Safe Reinforcement Learning through Phasic Safety Oriented Policy Optimization

Sumanta Dey[1,*], Pallab Dasgupta[1] and Soumyajit Dey[1]

[1]Indian Institute of Technology, Kharagpur, 721302, India

Abstract

Exploration is an essential feature of Reinforcement Learning (RL) algorithms, as they attempt to learn the optimal policy through trial and error. In safety-constrained environments, safety violations during exploration are a significant challenge when the training is online. In this context, this paper proposes Phasic Safety-oriented Policy Optimization (PSPO), where the policy learning is divided into multiple phases with safety updates. This approach utilizes an adaptive safety shield to minimize repetitive unsafe explorations of the RL agent by action-masking, and at the same time learn an auxiliary policy which provides safety updates to the main policy. Such periodic updates reduce the number of safety infractions during training, without compromising rewards as in purely conservative safety shield based approaches. We have demonstrated the effectiveness of our approach in multiple safety-critical environments. Our experimental results exhibit fewer failures during training while demonstrating similar or faster convergence than prior methods.

Keywords

Safe Reinforcement Learning, Safe Exploration, Safe Policy Learning

1. Introduction

In order to learn a policy that maximizes the total expected reward [1], a Reinforcement Learning (RL) agent operating in a model free environment needs to perform adequate exploration of its environment. In safety critical domains, the training phase poses a challenge if it is performed in the real environment, as the uninformed agent may lead itself to unsafe states during the exploration. A growing body of work addresses the safe RL problem from different directions, including the use of safety-shields, reward shaping, etc. [2]

The goal of a traditional RL agent is to learn an optimal policy (π^*) for a given starting state distribution (μ) that maximizes the overall expected return, G_t.

$$\pi^* \leftarrow \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}^{\pi}_{\mu}[G_t]$$

However, in safety constrained RL setups, the RL agent tries to learn an optimal policy (π^*_C) that maximizes the overall return (G_t) while also satisfying the safety constraints (C).

$$\pi^*_C \leftarrow \underset{\pi \in \Pi_C}{\operatorname{argmax}}\, \mathbb{E}^{\pi}_{\mu}[G_t]$$

where Π_C is the set of safe policies. It may so happen that the best policies in Π_C have trajectories that run close to unsafe states. In an attempt to remain safe, an RL agent may avoid safe states in the proximity of unsafe states, thereby missing out the better policies in Π_C.

Consider a simple volcanic grid-world in Figure 1. The robot tries to find the shortest path to the treasure in location, (a,2), from its initial location, (a,0). An active volcanic crater is present in location, (a,2), while the location, (c,1), is blocked. Consider the following trajectories of the agent, shown in green and black respectively:

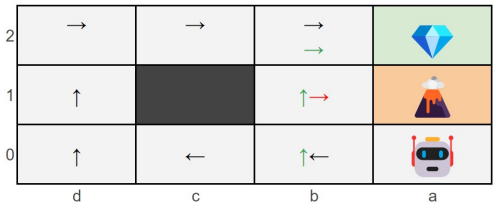


Figure 1: Volcanic Grid map [3]

Path-1: $(a,0) \xrightarrow{left} (b,0) \xrightarrow{up} (b,1) \xrightarrow{up} (b,2) \xrightarrow{right} (a,2)$

Path-2: $(a,0) \xrightarrow{left} (b,0) \xrightarrow{left} (c,0) \xrightarrow{left} (d,0) \xrightarrow{up} (d,1) \xrightarrow{up} (d,2) \xrightarrow{right} (c,2) \xrightarrow{right} (b,2) \xrightarrow{right} (a,2)$

Path-1 is optimal and significantly shorter than Path-2. During training, if the agent takes the *right* action from location, $(b,1)$, or the *up* action from location, $(a,0)$, the robot will fall in the volcanic cater and terminate that episode with failure. Experiencing failures from $(b,1)$ may result in the policy settling on Path-2 instead of Path-1. On the other hand, a safety shield that excludes the *right* action at $(b,1)$ will enable the agent to discover the

*Corresponding author.

sumanta.dey@iitkgp.ac.in (S. Dey); pallab@cse.iitkgp.ac.in (P. Dasgupta); soumya@cse.iitkgp.ac.in (S. Dey)

optimal path, Path-1, but will not make the policy safety aware in the absence of the shield.

In the literature, safety has been treated either as a discrete (safe/unsafe) binary, or as a continuous cost function. Methods such as Constrained Policy Optimization (CPO) [4] and Proximal Policy Optimization Lagrangian (PPO-Lagrangian)[5] are effective in reduction of safety infractions during online training, provided that safety is specified as a continuous cost function, that is, the environment returns a set of non-negative real valued costs for all the safety constraints, and a safety violation happens when the cumulative cost exceeds some defined threshold. In [6], the authors use a safety critic to guide the RL agent while learning to avoid unsafe instances. However, the safety critic must be trained in the pre-training phase with safe and unsafe states.

