDDPG and CPO) on the OpenAI safety benchmarks in Figure 3 and Tables 1 and 2. Additional training plots for policy return and policy cost can be found in Appendix A.2.1.

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 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.

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 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).

Constraint Satisfaction. Table 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 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 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 (Chow et al., 2018), 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 4.3 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 performance. We observe in Table 2 that LBPO achieves performance competitive to the baselines.

4.2 Backtracking Baseline

CPO (Achiam et al., 2017) and SDDPG (Chow et al., 2019) 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):

$$\pi = \begin{cases} \max_{\pi \in \mathcal{P}} \mathbb{E}_{s \sim \rho^{\pi_B}} \left[Q^R_{\pi_B}(s, \pi(s)) \right] & \text{if } \pi_B \text{ is safe} \\ \min_{\pi \in \mathcal{P}} \mathbb{E}_{s \sim \rho^{\pi_B}} \left[Q^C_{\pi_B}(s, \pi(s)) \right] & \text{if } \pi_B \text{ is unsafe} \end{cases} \tag{15}$$

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 1 and 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 A.2.2, we compare the number of backtracks performed by CPO, SDDPG and BACKTRACK.

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 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 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.

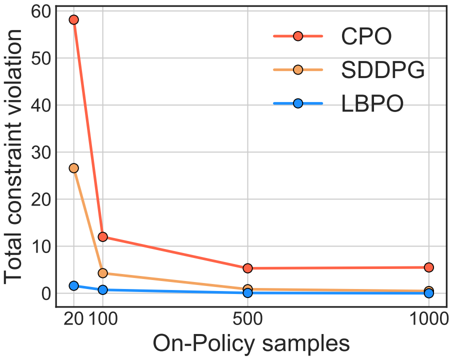

Figure 4: An analysis of the robustness of safe RL algorithms CPO, SDDPG, and LBPO to finite sample Q function errors for a simple didactic environment. Constraint violations in CPO and SDDPG increase quickly as the number of on-policy samples used to estimate the Q function decreases. Results are averaged over 5 seeds.

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 11 can be tuned; a larger β leads to more conservative policies. In Figure 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.

5 Related Work

Constrained Markov Decision Process CMDP's (Altman, 1998) have been a popular framework for incorporating safety in the form of constraints. In CMDP's the agent tries to maximize expected

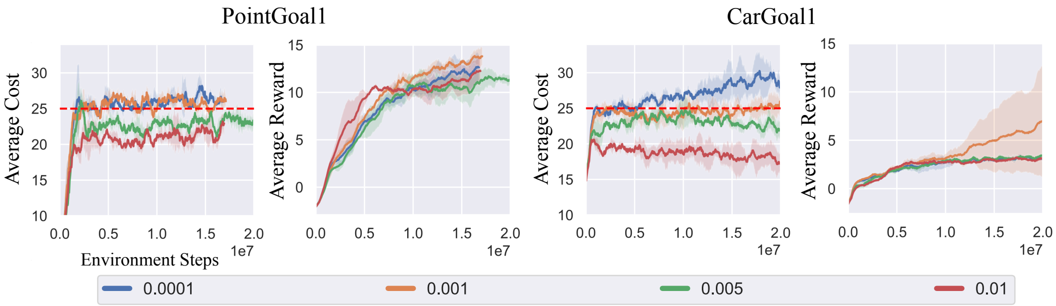

Figure 5: Increasing β parameter for the barrier increases the risk aversion of the agent as can be seen in the plots above.

returns by satisfying constraints on expectation of costs. Altman (1999) 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 (Altman, 1999; Geibel & Wysotzki, 2005; Chow et al., 2017). 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 (Achiam et al., 2017) 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 (Schulman et al., 2015). Chow et al. (2018; 2019) presents another class of method that formulates safe policy update under a Lyapunov constraint. Perkins & Barto (2002) explored the relevance of Lyapunov functions in control and (Berkenkamp et al., 2017) 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 (Pham et al., 2018; Dalal et al., 2018) 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. Dalal et al. (2018) satisfies state-based costs rather than trajectory-based costs. Thananjeyan et al. (2020) utilizes demonstrations to ensure safety in the model based framework, and Zhang et al. (2020) 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 (Rockafellar et al., 2000) 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 (Tang et al., 2019; Keramati et al., 2019; Tamar et al., 2014; Kalashnikov et al., 2018; Borkar & Jain, 2010; Chow & Ghavamzadeh, 2014).

