Constrained Markov Decision Processes via Backward Value Functions

Harsh Satija [1 2 3] **Philip Amortila** [1 2] **Joelle Pineau** [1 2 3]

Abstract

Although Reinforcement Learning (RL) algorithms have found tremendous success in simulated domains, they often cannot directly be applied to physical systems, especially in cases where there are hard constraints to satisfy (e.g. on safety or resources). In standard RL, the agent is incentivized to explore any behavior as long as it maximizes rewards, but in the real world, undesired behavior can damage either the system or the agent in a way that breaks the learning process itself. In this work, we model the problem of learning with constraints as a Constrained Markov Decision Process and provide a new on-policy formulation for solving it. A key contribution of our approach is to translate cumulative cost constraints into state-based constraints. Through this, we define a safe policy improvement method which maximizes returns while ensuring that the constraints are satisfied at every step. We provide theoretical guarantees under which the agent converges while ensuring safety over the course of training. We also highlight the computational advantages of this approach. The effectiveness of our approach is demonstrated on safe navigation tasks and in safety-constrained versions of MuJoCo environments, with deep neural networks.

1. Introduction

Reinforcement Learning (RL) provides a sound decision-theoretic framework to optimize the behavior of learning agents in an interactive setting (Sutton & Barto, 2018). Recently, the field of RL has found success in many high-dimensional domains, like video games, Go, robot locomotion and navigation. However, most of the success of RL algorithms has been limited to simulators, where the learning algorithm has the ability to reset the simulator. In the physical world, an agent will need to avoid harmful behavior (e.g. damaging the environment or the agent's hardware) while learning to explore behaviors that maximize the reward.

A few popular approaches for avoiding undesired behaviors for high-dimensional systems include reward-shaping (Moldovan & Abbeel, 2012), reachability-preserving algorithms (Mitchell, 2003; Eysenbach et al., 2017), state-level surrogate constraint satisfaction algorithms (Dalal et al., 2018), risk-sensitive algorithms (Tamar et al., 2013; Chow et al., 2015), apprenticeship learning (Abbeel & Ng, 2004) and high probability constraint satisfaction algorithms (Thomas et al., 2019). There also exists model-based Bayesian approaches that are focused on imposing the constraints via the dynamics (such as classifying parts of state space as unsafe) and then using model predictive control to incorporate the constraints in the policy optimization and planning (Turchetta et al., 2016; Berkenkamp et al., 2017; Wachi et al., 2018; Koller et al., 2018). An alternate way to model safety is via constraint satisfaction. A standard formulation for adding constraints to RL problems is the Constrained Markov Decision Process (CMDP) framework (Altman, 1999), wherein the environment is extended to also provide feedback on constraint costs. The agent must then attempt to maximize its expected cumulative rewards while also ensuring its expected cumulative constraint cost is less than or equal to some threshold.

A few algorithms have been proposed to solve CMDPs for high-dimensional domains with continuous action spaces - however they come with their own caveats. Reward Constrained Policy Optimization (Tessler et al., 2018) and Primal Dual Policy Optimization (Chow et al., 2015) do not provide any means to guarantee constraint satisfaction during the learning procedure, only on the final policy. Constrained Policy Optimization (Achiam et al., 2017) provides monotonic policy improvement but is computationally expensive due to requiring a backtracking line-search procedure and conjugate gradient algorithm for approximating the Fisher Information Matrix. Lyapunov-based Safe Policy Optimization (Chow et al., 2019) requires solving a Linear Program (LP) at every step of policy evaluation, although they show that there exists heuristics which can be substituted for the

Work partly done while HS was an intern at Facebook. [1]Department of Computer Science, McGill University, Montreal, Canada [2]Mila Québec AI Institute [3]Facebook AI Research, Montreal. Correspondence to: Harsh Satija <harsh.satija@mail.mcgill.ca>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

LP, at the expense of theoretical guarantees.

In this work, we propose an alternate formulation for solving CMDPs that transforms trajectory-level constraints into localized state-dependent constraints, through which a safe policy improvement step can be defined. In our approach, we define a notion of Backward Value Functions, which act as an estimator of the expected cost collected by the agent so far and can be learned via standard RL bootstrapping techniques. We provide conditions under which this new formulation is able to solve CMDPs without violating the constraints during the learning process. Our formulation allows us to define state-level constraints without explicitly solving a LP or its dual problem at every iteration. Our method is implemented as a reduction to any model-free on-policy bootstrap-based RL algorithm and can be applied for deterministic and stochastic policies and discrete and continuous action spaces. We provide empirical evidence of our approach with Deep RL methods on various safety benchmarks, including 2D navigation grid worlds (Leike et al., 2017; Chow et al., 2018), and MuJoCo tasks (Achiam et al., 2017; Chow et al., 2019).

2. Constrained Markov Decision Processes

We write $\mathscr{P}(\mathcal{S})$ for the set of probability distributions on a set $\mathcal{S}$. A Markov Decision Process (MDP) (Puterman, 2014) is a tuple $(\mathcal{X}, \mathcal{A}, \mathcal{P}, r, x_0)$, where $\mathcal{X}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P} : \mathcal{X} \times \mathcal{A} \to \mathscr{P}(\mathcal{X})$ is a transition probability function, $r : \mathcal{X} \times \mathcal{A} \to [0, \text{RMAX}]$ is a reward function, and x_0 is a starting state. For simplicity we assume a deterministic reward function and starting state, but our results generalize.

A Constrained Markov Decision Process (CMDP) (Altman, 1999) is an MDP with additional constraints which must be satisfied, thus restricting the set of permissible policies for the agent. Formally, a CMDP is a tuple $(\mathcal{X}, \mathcal{A}, \mathcal{P}, r, x_0, d, d_0)$, where $d : \mathcal{X} \to [0, \text{DMAX}]$ is the cost function and $d_0 \in \mathbb{R}^{\geq 0}$ is the maximum allowed cumulative cost. We note that the cost is only a function of states rather than state-action pairs. We consider a finite time horizon T after which the episode terminates. Then, the set of *feasible* policies that satisfy the CMDP is a subset of stationary policies:

$$\Pi_{\mathcal{D}} \coloneqq \Big\{\pi : \mathcal{X} \to \mathscr{P}(\mathcal{A}) \mid \mathbb{E}[\sum_{t=0}^{T} d(x_t) \mid x_0, \pi] \leq d_0\Big\}$$

The expected sum of rewards following a policy π from an initial state x is given by the value function $V^\pi(x) = \mathbb{E}[\sum_{t=0}^{T} r(x_t, a_t) \mid \pi, x_0 = x]$. Analogously, the expected sum of *costs* is given by the cost value function $V_{\mathcal{D}}^\pi(x) = \mathbb{E}[\sum_{t=0}^{T} d(x_t) \mid \pi, x_0 = x]$. The optimization problem in the CMDP is to find the feasible policy which maximizes expected returns from the initial state x_0, i.e.

$$\pi_{\mathcal{D}}^* = \arg\max_{\pi \in \Pi_{\mathcal{D}}} V^\pi(x_0)$$

An important point to note about CMDPs is that the cost function depends on immediate states but the constraint is cumulative and depends on the entire trajectory, rendering the optimization problem much harder.

In the case of MDPs, where a model of the environment is not known or is not easily obtained, it is still possible for the agent to find the optimal policy using Temporal Difference (TD) methods (Sutton, 1988). Broadly, these methods update the estimates of the value functions via bootstraps of previous estimates on sampled transitions (we refer the reader to Sutton & Barto (2018) for more information). In the on-policy setting, we alternate between estimating the state-action value function Q^π for a given π, and updating the policy to be greedy with respect to the value function.

3. Safe Policy Iteration via Backward Value Functions

Our approach proposes to convert the trajectory-level constraints of the CMDP into single-step state-wise constraints in such a way that satisfying the state-wise formulation will entail satisfying the original trajectory-level problem. The advantages of this approach are twofold: i) working with single-step state-wise constraints allows us to obtain analytical solutions to the optimization problem, and ii) the state-wise constraints can be defined via value-function-like quantities and can thus be estimated with well-studied value-based methods. The state-wise constraints we propose are defined via *Backward Value Functions* (cf. Section 3.2), which are value functions defined on a *Backward Markov Chain* (cf. Section 3.1). In Section 3.3 we provide a safe policy iteration procedure which satisfies said constraints (and thus the original problem).

3.1. Backward Markov Chain

Unlike in traditional RL, in the CMDP setting the agent needs to take into account the constraints which it has accumulated so far in order to plan accordingly for the future. Intuitively, the accumulated cost so far can be estimated via the cost value function $V_{\mathcal{D}}^\pi$ running "backward in time". To formally define the Backward Value Functions, we first need to introduce the concept of a Backward Markov Chain. Our approach is inspired by the work of Morimura et al. (2010), who also considered time-reversed Markov chains for the purpose of estimating the gradient of the log stationary distribution; we extend these ideas to TD methods. Recent work by Misra et al. (2019) also considers the use of backward dynamics for learning abstractions in Block MDPs.

Assumption 3.1 (Stationarity)**.** The MDP is ergodic for any policy π, i.e. the Markov chain characterized by the transition probability $\mathcal{P}^\pi(x_{t+1}|x_t) = \sum_{a_t \in \mathcal{A}} \mathcal{P}(x_{t+1}|x_t, a_t)\pi(a_t|x_t)$ is irreducible and aperiodic.

Let $\mathcal{M}(\pi)$ denote the Markov chain characterized by transition probability $\mathcal{P}^\pi(x_{t+1}|x_t)$. The above assumption implies that there exists a unique stationary distribution η^π satisfying:

$$\eta^\pi(x_{t+1}) = \sum_{x_t \in \mathcal{X}} \mathcal{P}^\pi(x_{t+1}|x_t)\eta^\pi(x_t). \tag{1}$$

Abusing notation, we also denote $\mathcal{P}^\pi(x_{t+1}, a_t|x_t) = \mathcal{P}(x_{t+1}|x_t, a_t)\pi(a_t|x_t)$.

We are interested in defining the *backward (or time-reversed)* probability $\overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}|x_t)$, which is the probability that a previous state-action pair (x_{t-1}, a_{t-1}) has led to the current state x_t. According to Bayes' rule, this probability is given by:

$$\overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}|x_t) = \frac{\mathcal{P}(x_t|x_{t-1}, a_{t-1}) \Pr(x_{t-1}, a_{t-1})}{\sum_{x', a' \in \mathcal{X}\times\mathcal{A}} \mathcal{P}(x_t|x', a') \Pr(x', a')}.$$

By Assumption 3.1, we have that:

$$\eta^\pi(x_{t-1}) = \frac{\Pr(x_{t-1}, a_{t-1})}{\pi(a_{t-1}|x_{t-1})},$$

and that:

$$\eta^\pi(x_t) = \sum_{x_{t-1}\in\mathcal{X}} \sum_{a_{t-1}\in\mathcal{A}} \mathcal{P}(x_t|x_{t-1}, a_{t-1}) \Pr(x_{t-1}, a_{t-1}).$$

Therefore, the backwards probability can be written as:

$$\overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}|x_t) = \frac{\mathcal{P}^\pi(x_t, a_{t-1}|x_{t-1})\eta^\pi(x_{t-1})}{\eta^\pi(x_t)}. \tag{2}$$

The forward Markov chain, characterized by the transition matrix $\mathcal{P}^\pi(x_{t+1}|x_t)$ runs forward in time, i.e., it gives the probability of the next state in which the agent will end up. Analogously, a backward Markov chain is denoted by the transition matrix $\overleftarrow{\mathcal{P}}^\pi(x_{t-1}|x_t) = \sum_{a_{t-1}\in\mathcal{A}} \overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}|x_t)$, and describes the state-action pair which the agent took to reach the current state.