In [7], for model-based MDPs (Markov Decision Process), the authors propose the use of a safety-shield derived from Linear Temporal Logic (LTL) [8] specifications to restrict the RL agent (by action shaping) to explore within a safe region. The model-based assumption makes this method infeasible for many real-world applications where the transition function is unknown. Also, this method suffers from scalability issues due to the product MDP construction in the shield synthesis phase. Decision trees may be used as safety shields, as in [9], but this work assumes the existence of a next state predictor which limits its generalizability.

In a completely unknown environment, the initial RL policy as well as the safety shield are oblivious of safety. As the exploration begins, the safety shield begins to take shape with every safety infraction, but this knowledge does not influence the policy learning directly. This paper bridges this gap to accelerate the convergence to a safe RL policy. In this paper, we propose the Phasic Safety-oriented Policy Optimization (PSPO) framework to reduce safety infractions during *exploration*. PSPO works on a model-free MDP with continuous space, continuous/discrete actions, and binary safety assumptions. The approach uses periodic safety updates using an auxiliary policy trained from safety infractions detected by the safety shield, The main features of each period of PSPO are as follows (Fig. 2):

1. A safety shield model is learned on-the-fly from state-to-unsafe action mapping from past exploration. The safety shield is continuously updated to adapt to newly visited unsafe states. The exploration is based on the current policy, which is not updated at this phase.
2. The policy network is updated in two separate phases as follows:
    - *Policy Training Phase:* In this phase, the policy network is trained only with the explored state-actions, and the primary objective is to optimize the policy (π) with respect to the reward function.
    - *Safety Optimization Phase:* In this phase, an auxiliary policy (π^{aux}) is trained with the explored state-actions along with the safety shield masked unsafe actions. This auxiliary policy (π^{aux}) is then used to induce safe behavior in the main policy (π) through behavioral cloning.

Since the safety shield prevents repeated failures, the policy learning does not get pushed to conservative sub-optimal trajectories. The periodic safety updates from the auxiliary policy induces safe behavior right from the inception of training.

We provide experimental results on several Gym Environments [10]. Our results demonstrate considerable reductions in safety infractions and high returns in episodic rewards in all of these environments.

2. Preliminaries

We use Constrained MDP (CMDP) [2] to define our problem setting and we use the Proximal Policy Optimization (PPO) [11] method in the back-end.

Constrained MDP (CMDP). As defined in [2] a CMDP is a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu, \mathcal{C})$, where $\mathcal{S}$ defines the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow [0, 1]$ is a transition function/matrix. $\mathcal{R}$ refers to the reward function defined as $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ and $\gamma \in (0, 1)$ and μ are the discount factor and starting state distribution respectively. Finally, $\mathcal{C} = \{(c_i : \mathcal{S} \leq \chi_i \rightarrow \{0, 1\}|\chi_i \in \mathbb{R}), i \in Z\}$ is a set of safety constraints that the RL agent must follow in order to be safe. c_i denotes the i-th constraint function, and χ_i denotes the maximal allowable limit of non-satisfaction in terms of the expected probability of failure.

In this paper, we consider the safety as a binary function $\{0,\ 1\}$. Therefore, c_i returns SAFE (0) or UNSAFE (1). If any of the constraint functions return 1 (UNSAFE) for a given state, then the state is treated as unsafe.

Proximal Policy Optimization (PPO). Proximal Policy Optimization (PPO) [11] is an advantage-based policy-gradient reinforcement learning algorithm proposed by OpenAI. The objective of commonly used policy gradient (PG) methods have the following form:

$$L^{PG}(\theta) = \mathbb{E}_t\big[log_{\pi_\theta}(a_t|s_t)\hat{A}_t\big]$$

Here, $\hat{A}_t$ denotes the estimated advantage [12] and π_θ is the policy parameterized by θ. On the other hand, the

