Projected Proximal Policy Optimization

Course project for Theory and Methods of Reinforcement Learning, EE-618 at EPFL

Sergei Volodin
Department of Computer Science
Swiss Federal Institute of Technology in Lausanne (EPFL)
1015-Lausanne
`sergei.volodin@epfl.ch`

Abstract

We propose Projected Proximal Policy Optimization, a novel Safe Reinforcement Learning algorithm for Constrained Markov Decision Processes. We show that it works in a proof-of-concept setting. We compare it with established in that domain baselines such as CPO and sDQN on simple environments such as Cartpole. We publish the code with safe RL benchmarks environments as a separate contribution.

1 Introduction

Artificial intelligence solves more and more problems and concerns were raised on whether or not it will be beneficial (1). Aside from long-term issues (2), already there are pressing problems in AI safety, for example, adversarial examples or safe exploration (1). For each of these issues, a tradeoff between performance and not causing unintended harm is required.

In this project we consider the issue of safe exploration, which is one of the two main directions in safe RL, as indicated in (3). Safe exploration, in its simplest form, means that an agent does not visit certain dangerous states. It can be achieved for example via constraints on the behavior, which can be either worst-case or on the expected value. Another possibility is to use inverse (imitative) reinforcement learning where rewards are learned from the provided "good" behavior which is assumed to be safe. The field of safe RL already has successful applications. A notable project tunes parameters of a quadcopter controller without causing harm to it (4; 5).

Currently, there are already methods for safe RL (6; 7). One of the main disadvantages is the complexity of implementation. All the methods use either an inner optimization problem including Hessians, or cumbersome closed-form solution to it. The complexity comes from the fact that the methods need to keep the solution close to the previous one, resulting in a Hessian computation or a dual problem. We instead circumvent this issue by using the same approach as in Proximal Policy Optimization (PPO) (8), which does not encourage the policy to differ from the current one instead of imposing constraints.

Currently, for discrete-state problems, there is a certain number of safe RL benchmarks (9) (i.e., benchmarks for which safety criteria which should be met are defined). On the other hand, this does not appear to be the case in the field of continuous-state problems. In the currently available research work concerning methods for safe continuous-state RL, the used benchmark environments are obtained by taking some available RL benchmarks studied in contexts not considering safety aspects and by then imposing constraints on them. We plan to use a set of simple environments from OpenAI Gym (Cartpole-v0, others planned) (10), add constraints to them and make the resulting environments available as open-source code thus going in a direction of standardized continuous-state safe RL benchmarks.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

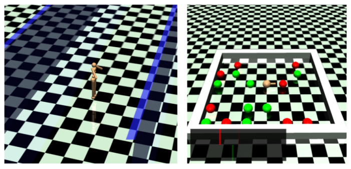
(a) Humanoid-Circle (b) Point-Gather

Figure 2. The Humanoid-Circle and Point-Gather environments. In Humanoid-Circle, the safe area is between the blue panels.

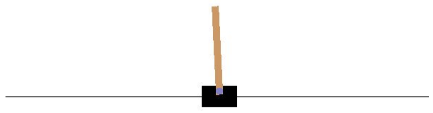

Figure 1: (Left) Safe RL environments from (7) (Right) CartPole-v0 environment

Contribution. We first propose a series of benchmarks for safe RL, like (9) but for continuous-state tasks. Then we compare existing algorithms (7; 6) on our benchmarks. After that we state Projected Proximal Policy Optimization, a novel safe RL algorithm and compare it with existing algorithms.

Outline. We first describe the motivation behind a set of benchmarks and our proposal in Section 2. Then we define the CMDP framework in Section 3 and existing agents (CPO, sDQN) in Section 3.1. Then we define PPPO in Section 3.2. Finally, in Section 4 we compare agents on our benchmarks. We conclude in Section 5. Appendices A contain the original project proposal and interim reports.

Our code is available at

`https://github.com/sergeivolodin/SafeContinuousStateRL`

2 Benchmarks for safe RL

In this section we state why safe RL needs standartized benchmarks, and then propose a solution to this problem.

