Safety First: The Effects of Hyperparameter Tuning in Safe Reinforcement Learning

Sam Bradie, Lily Haas, Andrew Hong, Ian Mortensen,
Emma Roskopf, Varun Saini

Department of Computer Science, Carleton College

Abstract

In this paper, we study and report on the differences between two conventional reinforcement learning (RL) algorithms and their respective safety constrained versions. Specifically, we focus on two simulated environments: one in which a moving, car-like agent learns to minimize travel time toward some destination and another in which a humanoid agent learns to run. Both of these environments originate from the Farama Foundation's Gymnasium, and we accessed them through OmniSafe, an infrastructural framework for safe RL research. While baseline RL algorithms tend to incentivize agents to pursue riskier strategies to maximize reward, we found that constrained RL algorithms still allow for learning while simultaneously constraining safety violations. We performed experiments in which we changed various hyperparameters to determine which hyperparameters most positively impacted the performance of the algorithms. The most impactful hyperparameter was the number of steps simulated per epoch. By increasing this value, we observed a drastic improvement in the performance of our safety algorithms.

1 Introduction

The field of reinforcement learning (RL) has expanded significantly in the past decade along with significant advancements in neural network architecture and computational resources [15]. Powerful environment simulators like the Farama Foundation's Gymnasium have allowed RL professionals to model real-world scenarios [7]. In order to learn the best strategy for completing a task, a reinforcement learning agent must explore its environment, which can be potentially dangerous for entities in the environment or even for the agent itself. Still, safety is often not considered when maximizing expected rewards in these simulated environments. While an agent's risky actions within a computerized simulation might not have any consequences, applying these same naive RL techniques to real-world problems could have drastic ramifications.

1.1 Safe Reinforcement Learning

At the heart of reinforcement learning is the balance between exploration and optimization [30]. That is, when should the agent exploit its current knowledge to optimize its reward, and when should it act randomly[1] in order to experience new states—potentially yielding more reward—that

[1]We note that utilizing randomness is but one of many possible exploration strategies.

it would not otherwise have visited? However, exploration is risky; in practice, this could lead to dangerous consequences. For instance, a self-driving car cannot afford to learn the layout of a neighborhood by navigating recklessly, both for its own structural integrity, and for the safety of its surroundings. Safe RL aims to mitigate the inherent danger that comes with exploration. One such method is constrained reinforcement learning [4], which involves maximizing reward while also minimizing a cost factor that is associated with dangerous behavior. This cost is meant to be adjusted during training in order to discourage instances where total reward is high enough to justify repeatedly incurring a fixed cost. This way, an agent must learn to perform safe actions while maximizing its reward.

2 Background

2.1 Markov Decision Processes

Fundamentally, reinforcement learning involves an agent that learns by exploring its environment. In this way, the agent learns through its own process of trial and error, rather than acting based on training from a provided dataset, as in traditional machine learning. The agent's overarching goal through this process is to learn to take actions that maximize its long-term total reward. In particular, taking an action moves the agent to a new state for which it receives an associated reward for entering. These components—actions, states, and rewards—constitute the basis for a Markov decision process (MDP) [30]. For many environments, state transitions are stochastic. That is, there is some probability associated with the state that the agent will transition to when it takes an action. For this reason, we also include in our MDP formulation a function P specifying this state transition probability.

Definition 2.1 (Markov Decision Process)**.** We formally define an *MDP* as a 4-tuple (S, A, P, R), where its components are defined as follows:

- S is the set of states in the environment.
- A is the set of actions our agent can choose from.
- $P(s_{t+1}|s_t, a_t)$ specifies the probability of transitioning to the next state s_{t+1} given that our agent is currently in state s_t and is about to take action a_t.
- $R(s_t, a_t, s_{t+1})$ is the reward for a state transition from s_t to s_{t+1} after taking action a_t.[2]

Here, we use t to denote the current time step in our environment.

2.2 Policy Optimization

Solving an MDP entails algorithmically optimizing a "policy" for taking actions. For example, our agent should learn to prefer taking actions with low reward in order to position itself into a path of fruitful rewards (to this end, an optimal policy should enjoy delayed gratification). Formally, this is defined as a function π that returns the probability of taking an action given a particular starting state.

[2]For notational simplicity, we often omit s_{t+1} from this formulation and instead conceptualize rewards as being earned as an immediate consequence of taking an action.

Sutton and Barto describe two methods for performing this task: value function estimation and direct policy search [30]. The former involves either calculating the optimal state-value function or optimal action-value function for a given Markov decision process, which can be estimated using value iteration or policy iteration. While we do not discuss these methods in this paper, value functions later play a role when optimizing our policy gradient equation. We therefore provide a definition below.

Definition 2.2 (Value Function). We define the *value function* of a state s as the expected sum of future rewards when starting in state s and acting based on the policy π. We write this formally as

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} R(s_t, a_t) \,\middle|\, s_0 = s\right], \tag{2.1}$$

where $\mathbb{E}_{\pi}$ denotes the expected value[3] over the distribution induced by the policy π.

The latter method involves searching in the policy space for an optimal policy, which can be achieved with gradient-based and gradient-free methods, depending on whether the objective function is differentiable with respect to the parameters of the policy. For the purposes of this paper, we focus primarily on deep reinforcement learning, which aligns most closely with gradient-based methods found in the latter framework.

3 Policy Gradient Methods

This section is modeled after the third video in Dr. Pieter Abbeel's lecture series on deep reinforcement learning [1, 2].

3.1 Formulating the Problem

In deep reinforcement learning, we generally represent a policy π with a neural network. At a high level, the input to the network is a state, and the output is the chosen action to be taken.[4] The neural network is parameterized by a vector θ, which encodes internal weights and biases, so we will therefore use the notation π_θ to represent a parameterized policy of this form.

When finding an optimal policy for an MDP, we solve the following problem:

$$\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right], \tag{3.1}$$

where T is the number of time steps in a given state-action sequence, and $\mathbb{E}_{\pi_\theta}$ is an expectation taken over the probability distribution induced by π_θ. In the interest of simplifying notation, we now introduce the notion of a trajectory.

[3]The notion of expected value is explained in more detail in Appendix A.1.

[4]More specifically, suppose states are n-dimensional, and actions are m-dimensional. Then, the input is an n-dimensional vector, and the output comprises two m-dimensional vectors that respectively represent the means and standard deviations of m normal distributions from which an action can be sampled. Due to this sampling, policies modeled by neural networks in this manner are non-deterministic.

Definition 3.1 (Trajectory)**.** A *trajectory* τ is formally defined as a tuple of states and actions $(s_0, a_0, \ldots, s_{T-1}, a_{T-1})$. The *reward* of a trajectory is defined as follows:

$$R(\tau) \coloneqq \sum_{t=0}^{T-1} R(s_t, a_t).$$

Using this new notation, Equation 3.1 can be simplified to the following equation:

$$\max_{\theta} \mathbb{E}_{\pi_\theta} \left[R(\tau) \right]. \tag{3.2}$$

In theory, a trajectory is any sequence of consecutive state-action pairs. In practice, however, we often use the terms *episode* and *trajectory* synonymously when discussing tasks that are simulated in fixed-length episodes, like those studied in this paper. When we assume that the length of a trajectory is equivalent to the length of an episode, solving Equation 3.2 becomes equivalent to maximizing an agent's expected returns for each episode.

3.2 Maximization by Gradient Ascent

In Section 3.1, we introduced Equation 3.2. To maximize this objective function, we can perform gradient ascent on θ. We apply the definition of expected value (Appendix A.1) to obtain the following identity:

$$\mathbb{E}_{\pi_\theta} \left[R(\tau) \right] = \sum_{\tau} P_{\pi_\theta}(\tau) R(\tau),$$

where $P_{\pi_\theta}(\tau)$ is defined as the probability of the trajectory τ occurring while following the policy π_θ. Applying this identity, we calculate the gradient as follows:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta} \left[R(\tau) \right] &= \nabla_\theta \sum_{\tau} P_{\pi_\theta}(\tau) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P_{\pi_\theta}(\tau) R(\tau) \\
&= \sum_{\tau} \frac{P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)} \nabla_\theta P_{\pi_\theta}(\tau) R(\tau) \\
&= \sum_{\tau} P_{\pi_\theta}(\tau) \frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)} R(\tau) \\
&= \sum_{\tau} P_{\pi_\theta}(\tau) \nabla_\theta \log P_{\pi_\theta}(\tau) R(\tau) \\
&= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log P_{\pi_\theta}(\tau) R(\tau) \right].
\end{aligned}$$

This value is called a *policy gradient* [31]. Empirically, we can approximate this by averaging over m trajectories collected under the policy π_θ:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta} \left[R(\tau) \right] &= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log P_{\pi_\theta}(\tau) R(\tau) \right] && (3.3) \\
&\approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P_{\pi_\theta}(\tau^{(i)}) R(\tau^{(i)}). && (3.4)
\end{aligned}$$

3.3 Reducing Variance with a Baseline

The policy gradient, as we have currently formulated it, attempts[5] to increase the probability of trajectories that resulted in positive rewards, and decrease the probability of those that resulted in negative rewards. However, simply shifting probabilities based on the sign of the reward introduces unwanted variance in our estimation. Additionally, if the rewards in our MDP are only positively-valued, then our gradient loses the ability to directly decrease the probability of undesirable trajectories. We thus seek a method to increase the probability of trajectories that, based on previous samples, yielded better than average rewards (and decrease those that were below average).

To do this, we introduce a baseline b to our policy gradient equation as follows:

$$\frac{1}{m}\sum_{i=1}^{m}\nabla_\theta \log P_{\pi_\theta}(\tau^{(i)})(R(\tau^{(i)}) - \boldsymbol{b}). \tag{3.5}$$

A natural concern might arise from this formulation: does the baseline b bias the gradient? In Appendix A.2, we show this is not the case on expectation, given that our choice of b at a given time step t is not dependent on a_t, the action chosen at that time step.

There are several formulations for b, of which we describe two. The first, and perhaps more natural, is precisely as we have described above—the average reward attained under the current policy over m sample trajectories:

$$b = \mathbb{E}_{\pi_\theta}[R(\tau)] \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)}).$$

More accurate, however, would be to allow b to depend on the current state in the form of a value function (recall Equation 2.1). For this, we return to our decomposition of trajectories into states and actions:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left[\left(\sum_{k=t}^{T-1} R(s_k^{(i)}, a_k^{(i)})\right) - V(s_t^{(i)})\right]. \tag{3.6}$$

Now, our baseline is an expectation of the rewards that we will experience starting in state s_t. In practice, we estimate this value function by formulating another neural network parameterized by a vector ϕ.[6] Similar to the notation π_θ, we use the notation V_ϕ to represent this neural network.

3.4 Advantage Estimation

In Equation 3.6, we compute the following value for the t^{th} time step of the i^{th} collected trajectory:

$$\left(\sum_{k=t}^{T-1} R(s_k^{(i)}, a_k^{(i)})\right) - V_\phi(s_t^{(i)}). \tag{3.7}$$

[5]There is some nuance here: as the probabilities must sum to 1, it may be the case that increasing the probability of one desirable trajectory inadvertently decreases the probability of another.

[6]At a high level, this neural network learns to minimize the difference between its estimates of expected sums of future rewards obtained from each given state and the true empirical sums of future rewards (as collected over multiple trajectories).

The first term of this equation represents the actual future rewards that were accumulated after starting from state s_t (as measured from the i^{th} collected trajectory), and the second term represents the expected future rewards accumulated from this same state (as approximated by the value function V_ϕ). These terms are similar, but the key distinction is that the first term is a measure of the current policy's empirical performance, while the second term is a measure of expected performance based on data collected from older policies. Thus, the difference between these terms intuitively represents the *advantage* that the current policy π_θ exhibits over its previous iterations. This difference is commonly denoted by $\hat{A}$ for this reason, as it is an empirical estimate of this advantage.

In Section 3.3, we outlined how adding a baseline term can reduce the variance of our gradient estimate. In this section, we aim to further reduce variance of the advantage equation above by focusing primarily on its first term: $\sum_{k=t}^{T-1} R(s_k^{(i)}, a_k^{(i)})$. This term is calculated using empirical data collected from a single trajectory, which makes it a high-variance estimate of the expected returns of trajectories that start at state s_t.

3.5 Discounting

One approach for reducing the variance of the term $\sum_{k=t}^{T-1} R(s_k^{(i)}, a_k^{(i)})$ is to introduce discounting. Specifically, we define a discount factor $\gamma \in (0, 1)$, and redefine the sum as follows:

$$\sum_{k=t}^{T-1} \gamma^{k-t} R(s_k^{(i)}, a_k^{(i)}). \tag{3.8}$$

Variance tends to increase with trajectory length, so we would intuitively expect $R(s_k^{(i)}, a_k^{(i)})$ to become a less reliable empirical estimate as k increases. By discounting each successive term in this sum by an additional factor of γ, we give less weight to high-variance rewards that are obtained further in the future,[7] which effectively reduces variance. We refer to the adjusted sum defined in Equation 3.8 as the sum of discounted future rewards obtained from state s_t.

Finally, the value function V_ϕ in our advantage equation (Equation 3.7) currently represents the expected sum of (undiscounted) future rewards, as defined in Equation 2.1. To keep the advantage estimate unbiased,[8] we additionally modify the definition of a value function by introducing a discount factor:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s \right]. \tag{3.9}$$

For the remainder of this paper, we implicitly assume that the parameterized function V_ϕ is an estimate of the value function described in Equation 3.9. In other words, V_ϕ is now assumed to be an estimate of the expected sum of *discounted* future rewards.