Definition 3.1 (Backward Markov Chain)**.** A backward Markov chain associated with $\mathcal{M}(\pi)$ is denoted by $\overleftarrow{\mathcal{B}}(\pi)$ and is characterized by the transition probability $\overleftarrow{\mathcal{P}}^\pi(x_{t-1}|x_t)$.

3.2. Backward Value Function

We define the *Backward Value Function* (BVF) to be a value function running on the backward Markov chain $\overleftarrow{\mathcal{B}}(\pi)$. A BVF is the expected sum of returns or costs collected by the agent *so far*. We are mainly interested in the BVF for the cost function in order to maintain estimates of the cumulative cost incurred at a state.

We note that, since every Markov chain $\mathcal{M}(\pi)$ is ergodic by Assumption 3.1, the corresponding backward Markov chain $\overleftarrow{\mathcal{B}}(\pi)$ is also ergodic (Morimura et al., 2010, Prop. B.1). In particular, every policy π can reach the initial state via some path in the transition graph of the backward Markov chain. Thus, the backwards Markov chain is also finite-horizon for some $T_{\mathcal{B}}$, with x_0 corresponding to the terminal state. We define a finite-horizon Backward Value Function for cost as:

$$\overleftarrow{V}^\pi_{\mathcal{D}}(x) = \mathbb{E}_{\overleftarrow{\mathcal{B}}(\pi)}\left[\sum_{k=0}^{T_{\mathcal{B}}} d(x_{t-k})|x_t = x\right], \tag{3}$$

where $\mathbb{E}_{\overleftarrow{\mathcal{B}}(\pi)}[\cdot]$ denotes the expectation over the backwards chain. By contrast, we will write $\mathbb{E}_{\mathcal{M}(\pi)}[\cdot]$ for expectations over the forward chain.

Proposition 3.1 (Sampling)**.** Samples from the forward Markov chain $\mathcal{M}(\pi)$ can be used directly to estimate the statistics of the backward Markov chain $\overleftarrow{\mathcal{B}}(\pi)$. In particular, for any $K \in \mathbb{N}$:

$$\mathbb{E}_{\overleftarrow{\mathcal{B}}(\pi)}\left[\sum_{k=0}^{K} d(x_{t-k})|x_t\right] = \mathbb{E}_{\mathcal{M}(\pi), x_{t-K}\sim\eta^\pi(\cdot)}\left[\sum_{k=0}^{K} d(x_{t-k})|x_t\right]. \tag{4}$$

Furthermore, equation (4) holds true even in the limit $K \to \infty$.

The proof is given in Appendix B.1. Using the above proposition, we get an interchangeability property that removes the need to sample from the backward chain. We can use the traditional RL setting and draw samples from the forward chain to estimate the BVFs. Equation (3) can be written recursively as:

$$\overleftarrow{V}^\pi_{\mathcal{D}}(x_t) = \mathbb{E}_{\overleftarrow{\mathcal{B}}(\pi)}\left[d(x_t) + \overleftarrow{V}^\pi_{\mathcal{D}}(x_{t-1})\right]. \tag{5}$$

As in the forward case, we can define a backwards Bellman operator $\overleftarrow{\mathcal{T}}^\pi_{\mathcal{D}}$:

$$(\overleftarrow{\mathcal{T}}^\pi_{\mathcal{D}}\overleftarrow{V}^\pi_{\mathcal{D}})(x_t) = \mathbb{E}_{x_{t-1}\sim\overleftarrow{\mathcal{P}}^\pi}\left[d(x_t) + \overleftarrow{V}^\pi_{\mathcal{D}}(x_{t-1})\right]. \tag{6}$$

The BVF for a policy π then corresponds to the fixed point equation

$$\overleftarrow{V}^\pi_{\mathcal{D}}(x_t) = \overleftarrow{\mathcal{T}}^\pi_{\mathcal{D}}\overleftarrow{V}^\pi_{\mathcal{D}}(x_t).$$

Proposition 3.2 (Fixed point)**.** For any policy π, the associated BVF $\overleftarrow{V}_{\mathcal{D}}^{\pi}$ is the unique fixed point of $\overleftarrow{V}_{\mathcal{D}}^{\pi} = \overleftarrow{\mathcal{T}}_{\mathcal{D}}^{\pi}\overleftarrow{V}_{\mathcal{D}}^{\pi}$. Furthermore, for any V_0 we have $\lim_{k\to\infty}\left(\overleftarrow{\mathcal{T}}^{\pi}\right)^k V_0 = \overleftarrow{V}^{\pi}$.

The proof is given in Appendix B.2. The above proposition allows us to soundly extend the RL methods based on Bellman operators towards the estimation of BVFs.

3.3. Safe Policy Improvement via BVF-based constraints

With the BVF framework, the trajectory-level optimization problem associated with a CMDP can be rewritten in state-wise form. Recall that a feasible policy must satisfy the constraint:

$$\mathbb{E}_{\mathcal{M}(\pi)}\left[\sum_{k=0}^{T} d(x_k) \mid x_0\right] \leq d_0.$$

Alternatively, for each timestep $t \in [0, T]$ of a trajectory:

$$\mathbb{E}\left[\sum_{k=0}^{t} d(x_k) \mid x_0, \pi\right] + \mathbb{E}\left[\sum_{k=t}^{T} d(x_k) \mid x_0, \pi\right] - \mathbb{E}\left[d(x_t) \mid x_0\right] \leq d_0. \tag{7}$$

The first and second terms can be estimated by the backward and forward value functions, respectively. The following proposition makes this formal, and relates the trajectory-level constraint of Equation (7) to a BVF formulation which only depends on the current state x_t.

Proposition 3.3. We have that:

$$\mathbb{E}_{\mathcal{M}(\pi)}\left[\sum_{k=0}^{T} d(x_k) \mid x_0\right] \leq \mathbb{E}_{x_t\sim\eta^{\pi}(\cdot)}\left[\overleftarrow{V}_{\mathcal{D}}^{\pi}(x_t) + V_{\mathcal{D}}^{\pi}(x_t) - d(x_t)\right]. \tag{8}$$

The proof is given in Appendix B.3. Thus, the constraint in the CMDP problem is less than the expected sum of the backwards and forwards value functions, when sampling a state from the stationary distribution.

These are the state-wise constraints that should hold at each step in a given trajectory - we refer to them as the *value-based constraints*. Satisfying the value-based constraints will also ensure that the given CMDP constraints are met, since the value-based formulation is an upper bound to the original CMDP problem.

This formulation allows us to introduce a policy improvement step, which maintains a safe feasible policy at every iteration by using the previous estimates of the forward and backward value functions. The policy improvement step is defined by a linear program, which performs a greedy update with respect to the current state-action value function subject to the value-based constraints:

$$\pi_{k+1}(\cdot|x) = \arg\max_{\pi\in\Pi} \langle \pi(\cdot|x), Q^{\pi_k}(x,\cdot)\rangle, \tag{SPI}$$
$$\texttt{s.t.}\ \langle \pi(\cdot|x), Q_{\mathcal{D}}^{\pi_k}(x,\cdot)\rangle + \overleftarrow{V}_{\mathcal{D}}^{\pi_k}(x) - d(x) \leq d_0, \forall x \in \mathcal{X}.$$

We show that the policies obtained by the policy improvement step will satisfy the safety constraints, as long as the policy updates are constrained to a small enough neighbourhood. For $p, q \in \mathscr{P}(\mathcal{S})$, we write $\mathrm{TV}(p, q) = \frac{1}{2}\sum_{s\in\mathcal{S}} |p(s) - q(s)|$ for their total variation distance.

Theorem 3.1 (Consistent Feasibility)**.** Assume that successive policies are updated sufficiently slowly, i.e. $\mathrm{TV}(\pi_{k+1}(\cdot|x), \pi_k(\cdot|x)) \leq \frac{d_0 - V_{\mathcal{D}}^{\pi_k}(x_0)}{2\mathrm{D}\textsc{max}T^2}$. Then the policy iteration step given by (SPI) is consistently feasible, i.e. if π_k is feasible at x_0 then so is π_{k+1}.

The assumption on the policy updates can be enforced, for example, by constraining the iterates to the prescribed neighbourhood. It is also possible to consider larger neighbourhoods for the policy updates, but at the cost of everywhere-feasibility – we present that result in Appendix C.

Next we show that the policy iteration step given by (SPI) leads to monotonic improvement.

Theorem 3.2 (Policy Improvement)**.** Let π_n and π_{n+1} be successive policies generated by the policy iteration step of (SPI). Then $V^{\pi_{n+1}}(x) \geq V^{\pi_n}(x)\ \forall x \in \mathcal{X}$. In particular, the sequence of value functions $(V^{\pi_n})_{n\geq 0}$ given by (SPI) monotonically converges.

Proofs for Theorems 3.1 and 3.2 are given in Appendix C. We note that the value-based constraints are in general a stronger constraint than the original CMDP formulation. Therefore, while Theorem 3.2 guarantees convergence, the final policy which is obtained from the (SPI) procedure may be suboptimal when compared to the optimal feasible policy $\pi_{\mathcal{D}}^*$. Finding the sub-optimality gap (if any) remains an interesting question for future work.

4. Practical Implementation Considerations

4.1. Discrete Action Space

In discrete action spaces, the updates (SPI) can be solved exactly as a Linear Programming problem. It is possible to approximate its analytical solution by casting it into the corresponding entropy-regularized counterpart (Neu et al., 2017; Chow et al., 2018). The details of the closed form solution can be found in Appendix D.

Furthermore, if we restrict the set of policies to be deterministic, then it is possible to have an in-graph solution as well. The procedure then closely resembles the Action Elimination Procedure (Puterman, 2014, Chapter 6), where non-optimal actions are identified as being those which violate the constraints.