Previously, algorithms for safe RL were evaluated on environments (7; 6) based on Gym with additional constraints added to them. Each project re-implements the constraints. However, the issue with this approach is that implementation might differ and therefore it might be unreasonable to compare methods just by looking at final results. RL already has standartized benchmarks, one of them is OpenAI Gym (10). It seems that Safe RL could benefit from a similar contribution.

We propose to create a set of environments for Gym which would follow the *decorator* design pattern (11). These would be classes having original Gym environment as a member. Constraint function would be a member function for that class. Constrained Environments would be instances of an abstract `ConstrainedEnvironment` class, which would differ from an original environment class in the `step()` method. Specifically, a constrained environment would also return the cost.

Papers (7) uses the following tasks based on Point, Ant and Humanoid environments from MuJoCo.

1. Circle: reward is given for running in a circle, constraint enforces staying in a smaller circle.
2. Gather: collecting green apples, avoiding red bombs (See Figure 1, left).

It is also worth including simpler environments from the classic control domain such as CartPole (Figure 1, right), LunarLander, etc because it is faster to train the agent on them. In this project, we only use CartPole. We propose to use $x \leq 0$ as a constraint for CartPole (the cart must stay on the left part) because of its simplicity.

Current progress. Currently, the code implements a `ConstrainedEnvironment` with CartPole as an example and a `ConstrainedAgent` with CPO and PPPO as examples, see `saferl.py`, `costs.py` and `baselines.py`.

We have defined the environments we will use and now we define the agents.

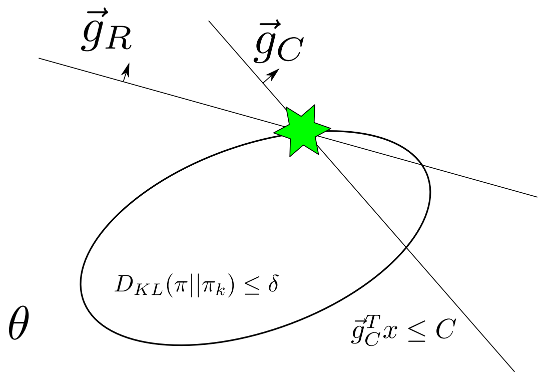

Figure 2: Constrained Policy Optimization (7)

3 Theory

In this section, we define the CMDP framework and state the implementation of existing algorithms as well as of our proposed algorithm

Definition 1. CMDP or an *environment* is a tuple $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \gamma, x_0, R, C, C_{\max}, P)$ where $\mathcal{S}$ is a set of *states*, $\mathcal{A}$ the set of *actions*, $\gamma \in (0, 1]$ the *discount factor*, x_0 the *initial state* distribution, *reward-like* $R, C\colon \mathcal{S} \times \mathcal{A} \to \Delta\mathbb{R}$[1] the *reward* and *cost* functions respectively. A stationary *policy* is a function $\pi\colon \mathcal{S} \to \Delta\mathcal{A}$. The *return operator* for a reward-like function T is defined as $J_{\mathcal{E},\pi}[T] = \sum\limits_{t=0}^{\infty} \mathbb{E}_{s_t \sim \pi \leftrightarrow \mathcal{E}} \gamma^t T(s_t)$ where $\pi \leftrightarrow \mathcal{E}$ is the distribution of states in an interaction between π and $\mathcal{E}$. The goal is to find a stationary policy maximizing reward return with constrained cost return:

$$\pi^* = \arg \max_{\pi\colon J_{\mathcal{E},\pi}[C] \leq C_{\max}} J_{\mathcal{E},\pi}[R]$$

In this project, we use $\mathcal{A}$ finite and $\mathcal{S} = \mathbb{R}^n$ finite-dimensional vector space.

3.1 Existing approaches

In this section, we go through different approaches to CMDPs and show their common elements.

