Exploration and Design of Modern Methods for Safe RL in an Autonomous Driving Environment

Abhiram Maddukuri **Rohit Chawla**

Abstract

Our work leverages the previous success of deep reinforcement learning (deep RL) for safe RL. Specifically, we focus on various deep RL methods which learn directly from pixels. We also investigate a pipeline for "pretraining" policies using offline RL and finetuning them in an online setting as a means to achieving safe RL during any environmental interaction. To test and analyze our investigations with safe RL, we deploy our policies on an autonomous driving simulator, where safety is especially crucial. We find that the online deep learning methods generalize best, both in terms of worst-case and average performance. We find mixed results for the pretraining pipeline, but acknowledge future work that can be done to improve upon this potentially powerful avenue for achieving fully safe RL policies.

1 Introduction

Recent success in reinforcement learning for optimal control has led to the widespread use of reinforcement learning in various contexts from video game AI to robotic manipulation. One such interesting and challenging application of reinforcement learning is in the context of self-driving cars. Learning a policy for a self-driving car involves optimizing for two objectives. Firstly, like other problems, the agent must learn a performant policy that maximizes expected reward. Secondly, and specific to the self-driving car context, the agent must learn a policy that is "safe", meaning the agent must minimize worst-case performance since any potential failure could be life-threatening when dealing with self-driving cars.

In this direction, we consider and devise various recent approaches which we believe to have strong average-case and worst-case performance in the highway-env roundabout-v0 Gym [3] environment and make our code available on GitHub.[1] Firstly, in all of our approaches, we look to replicate recent success in deep reinforcement learning from raw pixels [5]. Next, we consider deep reinforcement learning methods that are optimized for situations where worst-case performance can be considered. Specifically, we consider Deep Q-Networks (DQN) and Trust-Region Policy Optimization (TRPO) [8] as strong candidates for our purpose. We then devise a pipeline which attempts to mirror recent success in pretraining by using offline reinforcement learning [2] as a pretraining approach to achieve our what we hypothesize to be safest deployment of a policy.

We then compare our results to [4], which shows strong guarantees for worst-case policy performance. Overall, we find that the best policies come from purely online deep reinforcement learning policies. We then give future direction to optimize our offline pipeline to maximize performance in the safety context provided by using offline RL.

[1] `https://github.com/Abhiram824/CS378-Final-Project`

$$\hat{v}^r(\pi) \overset{\text{def}}{=} \sum_{t=0}^{H} \gamma^t \min_{s \in \Box S(t, s_0, \pi)} r(s, \pi(s))$$

2 Related Work

2.1 Approximate Robust Control of Uncertain Dynamical Systems

The first paper we looked at was [4]. Here, the authors tackle the problem in reinforcement learning where model-free algorithms are sample-inefficient and model-based algorithms can degrade easily under an erroneous dynamics model. The authors propose robust model-based algorithms with a guaranteed optimal lower bound on performance. Specifically, they seek to find a policy that $\max_\pi \min_T v_\pi^T$, where T is a given dynamics model.

The authors consider 2 algorithms for two cases. The first case is considered over discrete and finite dynamics and actions. Here, the authors consider the group of trees created by following the optimal policy across all dynamics models. They optimize over the lower and upper bound of the optimal value at each state and dynamics model pair node in the tree using the following equations.

$$b_i^r(n) \overset{\text{def}}{=} \begin{cases} u_i^r(n) + \frac{\gamma^d}{1-\gamma} & \text{if } i \in \mathcal{L}_n \,; \\ \max_{a \in \mathcal{A}} b_{ia}^r(n) & \text{if } i \in \mathcal{T}_n \setminus \mathcal{L}_n \end{cases}$$

$$u_i^r(n) \overset{\text{def}}{=} \begin{cases} \min_{m \in [1,M]} \sum_{t=0}^{d-1} \gamma^t r_t & \text{if } i \in \mathcal{L}_n \,; \\ \max_{a \in \mathcal{A}} u_{ia}^r(n) & \text{if } i \in \mathcal{T}_n \setminus \mathcal{L}_n \end{cases}$$

Where u is an upper bound on the true value, L is the leaf state induced by the queried trajectory, T is the tree produced by the trajectories, A is the action space, and i is the action sequence. Essentially they take the lower value bound by considering the values produced by all trees and taking the worst possible best value over all trees. This gives the following algorithm

```
Algorithm 1: Deterministic Robust Optimistic Planning
1 Initialize $\mathcal{T}$ to a root and expand it. Set $n = 1$.
2 while Numerical resource available do
3     Compute the robust u-values $u_i^r(n)$ and robust b-values $b_i^r(n)$.
4     Expand $\mathrm{argmax}_{i \in \mathcal{L}_n} b_i^r(n)$.
5     n = n + 1
6 return $\mathrm{argmax}_{a \in \mathcal{A}} u_a^r(n)$
```