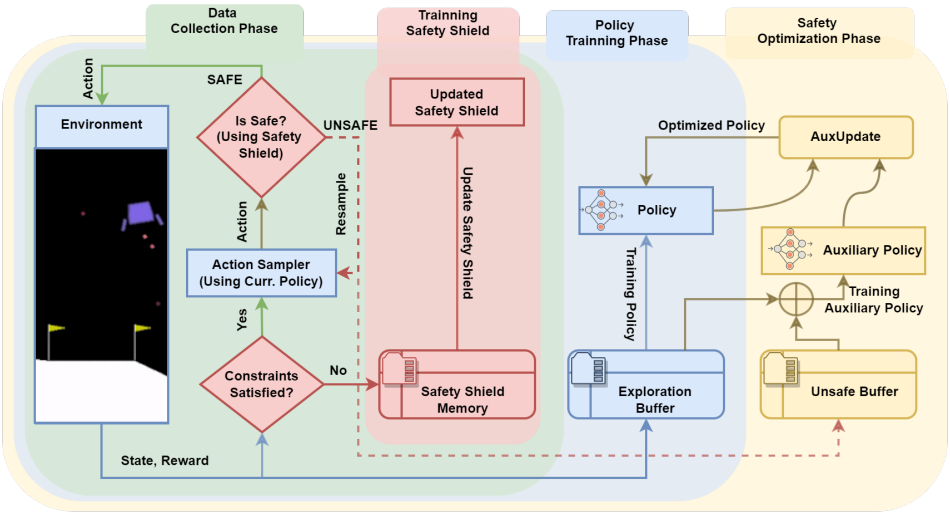


Figure 2: The diagram represents the overall process flow for one training epoch of the proposed PSPO framework. At first, in the Data Collection phase, the RL agent actively queries the environment based on the current policy and stores the State, Action, and Reward in Exploration Buffer under Safety Shield guidance. The Safety Shield also gets updated to incorporate new unsafe findings. Once the Exploration Buffer is filled, the Policy Training phase starts, and the Safety Optimization Phase follows it. This process repeats for each training epoch.

main objective in Proximal Policy Optimization (PPO) is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}_t\big[min(r_t(\theta)\hat{A}_t, \\ clip\big(r_t(\theta, 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\big]$$

Here, $r_t(\theta)$ denotes the probability ratio between new policy and old policy determined as $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. Finally, PPO restricted the policy update by using the clip method to directly limit the update range to [1 - ϵ, 1 + ϵ], where ϵ is a hyper-parameter that decides the clipping interval. With the above small change in the objective function, the PPO methods provide better stability and reliability over the vanilla policy gradient implementation.

3. Ideation

The task of inducing safe behavior into a RL policy requires careful balancing between safety and optimality. In an unknown model free environment an RL agent with limited domain knowledge will reach unsafe states during exploration, but it should ensure that it does not repeat the same mistakes. Relying solely on penalizing the agent for safety infractions may push the agent away from optimal paths that are on the border of unsafe regions, which then results in longer convergence times and sub-optimal policies. Learning safety shields from failures protects against future violations, but does not make the policy safety aware, and the agent continues to rely on the safety shield.

Our goal is to induce safe behavior by *correcting* a policy learned on the basis of reward. The pivot of the whole approach is a safety shield which gets updated whenever failures occur. The agent explores with the safety shield in place, and the trajectories are recorded. During this exploration, some of its actions may be thwarted by the safety shield – whenever this happens, the safety shield notes the exceptions. These exceptions are used to train an auxiliary policy, which essentially captures the behaviors in which we would like the current policy to behave differently. The auxiliary policy is therefore used periodically to update the current policy, thereby inducing safe actions in the relevant states, *without* affecting the rest of the policy. This ensures proximality with the optimal policy chosen by the safety agnostic learning algorithm.

An important benefit of the proposed approach is that the policy learning and the safety augmentation are separate phasic components. Therefore the method is adaptive to changes in either of the components. We can plug in any learning algorithm for the former, and handle additional safety corners when the agent reaches them.

Problem Statement. *Given a CMDP, the objectives of our proposed framework are as follows: 1) Learning the conditions that led the RL agent to failures from the past exploration history and later guide the RL agent through action masking to avoid the repetitive unsafe explorations. 2) Updating the policy network to incorporate the safety guidance from the previous step while minimally affecting the currently learned policy.*

4. Phasic Safety-Oriented Policy Optimization

This section discusses the overall process flow of our proposed Phasic Safety-oriented Policy Optimization (PSPO) framework. Figure 2 depicts the main components in this flow, and will be used as a reference in each of the following three phases:

- *Trajectory Collection:* This is done through exploration in the presence of the safety shield. Note that the safety shield is also learned/refined during the exploration. This exploration also collects the cases where the safety shield blocks actions proposed by the agent based on it's existing policy.
- *Policy Training Phase:* In this phase any standard algorithm may be used to train a policy based on the trajectories collected by the agent during exploration.
- *Safety Optimization Phase:* This involves the preparation of an auxiliary policy based on the blocked actions noted during exploration. The training of the auxiliary policy can work concurrently with the policy training. At the end of each training epoch, the auxiliary policy is used to update the learned policy.

We shall elaborate each of these phases now. Algorithm 1 summarizes the implementation of the proposed approach.

It may be noted that we address the problem in the online training setting only, with the goal of reducing safety infractions without distracting the algorithm for policy learning based on rewards.

