Learning Barrier Certificates: Towards Safe Reinforcement Learning with Zero Training-time Violations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Training-time safety violations have been a major concern when we deploy reinforcement learning algorithms in the real world. This paper explores the possibility of safe RL algorithms with zero training-time safety violations in the challenging setting where we are only given a safe but trivial-reward initial policy without any prior knowledge of the dynamics and additional offline data. We propose an algorithm, **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS),which iteratively *learns* barrier certificates, dynamics models, and policies. The barrier certificates are learned via adversarial training and ensure the policy's safety assuming calibrated learned dynamics. We also add a regularization term to encourage larger certified regions to enable better exploration. Empirical simulations show that zero safety violations are already challenging for a suite of simple environments with only 2-4 dimensional state space, especially if high-reward policies have to visit regions near the safety boundary. Prior methods require hundreds of violations to achieve decent rewards on these tasks, whereas our proposed algorithms incur zero violations.

1 Introduction

Researchers have demonstrated that reinforcement learning (RL) can solve complex tasks such as Atari games [Mnih et al., 2015], Go [Silver et al., 2017], dexterous manipulation tasks [Akkaya et al., 2019], and many more robotics tasks in simulated environments [Haarnoja et al., 2018]. However, deploying RL algorithms to real-world problems still faces the hurdle that they require many unsafe environment interactions. For example, a robot's unsafe environment interactions include falling and hitting other objects, which incur physical damage costly to repair. Many recent deep RL works reduce the number of environment interactions significantly (e.g., see Haarnoja et al. [2018], Fujimoto et al. [2018], Janner et al. [2019], Dong et al. [2020], Luo et al. [2019], Chua et al. [2018] and reference therein), but the number of unsafe interactions is still prohibitive for safety-critical applications such as robotics, medicine, or autonomous vehicles [Berkenkamp et al., 2017].

Reducing the number of safety violations may not be sufficient for these safety-critical applications—we may have to eliminate them. This paper explores the possibility of safe RL algorithms with *zero safety violations* in both training time and test time. We also consider the challenging setting where we are only given a safe but trivial-reward initial policy.

A recent line of works on safe RL design novel actor-critic based algorithms under the constrained policy optimization formulation [Thananjeyan et al., 2021, Srinivasan et al., 2020, Bharadhwaj et al., 2020, Yang et al., 2020, Stooke et al., 2020]. They significantly reduce the number of training-time safety violations. However, these algorithms fundamentally learn the safety constraints by contrasting

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

the safe and unsafe trajectories. In other words, because the safety set is only specified through the safety costs that are observed *postmortem*, the algorithms only learn the concept of safety through seeing unsafe trajectories. Therefore, these algorithms cannot achieve zero training-time violations. For example, even for the simple 2D inverted pendulum environment, these methods still require at least 80 unsafe trajectories (see Figure 2 in Section 6).

Another line of work utilizes ideas from control theory and model-based approach [Cheng et al., 2019, Berkenkamp et al., 2017, Taylor et al., 2019, Zeng et al., 2020]. These works propose sufficient conditions involving certain Lyapunov functions or control barrier functions that can certify the safety of a subset of states or policies [Cheng et al., 2019]. These conditions assume access to calibrated dynamical models. They can, in principle, permit safety guarantees without visiting any unsafe states because, with the calibrated dynamics, we can foresee future danger. However, control barrier functions are often non-trivially *handcrafted* with prior knowledge of the environments [Ames et al., 2019, Nguyen and Sreenath, 2016].

This work aims to design model-based safe RL algorithms that achieve zero training-time safety violations by *learning* the barrier certificates *iteratively*. We present the algorithm **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS), which alternates between *learning* barrier certificates that certify the safety of *larger* regions of states, optimizing the policy, collecting more data within the certified states, and refining the learned dynamics with data.

The work of Richards et al. [2018] is a closely related prior result, which learns a Lyapunov function given a fixed dynamics model via discretization of the state space. Our work significantly extends it with three algorithmic innovations. First, we use adversarial training to learn the certificates, which avoids discretizing state space and can potentially work with higher dimensional state space than the two-dimensional problems in Richards et al. [2018]. Second, we do not assume a given, globally accurate dynamics; instead, we learn the dynamics from safe explorations. We achieve this by co-learning the certificates, dynamics, and policy to iteratively grow the certified region and improve the dynamics and still maintain zero violations. Thirdly, the work Richards et al. [2018] only certifies the safety of some states and does not involve learning a policy. In contrast, our work learns a policy and tailors the certificates to the learned policies. In particular, our certificates aim to certify only states near the trajectories of the current and past policies—this allows us to not waste the expressive power of the certificate parameterization on irrelevant low-reward states.

We evaluate our algorithms on a suite of tasks, including a few where achieving high rewards requires careful exploration near the safety boundary. For example, in the ***Swing*** environment, the goal is to swing a rod with the largest possible angle under the safety constraints that the angle is less than 90°. We show that our method reduces the number of safety violations from several hundred to zero on these tasks.

2 Setup and Preliminaries

2.1 Problem Setup

We consider the standard RL setup with an infinite-horizon *deterministic* Markov decision process (MDP). An MDP is specified by a tuple $(\mathcal{S}, \mathcal{A}, \gamma, r, \mu, T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $0 \le \gamma < 1$ is the discount factor, μ is the distribution of the initial state, and $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the deterministic dynamics model. Let $\Delta(\mathcal{X})$ denote the family of distributions over a set $\mathcal{X}$. The expected discounted total reward of a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is defined as

$$J(\pi) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)\right],$$

where $s_0 \sim \mu, a_i \sim \pi(s_i), s_{i+1} = T(s_i, a_i)$ for $i \ge 0$. The goal is to find a policy π which maximizes $J(\pi)$.

Let $\mathcal{S}_{\text{unsafe}} \subset \mathcal{S}$ be the set of unsafe states specified by the user. The user-specified safe set $\mathcal{S}_{\text{safe}}$ is defined as $\mathcal{S} \backslash \mathcal{S}_{\text{unsafe}}$. A state s is (user-specified) safe if $s \in \mathcal{S}_{\text{safe}}$. A trajectory is safe if and only if all the states in the trajectory are safe. An initial state drawn from μ is assumed to safe with probability 1. We say a deterministic policy π is safe starting from state s, if the infinite-horizon trajectory obtained by executing π starting from s is safe. We also say a policy π is safe if it is safe starting from an initial state drawn from μ with probability 1. A major challenge toward safe RL is the existence of irrecoverable states which are currently safe but will eventually lead to unsafe states regardless of future actions. We define the notion formally as follows.

Definition 1. *A state s is* viable *iff there exists a policy π such that π is safe starting from s, that is, executing π starting from s for infinite steps never leads to an unsafe state. A user-specified safe state that is not viable is called an* irrecoverable *state.*

We remark that unlike Srinivasan et al. [2020], Roderick et al. [2020], we do not assume all safe states are viable. We rely on the extrapolation and calibration of the dynamics to foresee risks. A calibrated dynamics model $\widehat{\mathcal{T}}$ predicts a confidence region of states $\widehat{\mathcal{T}}(s, a) \subseteq \mathcal{S}$, such that for any state s and action a, we have $T(s, a) \in \widehat{\mathcal{T}}(s, a)$.

