Safe RL Algorithms: CPO and P3O

Daniel Alvarez[1], **Surabhi Gade**[2]

[1]Northeastern Universiy
[2]Northeastern Universiy
alvarez.d@northeastern.edu, gade.su@northeastern.edu

Abstract

This project aims on training an autonomous agent to navigate safely in environments with obstacles and hazards. The primary goal is to achieve optimal performance while rigorously adhering to predefined safety constraints. To accomplish this, we implemented and evaluated two state-of-the-art reinforcement learning algorithms proposed in distinct research papers: Constrained Policy Optimization (CPO) [1] and Policy-on Policy-off Policy Optimization (P3O)[3].

The experiments were conducted using the Safety Gymnasium environment, which is specifically designed for reinforcement learning tasks that emphasize safety. CPO directly incorporates safety constraints into policy updates, while P3O combines on-policy and off-policy optimization to balance learning efficiency and constraint satisfaction. The agents were trained to maximize reward while minimizing the cost of unsafe behavior. Experimental results demonstrated that both approaches achieved comparable improvements in safe navigation, with CPO showing superior constraint adherence and P3O excelling in computational efficiency. This work highlights the potential of safety-constrained reinforcement learning for practical applications requiring reliable and safe decision-making. The implementation of our project, including training scripts and example evaluation plots, is available on GitHub. The repository can be accessed at:

Github repository —
https://github.com/danielalvarez0/RL-5180-Project

Introduction

Deep reinforcement learning has achieved remarkable success in high-dimensional control tasks but often assumes unrestricted exploration. This assumption can be unsafe or impractical in many real-world scenarios, such as robotics, where safety constraints are critical. Existing approaches to constrained reinforcement learning either lack guarantees for constraint satisfaction during training or are computationally prohibitive for complex, high-dimensional policies. The paper CPO presents CPO as the first algorithm that ensures policy updates respect constraints throughout training, making it particularly suitable for real-world applications.

In Safe RL, the challenge lies in balancing the maximization of cumulative rewards with the satisfaction of safety constraints. For this purpose, we explored two advanced Safe RL algorithms: Constrained Policy Optimization (CPO) and Policy-on Policy-off Policy Optimization (P3O). CPO is a general-purpose algorithm designed for constrained RL, offering guarantees of near-constraint satisfaction during every training iteration. This makes it particularly suitable for high-dimensional control tasks, where maintaining safe behavior throughout the training process is critical. P3O, on the other hand, addresses limitations in other algorithms that struggle to efficiently update policies while satisfying hard constraints. Additionally, P3O can be extended to scenarios involving multiple constraints and agents, making it a versatile solution for complex environments.

To evaluate these algorithms, we used the Safety Gymnasium environment, a standardized platform for training agents to navigate safely in the presence of obstacles. By combining theoretical safety guarantees with practical implementations, this project aims to demonstrate its applicability to real-world problems. Our work highlights the potential of Safe RL techniques to enable reliable and safe decision-making in environments where safety is non-negotiable.

Background

Reinforcement Learning (RL) is a machine learning paradigm where agents learn to make decisions by interacting with an environment to maximize cumulative rewards. The agent observes the environment's state, takes an action, and receives feedback in the form of a reward and the next state. Over time, the agent learns a policy—a mapping from states to actions—that maximizes its expected rewards. However, traditional RL focuses solely on optimizing rewards without considering constraints, which limits its applicability in safety-critical domains.

Safe Reinforcement Learning (Safe RL) extends RL by incorporating constraints that the agent must satisfy while learning. These constraints can be defined over state variables, actions, or cumulative measures such as energy consumption or collision avoidance. Instead of designing behavior solely through a reward function, Safe RL explicitly separates the reward function from constraints. This distinction allows for more precise specification of safety requirements and improves the interpretability and reliability of trained agents.

The paper CPO[1] builds on the framework of Constrained Markov Decision Processes (CMDPs), which extend standard MDPs with constraints on auxiliary costs. CMDPs ensure that policies satisfy these constraints while maximizing rewards. Previous methods, such as primal-dual approaches and heuristic algorithms, have limitations in either scalability or guarantee of constraint satisfaction during training. Trust region methods and policy gradient techniques form the theoretical underpinnings of CPO, enabling efficient and safe updates.