4.1. Adaptive Safety Shield Framework

In RL, safety shields (coined by [7]) are used to block unsafe actions during exploration. Traditional safety shields deal with MDPs with discrete action space and model-based assumptions, which does not favor our model-free environment. Instead, we propose an Adaptive Safety Shield framework to *learn* a Safety Shield (SS) with the explored state-to-actions data collected on-the-fly during exploration. The explored state-action pairs are labeled based on the constraint violations in the next state during exploration.

$$(s_t, a_t) = \begin{cases} \text{UNSAFE} & \text{if } \exists \text{ i } \; c_i : s_{t+1} \rightarrow 1 \\ \text{SAFE} & \text{otherwise} \end{cases}$$

It is possible to bootstrap the initial safety shield based on prior knowledge on domain safety. It is also possible to start in the absence of prior knowledge.

In our implementation, the safety shield is a ML model trained to classify state-action pairs as safe/unsafe

Algorithm 1 Phasic Safe Policy Optimization (PSPO)

1: **for** `epoch = 1, 2, ...` **do**
2: `Init empty exploration buffer EB`
3: `Init empty unsafe buffer UB`
4: **%Data Collection Phase**
5: **for** `t = 1, 2, ..., EB.size` **do**
6: $a_t^{safe} = a_t \leftarrow SampleAction(\pi, s_t)$
7: **for** `iteration = 1,...,`N_{max_samp} **do**
8: **if** $SS.predict([s_t, a_t]) \leq \epsilon$ **then**
9: $a_t^{safe} \leftarrow a_t$
10: break
11: **else**
12: UB.append($s_t, a_t, PENALTY$)
13: $a_t \leftarrow SampleAction(\pi, s_t)$
14: **end if**
15: **end for**
16: $s_{t+1}, r_t \leftarrow env.step(a_t^{safe})$
17: **if** $s_{t+1} \not\models \varphi$ **then**
18: UB.append($s_t, a_t, PENALTY$)
19: EB.append($s_t, a_t, PENALTY$)
20: $SS.memory.append([s_t, a_t], UNSAFE)$
21: env.reset(); **%Start a new episode**
22: **else**
23: EB.append(s_t, a_t, r_t)
24: $SS.memory.append([s_t, a_t], SAFE)$
25: **end if**
26: **%Training Safety Shield**
27: Update SS after every N_{ss_update}-episodes
28: **end for**
29: **%Actor-Critic Training Phase**
30: `Perform rollouts on EB {`
31: Optimize θ_π wrt Policy Loss ($L^{clip} + \beta_s S[\pi]$)
32: Optimize θ_V wrt Value Loss (L^{value})
33: }
34: **%Auxiliary Policy Training**
35: `Perform rollouts on (EB+UB) {`
36: Optimize $\theta_{\pi^{aux}}$ wrt Policy Loss ($L^{clip} + \beta_s S[\pi]$)
37: }
38: **%AuxUpdate Phase**
39: `Perform rollouts on (EB+UB) {`
40: Optimize θ_π wrt Policy Distance (π, π^{aux}))
41: }
42: **end for**

(Algo. 1: Line 26-27). With new explorations, the model is updated with new state-action pairs.

The variable N_{ss_update} (Algo. 1: Line 27) is a hyperparameter that denotes the number of episodes after which the shield update is performed, that is, it controls the update frequency.

This safety shield is used to predict the probability (Algo. 1: Line 8) of reaching an unsafe state for a state-

Table 1
Gym environments with the respective safety criteria and the hyper-parameters

Environment Name	Safety Criteria	ϵ	N_{SS_update}	Penalty
Cartpole-v0	(-2.4 <**Position** <2.4) $\wedge$	1.00	100	-10
InvertedPendulam-v0	(-2.0 <**Momentum** <2.0) $\wedge$ (-0.2 <**Angle** <0.2)	0.90	200	-10
LunarLander-v2	(-0.2 <**PosX** <0.2) $\wedge$	0.85	50	-100
LunarLanderContinuous-v2	((**PosY** <0.1) $\rightarrow$ (**Angle** >-1 $\vee$ **Angle** <1))	1.00	50	-200

action pair proposed by the RL agent. If the predicted unsafe probability is less than the defined safety bound, ϵ, then the action is allowed. Otherwise, the proposed action is stored in an *unsafe buffer* (UB) (Algo. 1: Line 12) and the RL agent is asked to sample another action (Algo. 1: Line 13). This loop (Algo. 1: Line 7-15) continues until the safety shield finds the sampled action to be safe, or if the sample count exceeds a defined threshold, N_{max_samp}. In the latter case, the exploration continues with the first action proposed by the RL agent.