4.2. Extension to continuous control

For MDPs with only state-dependent costs, Dalal et al. (2018) proposed the use of safety layers, a constraint projection approach, that enables action correction at each step. At any given state, an unconstrained action is selected and is passed to the safety layer, which projects the action to the nearest action (in Euclidean norm) satisfying the necessary constraints. We extend this approach to stochastic policies to handle the corrections for the actions generated by stochastic policies. When the policy is parameterized with a Gaussian distribution, then the safety layer can still be used by projecting both the mean and standard-deviation vectors to the constraint-satisfying hyper-plane. A proof of this claim is given in Appendix E. In most cases, the standard-deviation vector is kept fixed or independent of the state (Kostrikov, 2018; Dhariwal et al., 2017), which allows us to formulate the problem as solving the following $\mathscr{L}_2$-projection of the mean of the Gaussian in Euclidean space. For $\mu_\pi(\cdot;\theta)$, at any given state $x \in \mathcal{X}$, the safety layer solves the following projection problem:

$$\begin{aligned}&\arg\min_{\mu} \left[\frac{1}{2}\left\|\mu - \mu_\pi(x)\right\|^2\right],\\ &\texttt{s.t.}\quad Q^\pi_{\mathcal{D}}(x,\mu) + \overleftarrow{V}^\pi_{\mathcal{D}}(x) - d(x) \le d_0.\end{aligned}$$

As previously shown, if the constraints have linear nature then an analytical solution exists (Dalal et al., 2018; Chow et al., 2019). In order to get a linearized version of the constraints (and simplify the projection), we can approximate the constraint with its first-order Taylor series at $\mu = \mu_\pi(x)$:

$$\begin{aligned}&\arg\min_{\mu} \left[\frac{1}{2}\left\|\mu - \mu_\pi(x)\right\|^2\right], \qquad (9)\\ &\texttt{s.t.}\quad \overleftarrow{V}^\pi_{\mathcal{D}}(x) - d(x) + Q^\pi_{\mathcal{D}}(x,\mu_\pi(x))\\ &\quad + (\mu - \mu_\pi(x))^T(\nabla Q^\pi_{\mathcal{D}}(x,\mu)|_{\mu=\mu_\pi(x)}) \le d_0.\end{aligned}$$

The above objective function is positive-definite and quadratic, and the constraints are linear. Though this problem can be solved by an in-graph QP solver, we derive an analytical solution:

Proposition 4.1. For any state $x \in \mathcal{X}$, let $g_{\mu,\mathcal{D}}(x) = \nabla Q^\pi_{\mathcal{D}}(x,\mu)|_{\mu=\mu_\pi(x)}$ and

$$\lambda^*(x) = \left(\frac{-(d_0 + d(x) - \overleftarrow{V}^\pi_{\mathcal{D}}(x) - Q^\pi_{\mathcal{D}}(x,\mu_\pi(x)))}{g_{\mu,\mathcal{D}}(x)^T g_{\mu,\mathcal{D}}(x)}\right)^+.$$

Then, the solution to Equation (9) is given by:

$$\mu^*(x) = \mu_\pi(x) - \lambda^*(x)\cdot g_{\mu,D}(x).$$

The derivation of the previous solution is given in Appendix F.

5. Related Work

Lagrangian-based methods: Initially introduced in Altman (1999), more scalable versions of the Lagrangian based methods have been proposed over the years (Moldovan & Abbeel, 2012; Tessler et al., 2018; Chow et al., 2015). The general form of the Lagrangian methods is to convert the problem to an unconstrained problem via Langrange multipliers. If the policy parameters are denoted by θ, then Lagrangian formulation becomes:

$$\begin{aligned}&\min_{\lambda\ge 0}\max_{\theta} L(\theta,\lambda)\\ &\quad = \min_{\lambda\ge 0}\max_{\theta}\left[V^{\pi_\theta}(x_0) - \lambda(V^{\pi_\theta}_{\mathcal{D}}(x_0) - d_0)\right],\end{aligned}$$

where L is the Lagrangian and λ is the Lagrange multiplier (penalty coefficient). The main problems of the Lagrangian methods are that the Lagrangian multiplier is either a hyper-parameter (without much intuition), or is solved on a lower time-scale. That makes the unconstrained RL problem a three time-scale problem, which makes it difficult to optimize in practice.[1] Another problem is that during the optimization, this procedure can violate the constraints. Ideally, we want a method that can respect the constraint throughout the training and not just at the final optimal policy.

Lyapunov-based methods: In control theory, the stability of the system under a fixed policy is computed using Lyapunov functions (Khalil, 1996). A Lyapunov function is a type of scalar potential function that keeps track of the energy that a system continually dissipates. Recently, Chow et al. (2018; 2019) provide a method of constructing the Lyapunov functions to guarantee global safety of a behavior policy using a set of local linear constraints. Their method requires the knowledge of closeness of the baseline policy and the optimal policy $\mathrm{TV}(\pi_0,\pi^*)$ to guarantee the theoretical claims. They further substitute the Lyapunov function with an approximate solution that requires solving a LP problem at every iteration. For the practical scalable versions, they use a heuristic constant Lyapunov function for all states that only depends on the initial state and the horizon. While our method also constructs state-wise constraints, there are two notable differences. Firstly, our theoretical results only require successive policies to be close (i.e. $\mathrm{TV}(\pi_k,\pi_{k+1})$) rather than $\mathrm{TV}(\pi_0,\pi^*)$. There are instances where this property is more desirable than the one in Chow et al. (2019), for instance in the case when the initial baseline policy is random and there is no knowledge of the optimal policy. Another difference is that our method does

[1]Classic Actor Critic is two time-scale (Konda & Tsitsiklis, 2000), and adding a learning schedule over the Lagrangian makes it three time scale.

Algorithm 1 A2C with Safety Layer - for each actor thread

```
1:  Input: $\pi(\cdot\,;\theta), V(\cdot\,;\phi), Q_{\mathcal{D}}(\cdot,\cdot\,;\theta_{\mathcal{D}}), \overleftarrow{V}_{\mathcal{D}}(\cdot\,;\phi_{\mathcal{D}}), n$
2:  for episode $e \in 1, ..., M$ do
3:    Get initial state $\{x_0\}$ ; $t \leftarrow 1$
4:    while $t < T$ do
5:      $t_{start} \leftarrow t$
6:      while $t < t_{start} + n$ or $t == T$ do
7:        Select $a_t$ using sampling from the projected mean $\mu_t$
          via the safety layer Eq.(9), execute $a_t$, observe $x_{t+1}$
          and reward $r_t$ and cost $d_t$.
8:        $t \leftarrow t + 1$
9:      end while
10:     Calculate the targets for $x_{t+1}$ using the current estimates
        for bootstrap:

        $R \leftarrow$ if $t == T$ then $0$ else $V(x_{t+1}, a_{t+1}; \phi)$ end if
        $R_{\mathcal{D}} \leftarrow$ if $t == T$ then $0$ else $Q_{\mathcal{D}}(x_{t+1}, \mu_{t+1}; \theta_{\mathcal{D}})$ end if
        $\overleftarrow{R}_{\mathcal{D}} \leftarrow$ if $t == 1$ then $0$ else $\overleftarrow{V}_{\mathcal{D}}(x_{t_{start}-1}; \phi_{\mathcal{D}})$ end if

11:     for $i \in \{t-1, \ldots, t_{start}\}$ do
12:       $R \leftarrow r_i + \gamma R$
13:       $R_{\mathcal{D}} \leftarrow d_i + \gamma R_{\mathcal{D}}$
14:       Accumulate the gradients w.r.t. $\theta, \phi, \theta_{\mathcal{D}}$:

            $d\phi \leftarrow d\phi + \partial(R - V(x_i \phi))^2 / \partial\phi$
            $d\theta_{\mathcal{D}} \leftarrow d\theta_{\mathcal{D}} + \partial(R_{\mathcal{D}} - Q_{\mathcal{D}}(x_i, \mu_i; \theta_{\mathcal{D}}))^2 / \partial\theta_{\mathcal{D}}$

15:       if the policy is in feasible space then
16:         Update the policy:

              $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i \mid x_i; \theta)(R - V(x_i; \phi))$

17:       else
18:         Use safeguard policy update to recover:

              $d\theta \leftarrow d\theta - \nabla_\theta \log \pi(a_i \mid x_i; \theta)(R_{\mathcal{D}})$

19:       end if
20:     end for
        {Update the backward estimator here}
21:     for $i \in \{t_{start}, \ldots, t\}$ do
22:       $\overleftarrow{R}_{\mathcal{D}} \leftarrow d_i + \gamma \overleftarrow{R}_{\mathcal{D}}$
23:       Accumulate the gradients w.r.t. $\phi_{\mathcal{D}}$:

            $d\phi_{\mathcal{D}} \leftarrow d\phi_{\mathcal{D}} + \partial(\overleftarrow{R}_{\mathcal{D}} - \overleftarrow{V}_{\mathcal{D}}(x_i; \phi_{\mathcal{D}}))^2 / \partial\phi_{\mathcal{D}}$

24:     end for
25:     Do synchronous batch update with the accumulated gra-
        dients to update $\theta, \phi, \theta_{\mathcal{D}}, \phi_{\mathcal{D}}$. using $d\theta, d\phi, d\theta_{\mathcal{D}}, d\phi_{\mathcal{D}}$.
26:   end while
27: end for
```

not require solving an LP at every update step to construct the constraint and as such the only approximation error that is introduced comes from the function approximation.

Conservative Policy Improvement: Constrained Policy Optimization (CPO) (Achiam et al., 2017) extends the Trust-Region Policy Optimization (TRPO) (Schulman et al., 2015) algorithm to satisfy constraints during training as well as after convergence. The slow update assumption in Theorem. 3.1 can be integrated with the conservative policy improvement framework (Kakade & Langford, 2002) to get a formulation similar to CPO. Solving such a formulation requires the knowledge of the exact Fisher Information Matrix (FIM) to claim any guarantees. For empirical purposes, as the FIM is not computationally tractable, an approximate estimate is used. Instead, our approach presents an alternate way to work with the analytical solutions of the state-dependent constraints.

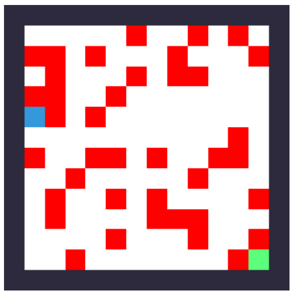
The CPO algorithm uses an approximation to the FIM which requires many steps of conjugate gradient descent (n_{cg} steps) followed by a backtracking line-search procedure (n_{ls} steps) for each iteration, so it is more expensive by $\mathcal{O}(n_{cg} + n_{ls})$ per update. Furthermore, accurately estimating the curvature requires a large number of samples in each batch (Wu et al., 2017).

6. Experiments

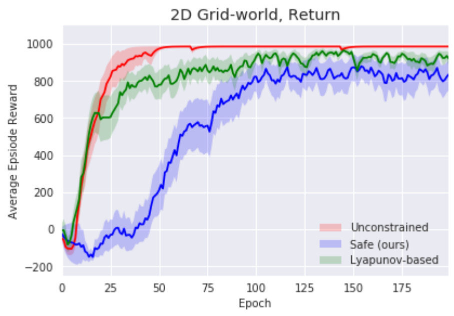
We empirically validate our approach on RL benchmarks to measure the performance of the agent with respect to the accumulated return and cost during training. We use neural networks as a function approximator. For each benchmark, we compare our results with the Lyapunov-based approach (Chow et al., 2018; 2019) and an unconstrained (unsafe) algorithm. Even though our formulation is based on the undiscounted case, we use discounting with $\gamma = 0.99$ for estimating the value functions in order to be consistent with the baselines. We acknowledge that there is a gap between the stationary assumption (Assumption 3.1) we build our work on, and in practice how the algorithms are implemented. The initial starting policy in our experiments is a random policy, and due to that we adopt a safe-guard (or recovery) policy update in the same manner as (Achiam et al., 2017; Chow et al., 2019), where if the agent ends up being in an infeasible policy space, we recover by an update that purely minimizes the constraint value.