2. Policy-on Policy-off Policy Optimization (P3O) is an algorithm designed to overcome the limitations of CPO in terms of computational efficiency and scalability to multi-constraint or multi-agent scenarios. P3O combines on-policy and off-policy methods, leveraging the stability of on-policy updates and the sample efficiency of off-policy updates. This hybrid approach enables efficient policy learning while satisfying constraints.

Safety Gymnasium is the environment used for this project. It provides a suite of benchmarks specifically designed for Safe RL. Agents operate in customizable environments with obstacles, goals, and safety constraints, making it an ideal testbed for evaluating algorithms like CPO and P3O. By using Safety Gymnasium, we ensure that our experiments are reproducible and directly relevant to real-world safety-critical scenarios.

This project builds on the foundation of Safe RL and constrained optimization to explore how advanced algorithms like CPO and P3O can be effectively applied to train safe and efficient agents in complex environments.

Related work

Reinforcement learning (RL) has achieved remarkable success in solving high-dimensional control tasks, such as Atari games, robotic locomotion, and strategic board games like Go. While RL enables agents to learn optimal policies through trial and error, unrestricted exploration during learning can lead to unsafe behaviors in real-world applications. Addressing this issue requires algorithms that enforce safety constraints while optimizing performance.

The Constrained Markov Decision Process (CMDP) framework (Altman, 1999)[2] provides a formal approach for incorporating safety into RL by introducing constraints on auxiliary costs. Methods such as primal-dual optimization also have been proposed to solve CMDPs, ensuring convergence to constraint-satisfying policies. However, these methods lack guarantees for constraint satisfaction during the learning process, making them unsuitable for safety-critical tasks.

To address this limitation, Achiam et al. (2017)[1] introduced Constrained Policy Optimization (CPO), the first policy search algorithm for continuous CMDPs that guarantees near-constraint satisfaction at every step of learning. CPO leverages trust region optimization (Schulman et al., 2015)[5] to derive a theoretically justified update rule that simultaneously maximizes rewards and enforces constraints. This approach has been successfully applied to high-dimensional tasks, such as simulated robotic locomotion, where safety constraints are critical.

While safe exploration is a primary focus of CPO, efficient use of data remains a broader challenge in RL. Traditional algorithms can be classified as either on-policy or off-policy. On-policy methods, such as Proximal Policy Optimization, prioritize stability and are easier to tune but are sample inefficient due to their inability to reuse past data. Off-policy algorithms, such as Deep Q-Networks (Mnih et al., 2015) and Deep Deterministic Policy Gradients (Lillicrap et al., 2016), improve sample efficiency by utilizing replay buffers but suffer from instability due to biased updates.

To bridge the gap between stability and efficiency, Fakoor et al. (2019)[3] proposed Policy-on Policy-off Policy Optimization (P3O). P3O combines on-policy updates for stability with off-policy updates for sample efficiency, using importance sampling (IS) to correct for off-policy biases. Additionally, P3O automatically tunes hyperparameters such as the IS clipping threshold and KL regularization coefficient using the normalized Effective Sample Size (ESS). This hybrid approach has demonstrated superior performance on discrete action tasks (e.g., Atari games) and continuous control tasks (e.g., MuJoCo environments).

These works collectively address critical challenges in reinforcement learning: safety during exploration (CPO) and efficient policy optimization (P3O). While CPO ensures constraint satisfaction throughout learning, P3O provides a framework for balancing stability and sample efficiency, making these algorithms foundational for advancing reinforcement learning in safety-critical and data-intensive applications.

Project Description/Methods

Problem Statement:

- **State Space** ($\mathcal{S}$): Agent's position(x,y), Orientation (yaw), Velocity of each wheel agent's lidar observations of goal, hazards and vases.
- **Action Space** ($\mathcal{A}$): Speed of right wheel and speed of left wheel.
- **Cost Function** (c_t): In our implementation, the exact cost function is defined by the aggregate cost signal provided by the Safety Gymnasium environment. (info['cost'])

$$c_t = \sum_{i=1}^{N} w_i \cdot I_i(\text{state}_t, \text{action}_t)$$

  Where:

  - N: The total number of unsafe elements in the environment (e.g., hazards, vases, gremlins).
  - w_i: A weight or penalty associated with the i-th unsafe element.
  - $I_i(\text{state}_t, \text{action}_t)$: An indicator function (or cost function) for the i-th unsafe element, which returns:
    - 1: If the agent's interaction with the unsafe element violates a constraint (e.g., entering a hazard zone).
    - 0: Otherwise.