3.6 Function Approximation

In addition to introducing discounting into our advantage equation, we can further reduce variance by approximating Equation 3.8 using information gained from past trajectories. If we expand the

[7] Recall that $\gamma \in (0, 1)$. Thus, whenever $\alpha > \beta$, we have $\gamma^\alpha < \gamma^\beta$.

[8] An unbiased advantage estimate should be equal to 0 when the current policy's performance is indistinguishable from the expectation.

sum in this equation, we have the following:

$$R(s_t^{(i)}, a_t^{(i)}) + \gamma R(s_{t+1}^{(i)}, a_{t+1}^{(i)}) + \gamma^2 R(s_{t+2}^{(i)}, a_{t+2}^{(i)}) + \cdots + \gamma^{(T-1)-t} R(s_{T-1}^{(i)}, a_{T-1}^{(i)}).$$

In Section 3.5, we redefined the value function V_ϕ to be an estimate of the expected sum of discounted future rewards. Therefore, we can reuse V_ϕ to estimate terms in the sum above. This is the formulation used by an algorithm called asynchronous advantage actor-critic (A3C) [17], and it relies on the following substitution:

$$\sum_{k=t}^{T-1} \gamma^{k-t} R(s_k^{(i)}, a_k^{(i)}) \approx \sum_{k=t}^{t+n-1} \gamma^{k-t} R(s_k^{(i)}, a_k^{(i)}) + \gamma^n V_\phi(s_{t+n}^{(i)}), \tag{3.10}$$

where n is a hyperparameter defined as the number of "lookahead" steps. Intuitively, n is the number of rewards from trajectory i that appear in the final sum of discounted rewards. All rewards from time steps greater than or equal to $t + n$ are omitted, with the discounted value function $\gamma^n V_\phi(s_{t+n}^{(i)})$ being used as a lower-variance substitute for the tail end of this sum. Therefore, there is an inherent trade-off with varying n: decreasing n reduces variance at the cost of discarding more data.[9] For a concrete example, see Appendix A.3.

Finally, we briefly discuss generalized advantage estimation (GAE), which builds on A3C [24]. Rather than choosing a specific n as a lookahead value, GAE computes an exponentially weighted average of all possible A3C advantage estimates obtained by varying n. Heuristically, we can imagine this as balancing the trade-off between reducing variance and discarding data that arises when choosing n for A3C.

4 Algorithms

This section is modeled after the fourth video in Dr. Pieter Abbeel's lecture series on deep reinforcement learning [1, 3].

4.1 Surrogate Objective

Up to this point, our gradient estimate has been based on an expectation with respect to π_θ, the current policy. As a result, the m trajectories in our empirical estimation must be resampled after each update to our policy. This is a costly step that we can address by modifying our gradient estimate to instead rely on a previous version of our policy, so that we may continue to make updates to the current policy without the need to resample as frequently. This takes the form of a probability ratio between the current policy and the "old" policy. The algorithms that we describe in this section make use of this ratio, whose derivation we now detail.

Recall from Section 3.2 that the expectation over π_θ in our objective comes from the step in

[9] As an extreme case, if n were theoretically set equal to 0, we would be discarding all collected data and simply using V_ϕ to estimate the sum of discounted rewards under the current policy. Since the advantage estimate (Equation 3.7) subtracts $V_\phi(s_t^{(i)})$, the calculated advantage would always be equal to 0, making the advantage estimate meaningless.

which we multiply by $\frac{P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)}$. If we replicate this step using $\pi_{\theta_{\text{old}}}$ as opposed to π_θ, we have

$$\begin{aligned}\sum_\tau \frac{P_{\pi_{\theta_{\text{old}}}}(\tau)}{P_{\pi_{\theta_{\text{old}}}}(\tau)} \nabla_\theta P_{\pi_\theta}(\tau) R(\tau) &= \sum_\tau P_{\pi_{\theta_{\text{old}}}}(\tau) \frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_{\theta_{\text{old}}}}(\tau)} R(\tau) \\ &= \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_{\theta_{\text{old}}}}(\tau)} R(\tau) \right].\end{aligned}$$

We note that due to the nature of the gradient, this probability ratio reduces to our previous estimate when trajectories are first sampled. That is, when $\pi_\theta = \pi_{\theta_{\text{old}}}$, we have:

$$\frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_{\theta_{\text{old}}}}(\tau)} = \nabla_\theta \log P_{\pi_\theta}(\tau).$$

Thus, we now have a generalization of our previous policy gradient estimate that allows us to achieve better sample efficiency while updating the policy. Moving forward, we refer to this as the *surrogate objective*, written as follows:

$$\max_\theta L(\pi_\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right]. \tag{4.1}$$

Since we now have an expectation over $\pi_{\theta_{\text{old}}}$, we can make multiple updates to π_θ by repeatedly reusing a single batch of trajectories collected under $\pi_{\theta_{\text{old}}}$ to compute the gradient of Equation 4.1. However, it is important to note that once $\pi_{\theta_{\text{old}}}$ and π_θ become distinct enough from each other, we can expect an increase in the variance of our gradient estimate.[10] Thus, it is crucial that data collection is periodically repeated when maximizing the surrogate objective.

4.2 Trust Region Policy Optimization (TRPO)

Building on the notion of surrogate advantage, Schulman et al. developed an algorithm called trust region policy optimization (TRPO) that utilizes both gradient-based optimization strategies and iterative policy improvement to craft an optimal policy [26]. At a high level, TRPO begins with a randomly initialized policy $\pi_{\theta_{\text{old}}}$ and collects a batch of trajectories under this policy. Using techniques resembling those described in Section 3, the policy gradient is then estimated empirically and used to update the policy. This gradient calculation and policy update is repeatedly performed to iteratively improve the current policy π_θ, without any data collection occurring between each update. However, as described in the previous section, it is still crucial for data to be collected periodically to reduce variance as π_θ begins to diverge substantially from $\pi_{\theta_{\text{old}}}$.

Therefore, TRPO's approach is to introduce a constraint to Equation 4.1:

$$\begin{aligned}&\max_\theta L(\pi_\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right] \\ &\text{subject to } \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[KL(\pi_\theta || \pi_{\theta_{\text{old}}}) \right] \le \varepsilon,\end{aligned} \tag{4.2}$$

[10] We can only collect a finite amount of data, so we will always experience some variance. As $\pi_{\theta_{\text{old}}}$ and π_θ diverge from one another, data collected under $\pi_{\theta_{\text{old}}}$ becomes less representative of data that would be expected to appear under π_θ.

where $KL(\pi_\theta || \pi_{\theta_{\text{old}}})$ is defined as the Kullback-Leibler (KL) divergence of π_θ from $\pi_{\theta_{\text{old}}}$, and $\varepsilon > 0$ is a hyperparameter.[11] Intuitively, by constraining $KL(\pi_\theta || \pi_{\theta_{\text{old}}})$, TRPO prevents π_θ and $\pi_{\theta_{\text{old}}}$ from diverging substantially as Equation 4.2 is optimized, preventing high variance. Once this method converges to an optimal solution to Equation 4.2, $\pi_{\theta_{\text{old}}}$ is redefined to be equal to the current policy π_θ, a batch of new trajectories is collected under $\pi_{\theta_{\text{old}}}$, and this iterative optimization process begins again. This cycle repeats until the current policy π_θ converges.

In practice, enforcing the KL divergence constraint requires calculating a *natural policy gradient* [11], which can become computationally expensive. Many implementations use the method of conjugate gradients as an approximate solution instead [33, 34]. The following algorithm substantially simplifies these calculations.

4.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) builds upon the ideas introduced in TRPO by modifying the objective function to address its limitations [25]. In particular, enforcing a trust region using KL divergence turns out to be a difficult task for neural networks with dropout,[12] as well as those with multiple outputs (such as our action probabilities π_θ and value function V_ϕ). Furthermore, as mentioned above, enforcing the KL divergence as a constraint introduces additional computational overhead to the optimization problem. TRPO's conjugate gradient method is complicated to implement, and prevents us from using popular existing first-order optimization methods such as stochastic gradient descent (SGD) [23], Adam [13], or RMSProp [9].

PPO proposes to forgo KL divergence and instead adopts a clipping function within the objective itself to establish the trust region. For the sake of notational simplicity, we denote our familiar probability ratio by $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$, so that $r(\theta_{\text{old}}) = 1$. We present our new objective as follows:

$$\max_\theta L^{CLIP}(\pi_\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}}[\min(r(\theta)A^{\pi_{\theta_{\text{old}}}}, \text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)A^{\pi_{\theta_{\text{old}}}})], \tag{4.3}$$

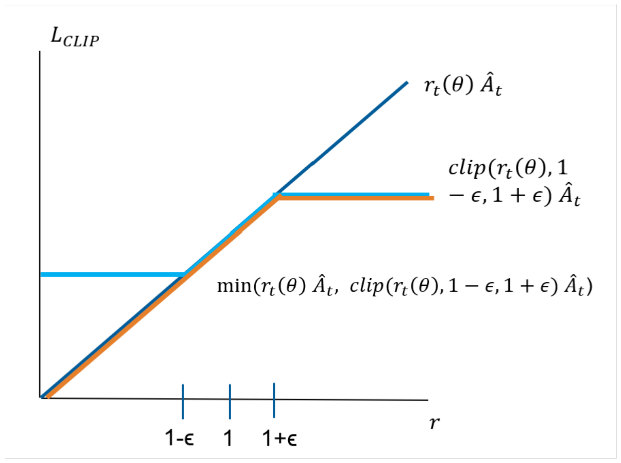
where $\varepsilon > 0$ is a hyperparameter. We now refer the reader to Figure 1 as we discuss Equation 4.3. Let us begin from the inside and work our way out. First, notice that $r(\theta)A^{\pi_{\theta_{\text{old}}}}$ is precisely the term used in TRPO's objective—we refer to this as the unclipped objective. The clipped objective, then, is the term $\text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)A^{\pi_{\theta_{\text{old}}}}$. Here, we prevent the probability ratio $r(\theta)$ from moving too far in a desirable direction by clipping the value of $r(\theta)$ at either $1-\varepsilon$ or $1+\varepsilon$, depending on the sign of $A^{\pi_{\theta_{\text{old}}}}$. We then take the minimum of these two objective functions in order to achieve a "pessimistic" value for the final objective. Specifically, we clip the probability ratio only in the direction that would improve the value of the objective, and allow it to change without bound when doing so would make the objective arbitrarily small.

For intuition on the choice to take a minimum of the clipped and unclipped objectives, recall Footnote 5, in which we mention that shifting the probability mass of certain state-action pairs may have adverse effects on that of others. By taking the minimum of $r(\theta)A^{\pi_{\theta_{\text{old}}}}$ and $\text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)A^{\pi_{\theta_{\text{old}}}}$, we allow the gradient to correct for these inadvertent shifts in probability mass.[13]

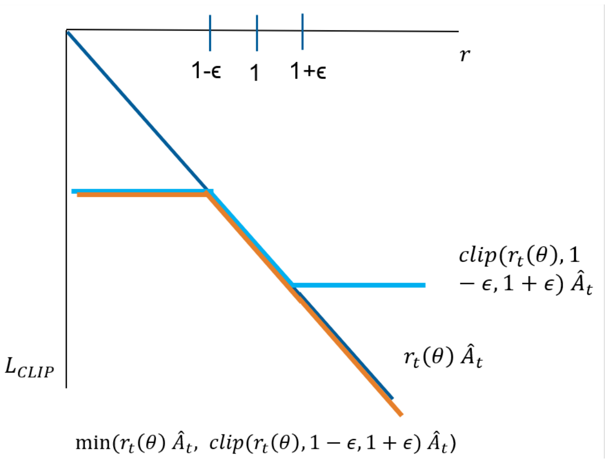
[11]The term $KL(\pi_\theta || \pi_{\theta_{\text{old}}})$ loosely represents the "difference" between π_θ and $\pi_{\theta_{\text{old}}}$.

[12]Dropout is a widely-used regularization method for reducing overfitting in neural networks [29].

[13]If we were to clip the "detrimental" direction as well, then (a) our probability ratio could end up in a region of slope 0, preventing future updates to it through our gradient, and (b) we would artificially place a lower bound on the value of the objective, instead of allowing it become arbitrarily small in order to penalize the aforementioned inadvertently bad updates.



(a) $A^{\pi_{\theta_{\text{old}}}} > 0$.



(b) $A^{\pi_{\theta_{\text{old}}}} < 0$.

Figure 1: The pieces of PPO's objective function in relation to advantageous (1a) and disadvantageous (1b) actions. In each subfigure, we see the main objective in orange, the unclipped objective in blue, and the clipped objective in light blue.

5 Ensuring Safety While Training

5.1 Definitions of Safety

Safety is highly context-dependent, and both reward and cost structures should reflect the current definition in each given environment [6]. For example, in the context of a Mars Rover, the most important measure of safety is the safety of the agent itself [18]. Violations could lead to catastrophic results (breaking the rover), therefore the safety of the rover should be implemented absolutely. While there could be smaller costs that the rover could incur without jeopardizing its mission, these should be classified differently because they are "acceptable" costs. On the other hand, moves that result in catastrophe should be weighted extremely highly or even made unavailable to the agent's decision-making process, discouraging the rover from ever taking an action that would lead to catastrophe.