6.1. Stochastic Grid World

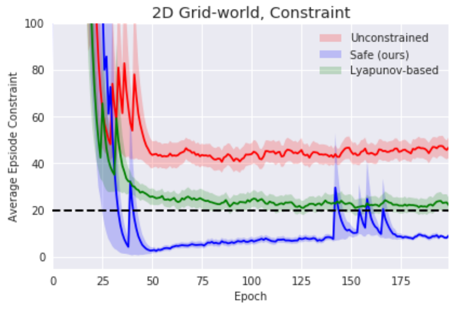
Motivated by the safety in navigation tasks, we first consider a stochastic 2D grid world (Leike et al., 2017; Chow et al., 2018). The agent (green cell in Fig. 1a) starts in the bottom-right corner, the safe region, and the objective is to move to the goal on the other side of the grid (blue cell). The



(a) 2D GridWorld



(b) Returns



(c) Constraints

Figure 1. (a) Example of the 2D Gridworld environment. (b,c) Performance over the training for Unconstrained (red), Lyapunov-based (green), and our method (blue) all trained with n-step SARSA on GridWorld task over 20 random seeds. The x-axis is the number of episodes in thousands. The dotted black line denotes the constraint threshold d_0. The bold line represents the mean and the shaded region denotes 80% confidence-intervals.

agent can only move in the adjoining cells in the cardinal directions. It gets a reward of $+1000$ on reaching the goal, and a penalty of -1 at every timestep. Thus, the task is to reach the goal in the shortest amount of time. There are a number of pits in the terrain (red cells) that represent the safety constraint and the agent gets a cost of 10 on passing through any pit cell. Occasionally, with probability $p = 0.05$, a random action will be executed instead of the one selected by the agent. Thus, the task is to reach to the goal in the shortest amount of time, while passing through the red grids at most $d_0/10$ times. The size of the grid is 12×12 cells, and the pits are randomly generated for each grid with probability $\rho = 0.3$. The agent starts at $(12, 12)$ and the goal is selected uniformly on $(\alpha, 0)$, where $\alpha \sim U(0, 12)$. The threshold $d_0 = 20$ implies the agent can pass at most two pits. The maximum horizon is 200 steps, after which the episode terminates.

We use the action elimination procedure described in Sec 4.1 in combination with n-step SARSA (Rummery & Niranjan, 1994; Peng & Williams, 1994) using neural networks and multiple synchronous agents as in (Mnih et al., 2016). We use ϵ-greedy exploration. The results are shown in Fig. 1. More experimental details can be found in Appendix G. We observe that the agent is able to respect the safety constraints more adequately than the Lyapunov-based method, albeit at the expense of some decrease in return, which is the expected trade-off for satisfying the constraints.

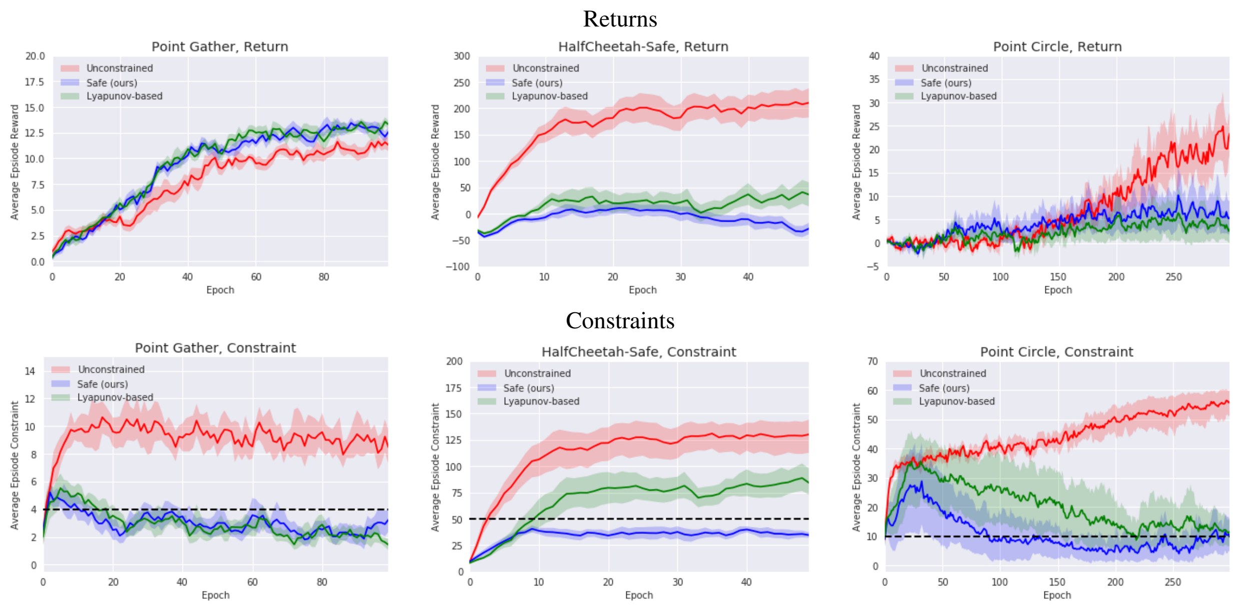
6.2. MuJoCo Benchmarks

Based on the safety experiments in Achiam et al. (2017); Chow et al. (2019), we design three simulated robot locomotion continuous control tasks using the MuJoCo simulator (Todorov et al., 2012) and OpenAI Gym (Brockman et al., 2016): (1) **Point-Gather**: A point-mass agent ($S \subseteq \mathbb{R}^9, A \subseteq \mathbb{R}^2$) is rewarded for collecting the green apples and constrained to avoid the red bombs; (2) **Safe-Cheetah**: A bi-pedal agent ($S \subseteq \mathbb{R}^{18}, A \subseteq \mathbb{R}^6$) is rewarded for running at high speed, but at the same time constrained by a speed limit; (3) **Point-Circle**: The point-mass agent ($S \subseteq \mathbb{R}^9, A \subseteq \mathbb{R}^2$) is rewarded for running along the circumference of a circle in counter-clockwise direction, but is constrained to stay within a safe region smaller than the radius of the circle.

We integrate our method on top of the A2C algorithms (Mnih et al., 2016) and PPO (Schulman et al., 2017), using the procedure described in Section 4.2. More details about the tasks and network architecture can be found in the Appendix H. Algorithmic details can be found in Algorithm 1 and in Appendix I. We discuss an alternate update for implementing the algorithms using target networks in Appendix J. The results with A2C are shown in Fig. 2 and the results with PPO are shown in Fig. 3. We observe that our Safe method is able to respect the safety constraint throughout most of the learning, and with much greater degree of compliance than the Lyapunov-based method, especially when combined with A2C. The one case where the Safe method fails to respect the constraint is in Point-Circle with PPO (Fig. 3(c)). Upon further examination, we note that the training in this scenario has one of two outcomes: some runs end with the learner in an infeasible set of states from which it cannot recover; other runs end in a good policy that respects the constraint. We discuss possible solutions to overcome this in the following section.

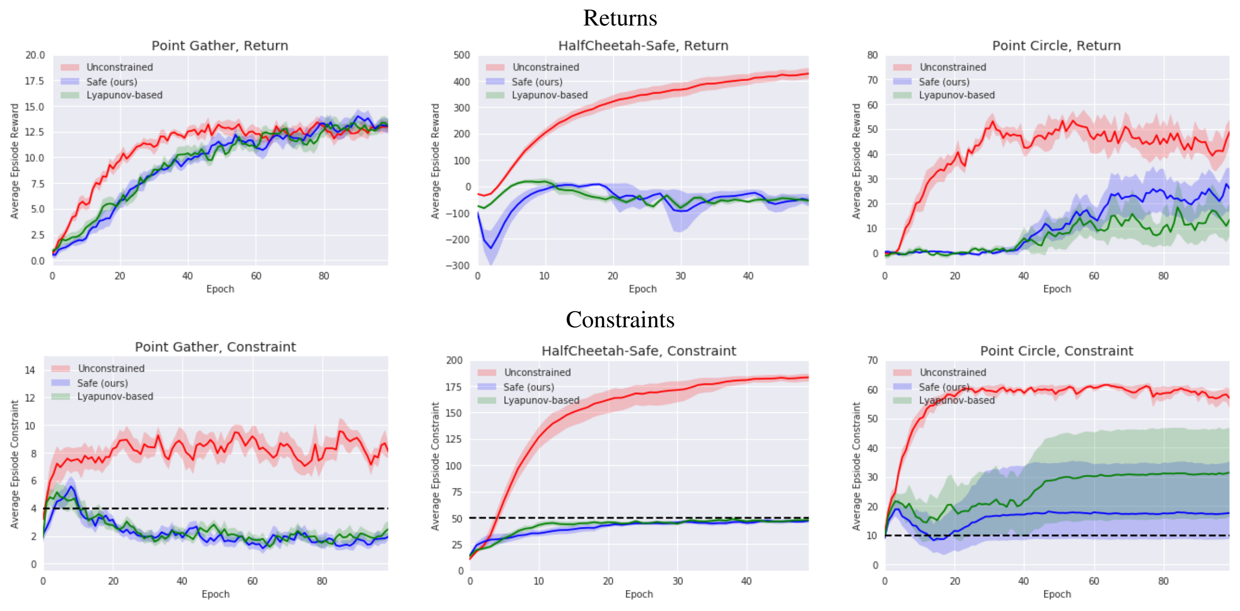
7. Discussion

We present a method for solving constrained MDPs that respects trajectory-level constraints by converting them into state dependent value-based constraints, and show how the method can be used to handle safety limitations in both discrete and continuous spaces. The main advantage of our approach is that the optimization problem is more easily



(a) Point-Gather (b) Safe-Cheetah (c) Point-Circle

Figure 2. A2C Performance over the training for Unconstrained (red), Lyapunov-based (green), and our method (blue) all trained with A2C on MuJoCo tasks over 10 random seeds. The x-axis is the number of episodes in thousands. The dotted black line denotes d_0. The bold line represents the mean, and the shaded region denotes the 80% confidence-intervals.



(a) Point-Gather (b) Safe-Cheetah (c) Point-Circle

Figure 3. PPO Performance over the training for Unconstrained (red), Lyapunov-based (green), and our method (blue) all trained with PPO on MuJoCo tasks over 10 random seeds. The x-axis is the number of episodes in thousands, and y-axis denotes the undiscounted accumulated returns. The dotted black line denotes d_0. The bold line represents the mean, and the shaded region denotes the 80% confidence-intervals.

solved with value-based constraints, while providing similar guarantees and requiring less approximations. Using backward value function allows us to satisfy the constraint in expectation and makes the system Markovian, which in turn allows us to derive analytical solutions to the state-dependent local constraints. This also enables us to use the Temporal Difference based bootstrapping techniques that allow us to fit the final algorithm in the Actor-Critic family of methods.

The empirical results presented show that our approach is able to solve the tasks with good performance while maintaining safety throughout training. It is important to note that there is a fundamental trade-off between exploration and safety. It is impossible to be 100% safe without some knowledge; in cases where that knowledge is not provided a priori, it must be acquired through exploration. We see this in some of our results (Gridworld, Point-Circle) where our safe policy goes above the constraint in the very early phases of training (all our experiments started from a random policy). We note that the other methods also suffer from this shortcoming. An open question is how to provide initial conditions or a priori knowledge to avoid this burn-in phase. Another complementary strategy to explore is to design better recovery methods to prevent an agent to get stuck in an unsafe or infeasible policy space. On the theoretical side, an important next step would be to quantify the sub-optimality induced by the constraints imposed in our safe policy iteration method.

