Distributional Safety Critic for Stochastic Latent Actor-Critic

Thiago S. Miranda[1], **Heder S. Bernardino**[1]

[1]Universidade Federal de Juiz de Fora (UFJF)

{thiago.miranda,heder}@ice.ufjf.br

Abstract. *When employing reinforcement learning techniques in real-world applications, one may desire to constrain the agent by limiting actions that lead to potential damage, harm, or unwanted scenarios. Particularly, recent approaches focus on developing safe behavior under partial observability conditions. In this vein, we develop a method that combines distributional reinforcement learning techniques with methods used to facilitate learning in partially observable environments, called distributional safe stochastic latent actor-critic (DS-SLAC). We evaluate the DS-SLAC performance on four Safety-Gym tasks and DS-SLAC obtained results better than those reached by state-of-the-art algorithms in two of the evaluated environments while being able to develop a safe policy in three of them. Lastly, we also identify the main challenges of performing distributional reinforcement learning in the safety-constrained partially observable setting.*

1. Introduction

The field of deep reinforcement learning (DRL) has enjoyed great research attention over the last few years. Particularly, DRL approaches achieved notable results when applied to games, including superhuman level play at games from the Atari 2600 console [Mnih et al. 2013] and beating the best Go player in the world [Silver et al. 2016]. However, as we aim to apply DRL techniques to real-world scenarios, the agent's performance ceases to be the only concern. In those circumstances, how safely the agent can learn and execute its assigned tasks can be just as important as how well it performs on the tasks themselves.

For instance, a robot that is controlled by an AI must never harm a human being, and a self-driving car must make every effort to avoid actions that cause harm to the vehicle, even during its training procedure. Safe Reinforcement Learning (Safe RL) [Garcıa and Fernández 2015] is the area concerned with addressing this problem. It aims to create agents which are robust enough to act in the real world without causing harm or performing unwanted actions. This is particularly challenging, as RL agents generally learn by trial and error. These agents explore their action space in order to find the optimal action in each situation. As such, they may execute undesirable actions before learning that these actions lead to poor outcomes.

Current approaches to the safe reinforcement learning problem mainly rely on Safety-Gym tasks [Ray et al. 2019] to benchmark performance and use the constrained Markov decision processes (CMDP) [Altman 1999] as their formalism. In CMDPs, beyond the usual reward signal, the RL agent also receives a cost signal, which encapsulates how unsafe or undesirable a transition is. Additionally, recent methods have focused on

performing safe reinforcement learning in environments with a high degree of partial observability. To this end, these methods learn from high dimensional sensory inputs as observations in a constrained partially observable Markov decision process (CPOMDP) [Isom et al. 2008, Lee et al. 2018] setting.

In this work, we develop a method that augments [Hogewind et al. 2022] approach by using distributional RL techniques [Bellemare et al. 2023], which are known to improve sample-efficiency and performance of RL agents [Dabney et al. 2018b, Hessel et al. 2018]. We call our method distributional safe stochastic latent actor-critic (DS-SLAC). We evaluate DS-SLAC using four Safety-Gym tasks, where it achieves comparable performance to state-of-the-art algorithms in two of the environments while being able to produce a safe policy in three of them. Ultimately, we identify some of the challenges involved in performing distributional safe reinforcement learning under high partial observability, as well as the main obstacles to producing risk-averse behavior.

2. Constrained Partially Observable Markov Decision Process

A constrained partially observable Markov decision process (CPOMDP) [Isom et al. 2008, Lee et al. 2018] is the formalism that combines CMDPs with the partial observability assumption. This work performs distributional reinforcement learning in the CPOMDP setting. CMDPs and POMDPs are described as follows.

2.1. Constrained Markov Decision Process

In the standard reinforcement learning setting, an agent learns by performing actions in an environment based on the environment's current state, which, in turn, changes the state of the environment and generates a reward signal that encapsulates how desirable a transition is [Sutton and Barto 2018]. In practice, RL practitioners have found difficulties in precisely expressing desired complex behavior through the use of a reward function, with reward shaping becoming an area of research of its own [Ng et al. 1999].

In the case of safe reinforcement learning problems, state-of-the-art algorithms avoid dealing with the difficulties of shaping the reward function to express safety constraints. Instead, these algorithms rely on the constrained Markov decision process (CMDP) [Altman 1999] formalism. This allows for a clear separation between the task objective that the agent is trying to optimize and the safety violations that the agent should aim to avoid. The framework also provides an easy way to regulate the tradeoff between these two, often conflicting, goals. We describe the mathematical details as follows.

In a CMDP, at every time step t, in addition to the reward signal r_t, the agent also receives a vector containing k different cost signals c_t^k. In this setting, the agent aims to find the optimal policy π^*, i.e. the policy that maximizes some metric $J_r(\pi)$. Where $J_r(\pi)$ is some function that depends on the reward induced from following policy π. However, unlike in MDPs, the set of possible policies Π is reduced to Π_C, such that only feasible policies are considered, meaning policies that satisfy a set of constraints. Thus, the CMDP objective is the following:

$$\pi^* = arg \max_{\pi \in \Pi_C} J_r(\pi) \tag{1}$$

and the set of constraint-satisfying policies can be defined as:

$$\Pi_C = \{\pi : J_{c_i}(\pi) \leq d_i, \quad i = 1, .., k\} \tag{2}$$

where J_{c_i} is some function that depends on the cost c_i induced from following policy π, and d_i is a hyperparameter that specifies a desired upper limit for $J_{c_i}(\pi)$.