In this case, this is a form of probability-one safety: we want to guarantee safety constraints are enforced with probability equal to 1. There are different scenarios in which probability-one safety is necessary, such as when protecting the safety of the agent itself; the agent should be safe with probability one and almost never violate its safety constraint.[14] In other situations, constraining violations on average, with some amount of acceptable variance, is preferable. For example, if we were training a moving agent to remain at an optimal speed, we would likely allow some fluctuations and might even prefer some variance so that the agent can adjust to its environment. Both probability-one and average safety constraints are necessary in different situations, and the formulation of safety in a given context should reflect this [28].

When considering multi-agent reinforcement learning (MARL), the definition of safety changes [6]. Consider an example in which we have eighty Roombas that are tasked with cleaning a warehouse. In this case, each Roomba simply plays a part in the larger group of Roombas. If there

[14]In this case, "almost never" is a formal mathematical term that means "with probability equal to 0."

were an action with a low risk of catastrophic damage but a high potential for completing the task, we might be able to justify a few Roombas taking this action.[15] Although in this situation, agent safety does matter, not every agent has to be entirely safe: if several Roombas break, this is not a catastrophic issue for the larger group of Roombas. Formulating a multi-agent problem such as this would require a different structure than before. Certain actions should be discouraged in this case rather than outright banned.

The most complex situation is environmental safety. Cars are an intuitive example of this. Although the car itself is valuable, the environment surrounding the car (pedestrians, other cars, etc.) is typically considered to be more important. Some hazards are so important to avoid that under no conditions can a collision be permitted (e.g. hitting pedestrians). This, then, is similar to the single-agent case in which catastrophic events should be banned.

When discouraging lower-level violations, the relative severity of these violations suggests a need for a formalized ranking of the importance of violations. For example, if the only choices (including staying still) are unsafe, which unsafe action should the agent take? This brings us into the realm of enforcing ethics when training reinforcement learning algorithms. Ethical discussions are necessary in complex decision making to provide rationale for the relative importance of each type of possible violation and how permissible each type should be relative to the others. We explore this topic further in Section 10.

5.2 Implementations of Safe Reinforcement Learning Algorithms

The types of safety relevant to a given situation determine how safety violations should be formulated. After formulating an optimization problem with a constraint on safety violations, we aim to compute the optimal solution to the problem. This can be achieved by utilizing cautious actions [8], state augmentation [28, 27], and Lagrange multipliers [10]. We discuss the latter two methods in this paper.

5.2.1 Sauté and Simmer

Sootla et al. introduce two strategies for enforcing safety constraints via state augmentation: Sauté (**Sa**fety **Au**gmen**te**d), and subsequently Simmer (**S**afe Policy **Im**prove**me**nt for **R**L) [27, 28]. Sauté incorporates a variable z_t into the state space. Much like traditional states, z_t provides information about the environment to the agent as a "safety" state. In particular, the set of safety states Z provides the agent with a constant measure of distance toward constraint violation. Simmer is merely an extension of the constrained optimization problem introduced in Sauté, of which we now provide an overview.

In order to formulate the problem, we introduce the notion of a *constrained* Markov decision process.

Definition 5.1 (Constrained Markov Decision Process)**.** We define a *constrained MDP (c-MDP)* as a 5-tuple (S, A, P, R, C), where the final term $C(s_t, a_t)$ is a nonnegative safety cost associated with taking action a_t from state s_t.

[15]This is explored more in MARL research, but is an interesting way to design different policies to optimize the whole group, rather than just each agent [6].

With the introduction of costs for taking actions, the associated optimization problem we now wish to solve is

$$\begin{aligned} &\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] \\ &\text{subject to } \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] \leq d, \end{aligned} \tag{5.1}$$

where $d \geq 0$ is the "safety budget" that our agent must not exceed for the purposes of satisfying the constraint.[16]

We now show that we can directly incorporate the constraint into the objective by augmenting the state space S with a safety state z_t associated with each s_t. We begin by introducing the term $w_t = d - \sum_{k=0}^{t} \gamma_c^k C(s_k, a_k)$ as the remaining safety budget at time t, where $\gamma_c \in (0, 1)$ is the discount factor for safety costs. For reasons that we clarify after the following series of calculations, we define z_t as a scaled version of w_t. In particular, we let $z_t = w_{t-1}/\gamma_c^t$, so that

$$\begin{aligned} z_{t+1} &= w_t/\gamma_c^{t+1} \\ &= (w_{t-1} - \gamma_c^t C(s_t, a_t))/\gamma_c^{t+1} \\ &= (z_t - C(s_t, a_t))/\gamma_c, \end{aligned}$$

where $z_0 = d$. The introduction of the scaling factor allows us to eliminate reliance on knowledge of the time step t to make updates to the value of z_t. That is, since we have shown that z_{t+1} depends solely on z_t, s_t, and a_t, we can implement z_t directly into our state space as mentioned above. In particular, we replace the state space S in our c-MDP with $\tilde{S} = S \times Z$, where Z is the set of safety states. As a result, we can now consider states as tuples of the form (s_t, z_t). In addition to receiving information about its environment via s_t, the agent also receives a distance to constraint violation via z_t. With this, we can modify the rewards in our MDP to be piecewise-defined by the value of z_t:

$$\tilde{R}((s_t, z_t), a_t) = \begin{cases} R(s_t, a_t) & z_t \geq 0 \\ -\infty & z_t < 0. \end{cases} \tag{5.2}$$

Our objective for this modified "Sauté" MDP is thus

$$\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \tilde{R}((s_t, z_t), a_t)\right]. \tag{5.3}$$

We note that in practice, the $-\infty$ in Equation 5.2 is replaced by a finite hyperparameter n.

Simmer extends upon the state augmentation strategy introduced in Sauté by allowing the safety budget d to shift dynamically during training.[17] This allows for fine tuning of the agent's behavior; the authors of [27] learned that agents tend to spend the entire safety budget in an initial training spike. To prevent this behavior, agents begin training under a strict safety budget d^{start} that is increased to some target budget d^{target}.

[16]For the sake of consistency with our unconstrained objective in Equation 3.1, we omit γ from this problem definition, though we include it in the coming definition of z_t.

[17]To control the safety budget dynamically, the authors of [27] implemented two controllers: one was based on a PI (proportional-integral) controller, and the other was a controller trained using Q-learning.

5.2.2 Lagrangian Formulation

Another approach for solving the constrained optimization problem outlined in Equation 5.1 is to transform it into an equivalent unconstrained optimization problem. We can achieve this by introducing a Lagrange multiplier λ.

Specifically, Equation 5.1 is formulated as follows:

$$\begin{aligned} \max_{\theta} \; & \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] \\ \text{subject to } & \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] \leq d, \end{aligned}$$

where $d \geq 0$ is our threshold of permitted accumulated costs. We claim that solving the problem above is equivalent to the following equation [14]:

$$\max_{\theta} \min_{\lambda \geq 0} \left(\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] - \lambda (\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] - d) \right). \tag{5.4}$$

To solve this, it has been shown that performing simultaneous gradient ascent on θ and gradient descent on λ (while enforcing $\lambda \geq 0$) converges to a solution [21]. For a breakdown of Equation 5.4, see Appendix A.4.

6 Methods

We analyzed the effects of different hyperparameters on the performance of various safe reinforcement learning algorithms in two simulated environments. After evaluating these algorithms under default hyperparameter configurations, we ran several initial experiments by adjusting one hyperparameter at a time. We constructed our set of final experiments as we interpreted these initial results and tried to explain unexpected outcomes in the context of our theoretical understanding.

6.1 Resources and Tools Used

To execute our experiments, we created a custom Docker image called `saferl/pytorch` for development, which is defined in our primary repository on GitHub.[18] The Xvfb display server is preinstalled on this Docker image for rendering graphics, along with dependencies such as the Farama Foundation's Gymnasium and PKU-Alignment's Safety-Gymnasium packages for simulating safe reinforcement learning environments [7, 20]. We utilized OmniSafe, which is an infrastructural framework that enables safe reinforcement learning research [10], to define and run our experiments. More specifically, we forked OmniSafe's primary repository[19] to add two of our own custom safe reinforcement learning algorithms, and we included our fork[20] as a preinstalled dependency on the `saferl/pytorch` Docker image. Our primary repository contains our experiment scripts, and we used Weights & Biases for aggregating and analyzing results [5].

[18]`https://github.com/Safe-Reinforcement-Learning/safe-reinforcement-learning`
[19]`https://github.com/PKU-Alignment/omnisafe`
[20]`https://github.com/Safe-Reinforcement-Learning/omnisafe`

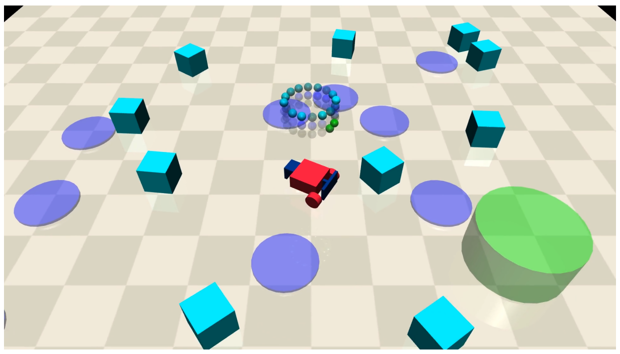


Figure 2: The *Car* environment. The green circle is the goal, the dark blue circles are hazards, and the light blue cubes are vases.

6.2 Algorithms

We compared a total of eight algorithms. Our base algorithms were TRPO and PPO, and we additionally analyzed three modified versions of each of these:

- TRPOSimmer and PPOSimmer (Simmer strategy from Section 5.2.1),
- TRPOLag and PPOLag (Lagrangian formulation from Section 5.2.2),
- TRPOLagSimmer and PPOLagSimmer (combination of the Lagrangian formulation and Simmer strategy).

Our two base algorithms are not designed with safety in mind, but all six of the modified algorithms incorporate safety constraints.

6.3 Environments

In order to compare the performances of these algorithms across distinct definitions of safety, we chose two fundamentally different environments from Safety Gymnasium's provided environments [7, 20]. Each environment pairs an agent with a task by defining specific reward and cost structures for the agent. One of these environments incentivizes minimizing harm to both the agent and the environment, while the other environment's definition of safety solely accounts for harm to the agent.

6.3.1 SafetyCarGoal2-v0

The first agent-task pair is the car agent run on the SafetyGoal2-v0 task (*Car*). In this environment, the car agent is placed in a space with a goal, hazards, and vases. The goal initializes to a random location at the start of an episode and is reset to a new location whenever the car agent successfully navigates to the goal. The hazards and vases are initialized to random locations at the beginning of the run, and hazards are stationary, while vases can move if the car agent collides with them. In terms of the car agent's reward structure, the car obtains a sparse reward whenever it successfully navigates to the goal. Every time step, the car also receives dense rewards proportional to the change in its distance to the goal. The car incurs costs when it drives over a hazard, collides

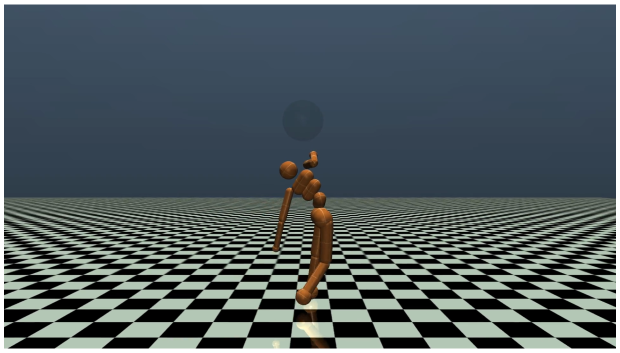


Figure 3: The *Humanoid* environment. The sphere above its head turns red when the velocity threshold has been exceeded.

with a vase, or moves a vase too quickly. In *Car*, the objective is for the agent to prioritize both environmental safety and its own safety. Hazards are meant to represent areas of potential harm to the agent while vases represent parts of the environment that could be harmed by the agent.

6.3.2 SafetyHumanoidVelocity-v1

The second agent-task pair is the humanoid agent run on the SafetyVelocity-v1 task (*Humanoid*). In this environment, the humanoid agent must learn to walk. It gains rewards for staying upright and moving forward, and it receives negative rewards when its control cost is high, encouraging an even distribution of force across its joints. The humanoid agent incurs costs when it violates a velocity threshold. Therefore, safety in *Humanoid* is solely determined by the safety of the agent: there are no obstacles present in the environment that could potentially be harmed by the agent's actions, so the agent's constraint is solely to prioritize its own safety.

7 Results for *Humanoid*

We initially trained policies with the default parameters for each algorithm for 10,000 epochs with 1024 steps each. This was based off the tutorial and paper published by Omnisafe [10].

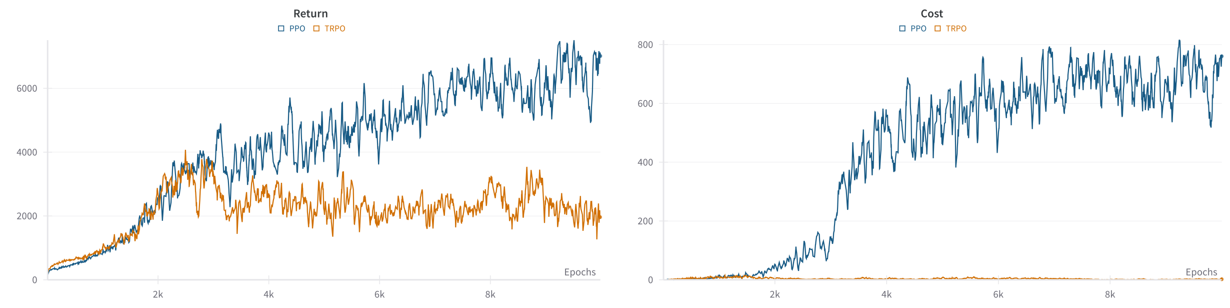


Figure 4: TRPO vs PPO trained for 10,000 epochs with 1024 steps per epoch.