Acknowledgements

The authors would like to thank NSERC (Natural Sciences and Engineering Research Council) for funding this research. We are thankful to Ahmed Touati, Joshua Romoff, Maxime Wabartha and Emmanuel Bengio for many helpful discussions about the work, and the anonymous reviewers for providing constructive feedback.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017.

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

Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pp. 908–919, 2017.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Chow, Y., Tamar, A., Mannor, S., and Pavone, M. Risk-sensitive and robust decision-making: a cvar optimization approach. In *Advances in Neural Information Processing Systems*, pp. 1522–1530, 2015.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.

Chow, Y., Nachum, O., Faust, A., Ghavamzadeh, M., and Duenez-Guzman, E. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

Chow, Y., Nachum, O., Faust, A., Duenez-Guzman, E., and Ghavamzadeh, M. Safe policy learning for continuous control, 2020. URL https://openreview.net/forum?id=HkxeThNFPH.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. https://github.com/openai/baselines, 2017.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Eysenbach, B., Gu, S., Ibarz, J., and Levine, S. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp. 267–274, 2002.

Khalil, H. K. Nonlinear systems. 1996.

Koller, T., Berkenkamp, F., Turchetta, M., and Krause, A. Learning-based model predictive control for safe exploration and reinforcement learning. *arXiv preprint arXiv:1803.08287*, 2018.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

Misra, D., Henaff, M., Krishnamurthy, A., and Langford, J. Kinematic state abstraction and provably efficient rich-observation reinforcement learning. *arXiv preprint arXiv:1911.05815*, 2019.

Mitchell, I. M. Application of level set methods to control and reachability problems in continuous and hybrid systems. 2003.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Moldovan, T. M. and Abbeel, P. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

Morimura, T., Uchibe, E., Yoshimoto, J., Peters, J., and Doya, K. Derivatives of logarithmic stationary distributions for policy gradient reinforcement learning. *Neural Computation*, 22(2):342–376, 2010. doi: 10.1162/neco.2009.12-08-922. URL https://doi.org/10.1162/neco.2009.12-08-922. PMID: 19842990.

Neu, G., Jonsson, A., and Gómez, V. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

Peng, J. and Williams, R. J. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pp. 226–232. Elsevier, 1994.

Pineau, J. The machine learning reproducibility checklist. https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf, 2018.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Rummery, G. A. and Niranjan, M. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tamar, A., Di Castro, D., and Mannor, S. Policy evaluation with variance related risk criteria in markov decision processes. *arXiv preprint arXiv:1301.0104*, 2013.

Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

Thomas, P. S., da Silva, B. C., Barto, A. G., Giguere, S., Brun, Y., and Brunskill, E. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Turchetta, M., Berkenkamp, F., and Krause, A. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems*, pp. 4312–4320, 2016.

Wachi, A., Sui, Y., Yue, Y., and Ono, M. Safe exploration and optimization of constrained mdps using gaussian processes. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288, 2017.

A. Reproducibility Checklist

We follow the reproducibility checklist (Pineau, 2018) and point to relevant sections explaining them here. For all algorithms presented, check if you include:

- **A clear description of the algorithm.** The algorithms are explained in Sec. I. Any additional details for Discrete methods are provided in Sec. 4.1, and for continuous Sec. 4.2.
- **An analysis of the complexity (time, space, sample size) of the algorithm.** As we build on the same methodology as the baseline safe-method (Lyapunov), the complexity is same as the baseline, but is higher than the unconstrained versions. In terms of computation time (for Deep-RL experiments) the newly proposed algorithms are almost identical to the safe baselines due to its parallelizable nature. We do not make any claims about the sample complexity.
- **A link to a downloadable source code, including all dependencies.** The code can be found at `https://github.com/hercky/cmdps_via_bvf`.

For any theoretical claim, check if you include:

- **A statement of the result.** See the main paper for all the claims we make. Additional details are provided in the Appendix.
- **A clear explanation of any assumptions.** See the main paper for all the assumptions.
- **A complete proof of the claim.** See the main paper. The cross-references to the proofs in the Appendix have been included in the main paper.

For all figures and tables that present empirical results, check if you include:

- **A complete description of the data collection process, including sample size.** For the base agent we standard benchmarks provided in OpenAI Gym (Brockman et al., 2016), and rllab (Duan et al., 2016). We use the code from Achiam et al. (2017) for building the Point-Circle and Point-Gather environments.
- **A link to downloadable version of the dataset or simulation environment.** See: github.com/openai/gym for OpenAI Gym benchmarks, github.com/jachiam/cpo for rllab based Circle and Gather environments.
- **An explanation of how samples were allocated for training / validation / testing.** We do not use a split as we run multiple runs over random seeds to examine the optimization performance.
- **An explanation of any data that were excluded.** NA
- **The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.** The default hyper-parameters for the MuJoCo baselines are taken from Kostrikov (2018). The ranges and parameters for Grid experiments are described in Sec. G, and for MuJoCo are described in Sec. H.
- **The exact number of evaluation runs.** The number of evaluation runs is mentioned in the caption corresponding to each result.
- **A description of how experiments were run.** See Experiments Sec. 6 in the main paper and in Appendix Sec. G and Sec. H.
- **A clear definition of the specific measure or statistics used to report results.** Un-discounted return and cost using the current policy over the horizon are plotted after every 1000 episodes are plotted. We use a linear-filter with 0.7 weight for smoothing. We use the smoothing algorithm provided by TensorBoard (`https://github.com/tensorflow/tensorboard`).
- **Clearly defined error bars.** Standard error used in all cases.

- **A description of results with central tendency (e.g. mean) and variation (e.g. stddev).** The bold lines in the figure represent the mean, and the shaded region denotes the 80% confidence interval.
- **A description of the computing infrastructure used.** We distribute all runs across 10 CPU nodes (Intel(R) Xeon(R) CPU E5-2650 v4) and 1 GPU (GP 100) per run for experiments.

B. Backward Value Functions

We have the following result from Proposition 1 from Morimura et al. (2010). We give the proof too for the sake of completeness.

Proposition B.1. Let the forward Markov chain $\mathcal{M}(\pi)$ be irreducible and ergodic, i.e., has a stationary distribution. Then the associated backward Markov chain $\overleftarrow{\mathcal{B}}(\pi)$ is also ergodic and has the same unique stationary distribution as $\mathcal{M}(\pi)$:

$$\eta^\pi(x) = \overleftarrow{\eta}^\pi(x), \qquad (\forall x \in \mathcal{X})$$

where $\eta^\pi(x)$ and $\overleftarrow{\eta}^\pi(x)$ are the stationary distributions of $\mathcal{M}(\pi)$ and $\overleftarrow{\mathcal{B}}(\pi)$.

Proof. Multiply both sides of Eq. (2) by $\eta^\pi(x_t)$ and sum over all actions $a_{t-1} \in \mathcal{A}$ we obtain detailed balance like equations (with respect to time):

$$\overleftarrow{\mathcal{P}}^\pi(x_{t-1}|x_t)\eta^\pi(x_t) = \mathcal{P}^\pi(x_t, a_{t-1}|x_{t-1})\eta^\pi(x_{t-1}). \qquad (\forall x_{t-1} \in \mathcal{X}, x_t \in \mathcal{X})$$

Sum over all possible x_t we have:

$$\sum_{x_t \in \mathcal{X}} \overleftarrow{\mathcal{P}}^\pi(x_{t-1}|x_t)\eta^\pi(x_t) = \eta^\pi(x_{t-1}).$$

The above equation indicates that $\overleftarrow{\mathcal{B}}(\pi)$ has same stationary distribution as $\mathcal{M}(\pi)$. In the matrix form the above equation can be written as $\eta\overleftarrow{P}^\pi = \eta$, that implies that η is stationary distribution with $\overleftarrow{P}^\pi$ transition matrix. □

B.1. Relation between forward and backward markov chains and backward Value Functions

Proof. We use the technique of Proposition 2 of Morimura et al. (2010) to prove this. Using the Markov property and then substituting Eq. (2) for each term we have:

$$\begin{aligned}
\overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}, \ldots, x_{t-K}, a_{t-K}|x_t) &= \overleftarrow{\mathcal{P}}^\pi(x_{t-1}, a_{t-1}|x_t)\ldots\overleftarrow{\mathcal{P}}^\pi(x_{t-K}, a_{t-K}|x_{t-K+1}),\\
&= \frac{\mathcal{P}^\pi(x_t, a_{t-1}|x_{t-1})\ldots\mathcal{P}^\pi(x_{t-K+1}, a_{t-K}|x_{t-K})\eta^\pi(x_{t-K})}{\eta^\pi(x_t)},\\
&\propto \mathcal{P}^\pi(x_t, a_{t-1}|x_{t-1})\ldots\mathcal{P}^\pi(x_{t-K+1}, a_{t-K}|x_{t-K})\eta^\pi(x_{t-K}).
\end{aligned}$$

This proves the proposition for finite K. Using the Prop. B.1, $K \to \infty$ case is proven too:

$$\begin{aligned}
\lim_{K\to\infty} \mathbb{E}_{\overleftarrow{\mathcal{B}}(\pi)}\left[\sum_{k=0}^{K} d(x_{t-k})|x_t\right] &= \lim_{K\to\infty} \mathbb{E}_{\mathcal{M}(\pi)}\left[\sum_{k=0}^{K} d(x_{t-k})|x_t, \eta^\pi(x_{t-K})\right],\\
&= \sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi(a|x)\eta^\pi(x)d(x).
\end{aligned}$$

□

B.2. TD for BVF

Proof. We use the same technique from Stochastic Shortest Path dynamic programming (Bertsekas et al., 1995, Vol 2, Proposition 1.1) to prove the above proposition. The general outline of the proof is given below, for more details we refer the reader to the textbook.

We have,

$$\overleftarrow{\mathcal{T}}^{\pi}\overleftarrow{V} = d + \overleftarrow{P}^{\pi}\overleftarrow{V}. \qquad \text{(Eq. (6) in matrix notation)}$$

Using induction argument, we have for all $\overleftarrow{V} \in \mathbb{R}^n$ and $k \geq 1$, we have:

$$\left(\overleftarrow{\mathcal{T}}^{\pi}\right)^{k}\overleftarrow{V} = \left(\overleftarrow{P}^{\pi}\right)^{k}\overleftarrow{V} + \sum_{m=0}^{k-1}\left(\overleftarrow{P}^{\pi}\right)^{m} d,$$

Taking the limit, and using the result, $\lim_{k\to\infty}\left(\overleftarrow{P}^{\pi}\right)^{k}\overleftarrow{V} = 0$, regarding proper policies from Bertsekas et al. (1995, Vol 2, Equation 1.2), we have:

$$\lim_{k\to\infty}\left(\overleftarrow{\mathcal{T}}^{\pi}\right)^{k}\overleftarrow{V} = \lim_{k\to\infty}\sum_{m=0}^{k-1}\left(\overleftarrow{P}^{\pi}\right)^{m} d = \overleftarrow{V}^{\pi},$$

Also we have by definition:

$$\left(\overleftarrow{T}^{\pi}\right)^{k+1}\overleftarrow{V} = d + \overleftarrow{P}^{\pi}\left(\overleftarrow{T}^{\pi}\right)^{k}\overleftarrow{V},$$

and by taking the limit $k \to \infty$, we have:

$$\overleftarrow{V}^{\pi} = d + \overleftarrow{P}^{\pi}\overleftarrow{V}^{\pi},$$

which is equivalent to,

$$\overleftarrow{V}^{\pi} = \overleftarrow{\mathcal{T}}^{\pi}\overleftarrow{V}^{\pi}.$$

To show uniqueness, note that if $\overleftarrow{V} = \overleftarrow{\mathcal{T}}^{\pi}\overleftarrow{V}$, then $\overleftarrow{V} = \left(\overleftarrow{\mathcal{T}}^{\pi}\right)^{k}\overleftarrow{V}$ for all k and letting $k \to \infty$ we get $\overleftarrow{V} = \overleftarrow{V}^{\pi}$.