Often, when dealing with safe reinforcement learning, a single cost function c is used to express unsafety, and the hyperparameter d is referred to as the budget. Additionally, $J_r(\pi)$ is defined to be the expected return achieved by following policy π and starting from the initial state s_0:

$$J_r(\pi) = v_\pi^r(s_0) \tag{3}$$

and $J_c(\pi)$ is defined in an analogous manner.

Thus, safe RL algorithms commonly use neural networks to approximate both the reward action value Q^r and the cost action value Q^c. In the case of actor-critic methods, Q^c is often referred to as a safety critic.

2.2. Partially Observable Markov Decision Process

To more accurately model complex scenarios, one can assume that the observations received by an RL agent do not contain all the relevant information necessary to choose the optimal action. In this setting, it assumed that, instead of receiving a state s_t at time step t, the agent receives an observation o_t that is a function of the true state of the environment $o_t \sim O(s_t)$. This is called a partially observable Markov decision process (POMDP) [Kaelbling et al. 1998].

In a POMDP the policy is generally a function of the full history of observations and actions (or a truncated part of it) $a_t \sim \pi(\cdot|o_{1:t}, a_{1:t-1})$. Recurrent neural networks have been used to perform end-to-end reinforcement learning to solve POMDPs [Hausknecht and Stone 2015, Zhu et al. 2017]. Model-based approaches have also been used with the same purpose [Watter et al. 2015, Karl et al. 2016]. In this work, we follow the authors of [Hogewind et al. 2022] and use a stochastic latent representation to mitigate the partial observability effects on the agent learning, inspired by the stochastic latent actor-critic (SLAC) algorithm [Lee et al. 2020].

3. Distributional Reinforcement Learning

Classic reinforcement learning is concerned with creating a policy that is able to maximize the expected return. Thus, it's only natural to ponder about what would happen if, instead of taking only the expected return into account, the full return distribution was considered. Intuitively, the distribution contains far more information than the singular scalar value. In practice, it's been shown that the use of distributional reinforcement learning increases both sample efficiency and performance of RL agents [Bellemare et al. 2017, Hessel et al. 2018].

Multiple approaches have been proposed to perform distributional reinforcement learning, such as C51 [Bellemare et al. 2017], QR-DQN [Dabney et al. 2018b] and FQF [Yang et al. 2019]. These approaches mainly differ in terms of the way they parameterize the return distribution and the distance metric that is used to measure the difference between two distributions. In this work, we follow the authors of [Yang et al. 2023] and utilize the implicit quantile network (IQN) [Dabney et al. 2018a] to approximate the cost return distribution.

3.1. Implicit Quantile Network

Implicit quantile network (IQN) is a distributional reinforcement learning algorithm based on deep Q-learning [Mnih et al. 2015]. It trains an implicit parametric function to reparameterize samples from a base distribution, usually the uniform distribution $U([0,1])$, to the quantile values of a target distribution, which can be either the reward return distribution $Z_r(s,a)$ or the cost return distribution $Z_c(s,a)$. For a given return distribution $Z(s,a)$, by approximating its quantile function F_Z^{-1}, sampling $\tau \sim U([0,1])$ and applying F_Z^{-1} would equate to sampling the original return distribution: $F_Z^{-1}(\tau)(s,a) \sim Z(s,a)$.

As a distance metric, the p-Wasserstein distance is employed to measure the difference between two distributions U and V:

$$W_p(U,V) = \left(\int_0^1 |F_U^{-1}(\omega) - F_V^{-1}(\omega)|^p d\omega \right)^{\frac{1}{p}} \tag{4}$$

where F is the c.d.f.

In practice, the Wasserstein distance is approximated by the Huber quantile regression loss [Huber 1992], with a threshold κ:

$$\rho_\tau^\kappa(\delta_{i,j}) = |\tau - \mathbb{I}\{\delta_{i,j} < 0\}| \frac{\mathcal{L}(\delta_{i,j})}{\kappa}, \quad \text{with} \tag{5}$$

$$\mathcal{L}(\delta_{i,j}) = \begin{cases} \frac{1}{2}\delta_{i,j}^2 & \text{if } |\delta_{i,j}| < \kappa \\ \kappa(|\delta_{i,j}| - \frac{1}{2}\kappa) & \text{otherwise} \end{cases} \tag{6}$$

Thus, to approximate the quantile function of a return distribution F_Z^{-1}, a neural network Q_ξ parameterized by ξ is employed. The neural network can be trained based on the sampled temporal difference error. At time step t, for two i.i.d. samples $\tau, \tau' \sim U([0,1])$ and a policy π, the sample difference temporal error is defined as:

$$\delta_t^{\tau,\tau'} = r_t + \gamma Q_{\xi\tau'}(s_{t+1}, a_{t+1}) - Q_{\xi\tau}(s_t, a_t), \quad a_{t+1} \sim \pi_\phi(\cdot|s_{t+1}) \tag{7}$$

where $Q_{\xi\tau}$ corresponds to the quantile function evaluated with sample τ.