Constrained Policy Optimization (7). This algorithm requires a policy network π_θ. After collecting a set of rollouts, it solves the following optimization problem: $\max g_R^T \Delta\theta$ s.t. $g_C^T \Delta\theta + J[C] \leq C_{\max}$ and $\Delta\theta^T \frac{\partial^2}{\partial\theta^2} D_{KL}(\theta'|\theta)\Delta\theta \leq \delta$ for $\Delta\theta = \theta' - \theta$. Here g_R and g_C are gradients of $J[R]$ and $J[C]$ estimated using the Policy Gradient theorem. Figure 5 demonstrates the problem solved. Intuitively, we want to optimize reward and stay within the cost constraints, and we use a first-order Taylor expansion to satisfy those. Moreover, we want the new policy $\pi_{\theta'}$ be not far from the original policy π_θ. This is achieved using the KL divergence. Using a simpler constraint $\Delta\theta^T \Delta\theta \leq \delta$ would be less efficient as a small change in the parameters can change the output probabilities significantly. The problem of having to compute a Hessian is resolved by switching to the dual space where the problem becomes 2-dimensional. An explicit solution is presented in the original paper. The method has a theoretical guarantee of the form "if δ is low enough, and second (third) order terms are small, the algorithm finds an improvement". In case if it was not possible to solve the problem, the method follows the natural policy gradient for $-J[C]$ to satisfy the constraint. The authors implement the method in RLLab, however, we were not successful in running it as RLLab is no longer supported. We provide our own implementation of CPO and note that it is not fully tuned.

Lyapunov-based methods (6). These methods are based on *Lyapunov* functions which are used to certify stability in dynamical systems. The method uses Lyapunov functions L of the form $J_\pi[L](x) \leq L(x)$. The Safe DQN algorithm works as follows. Having a safe policy π_k as a neural network, we estimate Q_R (reward Q-value), Q_C (cost Q-value) and Q_T (discounted stopping time Q-value) as neural networks via Bellman updates. Next, we construct a Lyapunov function

[1] ΔX means a set of all probability distributions over X

Q-value via $Q_L = Q_C + \varepsilon Q_T$ (correctness proven in the main paper) with $\varepsilon = C_{\max} - \frac{\pi_k^T Q_D}{\pi_k^T Q_T}$. At each step, we solve a linear program $\pi^T Q_R \to \max$ s.t. $(\pi - \pi_k)^T Q_L \le \varepsilon$ for π. In case if it is infeasible, we only learn Q_R, Q_C, Q_T using Bellman steps. After we make a supervised step $D_{JSD}(\pi|\pi_k) \to \min$[2]. Unfortunately, the authors do not provide the code with the paper and we have used own implementation. The version of the method with approximate Q-values does not have theoretical guarantees. A drawback of this method is that after each rollout, need to call a TensorFlow computation twice: once to get Q_R, Q_L and then to do the supervised JSD step, since TF cannot solve the LP. To solve the LP, we use the `pulp solver` as the problem is sparse. The paper does not describe in what sequence these quantities need to be trained. The issue is that to train a good π, we need a good Q and vice versa. In our implementation, we first train Qs with greedy action-selection and then switch on π training. However, it simply unlearns everything after enabling safety.

Common elements. We see that in order to solve Constrained MDPs, at each training step, a constrained optimization problem is solved. That problem approximates the original one with 1st or 2nd order Taylor approximation. It has three components: reward maximization, first-order hard constraint for cost, and closeness to original policy π_k.

Other methods. A trivial approach to solve CMDPs would be to consider a modified reward $R' = R - \lambda C$ with a learnable Lagrange multiplier λ. The problem with this method is that it becomes unstable (7).

We now have an understanding of how algorithms for Safe RL work and we will present our own.

3.2 Proposal: Projected PPO

In this section, we present PPPO, a novel algorithm for CMDPs.

The algorithms described in the previous section are quite complex. In this project, we try to propose a simpler algorithm. Specifically, we want to replace the inner constrained optimization problem (linear or quadratic) with projection. In order to ensure closeness to the current policy, we will use the PPO-clip objective (8) instead of constraints like in CPO. PPO shows promising results, therefore it makes sense to make it safe. Up to our knowledge, there is no safe easy-to-implement PPO algorithm.

Since PPO update enforces closeness, the return can be approximated using first-order Taylor expansion. Therefore constraint becomes linear: $J_\pi[C] + (\theta - \theta')^T g_C \le C_{\max}$. In order to satisfy it, after each policy improvement using the PPO-clip objective, the resulting θ is projected to that half-plane. See Algorithm 1 for details. The advantage of this method is that it is easier to implement than CPO or Lyapunov, because it does not have an inner optimization problem.