2.2 Preliminaries on Barrier Certificate

Barrier certificates are powerful tools to certify the stability of a dynamical system. Barrier certificates are often applied to a continuous-time dynamical system, but here we describe its discrete-time version where our work is based upon. We refer the readers to Prajna and Jadbabaie [2004], Prajna and Rantzer [2005] for more information about continuous-time barrier certificates.

Given a discrete-time dynamical system $s_{t+1} = f(s_t)$ *without control* starting from s_0, a function $h : \mathcal{S} \to \mathbb{R}$ is a barrier certifcate if for any $s \in \mathcal{S}$ such that $h(s) \geq 0$, $h(f(s)) \geq 0$. Zeng et al. [2020] considers a more restrictive requirement: For any state $s \in \mathcal{S}$, $h(f(s)) \geq \alpha h(s)$ for a constant $0 \leq \alpha < 1$.

it is easy to use a barrier certificate h to show the stability of the dynamical system. Let $\mathcal{C}_h = \{s : h(s) \geq 0\}$ be the superlevel set of h. The requirement of barrier certificates directly translates to the requirement that if $s \in \mathcal{C}_h$, then $f(s) \in \mathcal{C}_h$. This property of $\mathcal{C}_h$, which is known as the *forward-invariant* property, is especially useful in safety-critical settings: suppose a barrier certificate h such that $\mathcal{C}_h$ does not contain unsafe states and contains the initial state s_0, then it is guaranteed that $\mathcal{C}_h$ contains the entire trajectory of states $\{s_t\}_{t \geq 0}$ which are safe.

Finding barrier certificates requires a known dynamics f, which often can only be approximated in practice. This issue can be resolved by using a well-calibrated dynamics model $\hat{f}$, which predicts a confidence interval containing the true output. When a calibrated dynamics model $\hat{f}$ is used, we require that for any $s \in \mathcal{S}$, $\min_{s' \in \hat{f}(s)} h(s') \geq 0$.

Control barrier functions [Ames et al., 2019] are extensions to barrier certificates in the control setting. That is, control barrier functions are often used to *find* an action to meet the safety requirement instead of certifying the stability of a closed dynamical system. In this work, we simply use barrier certificates because in Section 3, we view the policy and the calibrated dynamics model as a whole closed dynamical system whose stability we are going to certify.

3 Learning Barrier Certificates via Adversarial Training

This section describes an algorithm that learns a barrier certificate for a fixed policy π under a calibrated dynamics model $\widehat{\mathcal{T}}$. Concretely, to certify a policy π is safe, we aim to learn a (discrete-time) barrier certificate h that satisfies the following three requirements.

R.1. For $s_0 \sim \mu$, $h(s_0) \geq 0$ with probability 1.

R.2. For every $s \in \mathcal{S}_{\text{unsafe}}$, $h(s) < 0$.

R.3. For any s such that $h(s) \geq 0$, $\min_{s' \in \widehat{\mathcal{T}}(s, \pi(s))} h(s) \geq 0$.

Requirement **R.1** and **R.3** guarantee that the policy π will never leave the set $\mathcal{C}_h = \{s \in \mathcal{S} : h(s) \geq 0\}$ by simple induction. Moreover, **R.2** guarantees that $\mathcal{C}_h$ only contains safe states and therefore the policy never visits unsafe states.

In the rest of the section, we aim to design and train such a barrier certificate $h = h_\phi$ parametrized by neural network ϕ.

h_ϕ **parametrization.** The three requirements for a barrier certificate are challenging to simultaneously enforce with constrained optimization involving neural network parameterization. Instead, we will parametrize h_ϕ with **R.1** and **R.2** built-in such that for any ϕ, h_ϕ always satisfies **R.1** and **R.2**.

We assume the initial state s_0 is deterministic (the parameterization can be extended to multiple initial states.) To capture the known user-specified safety set, we first handcraft a continuous function $\mathcal{B}_{\text{safe}} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ satisfying $\mathcal{B}_{\text{safe}}(s) \approx 0$ for typical $s \in \mathcal{S}_{\text{safe}}$ and $\mathcal{B}_{\text{safe}}(s) > 1$ for any $s \in \mathcal{S}_{\text{unsafe}}$.[1] The construction of $\mathcal{B}_{\text{safe}}$ does not need prior knowledge of irrecoverable states, but only the user-specified safety set $\mathcal{S}_{\text{safe}}$. To further encode the user-specified safety set into h_ϕ, we choose h_ϕ to be of form $h_\phi(s) = 1 - \text{Softplus}(f_\phi(s) - f_\phi(s_0)) - \mathcal{B}_{\text{safe}}(s)$, where f_ϕ is a neural network, and $\text{Softplus}(x) = \log(1 + e^x)$.

Because s_0 is safe and $\mathcal{B}_{\text{safe}}(s_0) \approx 0$, $h_\phi(s_0) \approx 1 - \text{Softplus}(0) > 0$. Therefore h_h satisfies **R.1**. Moreover, for any $s \in \mathcal{S}_{\text{unsafe}}$, we have $h_\phi(s) < 1 - \mathcal{B}_{\text{safe}}(s) < 0$, so h_ϕ in our parametrization satisfies **R.2** by design.

Training barrier certificates. We now move on to training ϕ to satisfy **R.3**. Let

$$U(s, a, h) := \max_{s' \in \widehat{\mathcal{T}}(s,a)} -h(s'). \tag{1}$$

Then, **R.3** requires $U(s, \pi(s), h_\phi) \leq 0$ for *any* $s \in \mathcal{C}_{h_\phi}$, The constraint in **R.3** naturally leads up to formulate the problem as a min-max problem. Define our objective function to be

$$C^*(h_\phi, U, \pi) := \max_{s \in \mathcal{C}_{h_\phi}} U(s, \pi(s), h_\phi) = \max_{s \in \mathcal{C}_{h_\phi}, s' \in \widehat{\mathcal{T}}(s, \pi(s))} -h(s'), \tag{2}$$

and we want to minimize C^* w.r.t. ϕ:

$$\min_\phi C^*(h_\phi, U, \pi) = \min_\phi \max_{s \in \mathcal{C}_{h_\phi}, s' \in \widehat{\mathcal{T}}(s, \pi(s))} -h(s'), \tag{3}$$

Our goal is to ensure the minimum value is less than 0. We use gradient descent to solve the optimization problem. We also derive the gradient of $C^*(L_\phi, U, \pi)$ w.r.t. ϕ :

$$\nabla_\phi C^*(h_\phi, U, \pi) = \nabla_\phi U(s^*, \pi(s^*), h_\phi) - \frac{\|\nabla_\phi U(s^*, \pi(s^*), h_\phi)\|_2}{\|\nabla_\phi h_\phi(s^*)\|_2} \nabla_\phi h_\phi(s^*), \tag{4}$$

where $s^* := \arg\max_{s: h_\phi(s) \leq 1} U(s, \pi(s), h_\phi)$ and we defer the derivation to Appendix A.

Computing the adversarial s^*.

Equation (4) requires us to compute s^* efficiently. Because the maximization problem with respect to s is nonconcave, there could be multiple local maxima. In practice, we find that it is more efficient and reliable to use multiple local maxima to compute $\nabla_\phi C^*$ and then average the gradient.