As a result, the complete loss for the IQN to train its action-state pair return distribution function is given by:

$$J_Q(\xi) = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\tau_i}^\kappa(\delta_t^{\tau_i,\tau_j'}) \tag{8}$$

where N and N' correspond to the number of samples $\tau, \tau' \sim U([0,1])$ used to estimate the loss.

3.2. Risk-averse safe reinforcement learning

Worst-case soft actor-critic (WCSAC) [Yang et al. 2023] is a soft actor-critic (SAC) [Haarnoja et al. 2018a, Haarnoja et al. 2018b] based algorithm that uses a distributional safety critic to produce risk-averse behavior. To this end, the upper tail of the estimated distribution is used. This is represented by the conditional Value-at-Risk (CVaR)

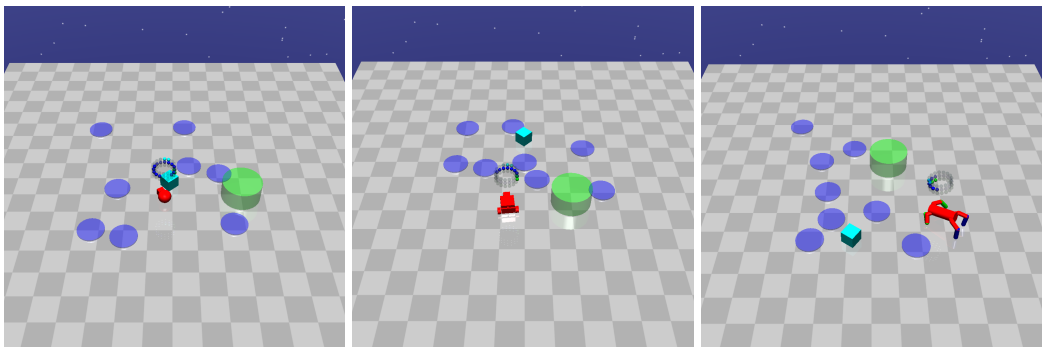
[Rockafellar et al. 2000]. The CVaR metric, for a cost signal c with random variable C that follows the return distribution Z_c induced by c when following policy π, is defined as:

$$\Gamma_\pi(s, a, \beta) = CVaR_\pi^\beta(C) = \mathbb{E}[C|C \geq F_{Z_c}^{-1}(1-\beta)] \tag{9}$$

where $\beta \in (0, 1]$ is a hyper parameter called risk level. Thus, when updating the agent's policy, the use of CVaR instead of the expectation over the entire distribution results in the agent adapting its behavior based on worst-case scenarios.

4. Related Work

To evaluate performance of safe reinforcement learning techniques, the authors of [Ray et al. 2019] developed a benchmark suite called Safety-Gym. Safety-gym contains a series of different robots (the agent's "body"), tasks, and hazards. Which can be combined to form environments with varying levels of difficulty for RL agents. Figure 1 shows some examples of these environments. Furthermore, they also provided baselines versions of the trust region policy optimization (TRPO) [Schulman et al. 2015] and proximal policy optimization (PPO) [Schulman et al. 2017] algorithms adapted to the CMDP setting. Subsequent work heavily relies on Safety-Gym in order to benchmark performance of CMDP DRL agents. We describe some of the approaches as follows.



(a) PointGoal1 (b) CarGoal1 (c) DoggoGoal1

Figure 1. Different Safety-Gym environments. In "Goal" tasks, the agent has to reach the goal (green cylinder) while trying to avoid hazards (blue circles and blue cube)

Constrained policy optimization (CPO) [Achiam et al. 2017] performs policy search by augmenting the usual local policy objective with CMDP constraints. In local policy search, the current policy is updated to the next policy by searching the space of all possible policies that have a similar distribution to the current one. Local search improves the stability of policy search methods and enables them to function in high dimensional function approximation settings. CPO augments the usual local policy search objective with CMDP constraints. The authors, then derive an approximate update that performs policy search, while respecting these constraints.

The Lagrangian model-based agent (LAMBDA) [As et al. 2022] is an approach for CPOMDPs. LAMBDA infers a Bayesian world model that is, then, used to simulate transitions and train the agent. Model-based techniques are used to improve sample efficiency of RL agents [Deisenroth and Rasmussen 2011]. The authors propose the use of

an optimistic pessimistic approach. To this end, they estimate uncertainty in their world model and use this uncertainty to generate multiple plausible instantiations of the model. Then, they generate a set of transitions for each model instantiation and use the maximum reward return and the maximum cost return to update the agent.