1. Constrained Policy Optimization :

CPO is an advanced policy optimization algorithm that ensures near-constraint satisfaction at every policy update by solving a constrained optimization problem. It builds upon the Proximal Policy Optimization (PPO) framework by explicitly incorporating constraints.

For large or continuous Markov Decision Processes (MDPs), finding the exact optimal policy is computationally intractable due to the curse of dimensionality, as discussed by Sutton and Barto (1998)[6]. To address this challenge, policy search algorithms constrain the optimization to a parameterized subset of policies $\Pi_\theta \subseteq \Pi$, where policies are parameterized by θ, such as neural networks with fixed architectures. In local policy search, the optimization iteratively updates the policy π_k to maximize the objective $J(\pi)$ within a localized neighborhood of π_k, as defined by a distance measure D and step size δ (Peters & Schaal, 2008)[4]:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} J(\pi) \quad \text{s.t.} \quad D(\pi, \pi_k) \leq \delta.$$

When the objective is linearized around π_k, it can be expressed as $J(\pi_k) + g^\top(\theta - \theta_k)$, where g is the policy gradient. The standard policy gradient update is obtained by setting $D(\pi, \pi_k) = \|\theta - \theta_k\|_2$.

For Constrained Markov Decision Processes (CMDPs), policy updates must additionally ensure feasibility under the constraints. This extends the optimization to the feasible policy space $\Pi_\theta \cap \Pi_C$:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} J(\pi) \quad \text{s.t.} \quad J_{C_i}(\pi) \leq d_i, \ \forall i \in \{1, \ldots, m\},$$

$$D(\pi, \pi_k) \leq \delta.$$

Directly implementing this update poses practical challenges, particularly the requirement to evaluate constraint functions for feasibility. In high-dimensional settings, sampling-based methods are typically used for policy updates, but this approach often necessitates off-policy evaluation, a notoriously difficult problem.

To address these challenges, we implemented the Constrained Policy Optimization (CPO) algorithm inspired by the work of Achiam et al. (2017)[1]. CPO introduces a principled approximation to the CMDP policy update by leveraging surrogate functions. These surrogate functions are computationally efficient to estimate using samples collected on the current policy π_k and serve as effective local approximations of both the objective and constraint functions.

In our implementation, we incorporated CPO's guarantees for near-constraint satisfaction and worst-case performance degradation, as derived through the theoretical bounds on policy divergence. Specifically, the algorithm employs trust region optimization to maintain updates within a bounded divergence while ensuring feasibility. This approach was further enhanced by adapting surrogate functions to minimize the likelihood of constraint violations.

Algorithm 1: Constrained Policy Optimization (CPO)

```
Require: Environment $env$, safety constraint $\delta$, discount
    factor $\gamma$, GAE parameter $\lambda$, max KL divergence max_kl
Ensure: Trained policy $\pi$
 1: Initialize policy $\pi$, value functions $V_r$ (rewards) and $V_c$
    (costs), and optimizers.
 2: for iteration $i = 1$ to $n\_iterations$ do
 3:     Collect Data:
 4:     Rollout trajectories in the environment to collect ob-
        servations, actions, rewards, and costs.
 5:     Populate buffers until batch size is reached.
 6:     Compute Advantages and Returns:
 7:     Use Generalized Advantage Estimation (GAE) to
        compute reward and cost advantages ($A_r$, $A_c$).
 8:     Compute discounted returns for rewards ($G_r$) and
        costs ($G_c$).
 9:     Update Value Functions:
10:     Update $V_r$ and $V_c$ by minimizing MSE loss between
        predictions and returns.
11:     Policy Update:
12:     Compute gradients of reward and cost advantages
        with respect to the policy.
13:     Solve for step direction using Conjugate Gradient
        (CG) with Fisher-Vector Product (FVP).
14:     Adjust step size to satisfy safety constraints and KL
        divergence.
15:     Update the policy parameters.
16:     Performance:
17:     Store average return and cost for this iteration.
18: end for
19: Output: Trained policy $\pi$
```

2. On-Policy Off-Policy Policy Optimization (P3O)

P3O is a hybrid approach to reinforcement learning that alternates between two modes of operation: learning directly from the current policy (*policy-on*) and utilizing off-policy data to improve sample efficiency (*policy-off*). This method balances exploration with the reuse of historical experience, aiming to leverage the benefits of both policy-gradient methods and value-based approaches.