Solving s^* is highly non-trivial, as it is a non-concave optimization problem with a constraint $s \in \mathcal{C}_{h_\phi}$. To deal with the constraint, we introduce a Lagrangian multiplier λ and optimize $U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_{h_\phi}}$ w.r.t. s without any constraints. However, it is still very time-consuming to solve an optimization problem independently at each time. Based on the observation that the parameters of h do not change too much by one step of gradient step, we can use the optimal solution from the last optimization problem as the initial solution for the next one, which naturally leads to the idea of maintaining a set of candidates of s^*'s during the computation of $\nabla_\phi C^*$.

We use Metropolis-adjusted Langevin algorithm (MALA) to maintain a set of candidates $\{s_1, \ldots, s_m\}$ which are supposed to sample from $\exp(\tau(U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_{h_\phi}}))$ for $\tau = 30$ and $\lambda = 33$. Here τ is the temperature indicating we want to focus on the samples with large $U(s, \pi(s), h_\phi)$. Although the indicator function always have zero gradient, it is still useful in the sense that MALA will reject $s_i \notin \mathcal{C}_{h_\phi}$. A detailed description of MALA is given in Appendix D.

We choose MALA over gradient descent because the maintained candidates are more diverse, approximate local maxima. If we use gradient descent to find s^*, then multiple runs of GD likely arrive at the same s^*, so that we lost the parallelism from simultaneously working with multiple

[1]The function $\mathcal{B}_{\text{safe}}(s)$ is called a barrier function for the user-specified safe set in the optimization literature. Here we do not use this term to avoid confusion with the barrier certificate.

Algorithm 1 Learning barrier certificate h_ϕ for a policy π w.r.t. a calibrated dynamics model $\widehat{\mathcal{T}}$.

Require: Temperature τ, Lagrangian multiplier λ, and optionally a regularization function Reg.
1: Let U be defined as in Equation (1).
2: Initialize m candidates of $s_1, \ldots, s_m \in \mathcal{S}$ randomly.
3: **for** n iterations **do**
4:     **for** every candidate s_i **do**
5:         sample $s_i \sim \exp(\tau U(s, \pi(s), h_\phi) - \lambda \mathbb{I}_{s \in \mathcal{C}_h})$ by MALA (Algorithm 5).
6:     $W \leftarrow \{s_i : h_\phi(s_i) \geq 0, i \in [m]\}$.
7:     Train ϕ to minimize $C^*(h_\phi, U, \pi) + \mathsf{Reg}(\phi)$ using all candidates in W.

Algorithm 2 CRABS: **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (Details in Section 4)

Require: An initial safe policy π_{init}.
1: Collected trajectories buffer $\widehat{D} \leftarrow \emptyset$; $\pi \leftarrow \pi_{\text{init}}$.
2: **for** T epochs **do**
3:     Invoke Algorithm 3 to safely collect trajectories (using π as the safeguard policy and a noisy version of π as the π^{expl}). Add the trajectories to $\widehat{D}$.
4:     Learn a calibrated dynamics $\widehat{\mathcal{T}}$ with $\widehat{D}$.
5:     Learn a barrier certificate h that certifies π w.r.t. $\widehat{\mathcal{T}}$ using Algorithm 1 with regularization.
6:     Optimize policy π (according to the reward), using data in $\widehat{D}$, with the constraint that π is certified by h.

local maxima. MALA avoids this issue by its intrinsic stochasticity, which can also be controlled by adjusting the hyperparameter τ.

We summarize our algorithm of training barrier certificates in Algorithm 1 (which contains optional regularization that will be discussed in Section 4.2). At Line 2, the initialization of s_i's is arbitrary, as long as they have a sort of stochasticity.

4 CRABS: Co-trained Barrier Certificate for Safe RL

In this section, we present our main algorithm, **C**o-t**ra**ined **B**arrier Certificate for **S**afe RL (CRABS), shown in Algorithm 2, to *iteratively* co-train barrier certificates, policy and dynamics, using the algorithm in Section 3. In addition to parametrizing h by ϕ, we further parametrize the policy π by θ, and parametrize calibrated dynamics model $\widehat{\mathcal{T}}$ by ω. CRABS alternates between training a barrier certificate that certifies the policy π_θ w.r.t. a calibrated dynamics model $\widehat{\mathcal{T}}_\omega$ (Line 5), collecting data safely using the certified policy (Line 3, details in Section 4.1), learning a calibrated dynamics model (Line 4, details in Section 4.3), and training a policy with the constraint of staying in the superlevel set of the barrier function (Line 6, details in Section 4.4). In the following subsections, we discuss how we implement each line in detail.

4.1 Safe Exploration with Certified Safeguard Policy

Safe exploration is challenging because it is difficult to detect irrecoverable states. The barrier certificate is designed to address this — a policy π certified by some h guarantees to stay within $\mathcal{C}_h$ and therefore can be used for collecting data. However, we may need more diversity in the collected data beyond what can be offered by the deterministic certified policy $\pi^{\text{safeguard}}$. Thanks to the contraction property **R.3**, we in fact know that any exploration policy π^{expl} within the superlevel set $\mathcal{C}_h$ can be made safe with $\pi^{\text{safeguard}}$ being a safeguard policy—we can first try actions from π^{expl} and

Algorithm 3 Safe exploration with safeguard policy $\pi^{\text{safeguard}}$

Require: (1) A policy $\pi^{\text{safeguard}}$ certified by barrier certificate h, (2) any proposal exploration policy π^{expl}.
Require: A state $s \in \mathcal{C}_{h_\phi}$.
1: Sample n actions $a_1, \ldots a_n$ from $\pi^{\text{expl}}(s)$.
2: **if** there exists an a_i such that $U(s, a_i, h) \leq 1$ **then**
3:     **return:** a_i
4: **else**
5:     **return:** $\pi^{\text{safeguard}}(s)$.

see if they stay within the viable subset $\mathcal{C}_h$, and if none does, invoke the safeguard policy $\pi^{\text{safeguard}}$. Algorithm 3 describes formally this simple procedure that makes any exploration policy π^{expl} safe. By a simple induction, one can see that the policy defined in Algorithm 3 maintains that all the visited states lie in $\mathcal{C}_h$.

The safeguard policy $\pi^{\text{safeguard}}$ is supposed to safeguard the exploration. However, activating the safeguard too often is undesirable, as it only collects data from $\pi^{\text{safeguard}}$ so there will be little exploration. To mitigate this issue, we often choose π^{expl} to be a noisy version of $\pi^{\text{safeguard}}$ so that π^{expl} will be roughly safe by itself. Moreover, the safeguard policy $\pi^{\text{safeguard}}$ will be trained via optimizing the reward function as shown in the next subsections. Therefore, a noisy version of $\pi^{\text{safeguard}}$ will explore the high-reward region and avoid unnecessary exploration.

Following Haarnoja et al. [2018], the policy π_θ is parametrized as $\tanh(\mu_\theta(s))$, and the proposal exploration policy π_θ^{expl} is parametrized as $\tanh(\mu_\theta(s) + \sigma_\theta(s)\zeta)$ for $\zeta \sim \mathcal{N}(0, I)$, where μ_θ and σ_θ are two neural networks. Here the $\tanh$ is applied to squash the outputs to the action set $[-1, 1]$.