[Hogewind et al. 2022] is another work that aims at solving partially observable problems in the context of safe reinforcement learning. The authors propose a new version of the Stochastic Latent Actor-Critic (SLAC) algorithm [Lee et al. 2020] called Safe SLAC, which is adapted to work under the CMDP framework. SLAC is an actor-critic method based on the SAC algorithm. It assumes partial observability and, in order to infer the true state of the environment $\mathbf{z}_t$, a sequential latent variable model is defined. Then, by predicting rewards and the observations based on the true hidden state, the model can be trained to find $\mathbf{z}_t$, such that the likelihood of o_t and r_t are maximized. This is done via variational inference [Kingma and Welling 2013]. In addition to the reward and observation, the Safe SLAC model is also trained to accurately predict the cost c_t.

5. Distributional Safe Stochastic Latent Actor-Critic

In the same vain as [As et al. 2022] and [Hogewind et al. 2022], we propose a safe reinforcement learning algorithm to operate in the CPOMDP setting. Our approach is closely related to [Hogewind et al. 2022] in the sense that it is based on SLAC. The main difference is the use of a distributional safety critic instead of an expectation-based one. Thus, we call the proposed approach distributional safe stochastic latent actor-critic (DS-SLAC). We describe DS-SLAC in detail as follows.

DS-SLAC relies on the same latent variable model as [Hogewind et al. 2022] to infer the true hidden state of the environment $\mathbf{z}_t$ given an observation o_t. It does so using variational inference [Kingma and Welling 2013] to optimize the model parameters in order to fit the observed data. The architecture is given by the following conditional probabilities:

$$
\begin{aligned}
&\mathbf{z}_1^1 \sim p(\mathbf{z}_1^1) \\
&\mathbf{z}_1^2 \sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
&\mathbf{z}_{t+1}^1 \sim p_\psi(\mathbf{z}_{t+1}^1|\mathbf{z}_t^2, a_t) \\
&\mathbf{z}_{t+1}^2 \sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t) \\
&o_t \sim p_\psi(o_t|\mathbf{z}_t^1, \mathbf{z}_t^2) \\
&r_t \sim p_\psi(r_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2) \\
&c_t \sim p_\psi(c_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2)
\end{aligned}
\qquad (10)
$$

$$
\begin{aligned}
&\mathbf{z}_1^1 \sim q_\psi(\mathbf{z}_1^1|o_1) \\
&\mathbf{z}_1^2 \sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
&\mathbf{z}_{t+1}^1 \sim q_\psi(o_{t+1}|\mathbf{z}_t^2, a_t) \\
&\mathbf{z}_{t+1}^2 \sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t)
\end{aligned}
\qquad (11)
$$

where Equation 10 corresponds to generative part of the model and Equation 11 corresponds to the posterior portion. Given this architecture it is possible to train the latent variable model according to:

$$
J_M(\psi) = \mathbb{E}_{\mathbf{z}_{1:\eta+1} \sim q_\psi} \left[\sum_{t=0}^{\eta} \begin{array}{c} -\log p_\psi(o_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(r_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(c_{t+1}|\mathbf{z}_{t+1}) \\ +D_{KL}(q_\psi(\mathbf{z}_{t+1}|o_t, \mathbf{z}_t, a_t)||p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t, a_t)) \end{array} \right] \qquad (12)
$$

This is the same loss as the one used to train the model in [Hogewind et al. 2022].

As DS-SLAC is also a SAC [Haarnoja et al. 2018a, Haarnoja et al. 2018b] based method, a reward critic is trained according to the maximum entropy framework. In this sense, the reward critic loss is similar to the usual SAC critic loss. It differs, however, in the fact that the hidden state produced by the latent variable model is used as input to the critic function. Thus, the loss used to update the parameters θ from the reward critic Q^r_θ is defined as:

$$J_{Q^r}(\theta) = \mathbb{E}_{a_t, r_t \sim \mathcal{D}, \mathbf{z}_t, \mathbf{z}_{t+1} \sim q_\psi} \left[(Q_\theta(\mathbf{z}_t, a_t) - \hat{Q}(\mathbf{z}_t, a_t))^2 \right] \tag{13}$$

where

$$\hat{Q}(\mathbf{z}_t, a_t) = r_t + \gamma(Q_\theta(\mathbf{z}_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|\mathbf{z}_{t+1})), \quad a_{t+1} \sim \pi(\cdot|\mathbf{z}_{t+1}) \tag{14}$$

and $\mathcal{D}$ is the replay buffer.

As mentioned before, a safety critic is also trained. Our reasons for adopting a distributional perspective on the safety critic are twofold. First, inspired by the authors of [Yang et al. 2023], we believe that it is particularly interesting to be able to produce risk-averse behavior in the context of safe reinforcement learning. Second, the use of a distributional safety critic can increase the accuracy of the predictions of cost returns, which, in turn, allows the agent to more accurately trade-off reward and cost in the CMDP setting.