Objective Function The objective of P3O is to maximize the value of the expected cumulative reward $J(\pi_\theta)$ where

$$J(\pi_\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

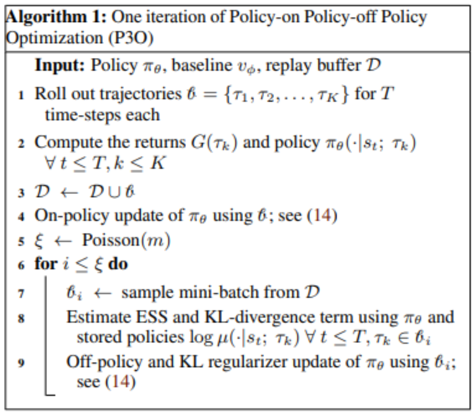
The two updates methods P3O uses are:

Policy-On When performing on-policy updates the current policy π_θ is modified using policy gradient methods. The gradient of $J(\pi_\theta)$ is approximated as:

$$\nabla_\theta J(\pi_\theta) \approx E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}(s_t, a_t)\right]$$

where $\hat{A}(s_t, a_t)$ is an estimate of the advantage

Policy-Off The off-policy updates are performed by taking a batch of previous experiences (s,a,r,s') and then estimating the value function $Q(s,a)$ via temporal difference learning:
$Q(s,a) = r + \gamma * max_{a'} Q(s', a')$.
Then optimize the policy π_θ by minimizing the KL divergence between the existing policy and one derived from $Q(s,a)$.



Algorithm 1: One iteration of Policy-on Policy-off Policy Optimization (P3O)

```
  Input: Policy $\pi_\theta$, baseline $v_\phi$, replay buffer $\mathcal{D}$
1 Roll out trajectories $\mathcal{B} = \{\tau_1, \tau_2, \ldots, \tau_K\}$ for $T$
    time-steps each
2 Compute the returns $G(\tau_k)$ and policy $\pi_\theta(\cdot|s_t;\ \tau_k)$
    $\forall\, t \leq T, k \leq K$
3 $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{B}$
4 On-policy update of $\pi_\theta$ using $\mathcal{B}$; see (14)
5 $\xi \leftarrow$ Poisson$(m)$
6 for $i \leq \xi$ do
7     $\mathcal{B}_i \leftarrow$ sample mini-batch from $\mathcal{D}$
8     Estimate ESS and KL-divergence term using $\pi_\theta$ and
        stored policies $\log \mu(\cdot|s_t;\ \tau_k)\ \forall\, t \leq T, \tau_k \in \mathcal{B}_i$
9     Off-policy and KL regularizer update of $\pi_\theta$ using $\mathcal{B}_i$;
        see (14)
```

Implementation

Initialization The algorithm begins by initializing the policy $\pi_\theta(a|s)$, which represents the probability distribution over actions a given a state s. This policy is parameterized by θ, typically represented by a neural network. Additionally, hyperparameters critical to learning are selected, including the learning rate α, the discount factor γ, the batch size N, and the total number of training episodes T. These parameters influence the pace of learning and the balance between short-term and long-term rewards.

Sampling Trajectories Once initialized, the agent interacts with the environment by following the policy π_θ. At the start of each episode, the environment is reset, and the agent samples actions based on the policy $\pi_\theta(a|s)$. For each time step t, the agent observes the current state s_t, selects an action a_t according to the policy, and receives a reward r_t and the subsequent state s_{t+1} from the environment. These interactions are recorded in a trajectory τ, comprising the sequence of states, actions, and rewards encountered during the episode. This trajectory collection continues until the episode terminates, either upon achieving a predefined goal or exceeding a maximum time horizon H.