In this initial experiment, displayed in Figure 4, we observed that PPO achieved much greater returns than TRPO with the trade-off of incurring much greater costs.

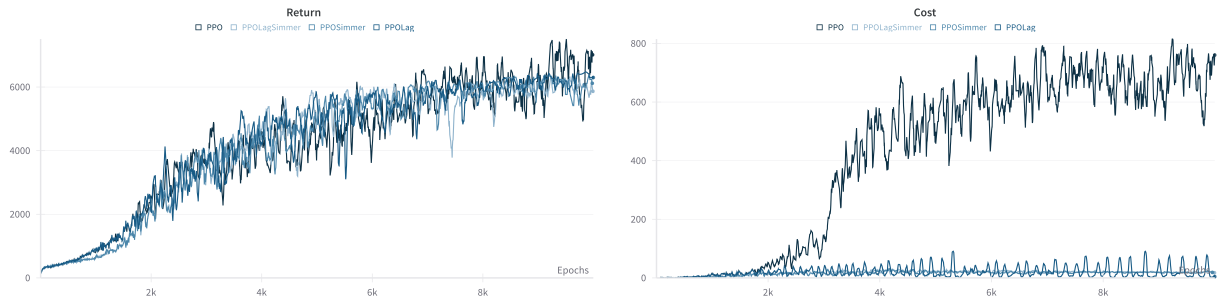


Figure 5: PPO, PPOLag, PPOSimmer, and PPOLagSimmer trained for 10,000 epochs with 1024 steps per epoch.

The returns from PPO and its constrained versions are displayed in Figure 5. Examining the differences in performance between PPO and its constrained versions reveal that each constrained version was able to achieve comparable returns while drastically limiting the incurred costs. PPOLag exhibited oscillating costs and returns, evidence that the Lagrange multiplier was increasing and decreasing as the policy violated and satisfied the acceptable cost constraint. Each time the policy exceeds the cost limit of 25, the returns and costs decrease. Once the costs have dipped below 25, the returns and costs begin to increase. PPOSimmer and PPOLagSimmer did not exhibit this same behavior, as the Simmer method attempts to prevent violations of the constraint in the first place.

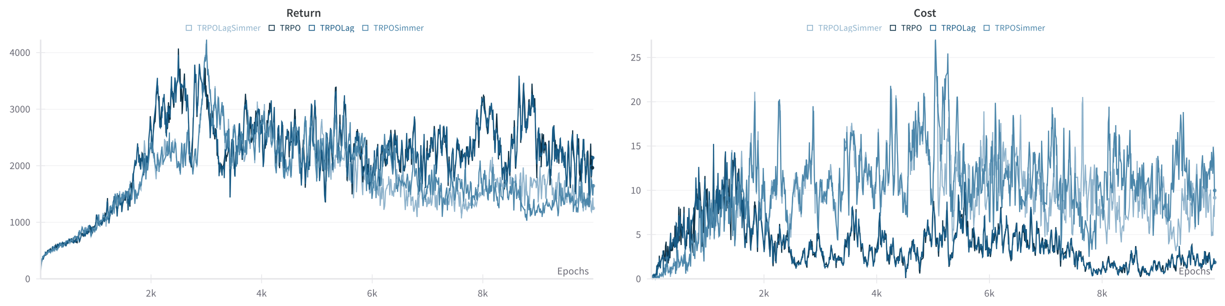


Figure 6: TRPO, TRPOLag, TRPOSimmer, and TRPOLagSimmer trained for 10,000 epochs with 1024 steps per epoch.

On the other hand, base TRPO did not violate the acceptable cost constraint in the first place (Figure 6). This led to TRPOLag behaving nearly identically to it, since the multiplier never increased. TRPOSimmer and TRPOLagSimmer incurred slightly higher costs than TRPO and TRPOLag but ultimately still adhered to the constraint and achieved similar returns.

7.1 Adjusting Acceptable Cost per Episode

As the baseline cost incurred by PPO is much higher than constraint, we conducted an experiment on PPO and TRPO and their safe variants to explore the impact of a less restrictive constraint. We increased the acceptable cost per episode to 100. In all graphs, if an error cloud is present it represents the standard error when the experiment was run using multiple seeds. Most of the initial experiments were run using a single seed to save time and computing power, and thus do not have standard error clouds.

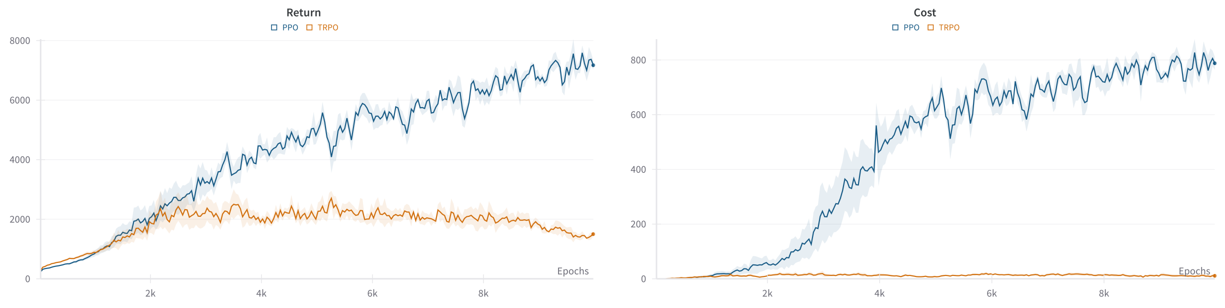


Figure 7: Unconstrained TRPO and PPO. The error cloud represents the standard error of different seeds.

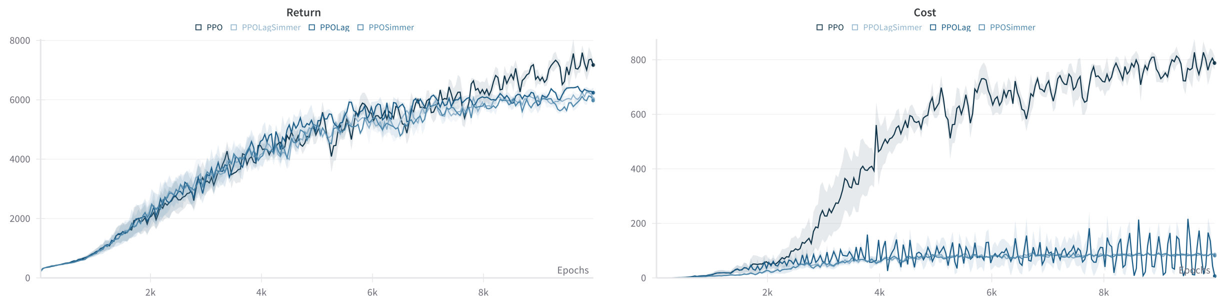


Figure 8: PPO versus PPOLag, PPOSimmer, and PPOLagSimmer with acceptable costs of 100.

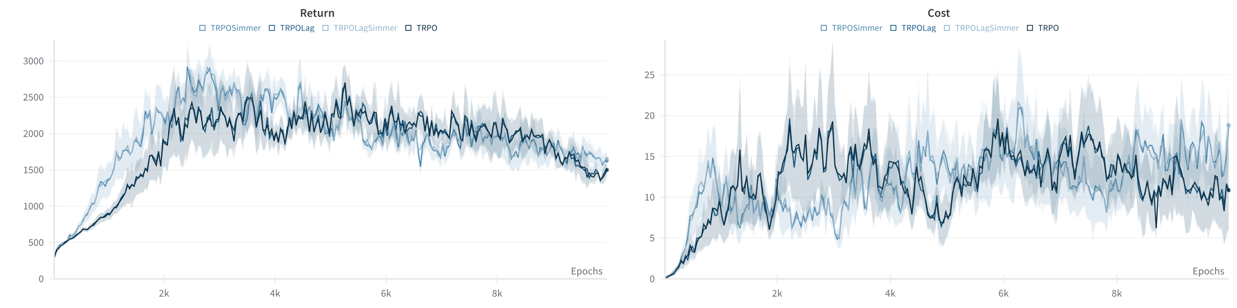


Figure 9: TRPO versus TRPOLag, TRPOSimmer, and TRPOLagSimmer with acceptable costs of 100.

When examining Figures 8 and 9, we see that an acceptable cost of 100 was enough to allow the safe versions of TRPO and PPO to achieve similar rewards to the unconstrained versions. TRPO already incurs costs below 100, so we only see measurable improvement in cost for PPO; when unconstrained, PPO incurred costs averaging 800. Figure 8 shows that base PPO achieves returns close to 7000 while its constrained versions achieve returns of 6000, but this slight reduction in returns is more than made up for in the drastic reduction in costs. Again, we observe a cyclical pattern for PPOLagand a more stable cost pattern from PPOSimmer.

8 Results for *Car*

As with *Humanoid*, we initially trained policies with the default parameters for each algorithm for 10,000 epochs with 1024 steps each.

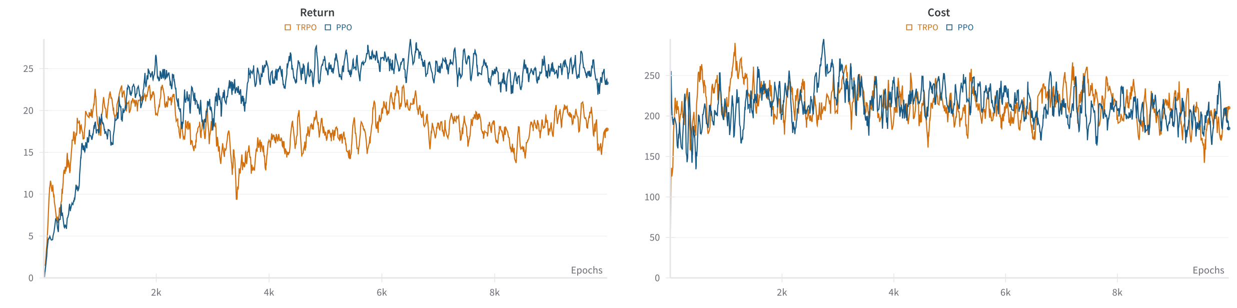


Figure 10: TRPO vs PPO trained for 10,000 epochs with 1024 steps per epoch

In Figure 10, we observed that TRPO on average returned around 18 while PPO on average returned around 25. Both incurred costs averaging 200.

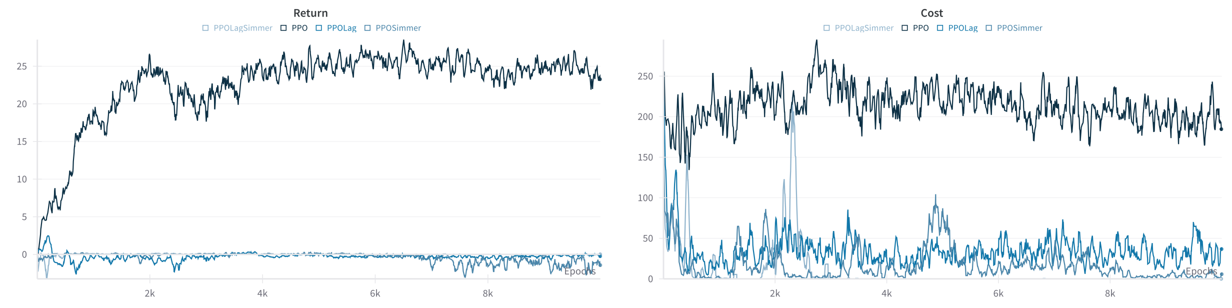


Figure 11: PPO, PPOLag, PPOSimmer, and PPOLagSimmer trained for 10,000 epochs with 1024 steps per epoch.

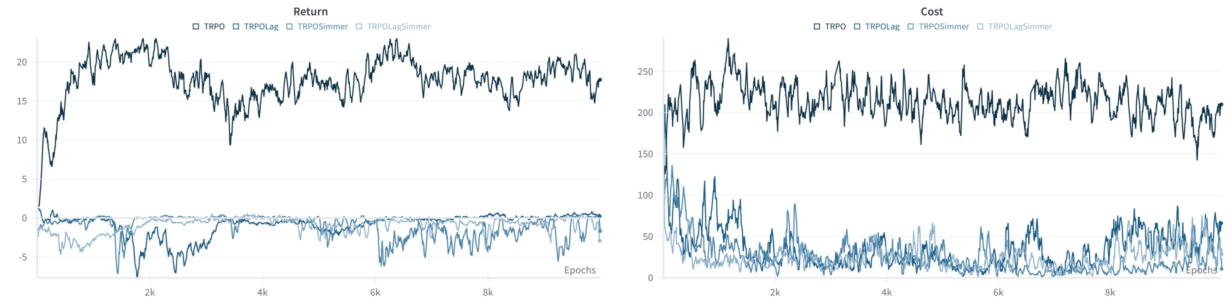


Figure 12: TRPO, TRPOLag, TRPOSimmer, and TRPOLagSimmer trained for 10,000 epochs with 1024 steps per epoch.