□

B.3. Value-based constraint

Proposition B.2. We have that:

$$\mathbb{E}_{\mathcal{M}(\pi)}\left[\sum_{k=0}^{T} d(x_k) \mid x_0\right] \leq \mathbb{E}_{x_t\sim\eta^{\pi}(\cdot)}\left[\overleftarrow{V}_{\mathcal{D}}^{\pi}(x_t) + V_{\mathcal{D}}^{\pi}(x_t) - d(x_t)\right]. \tag{10}$$

Proof. We show that $\mathbb{E}\left[\sum_{k=t}^{T} d(x_k) \mid x_0\right] \leq \mathbb{E}_{x_t\sim\delta_{x_0}(P^{\pi})^t}\left[V_{\mathcal{D}}^{\pi}(x_t)\right]$ and $\mathbb{E}\left[\sum_{k=0}^{t} d(x_k) \mid x_0\right] \leq \mathbb{E}_{x_t\sim\eta(\cdot)}\left[\overleftarrow{V}_{\mathcal{D}}^{\pi}(x_t)\right]$, where δ_{x_0} is a Dirac distribution at x_0. The first one follows since adding more steps to the trajectory (from $T-t$ steps to T) can only increase the expected total cost. In other words: $\mathbb{E}\left[\sum_{k=t}^{T} d(x_k) \mid x_0, \pi\right] = \delta_{x_0}(P^{\pi})^t\left(\sum_{k=t}^{T}(P^{\pi})^k\right) d \leq \delta_{x_0}(P^{\pi})^t\left(\sum_{k=t}^{T+t}(P^{\pi})^k\right) d = \mathbb{E}_{x_t\sim\delta_{x_0}(P^{\pi})^t}\left[V_{\mathcal{D}}^{\pi}(x_t)\right]$. We replace $\delta_{x_0}(P^{\pi})^t$ by η^{π} by taking a limit over t (by Assumption 3.1).

For the backwards case, we use the converse result to Proposition 3.1. Note that:

$$\begin{aligned}
\mathcal{P}^{\pi}(x_1, x_2, \ldots, x_t|x_0) &= \mathcal{P}^{\pi}(x_1|x_0)\mathcal{P}^{\pi}(x_2|x_1)\cdots\mathcal{P}^{\pi}(x_t|x_{t-1}) \\
&= \overleftarrow{\mathcal{P}}^{\pi}(x_0|x_1)\frac{\eta(x_1)}{\eta(x_0)}\overleftarrow{\mathcal{P}}^{\pi}(x_1|x_2)\frac{\eta(x_2)}{\eta(x_1)}\cdots\overleftarrow{\mathcal{P}}^{\pi}(x_{t-1}|x_t)\frac{\eta(x_t)}{\eta(x_{t-1})} \\
&= \overleftarrow{\mathcal{P}}^{\pi}(x_0|x_1)\overleftarrow{\mathcal{P}}^{\pi}(x_1|x_2)\cdots\overleftarrow{\mathcal{P}}^{\pi}(x_{t-1}|x_t)\frac{\eta(x_t)}{\eta(x_0)} \\
&= \overleftarrow{\mathcal{P}}^{\pi}(x_0, x_1, ..., x_{t-1}|x_t)\frac{\eta(x_t)}{\eta(x_0)}
\end{aligned}$$

This means that the forward probability for the trajectory $(x_0, x_1, ..., x_t)$ under policy π is equivalent the probability of sampling a state x_t from the stationary distribution and obtaining the trajectory $(x_t, x_{t-1}, ..., x_0)$ via the backwards chain. In other words:

$$\mathbb{E}_{\mathcal{M}(\pi)}\left[\sum_{k=0}^{t} d(x_k) \mid x_0\right] = \mathbb{E}_{\mathcal{B}(\pi), x_t \sim \eta(\cdot)}\left[\sum_{k=0}^{t} d(x_t) \mid x_0\right].$$

Since x_0 is a terminal state for the Backwards MDP we can replace the summation in the expectation with the backwards value function $\overleftarrow{V}_{\mathcal{D}}^{\pi}(x_t)$. ☐

C. Properties of the policy iteration (SPI)

Theorem C.1. Let $\sigma(x) \coloneqq \mathrm{TV}(\pi_{k+1}(\cdot|x), \pi_k(\cdot|x)) = (1/2)\sum_a |\pi_{k+1}(a|x) - \pi_k(a|x)|$ denote the total variation between policies $\pi_k(\cdot|x)$ and $\pi_{k+1}(\cdot|x)$. If the policies are updated sufficiently slowly and π_k is feasible, then so is π_{k+1}. More specifically:

(I) If π_k is feasible at x_0 and $\sigma(x) \leq \frac{d_0 - V_{\mathcal{D}}^{\pi_k}(x_0)}{2T^2 D_{\mathrm{MAX}}}$ $\forall x$ then π_{k+1} is feasible at x_0.

(II) If π_k is feasible everywhere (i.e. $V_{\mathcal{D}}^{\pi_k}(x) \leq d_0\ \forall x$) and $\sigma(x) \leq \frac{d_0 - V_{\mathcal{D}}^{\pi_k}(x)}{2T \max_{x'}\{d_0 - \overleftarrow{V}_{\mathcal{D}}^{\pi_k}(x') - d(x')\}}$ $\forall x$ then π_{k+1} is feasible everywhere.

We note that the second case allows the policies to be updated in a larger neighborhood but requires π_k to be feasible everywhere. By contrast the first item updates policies in a smaller neighbourhood but only requires feasibility at the starting state.

Proof. Similar to the analysis in Chow et al. (2018). We aim to show that $V_{\mathcal{D}}^{\pi_{k+1}}(x_0) \leq d_0$. For simplicity we consider $k = 0$, and by induction the other cases will follow. We write $P_0 = P^{\pi_0}, P_1 = P^{\pi_1}$, $\Delta(a|x) = \pi_1(a|x) - \pi_0(a|x)$, and $P_\Delta = \left[\sum_{a \in A} \Delta(a|x) P(x'|x,a)\right]_{\{x',x\}}$. Note that $(I - P_0) = (I - P_1 + P_\Delta)$, and therefore $(I - P_1 + P_\Delta)(I - P_0)^{-1} = I_{|\mathcal{X}|\times|\mathcal{X}|}$. Thus, we find

$$(I - P_0)^{-1} = (I - P_1)^{-1}(I_{|\mathcal{X}|\times|\mathcal{X}|} + P_\Delta(I - P_0)^{-1}).$$

Multiplying both sides by the cost vector d one has

$$V_{\mathcal{D}}^{\pi_0}(x) = \mathbb{E}\left[\sum_{t=0}^{T} d(x_t) + \varepsilon(x_t) \mid \pi_1, x\right],$$

for each x, where $\varepsilon(x) = \sum_{a \in A} \Delta(a|x) \sum_{x' \in \mathcal{X}} P(x'|x,a) V_{\mathcal{D}}^{\pi_0}(x')$. Splitting the expectation, we have

$$V_{\mathcal{D}}^{\pi_1}(x) = V_{\mathcal{D}}^{\pi_0}(x) - \mathbb{E}\left[\sum_{t=0}^{T} \varepsilon(x_t) \mid \pi_1, x\right]$$

For case **(I)** we note that $V_{\mathcal{D}}^{\pi_0}(x') \leq D_{\mathrm{MAX}} T$ and so $-2\sigma(x_t) D_{\mathrm{MAX}} T \leq \varepsilon(x_t)\ \forall x_t$. Using $\sigma(x_t) \leq (d_0 - V_{\mathcal{D}}^{\pi_k})/2D_{\mathrm{MAX}}T^2$ gives $V_{\mathcal{D}}^{\pi_1}(x_0) \leq V_{\mathcal{D}}^{\pi_0}(x_0) - 2D_{\mathrm{MAX}}T^2(d_0 - V_{\mathcal{D}}^{\pi_0}(x_0))/(2D_{\mathrm{MAX}}T^2) = d_0$, i.e. π_0 is feasible at x_0.
For case **(II)** we note that $V_{\mathcal{D}}^{\pi_0}(x) \leq \max_{x'}\{d_0 - \overleftarrow{V}_{\mathcal{D}}^{\pi_0}(x') - d(x')\} =: \Theta$ since π_0 is feasible at every x. As before, we have $-2\sigma(x_t)\Theta \leq \varepsilon(x_t)\ \forall x_t$ and so $V_{\mathcal{D}}^{\pi_1}(x) \leq V_{\mathcal{D}}^{\pi_0}(x) - 2\Theta T(d_0 - V_{\mathcal{D}}^{\pi_0}(x))/(2\Theta T) = d_0\ \forall x$, i.e. π_1 is feasible everywhere. ☐

Theorem C.2. Let π_n and π_{n+1} be successive policies generated be the policy iteration algorithm of (SPI). Then $V^{\pi_{n+1}} \geq V^{\pi_n}$.