Now we have defined the baselines and the proposed algorithm and we compare them in simple environments.

4 Experimental evaluation

In this section, we compare CPO, Lyapunov method sDQN, PPPO and a random agent on simple environments.

We use Cartpole-v0. We considering the problem solved if constraint was violated $< 1\%$ of the training time and a reward of at least 175 was achieved (maximum is 200). Agents are compared by the mean over repetitions, and the max over the training phase, reward for which the cost was satisfactory $< 100 = C_{\max}$. We use $\gamma = 1$. The Lyapunov method did not converge, see the description above.

For the CPO policy function we use a network with 10 hidden nodes and a sigmoid activation function with truncated normal initialization for both biases and weights. For the PPPO policy network we use a common layer of 10 nodes. Next, policy and value heads have other separate 10-node layers. Weights are initialized using the LeCun normal distribution and biases are set to 0. Activation is a sigmoid.

We tune hyperparameters in the following way. We fix the number of episodes to collect before training to 5. We repeat each experiment 5 times with different initializations and random seeds.

[2]Jensen-Shannon Divergence: $D_{JSD}(p, q) = \frac{1}{2}(D_{KL}(p||r) + D_{KL}(q||r))$ for $r = \frac{1}{2}(p + q)$

```
**Input:** Initial policy and value parameters $\pi_\theta$, $V_\varphi$;
**Result:** Safe Policy $\pi$ obtained via safe learning
**while** *not achieved the desired reward* **do**
    Collect a set of trajectories by running $\pi_{\theta_k}$ in $\mathcal{E}$;
    Compute rewards-to-go $\hat{R}_t$;
    Compute advantage estimates $\hat{A}_t$ based on $V_\varphi$;
    Compute the cost return $J_\pi[C]$ and the constraint gradient $g_C$;
    **if** *The current policy is safe:* $J_\pi[C] \le C_{\max}$ **then**
        Update the policy by optimizing PPO-clip using Adam:;
            $$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t,a_t), g(\varepsilon, A^{\pi_{\theta_k}}(s_t,a_t)) \right)$$
         Here $g(\varepsilon, A) = (1+\varepsilon)A$ if $A \ge 0$ or $(1-\varepsilon)A$ if $A < 0$;
        Project $\theta_{k+1}$ to a half-plane $(\theta_{k+1} - \theta_k)^T g_C + J_\pi[C] \le C_{\max}$;
    **else**
        Do a policy gradient step to minimize the cost:;
            $$\theta_{k+1} = \theta_k - \eta g_C$$
    **end**
    Fit the value function via Adam:
        $$\varphi_{k+1} = \arg\min_\varphi \frac{1}{|\mathcal{D}_k|T} \sum_{t\in\mathcal{D}_k} \sum_{t=0}^{T} (V_\varphi(s_t) - \hat{R}_t)^2$$
**end**
**Return** $\pi_\theta$
```

Algorithm 1: Projected Proximal Policy Optimization. Red shows differences between PPPO and PPO.

For CPO, we change the step size $\delta \in \{0.1, 0.5, 0.05, 0.01, 0.005, 0.001\}$ and the best option is $\delta^* = 0.05$ by the metric above. For PPPO, we set $\varepsilon = 0.1$ as recommended by (8). We use Adam for the PPO-clip objective with learning rates in $\gamma_p = \{0.1, 0.001\}$ and for the value function we use Gradient Descent with the same set of learning rates γ_v. For the fallback option we also use Gradient Descent with the same set of learning rates γ_s. We search for the optimal number of optimization steps $k \in \{1, 5, 10, 20\}$. The best options are: $\gamma_p = 0.001$, $\gamma_v = 0.1$, $\gamma_s = 0.001$ and $k = 5$.

The results for the working agents (CPO, PPPO, random) are shown in Figure 3. It shows that CPO has faster convergence than PPPO but it violates constraints more frequently. The performance is similar. Both agents are better than the random agent.