4.2 Regularizing Barrrier Certificates

The quality of exploration is directly related to the quality of policy optimization. In our case, the exploration is only within the learned viable set $\mathcal{C}_{h_\phi}$ and it will be hindered if $\mathcal{C}_{h_\phi}$ is too small or does not grow during training. To ensure a large and growing viable subset $\mathcal{C}_{h_\phi}$, we encourage the volume of $\mathcal{C}_{h_\phi}$ to be large by adding a regularization term

$$\mathsf{Reg}(\phi; \hat{h}) = \mathbb{E}_{s\in\mathcal{S}}[\text{relu}(\hat{h}(s) - h_\phi(s))],$$

Here $\hat{h}$ is the barrier certificate obtained in the previous epoch. In the ideal case when $\mathsf{Reg}(\phi; \hat{h}) = 0$, we have $\mathcal{C}_{h_\phi} \supset \mathcal{C}_{\hat{h}}$, that is, the new viable subset $\mathcal{C}_{h_\phi}$ is at least bigger than the reference set (which is the viable subset in the previous epoch.) We compute the expectation over $\mathcal{S}$ approximately by using the set of candidate s's maintained by MALA.

In summary, to learn h_ϕ in CRABS, we minimize the following objective (for a small positive constant λ) over ϕ as shown in Algorithm 1:

$$\mathcal{L}(\phi; U, \pi_\theta, \hat{h}) = C^*(L_\phi, U, \pi_\theta) + \lambda \mathsf{Reg}(\phi; \hat{h}). \tag{5}$$

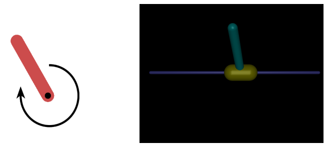
We remark that the regularization is not the only reason why the viable set $\mathcal{C}_{h_\phi}$ can grow. When the dynamics becomes more accurate as we collect more data, the $\mathcal{C}_{h_\phi}$ will also grow. This is because an inaccurate dynamics will typically make the $\mathcal{C}_{h_\phi}$ smaller—it is harder to satisfy **R.3** when the confidence region $\widehat{\mathcal{T}}(s, \pi(s))$ in the constraint contains many possible states. Vice versa, shrinking the size of the confidence region will make it easier to certify more states.

4.3 Learning a Calibrated Dynamics Model

It is a challenging open question to obtain a dynamics model $\widehat{\mathcal{T}}$ (or any supervised learning model) that is theoretically well-calibrated especially with domain shift [Zhao et al., 2020]. In practice, we heuristically approximate a calibrated dynamics model by learning an ensemble of probabilistic dynamics models, following common practice in RL [Yu et al., 2020, Janner et al., 2019, Chua et al., 2018]. We learn K probabilistic dynamics models $f_{\omega_1}, \dots, f_{\omega_K}$ using the data in the replay buffer $\widehat{D}$. (Interestingly, prior work shows that an ensemble of probabilistic models can still capture the error of estimating a deterministic ground-truth dynamics [Janner et al., 2019, Chua et al., 2018].) Each probabilistic dynamics model f_{ω_i} outputs a Gaussian distribution $\mathcal{N}(\mu_{\omega_i}(s, a), \text{diag}(\sigma^2_{\omega_i}(s, a)))$ with diagonal covariances, where μ_{ω_i} and σ_{ω_i} are parameterized by neural networks. Given a replay buffer $\widehat{D}$, the objective for a probabilistic dynamics model f_{ω_i} is to minimize the negative log-likelihood:

$$\mathcal{L}_{\widehat{\mathcal{T}}}(\omega_i) = -\mathbb{E}_{(s,a,s')\sim\widehat{D}}\left[-\log f_{\omega_i}(s'|s, a)\right]. \tag{6}$$

The only difference in the training procedure of these probabilistic models is the randomness in the initialization and mini-batches. We simply aggregate the means of all learn dynamics models as a coarse approximation of the confidence region, i.e., $\widehat{\mathcal{T}}(s, a) = \{\mu_{\omega_i}(s, a)\}_{i\in[K]}$.



(a) Pendulum (b) CartPole

Figure 1: Illustration of environments. The left figure illustrates the Pendulum environment, which is used by ***Upright*** and ***Tilt*** tasks. The right figer illustrates the CartPole environment, which is used by ***Move*** and ***Swing*** tasks.

4.4 Policy Optimization

We describe our policy optimization algorithm in Algorithm 4 . The desiderata here are (1) the policy needs certified by the current barrier certificate h and (2) the policy has as high reward as possible. We break down our policy optimization algorithm into two components: First, we optimize the total rewards $J(\pi_\theta)$ of the policy π_θ; Second, we use adversarial training to guarantee the optimized policy can be certified by h_ϕ. The modification of SAC is to some extent non-essential and mostly for technical convenience of making SAC somewhat compatible with the constraint set. Instead, it is the adversarial step that fundamentally guarantees that the policy is certified by the current h_ϕ.

Adversarial training. We use adversarial training to guarantee π_θ can be certified by h_ϕ. Similar to what we've done in training h_ϕ adversarially, the objective for training π_θ is to minimize $C^*(h_\phi, U, \pi_\theta)$. Unlike the case of ϕ, the gradient of $C^*(h_\phi, U, \pi_\theta)$ w.r.t. θ is simply $\nabla_\theta U(s^*, \pi_\theta(s^*), h_\phi)$, as the constraint $h_\phi(s)$ is unrelated to π_θ. We also use MALA to solve s^* and plug it into the gradient term $\nabla_\theta U(s^*, \pi_\theta(s^*), h_\phi)$.

Optimizing $J(\pi_\theta)$. We use a modified SAC [Haarnoja et al., 2018] to optimize $J(\pi_\theta)$. As the modification is for safety concerns and is minor, we defer it to Appendix B. As a side note, although we only optimize π_θ^{expl} here, π_θ is also optimized implicitly because π_θ^{expl} simply outputs the mean of π_θ deterministically.

5 High-risk, High-reward Environments

We design four tasks, three of which are high-risk, high-reward tasks, to check the efficacy of our algorithm. Even though they are all based on inverted pendulum or cart pole, we choose the reward function to be somewhat conflicted with the safety constraints. That is, the optimal policy needs to take a trajectory that is near the safety boundary. This makes the tasks particularly challenging and suitable for stress testing our algorithm's capability of avoiding irrecoverable states.

These tasks have state dimension dimensions between 2 to 4. We focus on the relatively low dimensional environments to avoid conflating the failure to learn accurate dynamics models from data and the failure to provide safety given a learned approximate dynamics. Indeed, we identify that the major difficulty to scale up to high-dimensional environments is that it requires significantly more data to learn a decent high-dimensional dynamics that can predict long-horizon trajectories. We remark that we aim to have zero violations. This is very difficult to achieve, even if the environment is low dimensional. As shown by Section 6, many existing algorithms fail to do so.

(a) ***Upright***. The task is based on Pendulum-v0 in Open AI Gym [Brockman et al., 2016], as shown in Figure 1a. The agent can apply torque to control a pole. The environment involves the crucial quantity: the tilt angle θ which is defined to be the angle between the pole and a vertical line. The safety requirement is that the pole does not fall below the horizontal line. Technically, the user-specified safety set is $\{\theta : |\theta| \le \theta_{\text{max}} = 1.5\}$ (note that the threshold is very close to $\frac{\pi}{2}$ which corresponds to $90°$.) The reward function r is $r(s, a) = -\theta^2$, so the optimal policy minimizes the angle and angular speed by keeping the pole upright. The horizon is 200 and the initial state $s_0 = (0.3, -0.9)$.