Consequently, DS-SLAC estimates the cost return distribution with an implicit quantile network discussed before. For brevity, we define $Q^c_{\xi\tau}$ as the function parameterized by ξ that approximates $F^{-1}_{Z_c}(\tau)$, where Z_c is the cost return distribution and F_{Z_c} is the c.d.f. of this distribution. In this manner, we train $Q^c_{\xi\tau}$ by sampling N and N' samples of $\tau, \tau' \sim U([0, 1])$ and estimating IQN the loss according to:

$$J_{Q^c}(\xi) = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho^\kappa_{\tau_i}(\delta_t^{\tau_i, \tau'_j}) \tag{15}$$

where

$$\delta_t^{\tau,\tau'} = r_t + \gamma Q^c_{\xi\tau'}(\mathbf{z}_{t+1}, a_{t+1}) - Q^c_{\xi\tau}(\mathbf{z}_t, a_t), \quad a_{t+1} \sim \pi_\phi(\cdot|\mathbf{z}_{t+1}) \tag{16}$$

and ρ^κ_τ is the Huber quantile regression loss described in equation 5.

Through estimating the cost return distribution and the expected reward return, the policy update can be performed according to:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{z}_t \sim q_\psi, a_t \sim \pi_\phi} \left[\alpha \log \pi_\phi(a_t|\mathbf{z}_t) - Q^r_\theta(\mathbf{z}_t, a_t) + \lambda Q^c_\xi(\mathbf{z}_t, a_t) \right] \tag{17}$$

with

$$Q^c_\xi(\mathbf{z}_t, a_t) = \frac{1}{K} \sum_{k=1}^{K} Q^c_{\xi\tau}(\mathbf{z}_t, a_t) \tag{18}$$

where K is the number of i.i.d. samples $\tau \sim U([0, 1])$ used to estimate $Q^c_\xi(\mathbf{z}_t, a_t)$, α is the parameter used to regulate the trade-off between reward and entropy, and λ is the Lagrange multiplier that, in turn, regulates the trade-off between reward and cost constraint.

Similarly to [Hogewind et al. 2022], we find during our preliminary experiments that updating the Lagrange multiplier with off-policy data (i.e., by sampling from the replay buffer $\mathcal{D}$) results in high instability during learning, often leading the agent to developing an unsafe policy. We found no other way of mitigating this stability besides updating the Lagrange multiplier with on-policy data, like it is done in [Hogewind et al. 2022]. For [Hogewind et al. 2022], this solution works well. In our case, however, this is particularly undesirable, as it prevents us from performing risk-averse updates by using the CVaR metric to update the Lagrange multiplier, as it is done by [Yang et al. 2023]. Thus, the update for the Lagrange multiplier is performed according to the following loss:

$$J_s(\lambda) = \mathbb{E}_\pi \left[\lambda \sum_{t=1}^{T} c_t - d \right] \tag{19}$$

where c_t is the cost induced by following policy π and d is the budget.

Consequently, DS-SLAC works in the following manner. First, the replay buffer $\mathcal{D}$ is initialized by interacting with the environment and selecting actions according to a random policy for W_p environments steps, in this way W_p transitions are stored in the replay buffer $\mathcal{D}$. Next, the latent variable model is pre-trained by sampling the transitions collected with the random policy from the replay buffer $(o_t, a_t, r_t, c_t) \sim \mathcal{D}$; this is performed for W_t times.

Afterward, the algorithm alternates between multiple iterations of two processes. The first consists of the agent interacting with the environment by selecting an action a_t based on the latent state z_t that is inferred by the latent model according to previous latent state z_{t-1} and current observation o_t. The selected action is executed in the environment and is stored in the replay buffer along with the received observation, reward, and cost $(o_{t+1}, r_{t+1}, c_{t+1})$. During this step, the Lagrange multiplier is updated, since it needs to be on-policy updated, as mentioned previously.

The second process consists of sampling a transition from the replay buffer $(o_t, a_t, r_t, c_t) \sim \mathcal{D}$, then calculating the losses for the latent model, reward critic, cost critic, and policy based on the sampled transition and according to the respective loss equations. Next, the respective weights are updated by performing a single gradient step. Lastly, the target networks for the reward and safety critic are also updated through exponential averaging [Mnih et al. 2015].

The alternation of these two processes can occur for as long as desired, essentially, until the agent develops a good enough policy. In practice, a maximum number of desired environment interactions (the number of times the first process is repeated) is defined and when this number is reached training stops. This is done to evaluate the agent sample efficiency, i.e. its performance relative to the number of samples of environment interactions used for learning.

Finally, DS-SLAC is described in detail in Algorithm 1, where θ_1 and θ_2 represent the double Q-learning networks [Fujimoto et al. 2018], while $\bar{\theta_1}$, $\bar{\theta_2}$ and $\bar{\xi}$ are the target networks [Mnih et al. 2015].