Return and Advantage Computation After gathering trajectories, the algorithm computes the cumulative return G_t for each time step in the trajectory. This return encapsulates the total discounted reward from time t onward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}.$$

In practice, the use of a baseline, such as a value function approximator $V(s_t)$, can improve training efficiency. The baseline serves to reduce variance in the policy gradient estimation. The advantage function $\hat{A}(s_t, a_t)$ is computed as the difference between the cumulative return G_t and the baseline $V(s_t)$:

$$\hat{A}(s_t, a_t) = G_t - V(s_t).$$

This advantage reflects how much better or worse an action a_t performed compared to the baseline expectation.

Policy Gradient Estimation Using the sampled trajectories, the algorithm estimates the policy gradient, which represents the direction in parameter space that most increases the expected cumulative reward. The policy gradient is given by:

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}(s_t, a_t) \right].$$

This expression relies on the likelihood ratio trick, which allows the gradient to be computed using the log-probabilities of actions under the current policy.

Policy Update With the policy gradient estimated, the parameters θ of the policy are updated using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta).$$

This step shifts the policy parameters in the direction of higher expected reward, reinforcing actions that lead to favorable outcomes and reducing the likelihood of less effective actions.

Iterative Optimization The process of sampling trajectories, computing returns and advantages, estimating gradients, and updating the policy is repeated iteratively for a predefined number of episodes T. Over time, the policy improves, converging toward an optimal behavior that maximizes the expected reward.

Experiments

We designed experiments that are easy to interpret and motivated by safety. We used the predefined environments available in safety gymnasium library such as 'SafetyCarGoal2-v0" and "SafetyCarCircle1-v0" . These experiments were structured to address the following key research questions:

1. Which algorithm, between CPO and P3O, performs better under specific conditions, and what factors influence their effectiveness?
2. How can we choose the most suitable algorithm based on specific task requirements, including safety constraints, sample efficiency, and learning stability?
3. How do CPO and P3O respond to variations in hyperparameters or environmental conditions, and what tradeoffs emerge in terms of safety and reward optimization?

Experimental Setup

Environment 1

- **Environment Name:** `SafetyCarGoal2-v0`.

- The agent (a car-like robot(Red)) is tasked with navigating to a predefined goal(green) while avoiding obstacles(blue and cyan).
- Collisions with obstacles incur costs, enforcing the safety constraint.

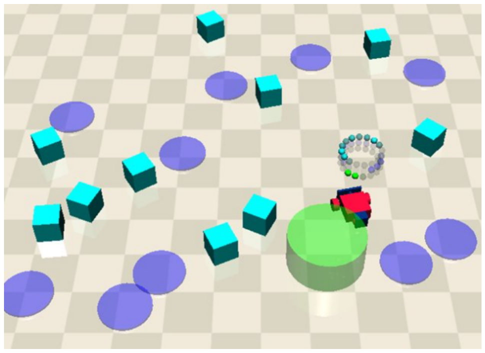


Figure 1: Env1: SafetyCarGoal2-v0

Environment 2

- **Environment Name:** `SafetyCarCircle1-v0`.
- Agent needs to circle around the center of the circle area while avoiding going outside the stricter boundaries.
- Crossing the boundary result in costs, emphasizing the importance of safe and efficient navigation within the area.

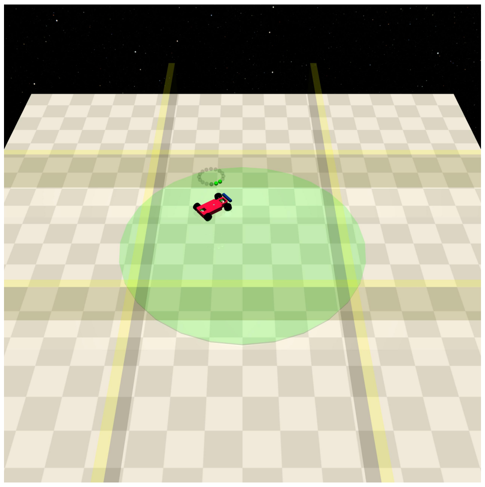


Figure 2: Env2:SafetyCarCircle1-v0

Training

- The agent was trained for 500 iterations, each collecting data through rollouts and updating the policy and value functions.
- Experiments were run using a fixed random seed (seed = 42) to ensure reproducibility.