(b) ***Tilt***. This action set, dynamics, and horizon, and safety set are the same as in *Upright*. The reward function is different: $r(s, a) = -(\theta_{\text{limit}} - \theta)^2$. The optimal policy is supposed to stay tilting near the angle $\theta = \theta_{\text{limit}}$ where $\theta_{\text{limit}} = -0.41151684$ is the largest angle the pendulum can stay balanced. The challenge is during exploration, it is easy for the pole to overshoot and violate the safety constraints.

(c) ***Move***. The task is based on a cart pole and the goal is to move a cart (the yellow block) to control the pole (with color teal), as shown in Figure 1b. The cart has an x position between -1 and 1, and the pole also has an angle $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ with the same meaning as ***Upright*** and ***Tilt***. The starting position is $x = \theta = 0$. We design the reward function to be $r(s, a) = x^2$. The user-specified safety set is $\{(x, \theta) : |\theta| \le \theta_{\text{max}} = 0.2, |x| \le 0.9\}$ where 0.2 corresponds to roughly $11°$. Therefore, the optimal policy needs to move the cart and the pole slowly in one direction, preventing the pole from falling down and the cart from going too far. The horizon is set to 1000.

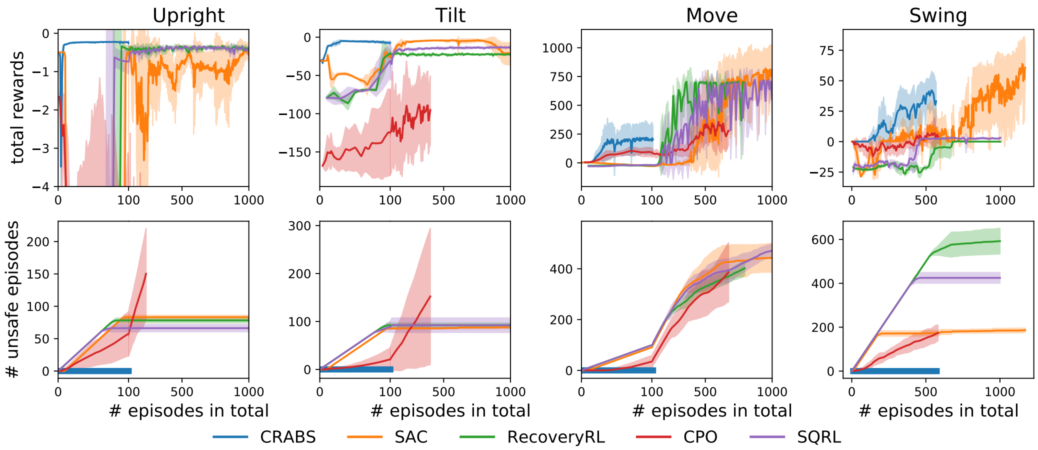
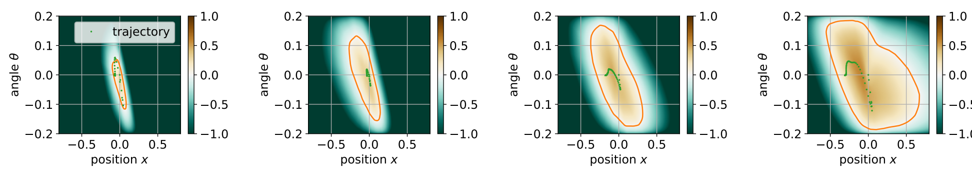


Figure 2: Comparision between CRABS and baselines. CRABS can learn a policy without any safety violations, while other baselines have a lot of safety violations. We run each algorithm four times with independent randomness. The solid curves indicate the mean of four runs and the shaded areas indicate one standard deviation around the mean.



(a) $\mathcal{C}_{h_\phi}$ after 0 epochs (b) $\mathcal{C}_{h_\phi}$ after 5 epochs (c) $\mathcal{C}_{h_\phi}$ after 10 epochs (d) $\mathcal{C}_{h_\phi}$ after 15 epochs

Figure 3: Visualization of the growing viable subsets learned by CRABS in ***Move***. To illustrate the 4-dimensional state space, we project a state from $[x, \theta, \dot{x}, \dot{\theta}]$ to $[x, \theta]$. The red curve encloses superlevel set $\mathcal{C}_{h_\phi}$, while the green points indicate the projected trajectory of the current safe policy. We can also observe that policy π learns to move left as required by the task. We note that shown states in the trajectory sometimes seemingly are not be enclosed by the red curve due to the projection.

(d) ***Swing***. This task is similar to ***Move***, except for a few differences: The reward function is $r(s, a) = \theta^2$; The user-specified safety set is $\{(x, \theta) : |\theta| \leq \theta_{\max} = 1.5, |x| \leq 0.9\}$. So the optimal policy will swing back and forth to some degree and needs to control the angles well so that it does not violate the safety requirement.

For all the tasks, once the safety constraint is violated, the episode will terminate immediately and the agent will receive a reward of -30 as a penalty. The number -30 is tuned by running SAC and choosing the one that SAC performs best with.

6 Experimental Results

In this section, we conduct experiments to answer the following question: Can CRABS learn a reasonable policy without safety violations in the designed tasks?

Baselines. We compare our algorithm CRABS against four baselines: (a) **Soft Actor-Critic** (SAC) [Haarnoja et al., 2018], one of the state-of-the-art RL algorithms, (b) **Constrained Policy Optimization** (CPO) [Achiam et al., 2017], a safe RL algorithm which builds a trust-region around the current policy and optimizes the policy in the trust-region, (c) **RecoveryRL** [Thananjeyan et al., 2021] which leverages offline data to pretrain a risk-sensitive Q function and also utilize two policies to achieving two goals (being safe and obtaining high rewards), and (d) **SQRL** [Srinivasan et al., 2020] which leverages offline data in an easier environment and fine-tunes the policy in a more difficult environment. SAC and CPO are given an initial safe policy for safe exploration, while RecoveryRL and SQRL are given offline data containing 40K steps from both mixed safe and unsafe trajectories which are free and are not counted. CRABS collects more data at each iteration in ***Swing*** than in other tasks to learn a better dynamics model $\widehat{\mathcal{T}}$. For SAC, we use the default hyperparameters because we found they are not sensitive. For RecoveryRL and SQRL, the hyperparameters are tuned in the same way as in Thananjeyan et al. [2021] . For CPO, we tune the step size and batch size. More details of experiment setup and the implementation of baselines can be found in Appendix C.

Results. Our main results are shown in Figure 2. From the perspective of total rewards, SAC achieves the best total rewards among all of the 5 algorithms in ***Move*** and ***Swing***. In all tasks, CRABS can achieve reasonable total rewards and learns faster at the beginning of training, and we hypothesize that this is directly due to its strong safety enforcement. RecoveryRL and SQRL learn faster than SAC in ***Move***, but they suffer in ***Swing***. RecoveryRL and SQRL are not capable of learning in ***Swing***, although we observed the average return during exploration at the late stages of training can be as high as 15. CPO is quite sample-inefficient and does not achieve reasonable total rewards as well.