4.2. Policy Training Phase

We use the standard Proximal Policy Optimization (PPO) algorithm for learning the policy for the RL agent (Algo. 1: Line 29-33). This phase in shown with blue background in Fig. 2. We consider only binary safety constraints. Hence, whenever the RL agent hits an unsafe state that violates the safety constraints, we end the episode. In general, in a reward shaping environment, an RL agent can repeatedly visit a previously explored unsafe state. In our case, this is prevented by the Adaptive Safety Shield.

4.3. Safety Optimization Phase

In this phase, an alternative policy network, called *auxiliary policy network*, is trained with traces from both the *exploration buffer* (EB) and the *unsafe buffer* (UB) (Algo. 1: Line 34-37). Then the PPO actor network is updated through behavioral cloning with the auxiliary policy network for the states in the unsafe buffer. Such updates may alter the learned policy distribution for the other states in the PPO actor network. Therefore to reduce such interference, we also consider the policy distributions for the states present in the exploration buffer (Algo. 1: Line 38-41). Hence the overall objective of AuxUpdate is:

$$L^{\pi}(\theta) = KL\_div\Big(\pi(.|[EB+UB]), \Big[\pi(.|[EB \backslash UB]) + \pi^{aux}(.|UB)\Big]\Big)$$

Here, the + and / signs are used to indicate list concatenation and subtraction operation, respectively. We use KL-divergence as the distance metric between the distributions. $\pi(.|[EB+UB])$ returns the action distributions of the states presented in the exploration buffer (EB) and the unsafe buffer (UB), $\pi(.|[EB \backslash UB])$ returns the action distributions of the states presented in the exploration buffer (EB) but not the unsafe buffer (UB). Finally, $\pi^{aux}(.|UB)$ returns the action distributions of the states present in the unsafe buffer (UB).

5. Experimental Setup

We provide empirical support to the following claims through experiments conducted on different gym environments with continuous and discrete action spaces:

- The PSPO approach reduces the number of failures, and
- The PSPO approach marginally affects the primary learning objective.

All the experiments were run on a machine with Ubuntu 20.04, Intel i7 processor, and GeForce RTX 2080-Ti Graphics unit.

We tested our framework against four different gym environments, two of which are with discrete action space and the other two with continuous action space. By default, the gym environments are not safety constrained. We have defined the custom binary safety constraints for all the environments following [13]. The name of the environment with associated safe constraints are given in Table 1. We consider the safety constraints such that few are perfectly aligned with the original goal of the underlying Gym environment, and few aren't exactly aligned with the original goal.

Cart Pole Environment: The cart pole environment is taken from Gym Classic control environments and is an environment with discrete action space [0, 1]. The aim is to keep the pole over the cart without falling by taking actions 0 and 1. We have considered the following set of safety constraints:

1. The cart **Position** should remain within -2.4 or +2.4
2. The cart **Momentum** should not be lesser than -2.0 or greater than 2.0
3. The pole **Angle** should not be greater than 0.2

Among these, the third safety constraint is directly aligned with the original goal of the cart pole environment, whereas the remaining two are not exactly aligned with the original goal.

Inverted Pendulum Environment: This environment is taken from the Gym MuJoCo environments. This environment is similar to the Cart Pole environment, except the action space of this environment is continuous. Here the pole on top of the cart is controlled by applying a force between (-1, +1) to the cart to prevent the pole from falling over. The safety constraints are identical to those for Cart Pole environment.

Lunar Lander Environment: The Lunar Lander environment is taken from the Box2D environments. We have applied our framework to both the continuous and discrete versions of the environment. This environment aims to land the lunar lander smoothly on the helipad marked with two flags by controlling the thrust of the rocket engines on the lander's left, right, and bottom. For this environment we consider the following set of safety constraints.

1. The lander should land only on the helipad, or the X-position **(PosX)** of the lander must be within [-0.2,+0.2].
2. The lander tilt **Angle** should not be beyond -1 or +1 if the lander Y-position **(PosY)** is less than 0.1 and the lander is just over the helipad.

Baselines. We have used the standard **PPO** with negative reward for constrained violation as Baseline 1 (**BASE1**), **SAC** (Soft Actor-Critic) [14] with negative reward for constrained violation as Baseline 2 (**BASE2**), and **VPG** (Vanilla Policy Gradient) [15] with negative reward for constrained violation as Baseline 3 (**BASE3**). We have implemented all baselines on OpenAI's Spinningup library[16]. Negative rewards on constraint violations are provided to all the baselines, and the proposed PSPO framework. The SAC algorithm does not support discrete action space; hence, BASE2 is not considered in Cart Pole and Lunar Lander (Discrete) environments.

There are other approaches for safe exploration in RL that are available. For example, Constrained Policy Optimization (CPO) and PPO-Lagrangian. However, both these methods assume safety as a continuous cost function that is returned by the environment, like the reward generated each time the agent applies an action. In our setup, we have considered safety as a binary function of a state.