1. CPO: Experiments and Results

Hyperparameters for CPO We selected 64 units per layer for the neural network, a standard choice in deep reinforcement learning, as it offers sufficient capacity without overfitting or excessive computational cost. For the discount factor (γ), we chose 0.99 because a value close to 1 promotes long-term planning, which is essential in Safety Gymnasium environments, while slightly reducing it prevents instability associated with infinite-horizon optimization. The GAE lambda provides stable learning by balancing bias and variance. A strict safety constraint ($\delta = 0.01$) was selected to minimize cost violations, which is crucial in safety-critical tasks. Additionally, a small maximum KL divergence (max_kl = 0.01) was used to ensure conservative policy updates, maintaining learning stability while preventing drastic changes that could compromise safety.safety constraints. 0.01 is a typical choice in trust-region-based methods like CPO and PPO.

As the first experiment, the agent was trained for varying numbers of iterations—specifically 5, 50, and 500—to observe the impact of training duration on cumulative returns and costs. During each configuration, data was collected through rollouts, and the policy and value functions were updated iteratively. This approach allowed us to analyze how the agent's performance evolved over time and the trade-offs between returns and adherence to safety constraints as training progressed.

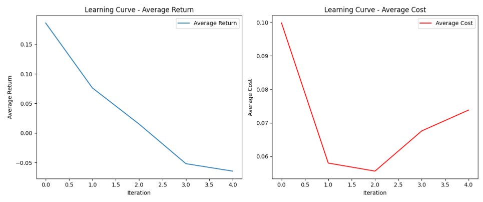


Figure 3: 5 iterations

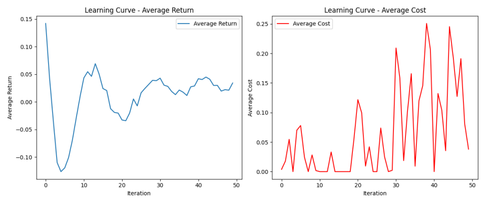


Figure 4: 50 iterations

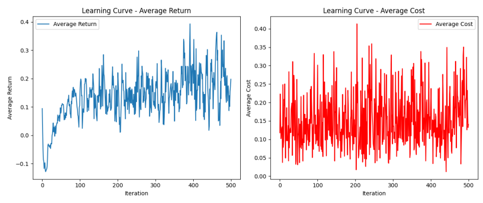


Figure 5: 500 iterations

In 5 iterations, the agent shows limited learning with declining returns and reduced costs. By 50 iterations, returns improve with fluctuations, while costs remain inconsistent; at 500 iterations, the agent achieves stable returns and better adherence to safety constraints.

For second experiment, we experimented with two learning rates, $\alpha = 1 \times 10^{-4}$ and $\alpha = 3 \times 10^{-4}$, to analyze their impact on the agent's performance. The lower learning rate demonstrated stable but slower convergence, while the higher learning rate accelerated learning with faster improvement in returns but exhibited greater variability in both performance and cost adherence. These results highlight the trade-off between learning speed and stability, and due to CPO's long convergence time(approx: 3 hrs), further exploration of learning rates was not pursued.

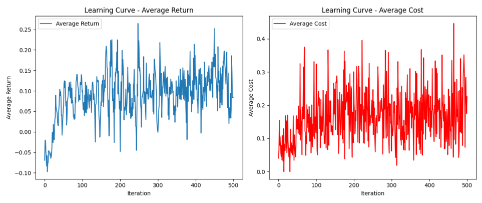


Figure 6: $\alpha = 1 \times 10^{-4}$

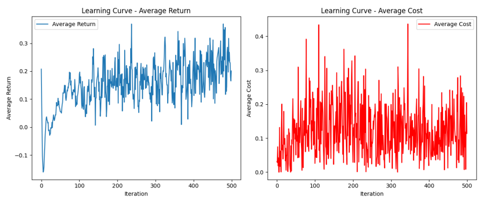


Figure 7: $\alpha = 3 \times 10^{-4}$

We experimented on different KL values such as 0.1 and 0.01. Due to time constraints we reduced the number of iterations to 100 for following results. We saw that KL = 0.01 resulted in a more stable and consistent improvement in the average return over iterations, with a noticeable upward trend and reduced fluctuations compared to KL = 0.1. In contrast, KL = 0.1 exhibited more oscillatory behavior in the average return, making it less stable. For the average cost, KL = 0.01 demonstrated slightly lower variance and smoother trends, although both KL values showed significant spikes in cost throughout the iterations. Overall, KL = 0.01 provided better performance stability and improved returns, making it a preferable choice in our experiments.