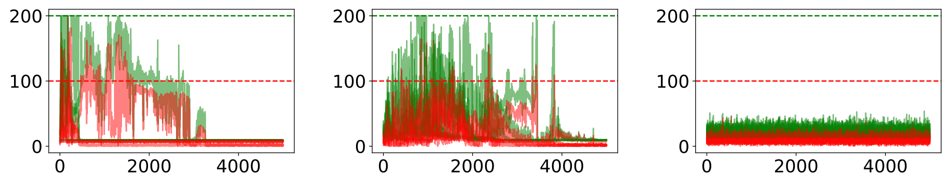

Figure 3: Agents: CPO, PPPO, Random on Cartpole-v0, best hyperparameters. 5 repetitions of a single experiment are shown on the same plot.

We compare agents on a simple environment and see that PPPO is better than CPO in terms of constraint satisfaction. However we note that our implementation of CPO is not well-tuned. To make the comparison more fair, it would make sense to use authors' implementation instead of our own.

5 Conclusion

We consider safe continuous-state reinforcement learning in the context of CMDPs. We first propose to standartize existing benchmarks as they vary from paper to paper. We implement existing safe RL algorithms such as CPO and sDQN. We propose our own algorithm, PPPO, which is simpler to implement. We compare the agents on a simple environment. We show that our implementation of PPPO shows a better balance between constraint satisfaction and reward. We note however that to make the comparison fair, CPO should be tuned in a much more thorough way.

One possible future direction is to finalize the set of proposed safe RL benchmarks by adding the environments from (7; 6) to it. In addition, a theoretical guarantee for PPPO convergence would make it a more reliable solution. Another extension is considering more complex environments for the comparison. In addition, it would make sense to finish the Lyapunov agent and release it as open-source code as the original paper does not provide it.

6 Acknowledgements

I thank Kamalaruban Parameswaran for useful discussions. I thank Ivan Pejcic, a former member of the project for the work on the introduction.

References

[1] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[2] T. Everitt, G. Lea, and M. Hutter, "Agi safety literature review," *arXiv preprint arXiv:1805.01109*, 2018.

[3] J. Garcia and F. Fernandez, "A Comprehensive Survey on Safe Reinforcement Learning," *The Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015.

[4] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 491–496, IEEE, 2016.

[5] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, pp. 908–918, 2017.

[6] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A Lyapunov-based Approach to Safe Reinforcement Learning," 2018.

[7] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained Policy Optimization," 2017.

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

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

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

[11] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[12] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via bayesian optimization," *CoRR*, vol. abs/1703.09260, 2017.

[13] M. Chen, *High Dimensional Reachability Analysis: Addressing the Curse of Dimensionality in Formal Verification*. PhD thesis, EECS Department, University of California, Berkeley, Jul 2017.

[14] I. M. Mitchell, “The flexible, extensible and efficient toolbox of level set methods,” *Journal of Scientific Computing*, vol. 35, pp. 300–329, Jun 2008.

[15] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. Duenez-Guzman, “Lyapunov-based Safe Policy Optimization for Continuous Control,” 2019.

[16] M. Wen and U. Topcu, “Constrained cross-entropy method for safe reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 7450–7460, 2018.

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

A Project proposal

Table A shows a planned timeline with an attempt to meet the NeurIPS submission deadline, which is May 23rd 2019. The planned activities can be approximately divided into three phases:

1. Phase 1: Familiarizing with several methods that seem fruitful for generating a novel idea. This phase would ideally take time until the end of March, but would take as long as an idea for moving into Phase 2 has not appeared. Currently planned flow of Phase 1 is divided into two parallel branches. The first involves implementation of the methods of (7) and (6), which were selected during preliminary considerations as potentially fruitful for development of a novel safe-RL concept. The second branch involves implementation of (12), which is a method based on Bayesian Optimization, and familiarization with (13) and related computational methods (14) which may be of use for safety aspect in Phase 2.
2. Phase 2: Development of the idea obtained from Phase 1. In case that a proper idea appears, it would be further developed in this phase from conceptual and theoretical point of view.
3. Phase 3: Experimental testing in simulation or on a physical system. The testings would first be first done numerically by simulation. There is also a certain probability of getting an opportunity to test the developed method on an inverted pendulum (which is therefore considered as a targeted system in Phase 1) or on a hovercraft.