Comparing PPO and TRPO to their safe variants (displayed in Figure 11 and Figure 12), we observe that constrained versions of these algorithms achieve low or negative returns. This implies that the agent is primarily driving away from the goals. Despite these low returns, the agents are incurring costs in excess of the specified acceptable cost of 25. So in addition to avoiding the goal, the agent is driving into hazards and colliding with vases. We believe that the cost limit was too low to allow for proper training. The algorithm could not properly explore before being heavily penalized for incurring high costs, and therefore developed a poorly optimized policy.

8.1 Adjusting Acceptable Cost per Episode

Seeing that the default acceptable cost of 25 was too low to allow for a policy that achieved measurable rewards, we decided to experiment with a higher acceptable cost. As an initial experiment, we ran PPOLag and TRPOLag with a cost limit of 100 to observe the increase in returns.

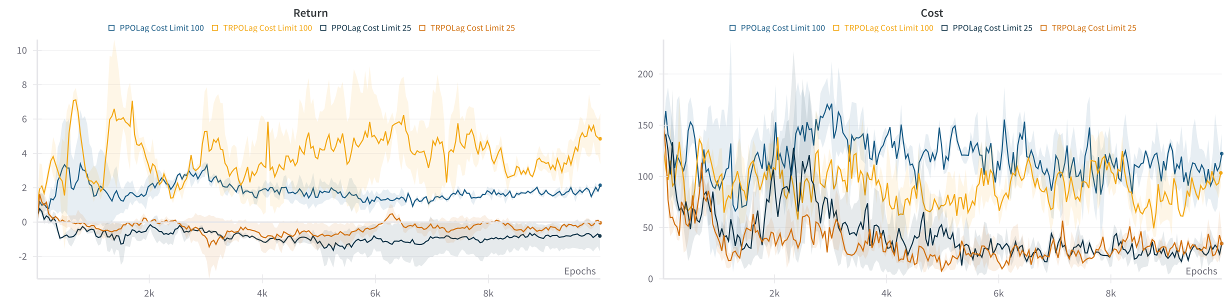


Figure 13: PPOLag and TRPOLagtrained for 10,000 epochs with 1024 steps per epoch with varying cost limits.

We observed that an increased cost limit allowed both PPOLag and TRPOLag to achieve higher returns, with the increased cost benefiting TRPOLag more (Figure 13). Additionally, TRPOLag seemed to be better constrained.

Therefore, we ran a new experiment to verify that changing the acceptable cost to 100 would yield similar results with TRPOSimmer, TRPOLagSimmer, PPOSimmer, and PPOLagSimmer.

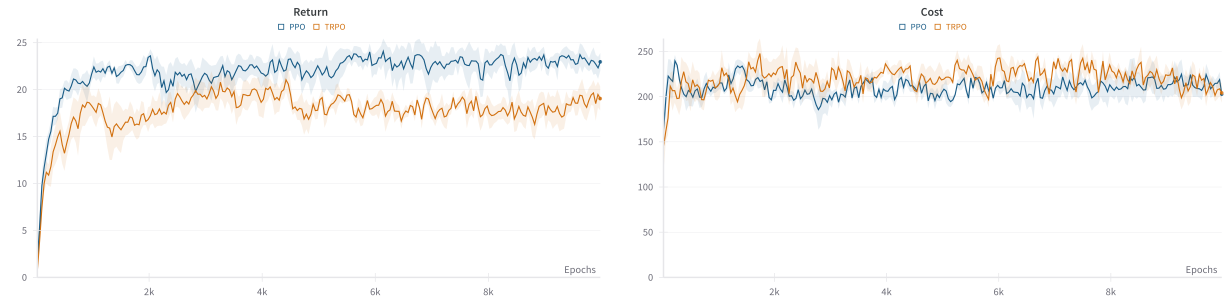


Figure 14: Unconstrained TRPO and PPO.

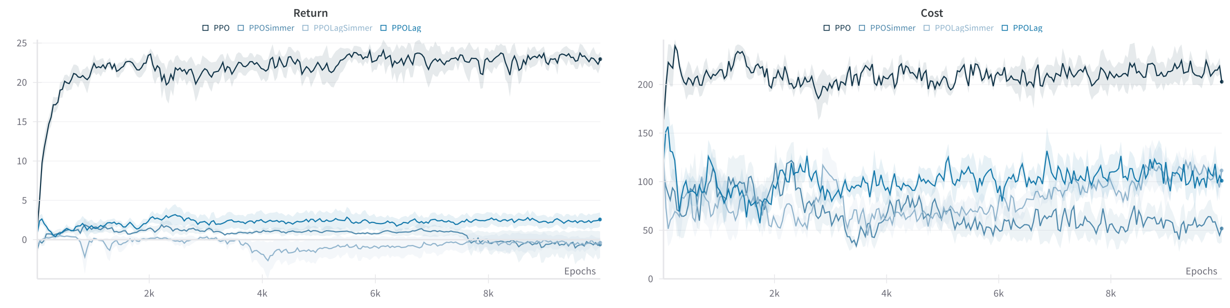


Figure 15: PPO versus PPOLag, PPOSimmer, and PPOLagSimmer with acceptable costs of 100.

We see that the PPO safe variants perform measurably worse than unconstrained PPO in Figure 15, achieving small or even negative returns. Of these, PPOLag performed the best with small, but consistently positive, returns. All safe variants were able to successfully constrain the cost to 100.

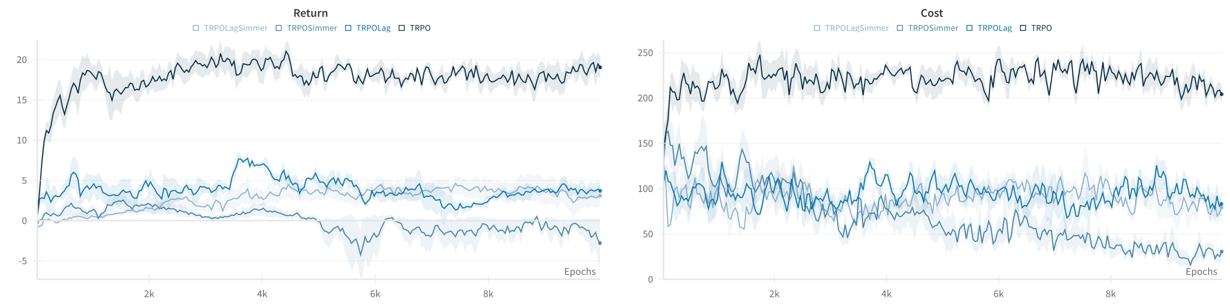


Figure 16: TRPO versus TRPOLag, TRPOSimmer, and TRPOLagSimmer with acceptable costs of 100.

We see that the TRPO safe variants also suffer in returns compared to unconstrained TRPO in Figure 16, but not nearly as much as the PPO safe variants. All but TRPOSimmer were able to consistently achieve positive returns. These returns were small, but greater than those seen by the safe PPO variants. As with PPO, all TRPO safe variants were able to successfully constrain the cost to 100.

To better understand what acceptable costs lead to the greatest increase in returns, we ran experiments with PPOLag, PPOSimmer, TRPOLag, and TRPOSimmer with varying acceptable costs.

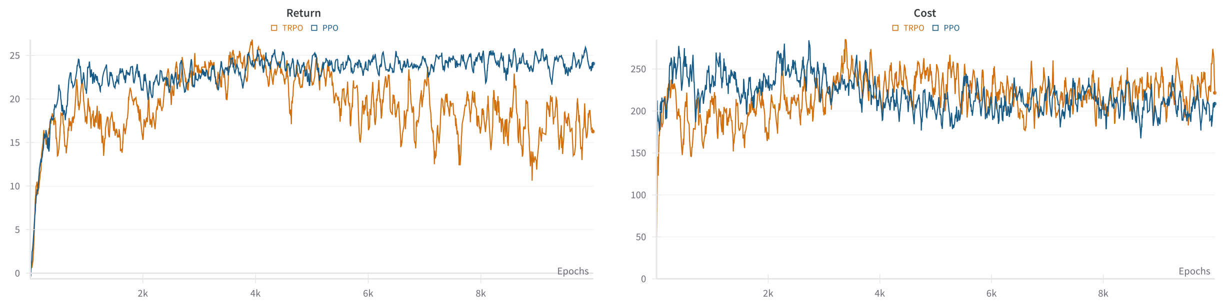


Figure 17: Unconstrained TRPO and PPO.

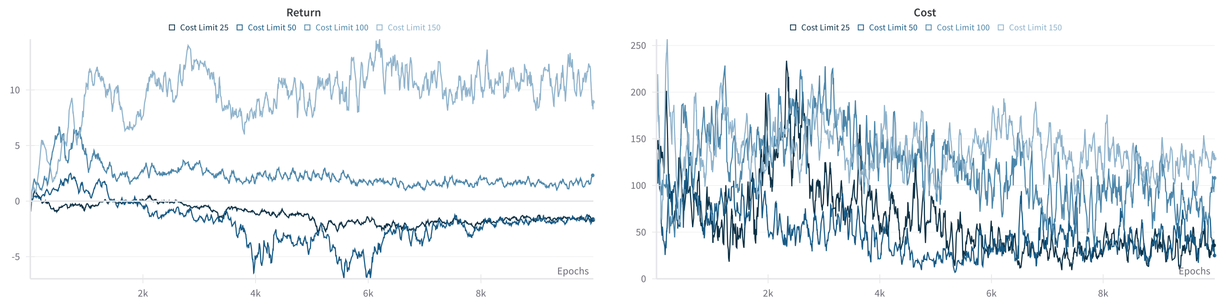


Figure 18: PPOLag with varying cost limits.

Similarly to before, we observe that cost limits 25 and 50 are unable to achieve positive returns in PPOLag (Figure 18). Additionally, cost limit 25 does not effectively constrain the costs under 25. Cost limits of 100 and 150 reliably achieve positive returns and constrain the costs to their specified limit.

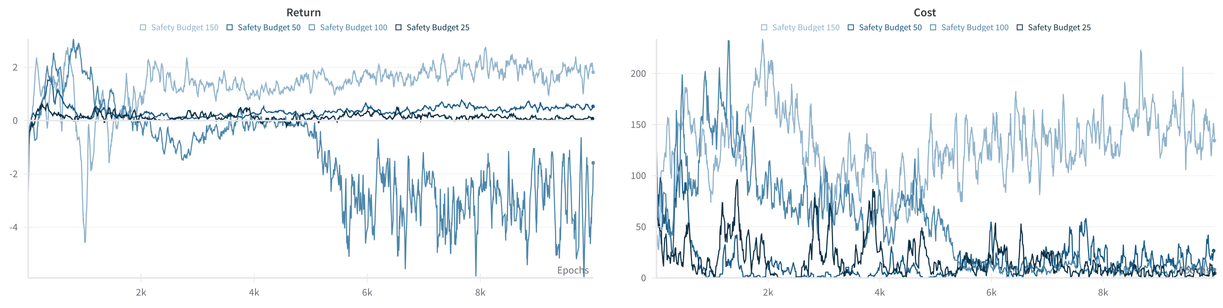


Figure 19: PPOSimmer with varying safety budgets.

PPOSimmer reliably achieves positive returns at all safety budgets, increasing with a higher safety budget, with one noticeable exception (Figure 19). Safety budget 100 achieved overall negative returns and constrained costs to much lower than 100, leveling out under 10.

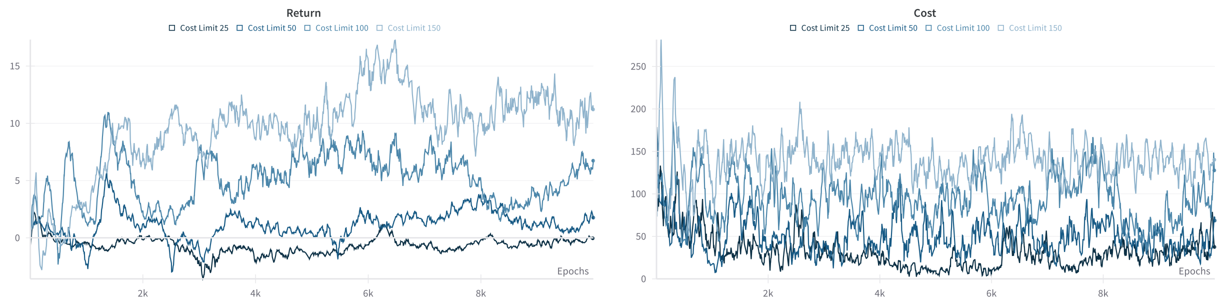


Figure 20: TRPOLag with varying cost limits.

With TRPOLag, cost limit 25 is unable to achieve reliably positive returns or fully constrain costs to 25 (Figure 20). Cost limits 50, 100, and 150 reliably achieve positive rewards that increase with the cost limit and all but 100 reliably constrain the cost.

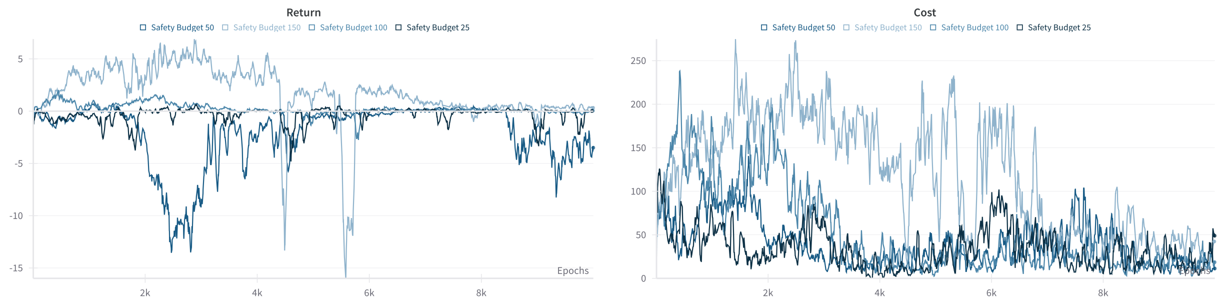


Figure 21: TRPOSimmer with varying safety budgets.