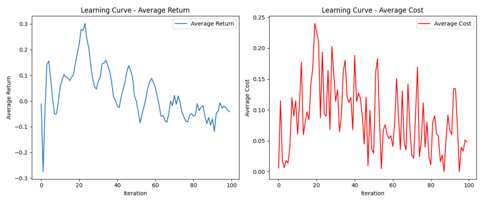


Figure 8: KL divergence: 0.1

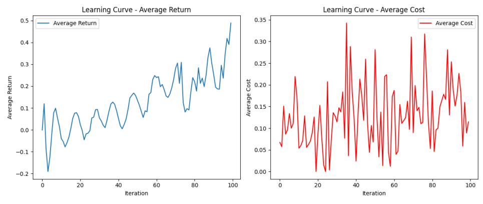


Figure 9: KL divergence: 0.01

We experimented with delta values of 0.1 and 0.01. While delta = 0.1 showed fluctuations in average return and high variance in average cost, delta = 0.01 provided a steadier improvement in return and smoother cost trends, indicating greater stability and effectiveness.

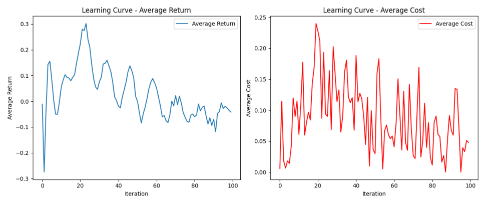


Figure 10: delta (δ)= 0.1

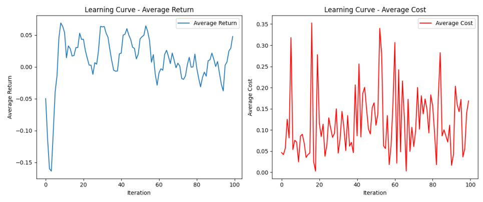


Figure 11: delta (δ)= 0.01

We saw that in the SafetyCarGoal2-v0 environment, the average return improved steadily over iterations but showed moderate fluctuations, while the average cost remained relatively high and variable, indicating challenges in achieving cost minimization. In contrast, in the SafetyCarCircle1-v0 environment, the average return increased significantly faster and reached higher values, showing better overall performance. However, this came at the expense of higher and more fluctuating average costs, highlighting the difficulty of balancing return and cost in this environment.

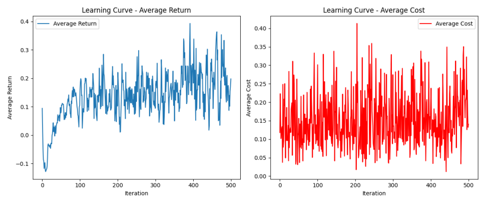


Figure 12: SafetyCarGoal2-v0

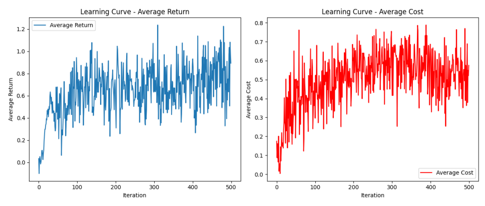


Figure 13: SafetyCarCircle1-v0

2. P3O: Experiments and Results

Hyperparameters The policy learning rate chosen was $3e-4$, the value learning rate was $1e-3$. For a discount factor .99 was chosen for the same reasons it was selected for CPO, it promotes long term planning. The dimensions of the hidden layers in policy and value networks is 64. Here again the choice was made because it is a standard value that is known to perform well. The cost factor chosen was 0.01. This was chosen because in early testing we found that if the c value was too large it would entirely prevent the agent from learning. This occurred when testing a c value of 0.04. The agent was trained on multiple times on the SafetyCarGoal2-v0 environment for 5, 50, and 500 iterations. After 4 iterations there was no visible improvement in the performance of the agent (note: all return and cost values have been normalized). After 50 iterations performance was still poor and very unstable, but during the last few episodes both performance and stability appear to have slightly improved. The total returns are on average better than at the start and while performance is still unstable the stability has increased.