The next algorithm the authors consider is for bounded continuous set of all dynamics models. They first consider the reachability set, which is the set of all states reachable given a policy and starting state along a dynamics model. Since this is a complex shaped set, the authors look to approximate it via an interval hull. They use this appromixated interval to optimize surrogate object $\hat{v}^r$ given by the following equation

Where s is a state sampled from the approximated reachability set. It can shown that the surrogate value is a lower bound of the true value. This gives us the following algorithm.

```
Algorithm 2: Interval-based Robust Control
1 Algorithm robust_control($s_0$)
2     Initialize a set $\Pi$ of policies
3     while resources available do
4         evaluate() each policy $\pi \in \Pi$ at current state $s_0$
5         Update $\Pi$ by policy search
6     end
7     return $\mathrm{argmax}_{\pi \in \Pi} \hat{v}^r(\pi)$
1 Procedure evaluate($\pi$, $s_0$)
2     Compute the state interval $\Box S(t, s_0, \pi)$ on a horizon $t \in [0, H]$
3     Minimize $r$ over the intervals $\Box S(t, s_0, \pi)$ for all $t \in [0, H]$
4     return $\hat{v}^r(\pi)$
```

2.2 Constrained Policy Optimization

The next paper we looked at was [1]. This approach involves detailed specification of the constraints for the system. The authors state that tweaking the reward function repeatedly is less productive and efficient compared to designing good constraints for the policies. While this paper doesn't focus on the application to the self-driving vehicle situations, one can certainly conclude that the designing of safety constraints could be helpful in these aforementioned environments.

The authors propose their solution of Constrained Policy Optimization (CPO) and demonstrate its effectiveness through simulated robot locomotion tasks, where safety constraints are declared. The key idea of this approach lies in its guarantees for near-constraint satisfaction at each iteration. In an autonomous vehicle application, this is vital to consider because any collision or break of constraint could be catastrophic for people involves and for the future of self-driving vehicles.

As is done in traditional reinforcement learning, the goal remains to select a policy that maximizes a total performance measure. In this case, we maximize J(π), or the infinite horizon discounted total return. This can be seen with this equation:

$$J(\pi) \doteq \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t, s_{t+1})\right].$$

Next, an equation expressing the difference between two policies' performances is given:

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\underset{\substack{s\sim d^{\pi'} \\ a\sim\pi'}}{\mathrm{E}}\left[A^{\pi}(s,a)\right]$$

The authors' primary guarantees are shown through a bound relating the expected rewards of two policies to an average divergence between them. The problem, along with its constraints, are formulated through the use of CMDPs (Constrained Markov Decision Processes). These are simply MDPs, but with a set of auxiliary cost functions, C1...Cm, and respective limits, d1...dm.

In general, the idea of constraining for safety purposes is sound. However, this approach lends itself better to less complex and dynamic environments since optimal safe RL policies usually aren't as simple. Additionally, using constraints may provide success for one environment but may struggle in a different one, as the process of defining constraints isn't very generalizable.

3 Methodology

We cast the self-driving car problem as an MDP with a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of states. For our environment, we consider states as a 4-stack of 128x64 grayscale images representing the full image of the roundabout.
- $\mathcal{A}$ is the set of actions that the agent can take. Our action space is discrete and consists of the following actions: LANE_LEFT, LANE_RIGHT, IDLE, FASTER and SLOWER.
- $\mathcal{P}$ is the state transition probability function: $\mathcal{P}(s'|s,a)$ represents the probability of transitioning to state s' when taking action a in state s.
- $\mathcal{R}$ is the reward function: $\mathcal{R}(s,a,s')$ represents the immediate reward received when transitioning from state s to state s' after taking action a. In this environment, the reward function consists of a velocity term and a collision term:

$$R(s,a) = a * \left(\frac{v - v_{\min}}{v_{\max} - v_{\min}}\right) - b * collision \tag{1}$$

  where v, $v_{\min}$, $v_{\max}$ are the current, minimum and maximum speed of the vehicle, respectively, and a, b are two coefficients. We will set b to be relatively high in order to emphasize safety.
- γ is the discount factor, a scalar value between 0 and 1, which determines the agent's preference for immediate rewards over future rewards.