6 Conclusion

In this work, 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.

REFERENCES

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

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

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.

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

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

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

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

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.

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*, pp. 8092–8101, 2018.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

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.

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

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.

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.

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

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)*, pp. 6236–6243. IEEE, 2018.

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

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

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

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

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.

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

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

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

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.

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*, pp. 5026–5033. IEEE, 2012.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *Advances in Neural Information Processing Systems*, pp. 2891–2901, 2019.

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.

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

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.

A Appendix

A.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:

$$\mathcal{L}_{\pi_B}(s_0, d_0) = \{L : S \to R_{\geq 0} : \mathcal{B}_{\pi_B,c}[L](s) \leq L(s), \forall s \in S; L(s_0) \leq d_0\} \tag{16}$$

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:

$$\mathcal{I}_{L_{\pi_B}} = \{\pi(.|s) \in \mathcal{P} : \mathcal{B}_{\pi,c}[L_{\pi_B}](s) \leq L_{\pi_B}(s) \forall s\} \tag{17}$$

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

$$V_\pi^c(s) = \lim_{k\to\infty} \mathcal{B}^k_{\pi,c}[L_{\pi_B}](s) \leq L_{\pi_B}(s) \; \forall s \in \mathcal{S} \tag{18}$$

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. 2.

In general, the optimal policy π^* does not belong the policies induced by the Lyapunov functions. Chow et al. (2018) 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:

$$L_{\pi_B,\epsilon}(x) = \mathcal{B}_{\pi_B,c+\epsilon}[L_{\pi_B,\epsilon}](s) \geq \mathcal{B}_{\pi_B,c}[L_{\pi_B,\epsilon}](s) \quad (\epsilon(s_t) > 0 \; \forall s_t). \tag{19}$$

It was shown in Chow et al. (2018) 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. 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 $\boldsymbol{P^{\pi_B}_{s,s'}}$ is the transition probability matrix from state s to s' under policy π_B. The vectors contain the function value at each state.

$$\boldsymbol{L_{\pi_B,\hat{\epsilon}}} = \boldsymbol{d} + \boldsymbol{\hat{\epsilon}} + \gamma \boldsymbol{P^{\pi_B}_{s,s'}} \boldsymbol{L_{\pi_B,\hat{\epsilon}}} \tag{20}$$

$$\boldsymbol{L_{\pi_B,\hat{\epsilon}}} = (\boldsymbol{I} - \gamma \boldsymbol{P^{\pi_B}_{s,s'}})^{-1} \boldsymbol{d} + (\boldsymbol{I} - \gamma \boldsymbol{P^{\pi_B}_{s,s'}})^{-1} \boldsymbol{\hat{\epsilon}} \tag{21}$$

$$\boldsymbol{1(s)^T L}_{\pi_B,\hat{\epsilon}} = \boldsymbol{1(s)^T D}_{\pi_B} + \boldsymbol{1}(s)^T (\boldsymbol{I} - \gamma \boldsymbol{P^{\pi_B}_{s,s'}})^{-1} \boldsymbol{\hat{\epsilon}} \tag{22}$$

where $\boldsymbol{1(s)^T}$ 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 22 we have:

$$L_{\pi_B,\hat{\epsilon}}(s_0) \leq d_0$$

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

Notice that $\mathbf{1}(s_0)^T(I - \gamma \boldsymbol{P}_{\boldsymbol{s},\boldsymbol{s'}}^{\boldsymbol{\pi_B}})^{-1}\mathbf{1}(s)$ represents the total discounted visiting probability $\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t \mathbf{1}(s_t = s)|s_0, \pi_B]$ 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:

$$\hat{\epsilon}(s) \leq (d_0 - D_{\pi_B}(s_0))/\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\right] = (1-\gamma)(d_0 - D_{\pi_B}(s_0)) \tag{23}$$