From the perspective of safety violations, CRABS surpasses all baselines **without a single safety violation**. The baseline algorithms always suffer from many safety violations. SAC, SQRL, and RecoveryRL have a similar number of unsafe trajectories in ***Upright, Tilt, Move***, while in ***Swing***, SAC has the fewest violations and RecoveryRL has the most violations. CPO has a lot of safety violations. We observe that for some random seeds, CPO does find a safe policy and once the policy is trained well, the safety violations become much less frequent, but for other random seeds, CPO keeps visiting unsafe trajectories before it reaches its computation budget.

Visualization of learned viable subset $\mathcal{C}_{h_\phi}$. We visualized the viable set $\mathcal{C}_{h_\phi}$ in Figure 3. As shown in the figure, our algorithm CRABS succeeds in certifying more and more viable states and does not get stuck locally, which demonstrates the efficacy of the regularization at Section 4.2.

Handcrafted barrier function h. To demonstrate the advantage of learning a barrier function, we also conduct experiments on a variant of CRABS, which uses a handcrafted barrier certificate by ourselves and does not train it, that is, Algorithm 2 without Line 5. The results show that this variant does not perform well: It does not achieve high rewards, and has many safety violations. We hypothesize that the policy optimization is often burdened by adversarial training, and the safeguard policy sometimes cannot find an action to stay within the superlevel set $\mathcal{C}_h$.

7 Related Work

Prior works about Safe RL take very different approaches. Dalal et al. [2018] adds an additional layer, which corrects the output of the policy locally. Some of them use Lagrangian methods to solve CMDP, while the Lagrangian multiplier is controlled adaptively [Tessler et al., 2018] or by a PID [Stooke et al., 2020]. Achiam et al. [2017], Yang et al. [2020] build a trust-region around the current policy. Eysenbach et al. [2017] learns a reset policy so that the policy only explores the states that can go back to the initial state. Turchetta et al. [2020] introduces a learnable teacher, which keeps the student safe and helps the student learn faster in a curriculum manner. Srinivasan et al. [2020] pre-trains a policy in a simpler environment and fine-tunes it in a more difficult environment. Bharadhwaj et al. [2020] learns conservative safety critics which underestimate how safe the policy is, and uses the conservative safety critics for safe exploration and policy optimization. Thananjeyan et al. [2021] makes use of existing offline data and co-trains a recovery policy.

Another line of work involves Lyapunov functions and barrier functions. Donti et al. [2020] constructs sets of stabilizing actions using a Lyapunov function, and project the action to the set, while Chow et al. [2019] projects action or parameters to ensure the decrease of Lyapunov function after a step. Ohnishi et al. [2019] is similar to ours but it constructs a barrier function manually instead of learning such one. Ames et al. [2019] gives an excellent overview of control barrier functions and how to design them. Perhaps the most related work to ours is Cheng et al. [2019], which also uses a barrier function to safeguard exploration and uses a reinforcement learning algorithm to learn a policy. However, the key difference is that we *learn* a barrier function, while Cheng et al. [2019] handcrafts one. The works on Lyapunov functions [Berkenkamp et al., 2017, Richards et al., 2018] require the discretizating the state space and thus only work for low-dimensional space.

8 Conclusion

In this paper, we propose a novel algorithm CRABS for training-time safe RL. The key idea is that we co-train a barrier certificate together with the policy to certify viable states, and only explore in the learned viable subset. The empirical rseults show that CRABS can learn some tasks without a single safety violation. We consider using model-based policy optimization techniques to improve the total rewards and sample efficiency as a promising future work. Another fascinating direction is how to deal with less accurate learned dynamics model in higher dimension environments.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 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.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. Conservative safety critics for exploration. *arXiv preprint arXiv:2010.14497*, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

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

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

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.

Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning. In *International Conference on Machine Learning*, pages 2627–2637. PMLR, 2020.

Priya L Donti, Melrose Roderick, Mahyar Fazlyab, and J Zico Kolter. Enforcing robust control guarantees within neural network policies. *arXiv preprint arXiv:2011.08105*, 2020.

Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

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.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJe1E2R5KX`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Quan Nguyen and Koushil Sreenath. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *2016 American Control Conference (ACC)*, pages 322–328. IEEE, 2016.

Motoya Ohnishi, Li Wang, Gennaro Notomista, and Magnus Egerstedt. Barrier-certified adaptive reinforcement learning with applications to brushbot navigation. *IEEE Transactions on robotics*, 35(5):1186–1205, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

Stephen Prajna and Anders Rantzer. On the necessity of barrier certificates. *IFAC Proceedings Volumes*, 38(1):526–531, 2005.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Spencer M. Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems, 2018.

Melrose Roderick, Vaishnavh Nagarajan, and J Zico Kolter. Provably safe pac-mdp exploration using analogies. *arXiv preprint arXiv:2007.03574*, 2020.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

Andrew J Taylor, Victor D Dorobantu, Hoang M Le, Yisong Yue, and Aaron D Ames. Episodic learning with control lyapunov functions for uncertain robotic systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6878–6884. IEEE, 2019.

Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3): 4915–4922, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Matteo Turchetta, Andrey Kolobov, Shital Shah, Andreas Krause, and Alekh Agarwal. Safe reinforcement learning via curriculum induction. *arXiv preprint arXiv:2006.12136*, 2020.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Accelerating safe reinforcement learning with constraint-mismatched policies. *arXiv preprint arXiv:2006.11645*, 2020.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

Jun Zeng, Bike Zhang, and Koushil Sreenath. Safety-critical model predictive control with discrete-time control barrier function. *arXiv preprint arXiv:2007.11718*, 2020.

Shengjia Zhao, Tengyu Ma, and Stefano Ermon. Individual calibration with randomized forecasting. *arXiv preprint arXiv:2006.10288*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Appendix E.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Appendix F.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Appendix C.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Gradient of $C^*(h_\phi, U, \pi)$

Recall the definition of C^* is

$$C^*(h_\phi, U, \pi) := \max_{s: h_\phi(s) \le 1} U(s, \pi(s), h_\phi),$$

The tricky part about optimizing C^* is that both the constraint and U function depend on the parameter ϕ, so the gradient of C^* is not merely just the gradient of U evaluated at the maximizer of U w.r.t s.

We first prove a more general lemma (Lemma 1). The gradient of $C^*(h_\phi, U, \pi)$ w.r.t. ϕ can be directly computed using Lemma 1.