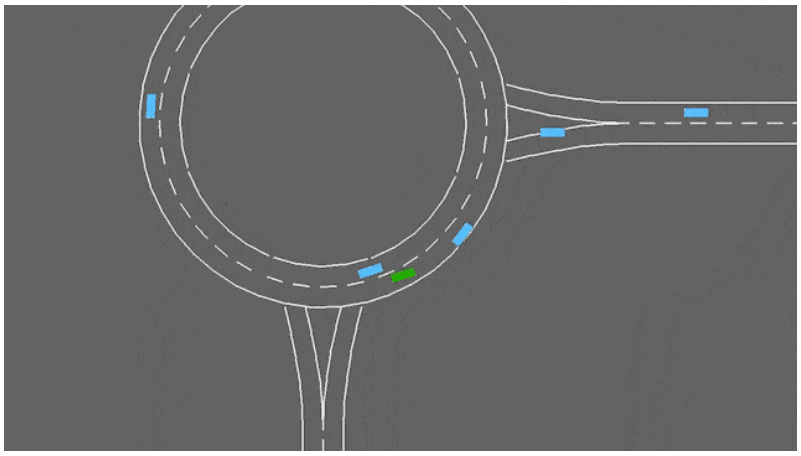


Figure 1: The roundabout-v0 environment

The goal in an MDP is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions, such that the cumulative expected reward is maximized:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathcal{R}(s_t, a_t, s_{t+1})\right]$$

where s_t represents the state at time step t, a_t is the action taken at time step t, and π^* denotes the optimal policy.

We hope to find such an optimal policy which also has strong worst-case performance. For all our approaches, we consider algorithms which use deep learning directly from raw pixels. We do so because neural networks have strong generalization properties; thus, a deep-learning policy trained with enough samples, can generalize to any relevant scenarios and will be very unlikely to fail, giving strong worst-case performance. From here, we consider specific approaches which we hypothesize will lend well for the safe reinforcement learning context.

3.1 Deterministic Robust Optimistic Planning

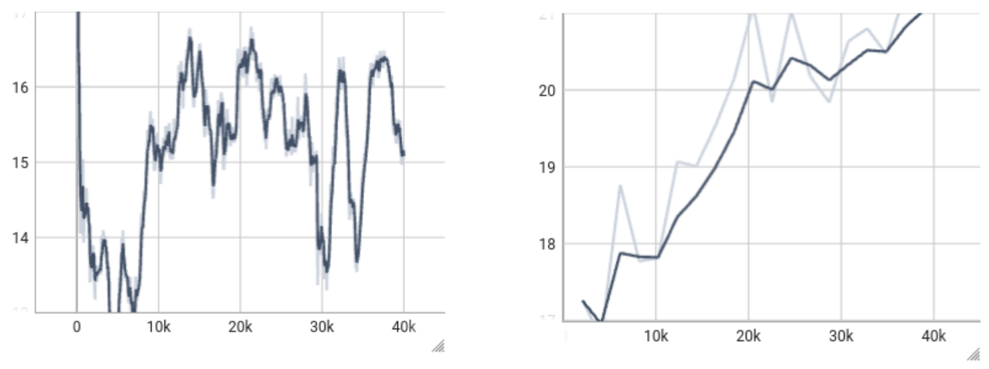
This is the baseline we compare to and is a model-based algorithm considered over discrete and finite dynamics and actions. The algorithm looks at the group of trees created by following the optimal policy across all dynamics models. They optimize over an estimate of the lower and upper bound of the optimal value at each state and dynamics model pair node in the tree. By providing tight bounds for the true value and optimizing the lower bound, the authors provide a robust algorithm for safe RL.

3.2 DQN

The DQN algorithm aims to learn the optimal action-value function in the same way as value iteration $Q^*(s,a) = E_s\left[r + \gamma \max_{a'} Q^*(s', a')\right]$. In the case of value iteration, an action value is learned separately for each state, but this is intractable when dealing with images. Thus, the authors combat this by using the generalization of CNN's as a value function approximator. When training the model we get the following loss function:

$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$, where $y_i = \mathbb{E}_{s_0\sim E}\left[r + \gamma \max_{a_0} Q(s_0, a_0; \theta_{i-1}) \mid s, a\right]$.

We specifically consider this algorithm because of it's strengths as an off-policy algorithm - when collecting data, there is an ϵ probability of taking a random action as opposed to following the policy; this is done to get a greater coverage of the states for training, allowing the network to generalize to a diverse set of states. By forcing the network to learn various states, the policy will be robust to new scenarios and will be less likely to crash or fail in unanticipated scenes.



(a) DQN Agent (b) TRPO Agent

Figure 2: Average Reward vs Timestep for Online Agents

3.3 TRPO

TRPO is a policy-gradient algorithm that restricts gradient updates to be within a trust region. They do so by optimizing a surrogate objective and a KL-Divergence term. We focus on TRPO because of its strong theoretical basis. By optimizing the surrogate objective, the authors show that each gradient update will result in a monotonic improvement of policies. With the strong theoretical basis of monotonic improvement, we can be confident that policy learning will be stable and we would not get a large unexpected divergence in certain scenarios, making it a strong candidate as a safe RL algorithm.