TRPOSimmer is not able to reliably achieve positive returns at any safety budget, though 150 was able to mostly achieve positive returns (Figure 21). Each safety budget seemed to constrain costs to 50 rather than the specified budget.

Generally, increasing the acceptable cost per run led to increased returns. By allowing the agent to choose less safe behaviors, it gained more rewards because it focused more time on driving toward the goal rather than avoiding hazards and vases. Those who are building policies for safe behavior will need to establish and justify their acceptable costs according to their definitions of safety.

8.2 Adjusting Lagrange Multiplier Learning Rate

Another effort to improve the performance of our safe algorithms was to adjust the Lagrange multiplier learning rate of PPOLag and TRPOLag. We hypothesized that the default Lagrange multiplier learning rate was attempting to constrain the costs of the policy too quickly, penalizing the agent too harshly before it had time to learn. We lowered the value to allow the agent to first learn how to gain rewards, and then later limit costs.

In our first experiment, we changed the Lagrange multiplier learning rate of PPOLag.

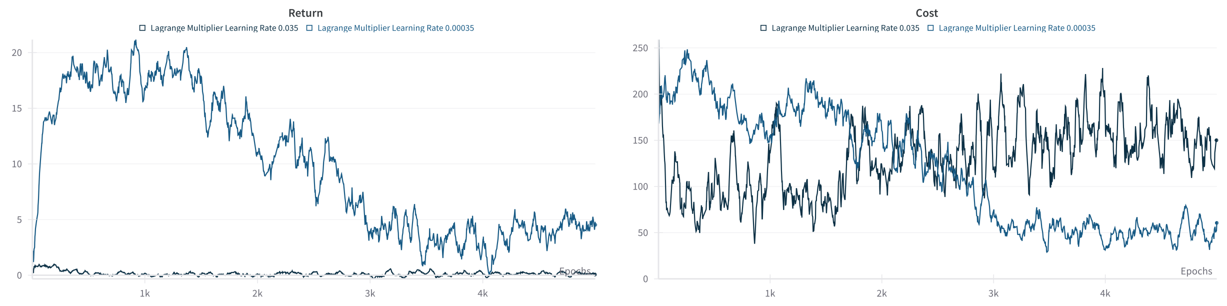


Figure 22: PPOLag trained for 10,000 epochs with 1024 steps per epoch with different Lagrange multiplier learning rates.

Initially, the policy with the slower learning rate was incurring high costs and high returns, as seen in Figure 22. As the Lagrange multiplier grew, the costs and returns both decreased. With the default learning rate, the were barely measurable and the costs were not constrained to the cost limit of 25. This showed that a slower learning rate yielded higher returns and more effectively constrained costs.

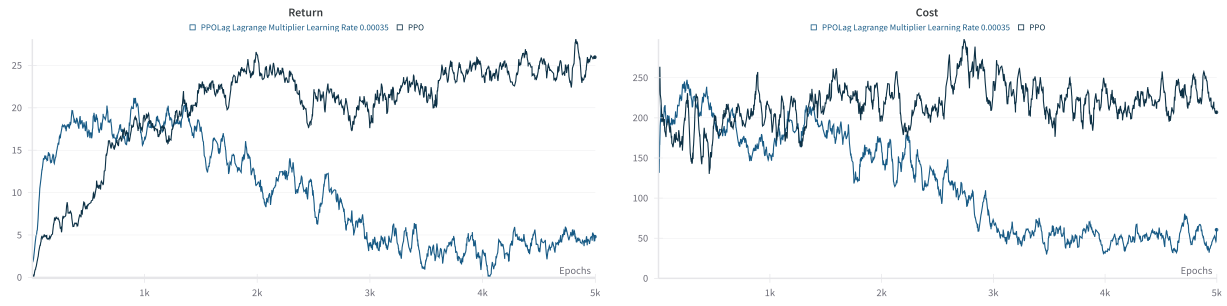


Figure 23: Unconstrained PPO compared to PPOLag with a slower Lagrange multiplier learning rate.

The slower Lagrange multiplier learning rate allowed the policy to more closely resemble one that would be produced by PPO in the beginning (Figure 23). This allowed the policy to develop to maximize returns before it began constraining costs. With the faster default learning rate, the policy was constrained before it was able to achieve significant returns.

Based on these findings, we ran an additional experiment with a cost limit of 100, Lagrange multiplier learning rates of 0.035, 0.0035, and 0.00035, and multiple seeds.

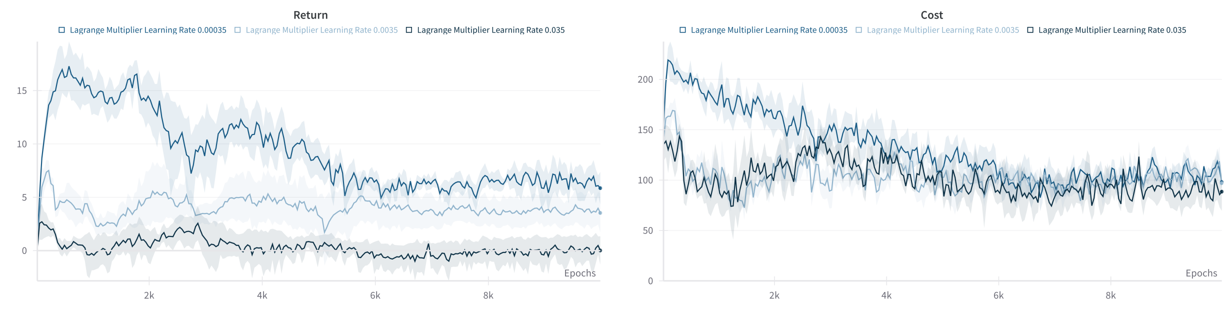


Figure 24: PPOLag with varying Lagrange multiplier learning rates.

Figure 24 shows that the slower two Lagrange multiplier learning rates of 0.0035, and 0.00035 were able to achieve higher returns than the default Lagrange multiplier learning rate of 0.035 while still constraining the costs to 100. This confirmed our initial hypothesis that a slower Lagrange multiplier learning rate would allow the policy to first learn how to achieve greater returns by initially disregarding the cost limit. After this initial unconstrained period, the policy was updated to enforce the safety constraint. By constraining the policy too early, the default Lagrange multiplier learning rate prevented the policy from achieving positive returns. We suspect that if the Lagrange multiplier learning rate was too slow, it would take the policy considerably longer to be constrained. Our best performing learning rate of 0.00035 was slow enough to allow for greater returns, but still constrained the costs within $10,000$ epochs of training. If we had more time, we would experiment with finding the optimal learning rate for different amounts of training.

9 Results for Policy Update Improvements

We found that the most effective way to increase returns without incurring more costs was to optimize how the algorithms update their policies. Compared to our previous hyperparameter tuning, this method requires less fine tuning to individual situations and does not run into ethical dilemmas with adjusting acceptable levels of cost.

9.1 *Humanoid* Policy Update Improvements

9.1.1 Adjusting Number of Steps per Epoch

Our first method of improving policy updates for *Humanoid* relied on basing each update on a greater number of episode samples. Since the environments cut runs off at a maximum of 1000 steps, increasing the number of steps per epoch past 1000 allows the agent to run and collect samples in the environment multiple times before the policy is updated. Thus, we increased the number of steps per epoch to allow the algorithm to collect more data before updating the policy.

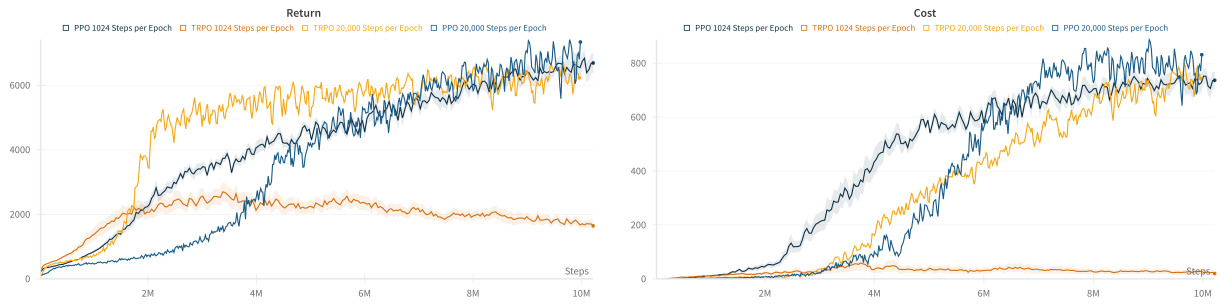


Figure 25: PPO and TRPO with varying steps per epoch.

Doubling the number of steps per epoch while maintaining the total number of steps drastically improved the returns for TRPO at the expense of incurring more cost (Figure 25). TRPO and PPO with 20,000 steps per epoch performed very similarly to PPO with 1024 steps per epoch.

Since our initial experiment only ran for 500 epochs, wanted to see if with 20,000 steps per epoch PPO would eventually overtake TRPO. We ran a new experiment for 5000 epochs to learn if the policies would continue to improve with more training.

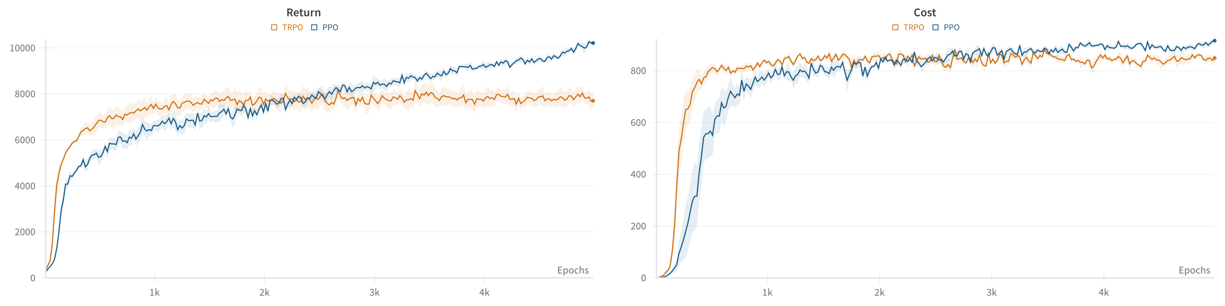


Figure 26: PPO and TRPO trained for 5000 epochs with 20,000 steps per epoch.

Figure 26 shows that returns and costs for TRPO plateaued at 1000 epochs of training. However, PPO continued to show increasing returns, with no sign of asymptotically leveling out.

These experiments reveal that increasing the number of steps per epoch, even when maintaining the total number of steps, leads to improved returns. Prioritizing longer epochs therefore leads to better policies.

9.1.2 Adjusting Allowed Policy Change per Update

Our second method of improving policy updates for *Humanoid* relied on altering how much the policy could change at each update. OmniSafe's implementation of PPO and TRPO use a target KL hyperparameter to accomplish this [10]. As a policy is represented as a distribution, the KL parameter limits how much a new policy distribution can diverge from the previous (section 3). For PPO, if the KL divergence between the previous policy and the new policy exceeds the target value, the run is cut short. For TRPO, the target value is used to establish the size of the trust regions.

In our initial experiment, we ran PPO with target KL values of 0.2, 0.02 (default), and 0.002.

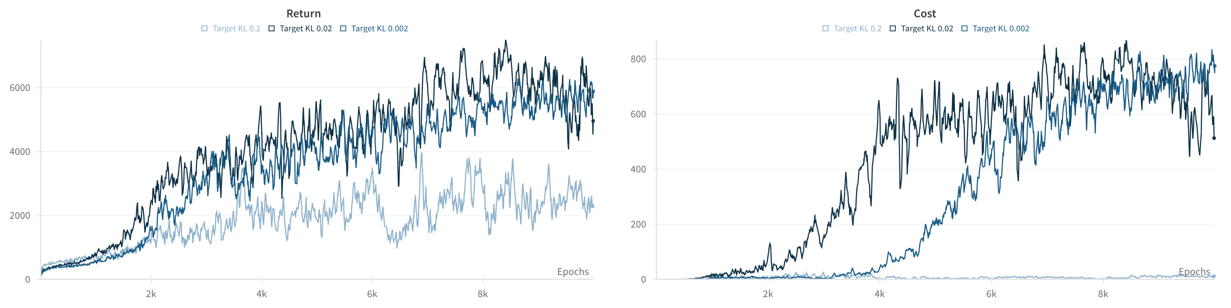


Figure 27: PPO with varying target KL values.

Figure 27 shows that for the smaller target KL values of 0.02 (default) and 0.002, returns and costs were very similar. We suspect that the larger value of 0.2 allowed for policy changes that were too large and changed too much, leading to a suboptimal policy.

To see if these results were consistent with TRPO, we ran another experiment where we varied the target KL value of both TRPO and PPO. For PPO the values we tested were 0.0002, 0.002, 0.02 (default), and 0.2. For TRPO the values we tested were 0.0001, 0.001, 0.01 (default), and 0.1.