Algorithm 1: Distributional Safe Stochastic Latent Actor-Critic

Hyperparameters: W_t, W_p, N, N', K

Initialize $\mathcal{D}$ by following a random policy
for *i=1 to W* **do**
  $(o_t, a_t, r_t, c_t) \sim \mathcal{D}$
  Update ψ according to Equation 12
end for
while *not converged* **do**
  for *each environment step* **do**
    $a_t \sim \pi_\phi(a_t|\mathbf{z}_t)$
    Obtain $(o_{t+1}, r_{t+1}, c_{t+1})$ by executing a_t
    $\mathcal{D} \leftarrow \mathcal{D} \cup (o_{t+1}, a_t, r_{t+1}, c_{t+1})$
    Update λ according to Equation 19
  end for
  for *each gradient step* **do**
    $(o_t, a_t, r_t, c_t) \sim \mathcal{D}$
    Update ψ according to Equation 12
    Update θ_1 and θ_2 according to Equation 13
    Update ξ according to Equation 15
    Update ϕ according to Equation 17
    $\bar{\theta_1} \leftarrow \nu\theta_1 + (1-\nu)\bar{\theta_1}$
    $\bar{\theta_2} \leftarrow \nu\theta_2 + (1-\nu)\bar{\theta_2}$
    $\bar{\xi} \leftarrow \nu\xi + (1-\nu)\bar{\xi}$
  end for
end while

6. Computational Experiments

In practice, DS-SLAC collects $W_p = 60K$ environments steps following a complete random policy. Next, for $W_t = 30K$ steps, the latent variable model is initialized based on the collected information. Additionally, during the training of the policy, DS-SLAC uses 100 environment steps and 100 gradient steps, meaning it collects 100 environment interactions and then performs 100 gradient updates, repeating this process until the maximum number of environment steps is reached.

We perform our experiments in a subset of the SG6 benchmark from [Ray et al. 2019]. In the same vein as [As et al. 2022] and [Hogewind et al. 2022], DS-SLAC learns from images in a first-person point of view of the Safety-Gym robots. Also following [As et al. 2022] and [Hogewind et al. 2022], we perform the evaluation of each run as follows: for each 25K environment steps, the agent performance is measured by collecting $E = 10$ different episodes of length of $T_{ep} = 1000$ with the current policy. Then, the undiscounted reward and cost return for each episode are calculated and the mean of these metrics across all episodes represent the current policy performance. In short, the reward performance is given by $\frac{1}{E}\sum_{e=1}^{E}\sum_{t=1}^{T_{ep}} r_t$ and the cost performance is given by $\frac{1}{E}\sum_{e=1}^{E}\sum_{t=1}^{T_{ep}} c_t$. The data used for evaluation is then discarded and is not used for training.