3.4 Offline Pre-training + Online Fine-tuning

We take inspiration from the strong performance gained by the pretraining paradigms used in works like GPT [6]. Specifically, we collect a dataset of expert demonstrations and train an offline policy from those demonstrations. We then fine-tune this pretrained policy in an online setting. We specifically look at CQL (Conservative Q-Learning) as our offline agent and transfer the learned value function to DQN for online fine-tuning. We consider this to be a potentially powerful approach to safe RL for two reasons.

1. **Deployability**: One glaring issue with online algorithms is the need for exploration; this can be dangerous in environments such as self-driving cars where exploration of various actions can be dangerous or even fatal. Offline reinforcement learning can leverage preexisting data and train a strong policy without dangerous or expensive exploration; afterwards, we can deploy and fine-tune this policy online without worrying about dangerous actions. While we follow experiments in a simulator, we still consider using offline reinforcement learning since we are looking at driving policies.

2. **CQL Safe Guarantees** It has been shown that CQL as an offline learning methodology achieves safe and high-confidence improvement over a behavior policy. This results from CQL's basis of producing a lower bound on the policy's value so that overestimation in offline learning doesn't occur. Thus, when learning offline from a comprehensive set of trajectories, we can obtain a provably strong policy without needing any live-environment interaction. Moreover, we can further improve this policy safely in an online setting.

4 Experiments and Analysis

We consider 2 main experiments/evaluations. (1) rollout evaluation during training, (2) rollout evaluation after training. For the TRPO and DQN implementations, we use [7], and we use [9] for CQL.

4.1 Experiment 1

For this experiment, we look at the rollout rewards generated during training of the DQN and TRPO agents. We train both algorithms for 40,000 timesteps. Results are shown in figure 2. We examine this because of our interest in safe RL. Specifically, the graph on rollout rewards reveals the behavior and stability of the online algorithms, which we hypothesized would be robust. Examining the graphs, we find that the DQN's episode return for each checkpoint fluctuates heavily from 13 to 17 and may not be stable where we stopped training. Despite the strong returns, we suspect the DQN may not be the perfect candidate for safe deployment. TRPO indicates a different story; staying true to the theory backing the algorithm, we see an almost monotonic increase in policy peformance, from 12 to more than 20 average reward, indicating stable policy learning. For deployment, we can fairly assume we will get stable behavior, even in the worst-case.

4.2 Experiment 2

For our next experiment, we conduct 50 rollouts for the final version of each trained agent; results are shown in table 1. For TRPO and DQN, we used the online trained models from experiment 1. For Interval-based Robust Control we reuse the results documented in [4]. For CQL, we collect an offline dataset of 1000 episodes of interaction data from the trained DQN. We then train the CQL agent for 10,000 timesteps. Lastly, we copy the learned Q-function from the CQL agent and finetune it for 40,000 timesteps for the CQL + DQN Finetuned agent.

Based on the stability and monotonicity of the TRPO graph, we get the expected result that the TRPO results outperform all other algorithms. While, the DQN performs better on average to the baseline, it has poor worst-case performance; we attribute this to the instability in training the DQN and hypothesize that more training iterations will make it more consistent. Surprisingly, the pretrained CQL + DQN agent is worse than the DQN agent. This could be due to a various set of factors such as the training dataset for CQL being not diverse enough or simply not enough training iterations for CQL to find a strong value function. Another possible explanation is that the value function transfer was not ideal and a policy function transfer would lead to more success; all of these investigations are potential areas for future works. Lastly, despite the pretraining + finetuning pipeline not working up to expectations, we see that a purely pretrained model also gets decent results, which is promising considering it had no online interaction.

Table 1: Rollout Reward Statistics

Agent	Worst-case return	Mean return
Interval-based Robust Control	8.99	10.78
CQL + DQN Finetuning	4.67	11.44
DQN	5.31	15.78
TRPO	9.25	17.69
CQL-Only	3.75	7.55

5 Conclusion

In this work we do the following: first, we introduce safe RL in the context of self-driving cars and propose justifications for various popular algorithms which could be performant We also propose a pipeline of pretraining + finetuning policies with the idea of safe RL in mind. We then evaluate each of these methods, analyze the results, and provide justifications and avenues for future work on further investigating the results.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. 2017.

[2] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[3] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[4] Edouard Leurent, Yann Blanco, Denis Efimov, and Odalric-Ambrym Maillard. Approximate robust control of uncertain dynamical systems. *arXiv preprint arXiv:1903.00220*, 2019.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training.

[7] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[8] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[9] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022.