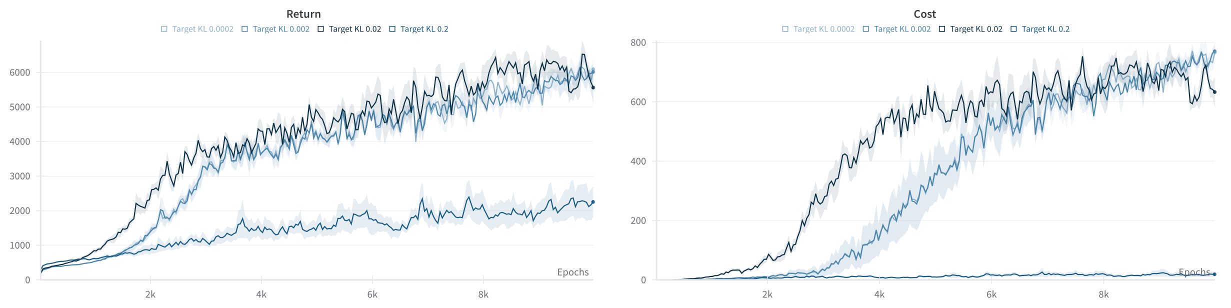


Figure 28: PPO with varying KL target values.

Figure 28 shows that the behavior of target values at or smaller than the default remains consistent while larger values prove less effective and experience smaller returns.

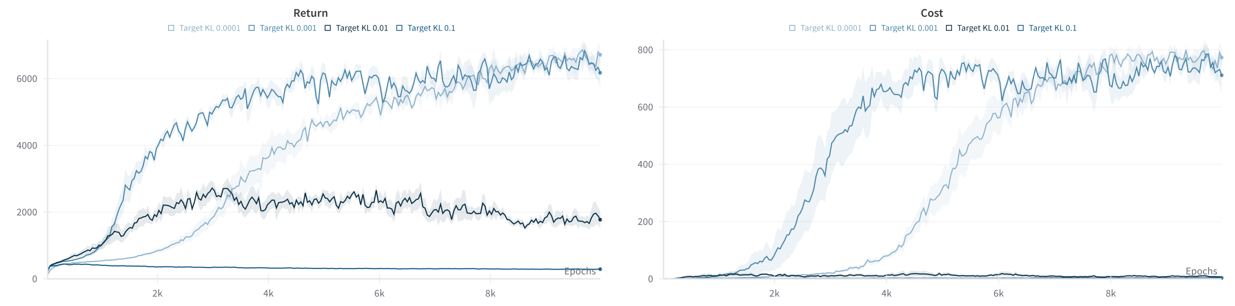


Figure 29: TRPO with varying KL target values.

Figure 29 shows that TRPO is more sensitive to differing target values, likely because this metric is used in the creation of the trust regions. The default value of 0.01 had both smaller returns and costs than 0.001 and 0.0001. The increase in both returns and costs suggests that the smaller target values allow the policy to learn to run, which gives positive returns and incurs costs when the velocity threshold is violated. As with PPO, the largest target value of 0.1 allowed policy changes too drastic to optimize returns.

9.2 *Car* Policy Update Improvements

For *Car* we worked on improving the policy updates by increasing the number of steps per epoch.

9.2.1 Adjusting Number of Steps per Epoch

For our first experiment we ran PPOLag and TRPOLag with 1024 and 2048 steps per epoch with a cost limit of 100.

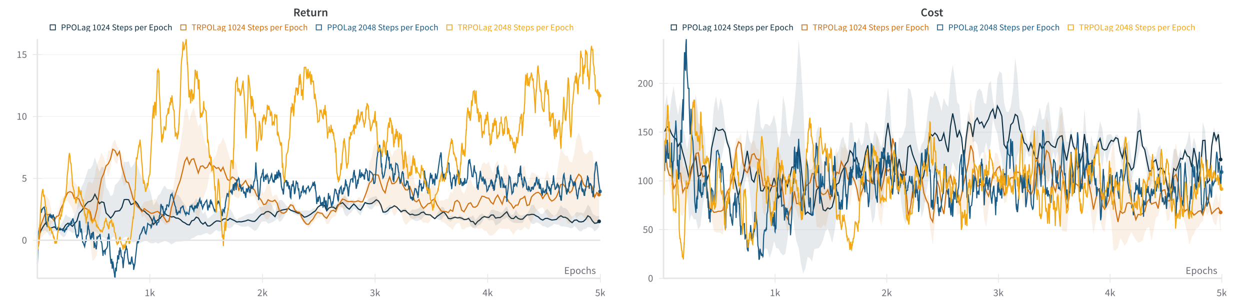


Figure 30: PPOLag and TRPOLag with varying steps per epoch with cost limit 100.

Figure 30 shows that both PPOLag and TRPOLag see increased returns when the number of steps is doubled.

We ran one final experiment with PPOLag and TRPOLag with 20,000 steps per epoch with a cost limit of 100.

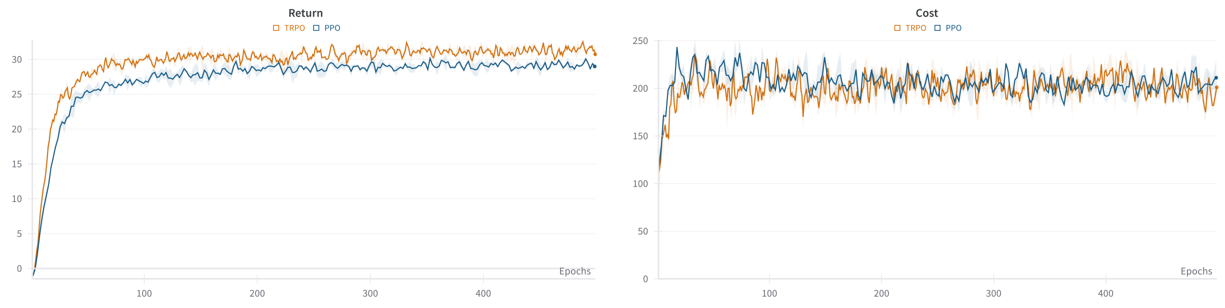


Figure 31: PPO and TRPO trained for 500 epochs with 20,000 steps per epoch.

Figure 31 shows that with 20,000 steps per epoch, TRPO achieves higher returns than PPO. This is the first experiment where we have seen TRPO outperform PPO, showing that TRPO benefits more from additional runs between policy updates than PPO does.

This could due to PPO's hypothesized sample efficiency. Higher sample efficiency would result in better performance in the 1024 steps per epoch runs and we would then observe less drastic improvement when the samples are doubled in the 2048 steps per epoch runs. This is exactly what the results show, but proving that this is due to sample efficiency is difficult. In their original paper, Schulman et al. found that in a continuous control experiments PPO performed similarly to sample efficient algorithms; this lead them to the conclusion that PPO is empirically more sample efficient, and this is the claim that they make [25]. We were unable to find any research that has successfully verified[21] this claim, especially when comparing PPO to TRPO. As such, we are unable to confidently claim that the difference between the algorithms different rates of improvement when adding more samples is due to PPO's relative sample efficiency. We will however echo Schulman et al.'s findings about the empirical differences; with our implementation of PPO and TRPO, PPO appears to have higher sample efficiency.

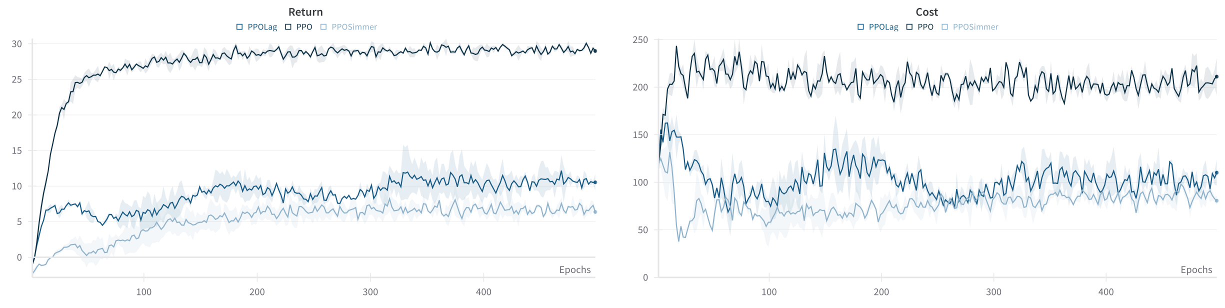


Figure 32: PPO versus PPOLAG and PPOSIMMER with acceptable cost 100 trained for 500 epochs with 20,000 steps per epoch.

Figure 32 shows that PPOLAG and PPOSIMMER benefit greatly from an increased epoch

[21]Most articles and blogs that claim that PPO is more sample efficient either provide no rationale or credit the use of a surrogate objective. As TRPO also uses a surrogate objective, this is not a satisfying argument for why PPO performs better with fewer samples.

length. Figure 15 showed that with 1024 steps per epoch and an acceptable cost of 100, PPOLag achieved returns under 5 and PPOSimmer did not reliably achieve returns greater than 0. With 20,000 steps per epoch, PPOLag achieves returns of 10 and PPOSimmer achieves returns greater than 5.

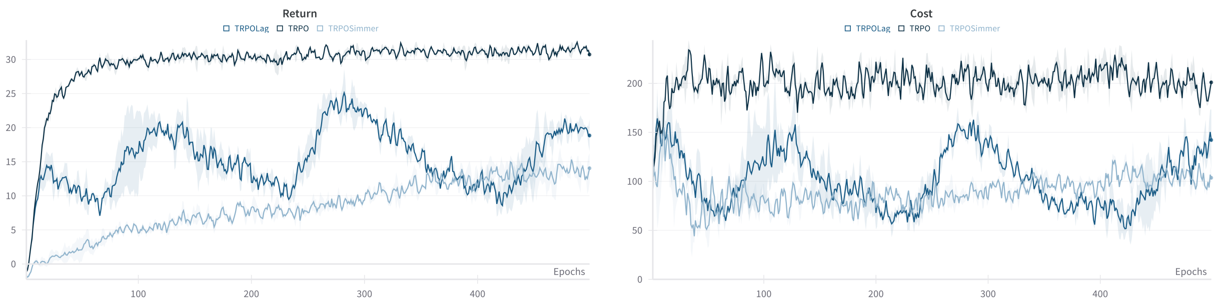


Figure 33: TRPO versus TRPOLag and TRPOSimmer with acceptable cost 100 trained for 500 epochs with 20,000 steps per epoch.

Figure 33 shows that TRPOLag and TRPOSimmer also benefited greatly from the longer epochs. Figure 16 showed that with 1024 steps per epoch and an acceptable cost of 100, TRPOLag achieved returns of less than 5 and TRPOSimmer consistently achieved returns less than 0. With 20,000 steps per epoch TRPOLag achieved returns ranging between 10 and 25 and TRPOSimmer achieved returns close to 15. TRPOLag again exhibits cyclical returns and costs. In our 500 epochs of training we do not see these metrics plateau, a possible extension could be running training for longer to see if they do eventually level out.

The single change of reducing epoch count while increasing epoch length was by far the most effective adjustment we made. It not only increased returns, it did not increase costs incurred by the constrained algorithms.

9.3 Policy Update Conclusions

As we saw with both *Humanoid* and *Car*, TRPO's policy updates were too extreme given 1024 steps per epoch and KL target 0.01. To correct this, we used two strategies: we increased the number of steps per epoch and decreased the target KL hyperparameter.

First, we found that increasing the number of steps per epoch, and consequently, the number of episode runs between policy updates, greatly improves returns. Intuitively, having the agent run the policy multiple times before an update provides a better sense of how the policy behaves. This additional data allows the algorithm to better redistribute the probabilities toward actions that yield higher rewards more frequently.

Second, we limited how much the policy could change by reducing the KL target in *Humanoid*. Although there is a high degree of variance when the number of episodes in between policy updates is small, reducing how much each policy update could change mitigated this issue. We suspect that with smaller, more frequent updates, even highly variable policy updates could move the policy in the right direction on average as no individual update could distort the policy distribution too much.

10 Discussion and Future Work

10.1 More Experiments

With more time and computing resources, we would have liked to re-run many of the experiments with more steps per epoch. As the final experiments show, increasing the steps per epoch was the most effective in improving performance. The initial experiments were all updated too frequently for optimal policy performance, while the more optimized algorithms may have yielded different results with the same hyperparameter changes. Unfortunately, we are constrained by time and our computers, and were unable to re-run all of the experiments.

10.2 More Environments

Experimenting with custom environments in OmniSafe would provide us with proof of environmental difference on algorithms and more results to draw conclusions from. A full comparison of all environments would reveal which types of environments the different algorithms did better on, and how the environment shapes the behavior of different types of safety. For example, in an environment with many obstacles, we would expect the safe versions to diverge more from the unsafe ones, as they are forced to deviate more from the reward-optimal path. We would also be able to see if the conclusions that we drew from the two environments that we used hold up in more complex and dynamic environments. Currently, this would have been computationally expensive as well as time-consuming, so we opted to create more in-depth experiments with only two environments.

10.3 Ethics

When considering agents that would directly interact with people, ethics are preferable over safety. Safety is a good goal to strive to, but when dealing with the complexity of society the agent needs to be able to make ethically sound and justifiable decisions. Neufeld et al. investigated enforcing ethics on safe reinforcement learning using defeasible deontic logic [19]. The rules that the agent needs to follow are structured so that some actions are allowed, permissible under certain circumstances, or forbidden. The supervisor/ethical component that enforces the rules on the agent also acts as a recorder, to provide an explanation behind the agent's actions. This is a major improvement on most safety formulations, because when forced to violate some safety constraint the agent can explain why it choose actions that may not appear safe to the outside viewer.