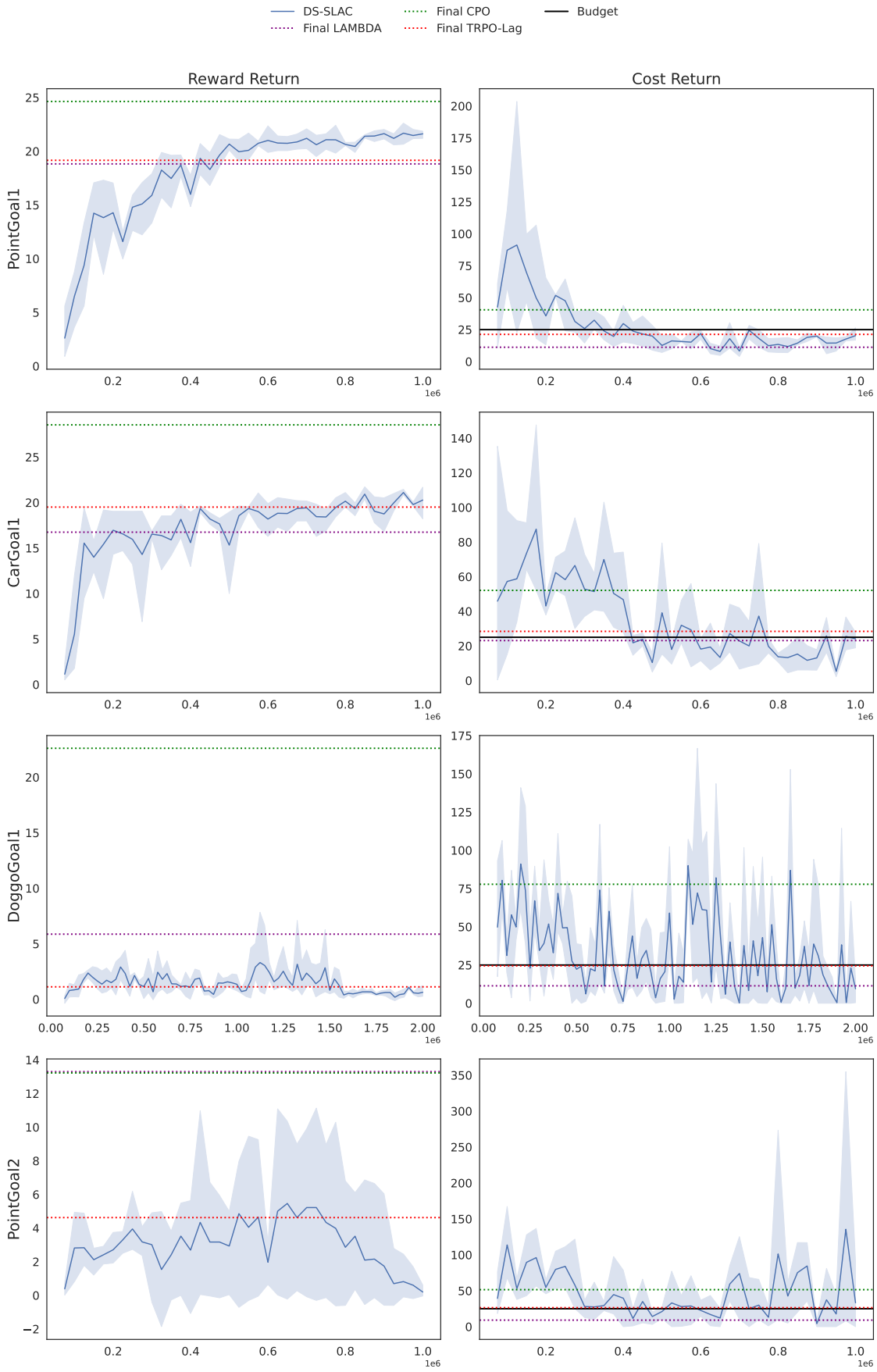


Figure 2. Learning curves for DS-SLAC in four different Safety-Gym environments.

Figure 2 presents the DS-SLAC learning curves for each Safety-Gym environment, generated by averaging the performance of three different random seeds. Dotted lines indicate the final results for LAMBDA, CPO, and TRPO-Lag that were reported by [As et al. 2022]. Both CPO and TRPO-Lag learn directly from sensors, with 10 million environment steps, meanwhile, LAMBDA and DS-SLAC learn for 1 million environment steps in PointGoal1, CarGoal1, and PointGoal2, and for 2 million steps in DoggoGoal1. [Hogewind et al. 2022] does not report the results for Safe-Slac in a tabular form, only through figures. As such, we do not include a direct comparison to Safe-Slac results. Nevertheless, it is still possible to get a sense of the comparative performance of both agents by looking at the graphics present here and in [Hogewind et al. 2022].

As proposed by [Ray et al. 2019], we also present a comparison of the normalized final reward and cost returns. Contrary to the learning curves, the normalized results are based on the final reward return and cost return after 1 million environment steps of training across all agents, including CPO and TRPO-Lag. The normalization is performed based on the result of an unconstrained proximal policy optimization (PPO) [Schulman et al. 2017] agent, also reported by [As et al. 2022]. Then, for each agent with a reward return $\hat{J}_r(\pi)$ and a cost return $\hat{J}_c(\pi)$, and for the reward return $\hat{J}_r^{PPO}$ and cost return $\hat{J}_c^{PPO}$ of the PPO agent, the normalized results are given by:

$$\begin{aligned} \bar{J}_r(\pi) &= \frac{\hat{J}_r(\pi)}{\hat{J}_r^{PPO}} \\ \bar{J}_c(\pi) &= \frac{max(0, \hat{J}_c(\pi) - d)}{max(10^{-6}, \hat{J}_c^{PPO} - d)} \end{aligned} \tag{20}$$

The normalized metrics are presented in Figure 3 and in Table 1.

Table 1. Normalized final results with each cell containing a tuple $(\bar{J}_r(\pi), \bar{J}_c(\pi))$.

	TRPO-Lag	CPO	LAMBDA	DS-SLAC
PointGoal1	0.51, 0.004	0.898, 0.302	1.077, **0.0**	**1.237**, **0.0**
CarGoal1	0.501, **0.0**	**1.579**, 0.604	1.284, **0.0**	1.555, **0.0**
DoggoGoal1	-1.257, 0.227	-0.723, 0.643	**5.400**, **0.0**	0.577, **0.0**
PointGoal2	0.119, 0.059	0.306, 0.132	**0.902**, **0.0**	0.014, 0.045

DS-SLAC demonstrates great performance in both PointGoal1 and CarGoal1. For either one of these environments, DS-SLAC outperforms all other methods by attaining the highest reward return, while also developing a safe policy.

On the other hand, for both DoggoGoal1 and PointGoal2, although DS-SLAC generates a safe policy for DoggoGoal1 and is very close to the budget value in the case of PointGoal2, our algorithm does not perform well with respect to reward return. In part, this occurs because these environments are significantly harder for agents in general. PointGoal2 contains a lot more hazards when compared to PointGoal1. Meanwhile, DoggoGoal1 robot has a large size, making it hard to avoid safety violations. Additionally, in the DoggoGoal1 environment, the robot's point of view shakes constantly when it moves, inducing a high degree of partial observability. The difficulty of these environments can also be attested by the performance of baseline algorithms, as seen in Figure 3.