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

$$Q_{L_{\pi_B,\epsilon}}(s,a) = d(s) + \hat{\epsilon}(s) + \gamma \sum_{s'} P(s'|s,a)L_{\hat{\epsilon}}^{\pi_B,\epsilon}(s') \tag{24}$$

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

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ s.t \int_{a \in \mathcal{A}} (\pi(a|s) - \pi_B(a|s))Q_{L_{\pi_B,\epsilon}}(s,a)da \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{25}$$

Using the Lyapunov function, the trajectory-based constraints of CMDP are converted to a per-state constraint (Eq.25), which are often much easier to deal with.

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

$$\pi_+(.|s) = \max_{\pi \in \mathcal{P}} J_\pi(s_0),\ \text{s.t}\ Q_{L_{\pi_B,\epsilon}}(s,\pi(s)) - Q_{L_{\pi_B,\hat{\epsilon}}}(s,\pi_B(s)) \leq \hat{\epsilon}(s)\ \forall s \in \mathcal{S} \tag{26}$$

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.

A.1.1 From Lyapunov Functions to cost Q Functions

Using the definition of $Q_{L_{\pi_B,\hat{\epsilon}}}(s,a)$ from Eq. 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 ($\sum_{t=0}^{\infty} \gamma^t \hat{\epsilon}$).

A.2 Additional Results

A.2.1 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 A.2.1 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 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 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).