The third test where the agent was trained on 500 iterations showed much clearer improvement. The average return improved by almost 0.2 by the end of the trial. this is close to a 50% increase over its performance for the first 100 iterations. The agent's has stability has also improved. The agent is still not truly stable, but the changes in return value form episode to episode have decreased significantly. The greatest change in returns between episodes in the last 100 iterations is 2.1. In the first 100 iterations that value is more than 6.

Costs did not show the same improvement that returns did. After 5 and 50 iterations there was no meaningful decrease in episode costs and even after 500 iterations the improvement was small. Something we observed which does not show up well on the plot is that in the final 100-200 iterations the final costs for an episode were extremely unstable. Approximately 50% of episodes had a cost value of 0. Another 37% had cost values over 80, and only the remaining 13% were in the 0 to 80 range. This is likely a result of cost not being directly constrained and only used to penalize an episodes final return value. The agent initially tries to avoid hazards, but after hitting one, it has a tendency to not correct its course and just continues forward, leading to many runs have no cost and a few have extremely high final costs.

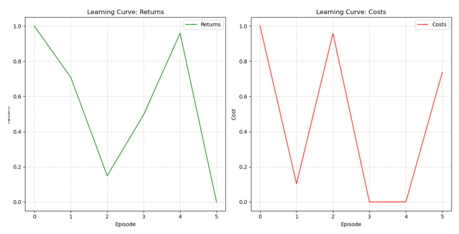


Figure 14: P3O 5 iterations

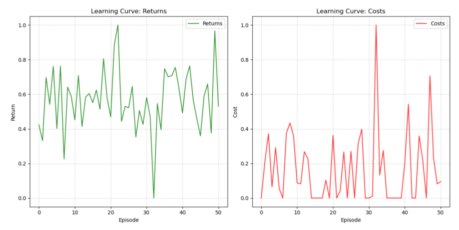


Figure 15: P3O 50 iterations

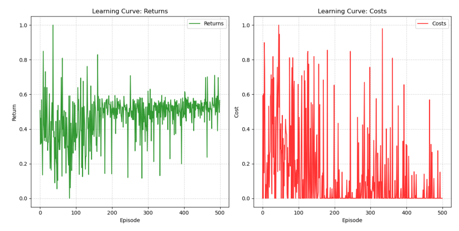


Figure 16: P3O 500 iterations

Conclusion

This project implemented and evaluated both algorithms for safe reinforcement learning in the environment. Based on the results obtained we would conclude that CPO performs better in tasks with strict safety constraints due to its explicit management of safety, making it ideal for safety-critical applications. P3O excels in tasks requiring faster learning and higher rewards, prioritizing flexibility and sample efficiency. Choosing between the two depends on task requirements: CPO is suited for environments prioritizing safety, while P3O fits scenarios emphasizing performance. Both are sensitive to hyperparameters and environmental variations, with CPO favoring safety over reward and P3O trading some safety for faster optimization. The decision should balance task-specific needs for safety, efficiency, and stability. Policy-on Policy-off Policy Optimization was implemented with moderate success, but also significant issues that are yet to be resolved at time of writing. The P3O algorithm was able to train an agent on the SafetyCarGoal2-v0 environment to reach the goal consistently and generally in an efficient manner. However, it did not succeed in training the agent to consistently avoid violating safety constraints. While training reduced the frequency of violations occurring and slightly reduced the magnitude of the violations when they did occur, the agent regularly

References

[1] Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained Policy Optimization. arXiv:1705.10528.

[2] Altman, E. 1999. Constrained Markov Decision Processes.

[3] Fakoor, R.; Chaudhari, P.; and Smola, A. J. 2019. P3O: Policy-on Policy-off Policy Optimization. arXiv:1905.01756.

[4] Peters, J.; and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, 21 4: 682–97.

[5] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust Region Policy Optimization. In Bach, F.; and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1889–1897. Lille, France: PMLR.

[6] Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.