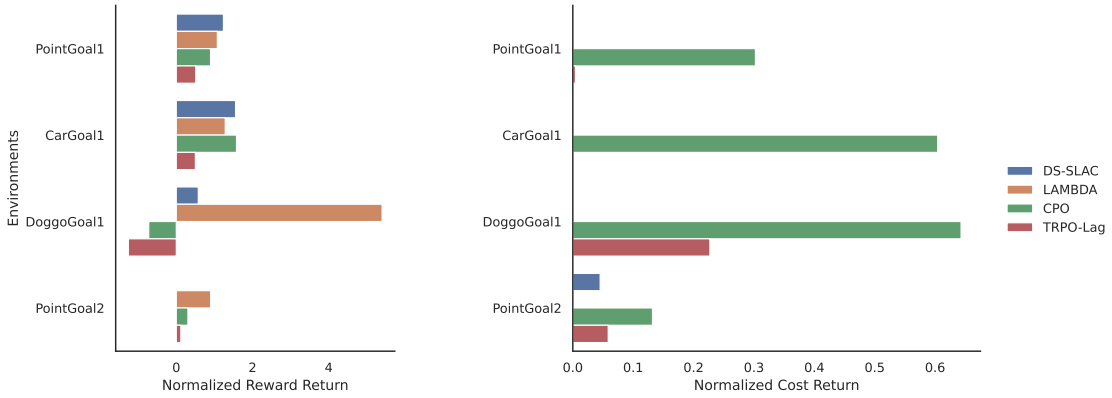


Figure 3. Normalized reward return and cost return.

Nevertheless, DS-SLAC still significantly underperforms when compared to both LAMBDA and Safe SLAC. We hypothesize that, for the more complex environments, DS-SLAC safety critic begins to overestimate cost values during training, eventually preventing the agent to seek higher reward return as the policy objective becomes too heavily influenced by the safety term. We believe one possible cause for this overestimation is the use of on-policy data to update the Lagrange multiplier, while the safety term in the actor loss is calculated in an off-policy manner. This factor can make the safety critic update unstable.

7. Conclusion

As reinforcement learning techniques increasingly transition from being applied to simulated environments to real-world applications, the topic of avoiding damage, risks, and unwanted scenarios during an agent's interaction with the environment becomes ever more prevalent. As such, there exists an intrinsic motivation for developing methods that are able to specify and enforce safety constraints in order to produce safe behavior.

Current state-of-the-art safe reinforcement learning algorithms work under the constrained Markov decision process (CMDP) formalism to create agents that perform well, even under a high degree of partial observability. We developed an algorithm named DS-SLAC, that combines techniques of amortized variational inference used to mitigate partial observability with a distributional reinforcement learning perspective. We obtain comparable results to state-of-the-art methods in some Safety-Gym environments, as DS-SLAC performs better than the other algorithms that were evaluated in two of these environments. Additionally, we identify some of the challenges in performing safe reinforcement learning with a distributional safety critic under the CPOMDP framework.

Future work can focus on improving performance of DS-SLAC in more complex environments. We hypothesize that updating the Lagrange multiplier with on-policy data while using off-policy data to calculate the safety term in the policy update results in an overestimation of the cost term used in the actor loss. As such, the cost term would dominate the policy objective, preventing the agent from focusing on accumulating reward.

Furthermore, we believe that being able to perform risk-averse constrained rein-

forcement learning is desirable in the safety setting. In the DS-SLAC case, we were not able to perform risk-averse constrained reinforcement learning, due to high instability when performing off-policy updates to the Lagrange multiplier, i.e., the term that regulates the trade-off between adhering to safety constraints and seeking a higher reward. Thus, encountering a way to perform the Lagrange multiplier update in an off-policy manner would unlock the ability to produce risk-averse behavior and, consequently, is a promising path for future work.

Acknowledgements

The authors thank the financial support provided by CNPq, FAPEMIG, CAPES, and UFJF.

References

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR.

Altman, E. (1999). *Constrained Markov decision processes*, volume 7. CRC press.

As, Y., Usmanova, I., Curi, S., and Krause, A. (2022). Constrained policy optimization via bayesian world models. *arXiv preprint arXiv:2201.09802*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR.

Bellemare, M. G., Dabney, W., and Rowland, M. (2023). *Distributional reinforcement learning*. MIT Press.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. (2018b). Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018b). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Hogewind, Y., Simao, T. D., Kachman, T., and Jansen, N. (2022). Safe reinforcement learning from pixels using a stochastic latent representation. *arXiv preprint arXiv:2210.01801*.

Huber, P. J. (1992). Robust estimation of a location parameter. *Breakthroughs in statistics: Methodology and distribution*, pages 492–518.

Isom, J. D., Meyn, S. P., and Braatz, R. D. (2008). Piecewise linear dynamic programming for constrained pomdps. In *AAAI*, volume 1, pages 291–296.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Karl, M., Soelch, M., Bayer, J., and Van der Smagt, P. (2016). Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. (2020). Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752.

Lee, J., Kim, G.-H., Poupart, P., and Kim, K.-E. (2018). Monte-carlo tree search for constrained pomdps. *Advances in Neural Information Processing Systems*, 31.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7(1):2.

Rockafellar, R. T., Uryasev, S., et al. (2000). Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42.

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

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

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

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28.

Yang, D., Zhao, L., Lin, Z., Qin, T., Bian, J., and Liu, T.-Y. (2019). Fully parameterized quantile function for distributional reinforcement learning. *Advances in neural information processing systems*, 32.

Yang, Q., Simão, T. D., Tindemans, S. H., and Spaan, M. T. (2023). Safety-constrained reinforcement learning with a distributional safety critic. *Machine Learning*, 112(3):859–887.

Zhu, P., Li, X., Poupart, P., and Miao, G. (2017). On improving deep reinforcement learning for pomdps. *arXiv preprint arXiv:1704.07978*.