Proof. Note that π_{n+1} and π_n are both feasible solutions of the LP (SPI). Since π_{n+1} maximizes V^π over all feasible solutions, the result follows. ☐

D. Analytical Solution of the Update - Discrete Case

We follow the same procedure as Chow et al. (2018, Section E.1) to convert the problem to its Shannon entropy regularized version:

$$\begin{aligned} \max_{\pi \in \Delta} \quad & \pi(.|x)^T (Q(x,.) + \tau \log \pi(.|x)), \\ \text{s.t.} \quad & \pi(.|x)^T Q_{\mathcal{D}}(x,.) + \overleftarrow{V}^{\pi}_{\mathcal{D}}(x) - d(x) \leq d_0, \end{aligned} \tag{11}$$

where $\tau > 0$ is a regularization constant. Consider the Lagrangian problem for optimization:

$$\max_{\lambda \geq 0} \max_{\pi \in \Delta} \Gamma_x(\pi, \lambda) = \pi(.|x)^T (Q(x,.) + \lambda Q_{\mathcal{D}}(x,.) + \tau \log \pi(.|x)) + \lambda(d_0 + d(x) - \overleftarrow{V}(x))$$

From entropy-regularized literature (Neu et al., 2017), the inner λ-solution policy has the form:

$$\pi^*_{\Gamma,\lambda}(.|x) \propto \exp\left(-\frac{Q(x,.) + \lambda Q_{\mathcal{D}}(x,.)}{\tau}\right)$$

We now need to solve for the optimal lagrange multiplier λ^* at x.

$$\max_{\lambda \geq 0} -\tau \,\text{log-sum-exp}\left(-\frac{Q(x,.) + \lambda Q_{\mathcal{D}}(x,.)}{\tau}\right) + \lambda(d_0 + d(x) - \overleftarrow{V}_{\mathcal{D}}(x)),$$

where $\text{log-sum-exp}(y) = \log \sum_a exp(y_a)$ is a convex function in y, and objective is a concave function of λ. Using KKT conditions, the ∇_λ gives the solution:

$$(d_0 + d(x) - \overleftarrow{V}_{\mathcal{D}}(x)) - \frac{\sum_a Q_{\mathcal{D}}(x,a) \exp(\left(-\frac{Q(x,a) + \lambda Q_{\mathcal{D}}(x,a)}{\tau}\right))}{\sum_a \exp(\left(-\frac{Q(x,a) + \lambda Q_{\mathcal{D}}(x,a)}{\tau}\right))} = 0$$

Using parameterization of $z = \exp(-\lambda)$, the above condition can be written as polynomial equation in z:

$$\sum_a \left(d_0 + d(x) - \overleftarrow{V}_{\mathcal{D}}(x) - Q_{\mathcal{D}}(x,a)\right) . \left(\exp(-\frac{Q(x,a)}{\tau})\right) z^{\frac{Q_{\mathcal{D}}(x,a)}{\tau}} = 0$$

The roots to this polynomial will give $0 \leq z^*(x) \leq 1$, using which one can find $\lambda^*(x) = -\log(z^*(x))$. The roots can be found using the Newton's method. The final optimal policy of the entropy-regularized process is then:

$$\pi^*_{\Gamma} \propto \exp\left(-\frac{Q(x,\cdot) + \lambda^* Q_{\mathcal{D}}(x,\cdot)}{\tau}\right)$$

E. Extension of Safety Layer to Stochastic Policies with Gaussian Paramterization

Consider stochastic gaussian policies parameterized by mean $\mu(x;\theta)$ and standard-deviation $\sigma(x;\phi)$, and the actions sampled have the form $\mu(x;\theta) + \sigma(x;\phi)\epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ is the noise. Here, $< \mu(x;\theta), \sigma(x;\phi) >$ are both deterministic w.r.t. the parameters θ, ϕ and x, and as such both of them together can be treated in the same way as deterministic policy $(\pi(x) =< \mu(x), \sigma(x) >)$. The actual action sampled and executed in the environment is still stochastic, but we have moved the stochasticity fron the policy to the environment. This allows us to define and work with action-value functions $Q_{\mathcal{D}}(x, \mu_\pi(x), \sigma_\pi(x))$. In this case, the corresponding projected actions have the form $\mu' + \sigma'\epsilon$. The main objective of the safety layer (without the constraints) can be further simplified as:

$$\arg\min_{\mu',\sigma'} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\frac{1}{2} \left\|(\mu' + \sigma'\epsilon) - (\mu_\pi(x) + \sigma_\pi(x)\epsilon)\right\|^2\right]$$

$$\arg\min_{\mu',\sigma'} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\frac{1}{2} \left\|(\mu' - \mu_\pi(x)) + ((\sigma' - \sigma_\pi(x))\epsilon)\right\|^2\right]$$

$$\arg\min_{\mu',\sigma'} \frac{1}{2}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\left\|\mu' - \mu_\pi(x)\right\|^2 + \left\|(\sigma' - \sigma_\pi(x))\epsilon\right\|^2 + \underbrace{2 < \mu' - \mu_\pi(x), (\sigma' - \sigma_\pi(x))\epsilon >}_{=0,\text{due to linearity of expectation},\epsilon\sim\mathcal{N}(0,I)}\right]$$

$$\arg\min_{\mu',\sigma'} \frac{1}{2}\left(\left\|\mu' - \mu_\pi(x)\right\|^2 + \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[\left\|(\sigma' - \sigma_\pi(x))\epsilon\right\|^2\right]\right)$$

$$\arg\min_{\mu',\sigma'} \frac{1}{2}\left(\left\|\mu' - \mu_\pi(x)\right\|^2 + \left\|(\sigma' - \sigma_\pi(x))\right\|^2 \underbrace{\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[\left\|\epsilon\right\|^2\right]}_{=1,\text{second moment of }\epsilon}\right)$$

$$\arg\min_{\mu',\sigma'} \frac{1}{2}\left(\left\|\mu' - \mu_\pi(x)\right\|^2 + \left\|(\sigma' - \sigma_\pi(x))\right\|^2\right)$$

As both $\mu_\pi(.;\theta)$ and $\sigma_\pi(.;\phi)$ are modelled by independent set of parameters (different neural networks, usually) we can solve each of the safety layer problem independently, w.r.t. only those parameters.

F. Analytical Solution in Safety Layer

The proof is similar to the proof of the Proposition 1 of Dalal et al. (2018). We have the following optimization problem:

$$\arg\min_{\mu} \left[\frac{1}{2}\left\|(\mu - \mu_\pi(x)\right\|^2\right],$$
$$\texttt{s.t.}\quad \overleftarrow{V}^{\pi}_{\mathcal{D}}(x) - d(x) + Q^{\pi}_{\mathcal{D}}(x, \mu_\pi(x)) + (\mu - \mu_\pi(x))^T(\nabla Q^{\pi}_{\mathcal{D}}(x,\mu)|_{\mu=\mu_\pi(x)}) \leq d_0$$

As the objective function and constraints are convex, and the feasible solution, μ^*, λ^*, should satisfy the KKT conditions. We define $\epsilon(x) = (d_0 + d(x) - \overleftarrow{V}^{\pi}_{\mathcal{D}}(x) - Q^{\pi}_{\mathcal{D}}(x, \mu_\pi(x)))$, and $g_{\mu,\mathcal{D}}(x) = \nabla Q^{\pi}_{\mathcal{D}}(x,u)|_{u=\mu_\pi(x)}$. Thus, we can write the Lagrangian as:

$$L(\mu,\lambda) = \frac{1}{2}\left\|(\mu - \mu_\pi(x)\right\|^2 + \lambda((\mu - \mu_\pi(x))^T g_{\mu,\mathcal{D}}(x) - \epsilon(x))$$

From the KKT conditions, we get:

$$\nabla_\mu L = \mu - \mu_\pi(x) + \lambda g_{\mu,\mathcal{D}}(x) = 0 \tag{12}$$
$$(\mu - \mu_\pi(x))^T g_{\mu,\mathcal{D}}(x) - \epsilon(x) = 0 \tag{13}$$

From Eq. (12), we have:

$$\mu^* = \mu_\pi(x) - \lambda^*(x)\cdot g_{\mu,D}(x) \tag{14}$$

Substituting Eq. (14) in Eq. (13), we get:

$$-\lambda^*(x)\cdot g_{\mu,D}(x)^T g_{\mu,D}(x) - \epsilon(x) = 0$$
$$\lambda^* = \frac{-\epsilon(x)}{g_{\mu,D}(x)^T g_{\mu,D}(x)}$$

When the constraints are satisfied ($\epsilon(x) > 0$), the λ should be inactive, and hence we have $()^+$ operator, that is 0 for negative values.

G. Details of Grid-World Experiments

G.1. Architecture and Training details

We use one-hot encoding of the agent's location in the grid as the observation, i.e. x is a binary vector of dimension $\mathbb{R}^{12\times 12}$. The agent is trained for 200k episodes, and the current policy's performance is evaluated after every 1k episodes.

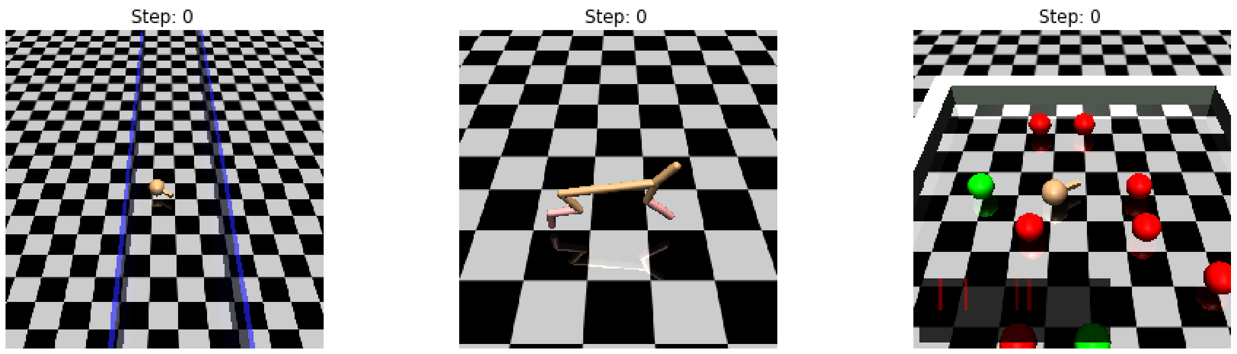
The same three layer neural network with the architecture is used for state encoding for all the different the estimators. The feed-forward neural network has hidden layers of size 64, 64, 64, and relu activations. For the state-action value based estimators, the last layer is a linear layer with 4 outputs, for each action. For value function based estimators the last layer is linear layer with a single output.

We use Adam Optimizer for training all the estimators. A learning rate of 1e-3 was selected for all the reward based estimators and a learning rate of 5e-4 was selected for all the cost based estimators. The same range of learning rate parameters for considered for all estimators i.e. {1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1}.

We use n-step trajectory length in A2C with $n = 4$, i.e., trajectories of length n were collected and the estimators were updated to used via the td-errors based on that. We use the number of parallel agents 20 in all the experiments. The range of parameters considered was $n \in \{1, 4, 20\}$. The same value of n was used for all the baselines.

H. Details of the MuJoCo Experiments

H.1. Environments Description



(a) Point-Circle (b) Safe-Cheetah (c) Point-Gather

Figure 4. MuJoCo Safety Environments

- **Point-Gather**: The environment (Fig.4c) is taken from Achiam et al. (2017), where the point mass agent gets a reward of $+10.0$ for collecting a green apple, and a cost of 1 for collecting a red bomb. Two apples and eight bombs are spawned randomly at the start of each episode. The constraints are defined over the nmber of bombs collected over the episode. Episode horizon is 15 and threshold $d_0 = 4$.
- **Safe-Cheetah**: This environment (Fig.4b) is taken from Chow et al. (2019). A bi-pedal agent (HalfCheetah-v0) is augmented with speed safety constraints. The agent gets the reward based on the speed with which it runs, and the constrain is define on the speed to be less than 1, i.e., it gets a constraint cost based on $\mathbb{1}[|v| > 1]$, where v is the velocity at the state. The maximum length of the episode is 200 and the constraint threshold is $d0 = 50$.
- **Point-Circle**: This environment (Fig.4a) is taken from Achiam et al. (2017). The point-mass agent is rewarded for running along the circumference of a circle of radius 15 in counter-clockwise direction, with the reward and cost function:

$$R(s) = \frac{v^T[-y, x]}{1 + |\,\|[x, y]\|_2 - 15|},$$
$$C(s) = \mathbb{1}[|x| > 2.5],$$

where x, y are coordinates in the plane and v is the velocity. The length of the episode is 65 and the constraint threshold $d_0 = 10.0$.