Lemma 1. *Let $f, g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be two differentiable functions. For any $y \in \mathbb{R}^m$, define $x^* = \arg\max_{x: g(x,y) \le 0} f(x, y)$ and $f^* = f(x^*, y)$. The gradient of $f^*(y)$ w.r.t. y is given by:*

$$\nabla_y f^*(y) = \nabla_y f(x, y)|_{x^*} - \frac{\|\nabla_x f(x, y)|_{x^*}\|_2}{\|\nabla_x g(x, y)|_{x^*}\|_2} \nabla_y g(x, y)|_{x^*}.$$

Proof. Let $L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$ be the Lagrangian. For a suitable choice of λ, we have

$$f^*(y) = \min_x L(x, y, \lambda) = \min_x f(x, y) - \lambda g(x, y).$$

In this way, we remove the constraint from x, so

$$\begin{aligned} \nabla_y f^*(y) &= \nabla_y [\min_x f(x, y) - \lambda g(x, y)] \\ &= \nabla_y f(x, y)|_{x^*} - g(x^*, y) \nabla_y \lambda - \lambda \nabla_y g(x, y)|_{x^*}. \end{aligned} \tag{7}$$

Now we're going to simplify the second term $g(x^*, y)\nabla_y \lambda$. In the case $g(x^*, y) = 0$, $g(x^*, y)\nabla_y \lambda$ is definitely 0. In the case $g(x^*, y) \ne 0$, x^* is in the interior of the feasible set $\{x : g(x, y) \le 0\}$. By KKT condition, $\lambda = 0$. To analyze $\nabla_y \lambda$ in this case, we consider a neighbor $y + \Delta y$ of y. When $\|\Delta y\|$ is small enough, x^* is still in the feasible set $\{x : g(x, y + \Delta y) \le 0\}$ so λ does not change, which means $\nabla_y \lambda = 0$. In both cases, we have $g(x^*, y)\nabla_y \lambda = 0$. Therefore we can simplify (7):

$$\nabla_y f^*(y) = \nabla_y f(x, y)|_{x^*} - \lambda \nabla_y g(x, y)|_{x^*}. \tag{8}$$

Once again by KKT condition:

$$\nabla_x f(x, y)|_{x^*} - \lambda_y \nabla_x g(x, y)|_{x^*} = 0,$$

so

$$\lambda = \frac{\|\nabla_x f(x, y)|_{x^*}\|_2}{\|\nabla_x g(x, y)|_{x^*}\|_2}, \tag{9}$$

The last step is simply plugging (9) into (8). $\square$

B Reward Optimizing in CRABS

As in original SAC, we maintain two Q functions Q_{ψ_i} and their target networks $Q_{\bar{\psi}_i}$ for $i \in \{1, 2\}$, together with a learnable temperature α. The objective for the policy is to minimize

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s \sim \widehat{D}, a \sim \pi_\theta} \left[\alpha \log \pi_\theta^{\text{expl}}(a|s) - \hat{Q}_{\psi_1}(s, a) \right], \tag{10}$$

where $\hat{Q}_{\psi_1}(s, a) = Q_{\psi_1}(s, a)$ if $U(s, a, h) \le 0$, otherwise $\hat{Q}_{\psi_1}(s, a) = -C - U(s, a, h)$ for a large enough constant C. The heuristics behind the design of $\hat{Q}_{\psi_1}$ is that we should lower the probability of π_θ^{expl} proposing an action which will possibly leave the superlevel set $\mathcal{C}_{h_\phi}$ to reduce the frequency of invoking the safeguard policy during exploration.

The temporal difference objective for the Q function is

$$\mathcal{L}_Q(\psi_i) = \mathbb{E}_{(s,a,r,s') \sim \widehat{D}} \mathbb{E}_{a' \sim \pi_\theta^{\text{expl}}(s')} \left[(Q_{\psi_i}(s, a) - (r + \gamma \min_{i \in \{1,2\}} Q_{\bar{\psi}_i}(s, a)))^2 \mathbb{I}_{U(s', a', h_\phi) \le 0} \right], \tag{11}$$

We remark that we reject all $a' \sim \pi_\theta^{\text{expl}}(s')$ such that $U(s', a', h_\phi) > 0$, as our safe exploration algorithm (Algorithm 3) will reject all of them eventually. The temperature α is learned the same as in Haarnoja et al. [2018]:

$$\mathcal{L}_\alpha(\alpha) = \mathbb{E}_{s \sim \widehat{D}} [-\alpha \log \pi_\theta^{\text{expl}}(a|s) - \alpha \bar{\mathcal{H}}], \tag{12}$$

where $\bar{\mathcal{H}}$ is hyperparameter, indicating the target entropy of the policy π_θ^{expl}.

Algorithm 4 Modified SAC to train a policy while constraining it to stay within $\mathcal{C}_{h_\phi}$

input A policy π, the replay buffer $\widehat{D}$

1: Sample a batch $\mathcal{B}$ from buffer $\widehat{D}$.
2: Train θ to minimize $\mathcal{L}_\pi(\theta)$ using $\mathcal{B}$.
3: Train Q to minimize $\mathcal{L}_Q(\psi_i)$ for $i \in \{1, 2\}$ using $\mathcal{B}$.
4: Train α to minimize $\mathcal{L}_\alpha(\alpha)$ using $\mathcal{B}$.
5: Invoke MALA to training s^* adversarially (as in L4-5 in Algorithm 1).
6: Train θ minimize $C^*(h_\phi, U, \pi_\theta)$.
7: Update target network $\bar{\psi}_i$ for $i \in \{1, 2\}$.

C Experiment Details

Our code is implemented by Pytorch [Paszke et al., 2019] and runs in a single RTX-2080 GPU. Typically it takes 12 hours to run one seed for ***Upright***, ***Tilt*** and ***Move***, and for ***Swing*** it takes around 60 hours.

C.1 Environment

All the environments are based on OpenAI Gym [Brockman et al., 2016] where MuJoCo [Todorov et al., 2012] serves as the underlying physics engine. We use discount $\gamma = 0.99$.

The tasks ***Upright*** and ***Tilt*** are based on `Pendulum-v0`. The obsevation is $[\theta, \dot{\theta}]$ where θ is the angle between the pole and a vertical line, and $\dot{\theta}$ is the angular velocity. The agent can apply a torque to the pendulum. The task ***Move*** and ***Swing*** is based on `InvertedPendulum-v2` with observation $[x, \theta, \dot{x}, \dot{\theta}]$. The agent can control how the cart moves.

As all of the constraints are in the form of $\|\theta\| \le \theta_{\text{max}}$ and $|x| \le x_{\text{max}}$. For each type of constraint, we design $\mathcal{B}_{\text{safe}}$ to be

$$\mathcal{B}_{\text{safe}}(s) = \max\left(\omega\left(\theta/\theta_{\text{max}}\right), \omega\left(x/x_{\text{max}}\right)\right),$$

with $\omega(x) = \max(0, 100(|x| - 1))$. If there is no constraint of x, we just take $\mathcal{B}_{\text{safe}}(s) = \omega\left(\theta/\theta_{\text{max}}\right)$. One can easy check that $\mathcal{B}_{\text{safe}}(s)$ is continuous and equals to 1 at the boundary of safety set.

C.2 Hyperparameters

Policy We parametrize our policy using a feed-forward neural network with ReLU activation and two hidden layers, each of which contains 256 hidden units. Similar to Haarnoja et al. [2018], the output of the policy is squashed by a tanh function.

The initial policy is obtained by running SAC for 10^5 steps, checking the intermediate policy for every 10^4 steps and picking the first safe intermediate policy.

In all tasks, we optimize the policy for 2000 steps in a single epoch.

Dynamics Model We use an ensemble of five learned dynamics models as the calibrated dynamcis model. Each of the dynamics model contains 4 hidden layers with 400 hidden units and use Swish as the activation function [Ramachandran et al., 2017]. Following Chua et al. [2018], we also train learnable parameters to bound the output of σ_ω. We use Adam [Kingma and Ba, 2014] with learning rate 0.001, weight decay 0.000075 and batch size 256 to optimize the dynamics model.