Week	Sergei	Ivan	Summary
19 March			
26 March			Reading on safe RL
5 March			Reading on safe RL
12 March	Policy gradients for CartPole		Implementing simple methods for simple environments without safety. Project description.
19 March	Implementing paper (7)	Implementing paper (12)	Implementing safe RL methods
26 March	Implementing paper (7)	Reachability analysis (13), level set toolbox (14)	Implementing safe RL methods
2 April	Implementing paper (7)		Formulating safe continuous RL benchmark
9 April	Implementing paper (6)		Formulating safe continuous RL benchmark
16 April	Implementing paper (6)		Evaluating all methods on benchmark
23 April	Implementing paper (6)		Coming up with an extension method
30 April			Evaluating the extension method
7 May			Reserved for unforeseen changes
14 May			Writing and proofreading
21 May			Writing and proofreading

B Interim results I (April 26th)

We have defined safety for the Cartpole-v0 environment as having $x \geq 0$ and $\varphi \geq 0$. We have *trained* Constrained Policy Optimization (7) with vanilla policy gradient (using G_t as a critic) and no backtracking line search. Figure 5 shows that the method works, meaning that it was possible to obtain low cost with high reward.

We are using $k = 20$ episodes to estimate policy gradient and use matrix conditioner of 0.01, discount factor $\gamma = 0.99$, network with sigmoid activations for hidden layers and linear output layer with configuration $[4, 5, 2]$ and CPO step $\delta = 0.01$ for $n = 30$ steps of optimization

The main problem was that implementing (7) took 3 weeks instead of planned 1, the difficulties were caused by many things. First, it required learning pycvx. Moreover, originally method did not average gradients over episodes resulting in a noisy estimate. Moreover, there were issues with implementing vanilla (non-safe) policy gradients.

We are looking at available CPO source code to replace our implementation for a fair comparison, however, it is written for RLLab which requires an older version of Ubuntu to work with.

Currently we have implemented Advantage Actor-Critic method and Proximal Policy Optimization (Figure 4) as step for implementing (6) since it is based on these algorithms. An unpredicted challenge was that implementing the algorithms also takes time.

Our code is available at

`https://github.com/sergeivolodin/SafeContinuousStateRL`

In order to implement (6) (no code is provided with the original paper) we intend to take an existing implementation of PPO and then make it safe as the existing implementation would already possess the necessary additional features such as parallelism and efficient implementation. It does not make sense to make our own implementation that powerful because the goal of familiarizing ourselves with PG, PPO, A2C was achieved.

Plan corrections. We are significantly behind the schedule and currently we plan to remove the Phase 3 (deployment on a physical system). Moreover, Ivan had a thesis deadline and couldn't do his part before May. We still plan to compare methods on the simulated tasks and still plan to try to come up with a theoretical extension to them.

C Interim report 2

Paper (15) is actually for continuous-action spaces. The right paper is (6) and also (16; 17) for entropy and reward-constrained.

Plan: take SDQN code from (6) and implement it, first implementing DQN (should be easy...). Estimated time: 3 hours.

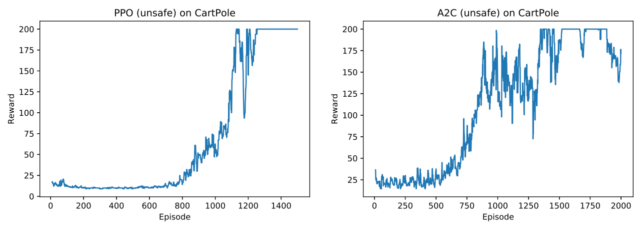

Figure 4: PPO and A2C (our implementation, both unsafe) for CartPole

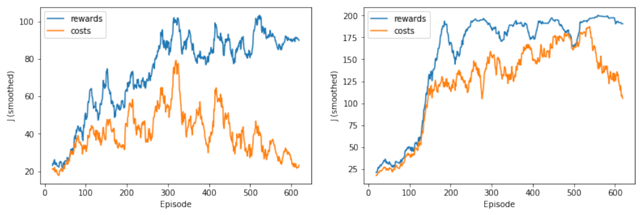

Figure 5: CPO (our implementation, safe) with $d = 20$ (left) and $d = 200$ (right)