Furthermore, when discussing ethical actions, the concept of "lesser evil" comes up when weighing one bad action against another [19]. It is nearly impossible to design an agent that will never violate any constraint, there are many situations where the agent has no safe options. There needs to be a consistent and defensible strategy for the agent to then decide on the best option given the circumstances. They explore this concept of "lesser evil" much more in-depth in their paper. With more time, we would have liked to implement ethics on our learning model rather than safety, and we believe this would be a value extension on our work.

We also would have liked to run ethical agents in custom environments. Moving obstacles would force the agent into situations where not moving is a cost because it results in environmental objects colliding with the agent. It also makes the environment more variable, because there is another actor in the environment with the potential to move objects. If this other obstacle was another agent, we could also impose extra costs for colliding with the other agent, as it impairs both agent's ability to complete their task. This could yield interesting results for how the algorithms and agents learn how to operate under more difficult situations and complex rules.

11 Final Remarks

Thoughtful and extensive work should be put into examining that "safety" truly looks like for digital agents as they become more prevalent. We have seen many recent examples in the news of improperly implemented agents that cause havoc in society, such as an airline chatbot giving incorrect information [16] and self-driving cars causing a myriad of issues [12]. This is especially concerning for cars, as they can cause traffic incidents, block roads, and injure people. For society to function with these new digital agents, there needs to be accountability and transparency, which is currently lacking. This fosters distrust, which could be at least partially addressed by more communication about what ethics should (and do) look like with digital agents, and what society can expect from them. While solving these problems is not easy, it is incredibly important for both society and the future of RL that agents are safe and ethical.

Acknowledgments

We would like to thank our advisor, Anna Rafferty, for her excellent advice and support throughout the entire comps process. We would also like to thank the Carleton Computer Science department faculty and staff for technical help and general resources. Outside of Carleton, our project would not have been feasible without the work of the OmniSafe team, the Weights and Biases team, and Dr. Pieter Abbeel. We greatly appreciate the larger reinforcement learning community for being supportive and welcoming to beginners.

References

[1] Pieter Abbeel. *Foundations of Deep RL – 6-lecture series by Pieter Abbeel.* YouTube. Aug. 24, 2021. URL: https://www.youtube.com/playlist?list=PLwRJQ4m4UJjNymuBM9RdmB3Z9N5-0IlY0.

[2] Pieter Abbeel. *L3 Policy Gradients and Advantage Estimation (Foundations of Deep RL Series).* YouTube. Aug. 25, 2021. URL: https://www.youtube.com/watch?v=AKbX1Zvo7r8.

[3] Pieter Abbeel. *L4 TRPO and PPO (Foundations of Deep RL Series).* YouTube. Aug. 25, 2021. URL: https://www.youtube.com/watch?v=KjWF8VIMGiY.

[4] Joshua Achiam, Alex Ray, and Dario Amodei. *Safety Gym.* en-US. URL: https://openai.com/research/safety-gym (visited on 01/14/2024).

[5] Lukas Biewald. *Weights and Biases.* 2020. URL: https://www.wandb.com/.

[6] Shangding Gu et al. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications.* arXiv:2205.10330 [cs]. Feb. 2023. URL: http://arxiv.org/abs/2205.10330 (visited on 01/08/2024).

[7] *Gymnasium Documentation.* URL: https://gymnasium.farama.org/index.html (visited on 01/14/2024).

[8] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. *Cautious Reinforcement Learning with Logical Constraints*. arXiv:2002.12156 [cs, eess, stat]. Mar. 2020. DOI: 10.48550/arXiv.2002.12156. URL: http://arxiv.org/abs/2002.12156 (visited on 02/16/2024).

[9] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*. 2012.

[10] Jiaming Ji et al. *OmniSafe: An Infrastructure for Accelerating Safe Reinforcement Learning Research*. arXiv:2305.09304 [cs]. May 2023. DOI: 10.48550/arXiv.2305.09304. URL: http://arxiv.org/abs/2305.09304 (visited on 01/23/2024).

[11] Sham M Kakade. "A Natural Policy Gradient". In: *Advances in Neural Information Processing Systems*. Vol. 14. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper_files/paper/2001/hash/4b86abe48d358ecf194c56c69108433e-Abstract.html (visited on 03/12/2024).

[12] Dara Kerr. "Armed with traffic cones, protesters are immobilizing driverless cars". en. In: *NPR* (Aug. 2023). URL: https://www.npr.org/2023/08/26/1195695051/driverless-cars-san-francisco-waymo-cruise (visited on 03/08/2024).

[13] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[14] David Knowles. "Lagrangian Duality for Dummies". In: ().

[15] Yuxi Li. "Reinforcement Learning Applications". In: *CoRR* abs/1908.06973 (2019). arXiv: 1908.06973. URL: http://arxiv.org/abs/1908.06973.

[16] Kyle Melnick. "Air Canada chatbot promised a discount. Now the airline has to pay it." en-US. In: *Washington Post* (Feb. 2024). ISSN: 0190-8286. URL: https://www.washingtonpost.com/travel/2024/02/18/air-canada-airline-chatbot-ruling/ (visited on 03/08/2024).

[17] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. June 16, 2016. arXiv: 1602.01783[cs]. URL: http://arxiv.org/abs/1602.01783 (visited on 03/11/2024).

[18] Teodor Mihai Moldovan and Pieter Abbeel. *Safe Exploration in Markov Decision Processes*. arXiv:1205.4810 [cs]. July 2012. URL: http://arxiv.org/abs/1205.4810 (visited on 01/07/2024).

[19] Emery A. Neufeld et al. *Enforcing ethical goals over reinforcement-learning policies*. en. Sept. 2022. DOI: 10.1007/s10676-022-09665-8. URL: https://doi.org/10.1007/s10676-022-09665-8 (visited on 02/16/2024).

[20] PKU-Alignment. *Safety-Gymnasium Documentation*. URL: https://safety-gymnasium.readthedocs.io/en/latest/ (visited on 03/12/2024).

[21] B. T. Polyak. "Iterative methods using Lagrange multipliers for solving extremal problems with constraints of the equation type". In: *USSR Computational Mathematics and Mathematical Physics* 10.5 (1970), pp. 42–52. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(70)90036-4. URL: https://www.sciencedirect.com/science/article/pii/0041555370900364.

[22] *Proximal Policy Optimization — Spinning Up documentation.* 2018. URL: `https://spinningup.openai.com/en/latest/algorithms/ppo.html` (visited on 01/14/2024).

[23] Herbert E. Robbins. "A Stochastic Approximation Method". In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407. URL: `https://api.semanticscholar.org/CorpusID:16945044`.

[24] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation.* Oct. 20, 2018. arXiv: `1506.02438[cs]`. URL: `http://arxiv.org/abs/1506.02438` (visited on 03/11/2024).

[25] John Schulman et al. *Proximal Policy Optimization Algorithms.* 2017.

[26] John Schulman et al. "Trust Region Policy Optimization". In: *International conference on machine learning.* PMLR. 2015, pp. 1889–1897.

[27] Aivar Sootla et al. *Effects of Safety State Augmentation on Safe Exploration.* arXiv:2206.02675 [cs]. Oct. 2022. URL: `http://arxiv.org/abs/2206.02675` (visited on 01/24/2024).

[28] Aivar Sootla et al. "Sauté RL: Almost Surely Safe Reinforcement Learning Using State Augmentation". In: *International Conference on Machine Learning.* PMLR. 2022, pp. 20423–20443.

[29] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[31] Richard S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems.* Vol. 12. MIT Press, 1999. URL: `https://proceedings.neurips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html` (visited on 03/12/2024).

[32] OmniSafe Team. *Proximal Policy Optimization -.* 2022. URL: `https://www.omnisafe.ai/en/latest/baserl/ppo.html#` (visited on 03/03/2024).

[33] OmniSafe Team. *Trust Region Policy Optimization -.* 2022. URL: `https://www.omnisafe.ai/en/latest/baserl/trpo.html` (visited on 03/03/2024).

[34] *Trust Region Policy Optimization — Spinning Up documentation.* 2018. URL: `https://spinningup.openai.com/en/latest/algorithms/trpo.html` (visited on 01/14/2024).

[35] Xinglong Zhang et al. *Model-Based Safe Reinforcement Learning with Time-Varying State and Control Constraints: An Application to Intelligent Vehicles.* arXiv:2112.11217 [cs]. Aug. 2023. DOI: `10.48550/arXiv.2112.11217`. URL: `http://arxiv.org/abs/2112.11217` (visited on 01/14/2024).

Appendices

A Theoretical Background

A.1 Expected Value

Given a random variable X with a countably infinite set of possible outcomes $x_1, x_2, \ldots$, the *expected value* of X under a probability distribution p is defined as follows:

$$\mathbb{E}_p[X] \coloneqq \sum_{i=1}^{\infty} p(x_i)x_i.$$

If X were an uncountably infinite set, we could generalize this definition by introducing an integral. In practice, however, computational constraints require us to compute finite sums when estimating expected values, so we choose to omit this generalized definition.

A.2 Baseline Vanishes on Expectation

In Section 3.3 we claim that adding the baseline b to the policy gradient equation does not introduce bias on expectation. To prove this, we rewrite Equation 3.5 in terms of expected value, giving us the following:

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)(R(\tau)-b)\right] &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)R(\tau) - \nabla_\theta \log P_{\pi_\theta}(\tau)b\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)R(\tau)\right] - \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)b\right].\end{aligned}$$

Hence, showing that $\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)b\right] = 0$ proves that Equation 3.5 is equal to the gradient expressed in Equation 3.3 on expectation and is thus unbiased. We prove this in the simple case in which b is constant:

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log P_{\pi_\theta}(\tau)b\right] &= \mathbb{E}_{\pi_\theta}\left[\frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)}b\right] \\ &= \sum_\tau P_{\pi_\theta}(\tau)\frac{\nabla_\theta P_{\pi_\theta}(\tau)}{P_{\pi_\theta}(\tau)}b \\ &= \sum_\tau \nabla_\theta P_{\pi_\theta}(\tau)b \\ &= \nabla_\theta \sum_\tau P_{\pi_\theta}(\tau)b \\ &= b\nabla_\theta \sum_\tau P_{\pi_\theta}(\tau) \\ &= b\nabla_\theta(1) \\ &= b \cdot 0 \\ &= 0.\end{aligned}$$

This derivation can be generalized to all choices of b that do not depend on the policy parameter θ. In other words, as long as our choice of b at a given time step is not dependent on the chosen action at that time step (which depends on π_θ and thus indirectly on θ), we can safely use Equation 3.5 as an unbiased empirical estimate of the gradient.

A.3 Asynchronous Advantage Actor-Critic Example

Here, we outline the advantage estimation technique from [17] with a concrete example. If $n = 2$, the estimation discussed in Equation 3.10 is equivalent to

$$R(s_t^{(i)}, a_t^{(i)}) + \gamma R(s_{t+1}^{(i)}, a_{t+1}^{(i)}) + \gamma^2 V_\phi(s_{t+2}).$$

When we subtract $V_\phi(s_t^{(i)})$ to compute advantage, we get the following:

$$\begin{aligned}\left(\sum_{k=t}^{T-1} \gamma^k R(s_k^{(i)}, a_k^{(i)})\right) - V_\phi(s_t^{(i)}) &\approx R(s_t^{(i)}, a_t^{(i)}) + \gamma R(s_{t+1}^{(i)}, a_{t+1}^{(i)}) + \gamma^2 V_\phi(s_{t+2}) - V_\phi(s_t^{(i)}) \\ &= \left[R(s_t^{(i)}, a_t^{(i)}) + \gamma R(s_{t+1}^{(i)}, a_{t+1}^{(i)})\right] - \left[V_\phi(s_t^{(i)}) - \gamma^2 V_\phi(s_{t+2})\right].\end{aligned}$$

The first group $R(s_t^{(i)}, a_t^{(i)}) + \gamma R(s_{t+1}^{(i)}, a_{t+1}^{(i)})$ represents the first two terms in the empirical sum of future discounted rewards, and the second group $V_\phi(s_t^{(i)}) - \gamma^2 V_\phi(s_{t+2})$ represents the expectation for this two-term sum, as estimated by the value function V_ϕ.

A.4 Breakdown of Lagrangian Formulation

Equation 5.4 is defined as follows:

$$\max_\theta \min_{\lambda \geq 0} \left(\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] - \lambda(\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] - d)\right).$$

To understand what this equation represents, it is useful to analyze the inner minimum function in two cases:

$$\min_{\lambda \geq 0} \left(\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] - \lambda(\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] - d)\right). \tag{A.1}$$

If $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] - d > 0$, then the cost constraint has been violated. Since Equation A.1 minimizes over all nonnegative values of λ, we let λ be arbitrarily large. Loosely speaking, this yields a value of $-\infty$. Alternatively, if $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} C(s_t, a_t)\right] - d \leq 0$, then the cost constraint has not been violated. In this case, we minimize Equation A.1 by setting λ equal to 0. Thus, the minimization yields a value of $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right]$.
By combining these two cases, we have that Equation A.1 is equivalent to the following:

$$\begin{cases} -\infty & \text{if the constraint is violated on expectation} \\ \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] & \text{otherwise.}\end{cases}$$

Maximizing this piecewise function over θ guarantees that the constraint is not violated on expectation.[22] This is exactly why the objective described in Equation 5.4 takes the maximum of Equation A.1 over θ.

[22]If the constraint were violated, the piecewise function above would equal $-\infty$ and thus fail to be maximized.