H.2. Network Architecture and training details

The architecture and the training procedure is based on the open-source implementations (Kostrikov, 2018). All the value based estimators use a network architecture of 2 hidden layers of size 200, 50 hidden units with tanh non-linearity, followed by a linear layer with single output. For the actor, we model mean using a network architecture of 2 hidden layers of size 100, 50 hidden units with tanh non-linearity, followed by a linear layer with dimensions of the action-space and tanh non-linearity. For the $Q(x, \mu)$ we also a 2 layer neural network with 200, (50 + action-dimension) hidden units and tanh non-linearity. We concatenate the mean in the second layer, and add a linear layer with single output in the end.

Entropy regularization with $\beta = 0.001$ was used for all the experiments and the baselines. The trajectory length for different environments. For PPO, GAE with $\lambda = 0.95$ was used for every algorithm. 20 parallel actors were used for every algorithm for each experiment. We searched the trajectory length hyper-parameter in the range 5,20,1000 for every environment.

We use Adam Optimizer for training all the estimators. The learning rate of the critic is always 0.5 the learning rate of the actor. For the cost estimators, the same learning rate was used for forward and backward estimators. The same range of learning rate parameters for considered for all estimators i.e. {1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1}.

H.3. Other details

We run the experiments with and without the use of recovery policies (in the same procedure as the baselines), and chose the run that performs the best. In order to take error due to function approximation into account, Achiam et al. (2017) use cost-shaping to smooth out the sparse constraint, and Chow et al. (2019) use a relaxed threshold, i.e. $d_0 \cdot (1 - \delta)$, instead of d_0, where $\delta \in (0, 1)$. We run experiments with $\delta = \{0.0, 0.2\}$ for each algorithms, and use the best among them. We found that empirically, only for Safe-Cheetah $\delta = 0.2$ works better compared to $\delta = 0.0$.

I. Algorithm Details

I.1. n-step Synchronous SARSA

The algorithm for n-step Synchronous SARSA is similar to the n-step Asynchronous Q-learning of Mnih et al. (2016), except that it uses SARSA instead of Q-learning, is synchronous, and instead of greedy maximization step of ϵ-greedy we use (SPI). When working with discrete actions and deterministic policies, this can be solved as part of the computation-graph itself.

I.2. A2C

In Actor Critic (Konda & Tsitsiklis, 2000) algorithms, the parameterized policy (actor) is denoted by $\pi(a|x;\theta)$, and is updated to minimizing the following loss:

$$L(\theta) = \mathbb{E}[-\log \pi(a_t|x_t;\theta)(r_t + \gamma V^{\pi}(x_{t+1} - V_{x_t}))]$$

The algorithm for A2C with Safety Layer given by Eq. (9) is similar to the Synchronous version of Actor-Critic (Mnih et al., 2016), except that it has estimates for the costs and safety layer. Note that due to the projection property of the safety layer, it is possible to sample directly from the projected mean. Also, as the projection is a result of vector products and max, it is differentiable and and computed in-graph (via relu). The algorithm is presented in Algorithm 1.

I.3. PPO

The PPO algorithm build on top of the Actor-Critic algorithm and is very similar to Algorithm 1. The main difference is how the PPO loss for the actor is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}[\min(\rho_t(\theta)A_t, clip(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)],$$

where the likelihood ration is $\rho_t(\theta) = \frac{\pi_\theta(a_t|x_t)}{\pi_{\theta_{old}}(a_t|x_t)}$, with π_{old} being the policy parameters before the update, $\epsilon < 1$ is a hyper-parameters that controls the clipping and A_t is the generalized advantage estimator:

$$A_t^{GAE(\lambda,\gamma)} = \sum_{k=0}^{T-1} (\lambda\gamma)^k \delta_{t+k}^{V^\pi},$$

where T is the maxmimum number of timestamps in an episode trajectory, and δ_j denotes the TD error at j. The value function is updated using the $\gamma\lambda$-returns from the GAE:

$$L(\phi) = \mathbb{E}[(V^\pi(x;\phi) - (V^\pi(x;\phi_{old}) + A_t))^2].$$

Similar to the the forward value estimates the backward value estimates are defined in the similar sense. One way to think of it is to assume the trajectories are reversed and we are doing the regular GAE estimation for the value functions.

The GAE updates for the regular value function can be seen in the λ-operator form as:

$$\mathcal{T}_\lambda^\pi v^\pi = (I - \gamma\lambda P^\pi)^{-1}(r^\pi + \gamma P^\pi v^\pi - v^\pi) + v^\pi.$$

In similar spirit it can be shown that the λ-operator for SARSA has the form:

$$\mathcal{T}_\lambda^\pi q^\pi = (I - \lambda\gamma P^\pi)^{-1}(\mathcal{T}^\pi q^\pi - q^\pi) + q^\pi,$$

where $(\mathcal{T}^\pi q^\pi - q^\pi)$ denotes the TD error. Thus, the GAE estimates can be applied for the Q-functions in the similar form, i.e.

$$B_t^{GAE(\lambda,\gamma)} = \sum_{k=0}^{T-1} (\lambda\gamma)^k \delta_{t+k}^{Q_\mathcal{D}^\pi},$$
$$L(\theta_\mathcal{D}) = \mathbb{E}[(Q_\mathcal{D}^\pi(x,a;\theta_\mathcal{D}) - (Q_\mathcal{D}^{\theta_\mathcal{D}}(x,a;\theta_{\mathcal{D}_{old}}) + B_t))^2].$$

J. Target based updates

An alternate way to implement the safety layer can be done as proposed by Chow et al. (2020) using target networks (Mnih et al., 2016). A target network for a policy, π_ω, denotes a copy of the parameterized policy at a previous iteration that is used for calculating the safety constraint in the safety layer. Using this notion, the safety layer projection objective as defined in Equation 9 can be written as:

$$\arg\min_\mu \left[\frac{1-\kappa}{2}\left\|\mu - \mu_\pi(x)\right\|^2\right] + \left[\frac{\kappa}{2}\left\|\mu - \mu_{\pi_\omega}(x)\right\|^2\right],$$
$$\texttt{s.t.}\quad \overleftarrow{V}_\mathcal{D}^{\pi_\omega}(x) - d(x) + Q_\mathcal{D}^{\pi_\omega}(x,\mu_{\pi_\omega}(x)) + (\mu - \mu_{\pi_\omega}(x))^T(\nabla Q_\mathcal{D}^{\pi_\omega}(x,\mu)|_{\mu=\mu_{\pi_\omega}(x)}) \le d_0,$$

where κ is a hyper-parameter that controls the trade-off between the unconstrained policy and the target policy.

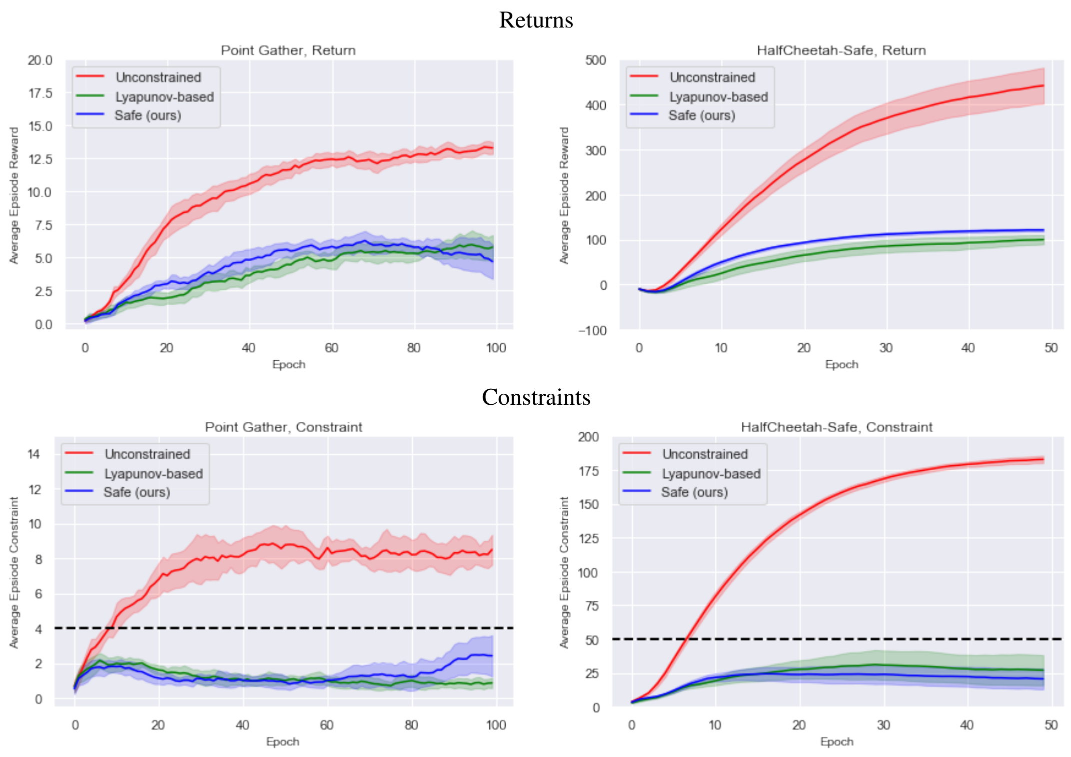
Using the exact same procedure in Appendix F, we can get that for any state $x \in \mathcal{X}$ the analytical solution to the equation above is given by:

$$\mu^*(x) = (1-\kappa)\mu_\pi(x) + \kappa\,\mu_{\pi_\omega}(x) - \lambda^*(x)\cdot g_{\mu_\omega,\mathcal{D}}(x),$$

where,

$$g_{\mu_\omega,\mathcal{D}}(x) = \nabla Q_\mathcal{D}^{\pi_\omega}(x,\mu)|_{\mu=\mu_{\pi_\omega}(x)},$$
$$\epsilon(x) = (d_0 + d(x) - \overleftarrow{V}_\mathcal{D}^{\pi_\omega}(x) - Q_\mathcal{D}^{\pi_\omega}(x,\mu_{\pi_{target}}(x)),$$
$$\lambda^*(x) = \left(\frac{(1-\kappa)g_{\mu_\omega,\mathcal{D}}(x)^T(\mu - \mu_{\pi_\omega}) - \epsilon(x)}{g_{\mu_\omega,\mathcal{D}}(x)^T g_{\mu_\omega,\mathcal{D}}(x)}\right)^+.$$

The target-network based implementation is easier to implement and computationally faster, however at the same time it introduces the hyper-parameter κ that controls the trade-off between reward maximization and constraint satisfaction, and another hyper-parameter for the rate at which the the target networks are updated. We tested this implementation with PPO on Point-Gather and Safe-Cheetah MuJoCo environments. We report the empirical results in Figure 5. We observe that the trends are similar with respect to constraint satisfaction, even though there is some variation in the magnitude of actual return collected by the agent.



(a) Point-Gather (b) Safe-Cheetah

Figure 5. PPO Performance over the training for Unconstrained (red), Lyapunov-based (green), and our method (blue) all trained with PPO and using a target network based update rule, on MuJoCo tasks over 10 random seeds. The x-axis is the number of episodes in thousands, and y-axis denotes the undiscounted accumulated returns. The dotted black line denotes d_0. The bold line represents the mean, and the shaded region denotes the 80% confidence-intervals.