In the experiment ***Move*** and ***Swing***, the initial model is obtained by traininng one a data for 20000 steps with 500 safe trajectories, obtained by adding different noises to the initial safe policy.

At each epoch, we optimize the dynamics models for 1000 steps.

Barrier certificate h The barrier certificate is parametrized by a feed-forward neural network with ReLU activation and two hidden layers, each of which contains 256 hidden units. The coefficient λ in Equation (5) is set to 0.001.

Collecting data. In ***Upright***, ***Tilt*** and ***Move***, the Line 3 in Algorithm 2 collects a single episode. In ***Swing***, the Line 3 collects six episodes, two of which are from Algorithm 3 with a uniform random policy, another two are from the current policy, and the remaining two are from the current policy but with more noises. In Algorithm 3, we first draw $n = 100$ Gaussian samples $\zeta_i \sim \mathcal{N}(0, I)$, and the sampled actions are $a_i = \tanh(\mu_\theta(s) + \zeta_i \sigma_\theta(s))$, where $\sigma_\theta(s)$ and $\mu_\theta(s)$ are the outputs of the exploration policy π^{expl}.

C.3 Baselines

RecoveryRL We use the code in `https://github.com/abalakrishna123/recovery-rl`. We remark that when running experiments in Recovery RL, we do not add the violation penalty for an unsafe trajectory. We set $\epsilon_{\text{risk}} = 0.5$ (chosen from $[0.1, 0.3, 0.7, 0.7]$) and discount factor $\gamma_{\text{risk}} = 0.6$ (chosen from $[0.8, 0.7, 0.6, 0.5]$). The offline dataset $\mathcal{D}_{\text{offline}}$, which is used to pretrain the Q^{π}_{risk}, contains 20K transitions from a random policy and another 20K transitions from the initial (safe) policy used by CRABS. The violations in the offline dataset is **not** counted when plotting.

Unfortunately, with chosen hyperparameters, we do not observe reasonable high reward from the policy, but we do observe that after around 400 episodes, RecoveryRL visits high reward (15-20) region in the ***Swing*** task and there are few violations since then.

SAC We implement SAC ourselves with learned temperature α, which we hypothesize is the reason of it superior performance over RecoveryRL and SQRL. The violation penalty is chosen to be 30 from $[3, 10, 30, 100]$ by tuning in the ***Swing*** and ***Move*** task. We found out that with violation penalty being 100, SAC has slightly fewer violations (around 167), but the total reward can be quite low (< 2) after 10^6 samples, so we choose to show the result of violation penalty being 30.

SQRL We use code provided by RecoveryRL with the same offline data and hyperparameters. However, we found out that the ν parameter (that is, the Lagrangian multiplier) is very important and tune it by choosing the optimal one from $[3, 10, 30, 100, 300]$ in ***Swing***. The optimal ν is the same as that for SAC, which is 30. As SQRL and RecoveryRL use a fixed temperature for SAC, we find it suboptimal in some cases, e.g., for ***Swing***.

CPO We use the code in `https://github.com/jachiam/cpo`. To make CPO more sample efficient and easier to compare, we reduce the batch size from 50000 to 5000 (for ***Move*** and ***Tilt***) or 1000 (for ***Tilt*** and ***Upright***). We tune the step size in $[0.02, 0.05, 0.005]$ but do not find substantial difference, while tuning the batch size can significantly reduce its sample efficiency, although it is still sample-inefficient.

D Metropolis-Adjusted Langevin Algorithm (MALA)

Given a probability density function p on $\mathbb{R}^d$, Metropolis-Adjusted Langevin Algorithm (MALA) obtains random samples $x \sim p$ when direct sampling is difficult. it is based on Metropolis-Hastings algorithm which generates a sequence of samples $\{x_t\}_t$. Metropolis-Hastings algorithm requires a *proposal distribution* $q(x'|x)$. At step $t \geq 0$, Metropolis-Hastings algorithm generates a new sample $\hat{x}_{t+1} \sim q(\cdot|x_t)$ and accept it with probability

$$\alpha(x \to x') = \min\left(1, \frac{p(x')q(x|x')}{p(x)q(x'|x)}\right).$$

If the sample $\hat{x}_{t+1}$ is accepted, we set $x_{t+1} = \hat{x}_{t+1}$; Otherwise the old sample x_t is used: $x_{t+1} = x_t$. MALA considers a special proposal function $q_\tau(x'|x) = \mathcal{N}(x + \tau \nabla p(x), 2\tau I_d)$. See Algorithm 5 for the pseudocode.

For our purpose, as we seek to compute $C^*(h_\phi, U, \pi_\theta)$, we maintain $m = 10^4$ sequences of samples $\{\{s_t^{(i)}\}_t\}_{i \in [m]}$. Recall that C^* involves a constrained optimization problem:

$$C^*(h_\phi, U, \pi_\theta) := \max_{s: h_\phi(s) \leq 1} U(s, \pi_\theta(s), h_\phi),$$

so for each $i \in [m]$, the sequence $\left\{s_t^{(i)}\right\}_t$ follows the Algorithm 5 to sample $s \sim \exp(\lambda_1 U(s, \pi_\theta(s), h_\phi) - \lambda_2 \mathbb{I}_{s \in \mathcal{C}_h})$ with $\lambda_1 = 30, \lambda_2 = 1000$. The step size τ is chosen such

Algorithm 5 Metropolis-Adjusted Langevin Algorithm (MALA)

Require: A probability density function p and a step size τ.

```
Initialize x_0 arbitrarily.
for t from 0 to ∞ do
    Draw ζ_t ~ N(0, I_d).
    Set x̂_{t+1} = x_t + τ∇ log p(X_t) + √(2τ) ζ_t.
    Draw u_t ~ Uniform[0, 1].
    if u_t ≥ α(x_t → x̂_{t+1}) then
        Set x_{t+1} = x̂_{t+1}.
    else
        Set x_{t+1} = x_t.
```

that the acceptance rate is approximately 0.6. In practice, when $s_t^{(i)} \notin \mathcal{C}_h$, we do not use MALA, but use gradient descent to project it back to the set $\mathcal{C}_h$.

E Limitations

- Our work relies on learning a calibrated dynamics model. However, as we pointed out in Section 4.3, it is often very difficult to learn a well-calibrated dynamics model. The fact that our algorithm of training a barreir certificate certifies the safety of a policy for infinition horizon requires a very well-calibraetd dynamics model. This limitation can be possibly reduced by leveraging domain knowledge.
- CRABS has a very slow training speed, mostly due to adversarial training and the use of an ensemble of dynamics model.
- CRABS did not find the optimal policy in the sense that the total reward is lower than some baselines (SAC).
- We do not guarantee that the Algorithm 1, which learns a barrier certificate for a policy w.r.t. a calibrated dynamics model, can always succeed, even if the policy is safe in the real environment.

F Negative Social Impact

Our algorithm aims to achieve zero safety violations, but we only tested our algorithm on simulated environments. So the algorithm cannot be applied to safety-critical real environments directly without further testing. If the algorithm is deployed without further testing, there might be undesirable consequences that have negative social impacts, e.g., the valuable devices might be broken, or when applied to health care, it might endanger the patients.