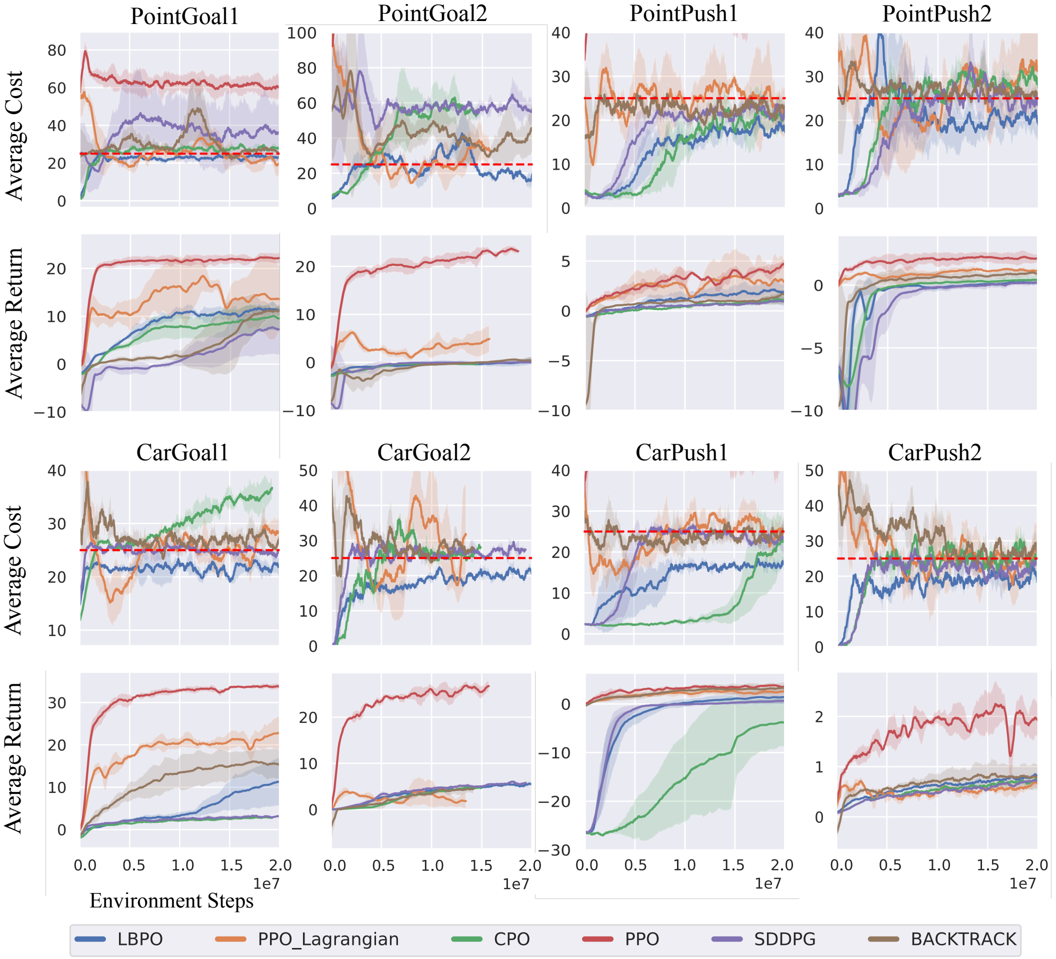

Figure 6: Training curved for LBPO in comparison to baselines: PPO, PPO-lagrangian, CPO, SDDPG. We also compare against our simple baseline BACKTRACK here. For each environment, the top row shows the Average undiscounted cumulative cost during training, and bottom row shows the Average undiscounted return. 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.

A.2.2 Backtracks in CPO and SDDPG

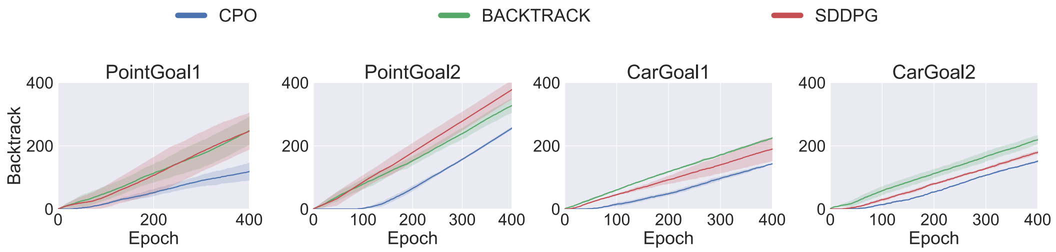

Figure 7: We compare the cumulative number of backtracking steps taken by CPO and SDDPG to BACKTRACK method for the first 400 epochs/policy updates.

Figure 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.

A.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 1: LBPO

```
Initialize parameterized actor $\pi_\phi$ with a safe initial policy, reward Q-function $Q_\theta^R$ and a cost Q
  function $Q_\theta^C$
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-($\lambda$) estimate.
    Step 3: Update the policy parameters by minimizing the objective in Eq 12.
        $\min_\phi \mathbb{E}_{s\sim\mathcal{R}}\Big[-Q^R_{\pi_{\phi,i-1}}(s,\pi_\phi(s)) + \psi(Q^C_{\pi_{\phi,i-1}}(s,\pi_\phi(s)))\Big]$
        $\text{s.t } \mathrm{D_{KL}}(\pi_\phi + \mathcal{N}(0,\delta) \| \pi_{\phi,i-1} + \mathcal{N}(0,\delta)) < \mu$
    Step 4: Set $\pi_{\phi,i}$ to be the safe policy resulting from the update in Step 3 $\pi_\phi$.
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 (Achiam et al., 2017) significantly differ from the benchmarks shown in (Ray et al., 2019) 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.

Table 4: LBPO Hyperparameters

Hyperparamater	**Value**
N	30
β	0.005[1]
β-thres	0.05
Policy learning rate	3e-4
Q-function learning rate	1e-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 (Chow et al., 2019). A brief discussion of practical issue faced in the implementation is present in Section 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.

[1] β is set to 0.005 for most environments. Appendix A.3 describes specific value of β for each environment.