Implementation Details. For the environments, we considered using Random Forest [17] based on the ensemble method as Adaptive Safety Shield(SS). The advantages of using Random Forest for safety predictors are the following:

- ***Explainable.*** The individual decision trees of Random forest are explainable and can easily be interpreted and verified by human experts. They can also be factored into a rule-based system.
- ***Efficient.*** In a robust system, failures are rare, and thereby the number of updates needed for the safety shield declines rapidly.
- ***Augmentable.*** Individual Decision trees can easily be augmented with known safety constraints [18], which enables us to include safety constraints provided by domain experts.

One problem of using a random forest or decision tree as the Safety Shield, mainly in the case of discrete environments, is not considering the action in each decision branch. If an action from a state is found unsafe, then the safety shield could predict all the remaining actions as unsafe, whereas there could be an unexplored action that is safe. To avoid such issues, we use a simple re-labeling trick, where we use label (i+1) if the i-th action (starting index from 0) is unsafe, and we use label 0 if the action is safe. During prediction, if the (i+1)-th label prediction probability is greater than the provided safety threshold (ϵ), then the i-th action is considered unsafe; otherwise, the action is considered safe.

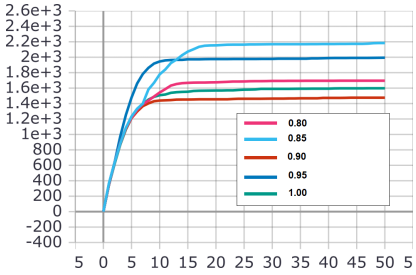
We use the standard PyTorch implementation of the PPO algorithm provided in OpenAI's Spinningup library as the Policy Learner. The Auxiliary Policy Network is constructed by replicating the PPO's Actor network. We have also done experiments of our PSPO method with different safety bound (ϵ) values to show the impact of the safety bound hyperparameter in the PSPO framework.

6. Results and Discussion

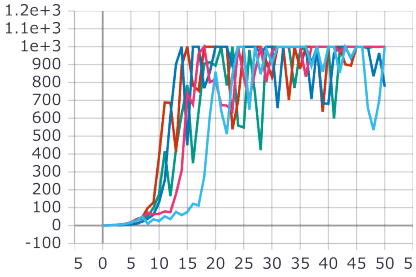
In this section we describe the results we have obtained in comparison to the baselines described in the previous section. Figure 4 shows Epoch (X-axis) vs Total Number of Safety Violations (Y-axis) for all four Gym environments. All the graphs show that the PSPO framework has significantly fewer safety violations during training than the BASE1 method. These figures also show that the average number of safety violations per epoch eventually tends to zero, as the PSPO graphs are slowly entering the lag phase. In these figures, we did not consider the other two baselines due to their large number of safety violations. Table 2 shows the number of safety violations

Table 2
Contains the number of safety infractions vs. the total number of episodes for different baselines and PSPO framework for fixed number of training epochs. Episode count varies as number of steps is fixed for all epochs.

Algorithms Failed / (Total) Episodes	**BASE1 (PPO)**	**BASE2 (SAC)**	**BASE3 (VPG)**	**PSPO**
CartPole-v1 (100 Epochs)	813 / (1510)	-	5011 / (5054)	**715** / (1408)
InvertedPendulum-v2 (100 Epochs)	1807 / (2120)	2056 / (2397)	7038 / (7038)	**1505** / (1823)
LunarLander-v2 (200 Epochs)	552 / (1108)	-	2092 / (3682)	**421** / (1144)
LunarLanderContinuous-v2 (200 Epochs)	591 / (1417)	307 / (891)	1107 / (2282)	**263** / (1099)



(a) Epochs vs. Cumulative Safety Violation



(b) Epochs vs. Average Episodic Reward

Figure 3: (a)-(b) shows the Epoch vs. Cumulative Safety Violations and Average Episodic Return for Inverted Pendulum Environment.

or unsafe episodes (both are the same as we terminate an episode for safety violation) for PSPO along with all three baseline methods. For all four environments, it is clear that PSPO has the least number of safety violations. This supports our first claim, that the PSPO framework helps to reduce the number of failures.

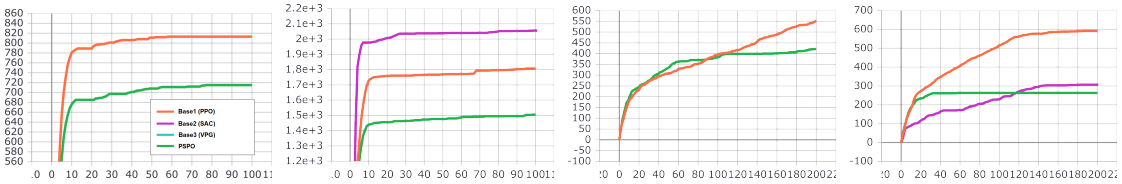
Figure 5 shows the Epoch (X-axis) vs. Average Return of the PSPO framework against the baselines for all four Gym environments. In Fig 5a and Fig 5b, the average return of PSPO is similar to BASE1 (PPO), and is better than the other baselines. In Fig 5d average return of PSPO is better than other baselines. This evidence supports our second claim.

Figure 3 shows the impact of the safety bound, ϵ, in the PSPO framework in terms of total safety violations and average episodic return. With higher values of this hyperparameter, ϵ, the safety shield intervenes/corrects fewer unsafe actions. In contrast, with the low values for ϵ, the safety shield unnecessarily corrects a higher number of safe actions. Hence, the safety bound, ϵ, controls the tradeoff between False Safe versus False Unsafe. In this case, the safety bound, $\epsilon = 0.9$ provided the best results, both for episodic returns and fewer safety violations.

7. Related Work

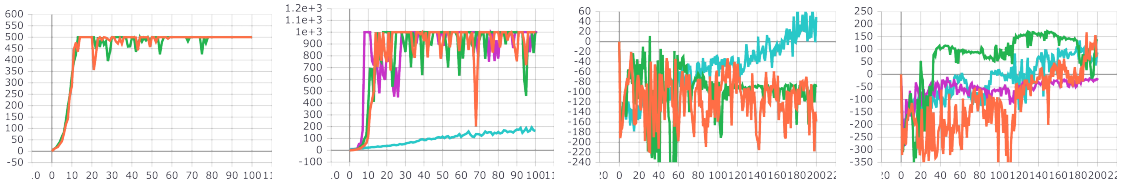
Several approaches for safe exploration in reinforcement learning problems have been studied theoretically and across application domains. The possibilities of safe exploration using safe baselines/backup policies whenever a safety violation is detected have been demonstrated in [19, 20]. On the other hand, in [21], the RL agent seeks expert advice for unknown/unsafe situations. Reward shaping techniques for safe RL has been studied in [22, 23, 24], where safety constraints are included in the reward function.

In [25, 26, 27, 28, 29, 7], prior domain knowledge is used to ensure safety during exploration. [25, 26] use a defined safe set of environment states and a safe backup policy for safe exploration in model-based MDP. In [27], the authors use a set of user demonstrations and an oracle to determine whether a state is safe or not. [28, 29, 7] use action shaping techniques to restrict unsafe actions. [28, 29] use model checking to verify an action for safety consequences against a specification before applying, whereas [7] constructs a reactive system based *safety shield* from the product automaton of the safety constraints, modeled as a temporal logic constraint [30] and uses the shield to block unsafe actions of the RL



(a) Cart Pole Environment (b) Inverted Pendulum (c) Lunar Lander (Discrete) (d) Lunar Lander (Continuous)

Figure 4: (a)-(d) shows the Epoch vs. Cumulative Safety Violation for different Gym Environments. SAC algorithm does not support discrete action space; hence, BASE2 is not considered in Cart Pole and Lunar Lander (Discrete) environments. While BASE3 has very large safety infractions counts, hence not shown in these figures.



(a) Cart Pole Environment (b) Inverted Pendulum (c) Lunar Lander (Discrete) (d) Lunar Lander (Continuous)

Figure 5: (a)-(d) shows the Epoch vs. Average Episodic Reward for different Gym Environments.

agent.

[4, 5, 31] consider safety as a continuous cost function, and the cumulative cost should be within a specified limit to be safe. In [6, 32], a safety-critic-based approach is proposed, where if the safety critic predicts an action as unsafe, the agent samples a different action. In [32], the safety critic is learned along with policy, whereas in [6], the safety critic is learned separately.

Another line of work can be found in [13], where authors propose a method to incorporate the safety in a learned policy by finding the counterexamples or the failure states and then modifying the policy for the corresponding states minimally.

8. Conclusion

We have presented a method for safe exploration in RL. We use an Adaptive Safety Shield to learn the state-to-unsafe action mapping from the past exploration and provide guidance to the RL agent to avoid repeating its mistakes. We have provided an auxiliary policy based update method to incorporate the safety guidance provided by the safety shield into the RL agent while minimally affecting the policy network for other state-actions. We have also presented various experiments which empirically validate that our method incurs fewer safety incidents while achieving higher or similar performance.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction (2018).

[2] J. Garcıa, F. Fernández, A comprehensive survey on safe reinforcement learning, Journal of Machine Learning Research 16 (2015) 1437–1480.

[3] P. Abbeel, Cs 188: Introduction to artificial intelligence, 2018. URL: https://inst.eecs.berkeley.edu/~cs188/fa18/, accessed: 2022-08-06.

[4] J. Achiam, D. Held, A. Tamar, P. Abbeel, Constrained policy optimization, in: International conference on machine learning, PMLR, 2017, pp. 22–31.

[5] A. Ray, J. Achiam, D. Amodei, Benchmarking Safe Exploration in Deep Reinforcement Learning (2019).

[6] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, C. Finn, Learning to be safe: Deep RL with a safety critic, CoRR abs/2010.14603 (2020). URL: https://arxiv.org/abs/2010.14603. arXiv:2010.14603.

[7] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu, Safe reinforcement learning via shielding, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[8] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, IEEE Computer Soci-

ety, 1977, pp. 46–57. URL: https://doi.org/10.1109/SFCS.1977.32. doi:10.1109/SFCS.1977.32.

[9] S. Dey, A. Mujumdar, P. Dasgupta, S. Dey, Adaptive safety shields for reinforcement learning-based cell shaping, IEEE Transactions on Network and Service Management (2022) 1–1. doi:10.1109/TNSM.2022.3194566.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, 2016.

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

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, arXiv preprint arXiv:1506.02438 (2015).

[13] B. Gangopadhyay, P. Dasgupta, Counterexample guided rl policy refinement using bayesian optimization, Advances in Neural Information Processing Systems 34 (2021) 22783–22794.

[14] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in: Y. Bengio, Y. LeCun (Eds.), 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016. URL: http://arxiv.org/abs/1506.02438.

[15] R. S. Sutton, D. A. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: S. A. Solla, T. K. Leen, K. Müller (Eds.), Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999], The MIT Press, 1999, pp. 1057–1063.

[16] J. Achiam, Spinning Up in Deep Reinforcement Learning (2018).

[17] T. K. Ho, Random decision forests, in: Proceedings of 3rd International Conference on Document Analysis and Recognition, volume 1, 1995, pp. 278–282 vol.1. doi:10.1109/ICDAR.1995.598994.

[18] S. Dey, P. Dasgupta, B. Gangopadhyay, Safety augmentation in decision trees, in: Proceedings of the Workshop on Artificial Intelligence Safety 2020 colocated with the 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI 2020), Yokohama, Japan, January, 2021, volume 2640 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2640/paper_13.pdf.

[19] S. Feghhi, E. Aumayr, F. Vannella, E. A. Hakim, G. Iakovidis, Safe reinforcement learning for antenna tilt optimisation using shielding and multiple baselines, arXiv preprint arXiv:2012.01296 (2020).

[20] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in ai safety, arXiv preprint arXiv:1606.06565 (2016).

[21] P. Abbeel, A. Coates, A. Y. Ng, Autonomous helicopter aerobatics through apprenticeship learning, International Journal of Robotics Research (IJRR) 29 (2010).

[22] T. J. Perkins, A. G. Barto, Lyapunov design for safe reinforcement learning, Journal of Machine Learning Research 3 (2002) 803–832.

[23] F. Berkenkamp, R. Moriconi, A. P. Schoellig, A. Krause, Safe Learning of Regions of Attraction for Uncertain, Nonlinear Systems with Gaussian Processes, 2016 IEEE 55th Conference on Decision and Control, CDC 2016 (2016) 4661–4666. doi:10.1109/CDC.2016.7798979. arXiv:1603.04915.

[24] Y. Chow, O. Nachum, E. Duenez-Guzman, M. Ghavamzadeh, A lyapunov-based approach to safe reinforcement learning, arXiv preprint arXiv:1805.07708 (2018).

[25] T. Koller, F. Berkenkamp, M. Turchetta, A. Krause, Learning-based model predictive control for safe exploration, in: 2018 IEEE conference on decision and control (CDC), IEEE, 2018, pp. 6059–6066.

[26] F. Berkenkamp, M. Turchetta, A. Schoellig, A. Krause, Safe model-based reinforcement learning with stability guarantees, Advances in neural information processing systems 30 (2017).

[27] B. Thananjeyan, A. Balakrishna, U. Rosolia, F. Li, R. McAllister, J. E. Gonzalez, S. Levine, F. Borrelli, K. Goldberg, Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks, IEEE Robotics and Automation Letters 5 (2020) 3612–3619.

[28] N. Fulton, A. Platzer, Safe reinforcement learning via formal methods: Toward safe control through proof and learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[29] A. Nikou, A. Mujumdar, M. Orlic, A. V. Feljan, Symbolic reinforcement learning for safe ran control, arXiv preprint arXiv:2103.06602 (2021).

[30] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT press, 2008.

[31] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, M. Ghavamzadeh, Lyapunov-based safe policy optimization for continuous control, arXiv preprint arXiv:1901.10031 (2019).

[32] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, A. Garg, Conservative safety critics for exploration, in: International Conference on Learning Representations, 2021. URL: https://openreview.net/forum?id=iaO86DUuKi.