Navigation on OpenAIGym with Deep Reinforcement Learning

YANG LONGWEI
Department of Computer Science
Hanyang University

ZHOU ZHUGUI
Department of Computer Science
Hanyang University

ABSTRACT

This project uses the classic environment CarRacing-v0 from OpenAI, a 2D autonomous vehicle environment, alongside custom based environment modifications, a Proximal Policy Optimization(PPO) algorithm based model is created to solve the autonomous car problems. Through exploration, the results show that a significant result is obtained by using PPO with other mechanisms which can almost complete all the track (700 score) and trained in only 40 minutes. We analyzed and concluded that compared to other algorithms. Thereby, the model is able to regularly surpass a reward count of 600.

INTRODUCTION

BACKGROUND

Simulating realistic virtual characters is an important research topic in many fields of study, such as film, computer games, urban engineering, and behavioral science. In a game scene, each character must reach a target location or perform a specific action without colliding with other characters or obstacles. Current character control algorithms mainly use optimization or search algorithms. The advantage of this algorithm is that simply by designing the cost function, it is possible to create a motion planning algorithm that performs consistently and robustly in a variety of scenarios. However, in order to use the search algorithm in applications where real-time is important, it is important to reduce the computation time. More recently, optimal controllers using deep reinforcement learning are being investigated to create a real-time controller for realistically moving characters in complex environments. So, in this project, we present a deep reinforcement learning approach for navigation in car racing simulation, which has a better performance compared with existing learning algorithms or search algorithms.

Gym

Gym, launched by OpenAI, is a standard API for reinforcement learning, and a diverse collection of reference environments. It is one of the most widely used reinforcement learning experimental environments, with hundreds of built-in experimental environments, such as the movement of some simple geometric bodies, some simple games represented by text, or experimental environments such as grasping and control of robotic arms.

PPO

The Reinforcement learning algorithm used in this study is Proximal Policy Optimization (PPO)—an architecture that improves the stability of agent training by avoiding too large policy updates. To do this, a ratio is used to indicate the difference between the current policy and the old policy, and the ratio is clipped from a certain range

$$[1-\epsilon, 1+\epsilon].$$

Doing so ensures that policy updates are not too large, and the training is more stable. Another advantage of PPO is that it can accommodate asynchrony, which makes the possibility of multiple parallel environments improve convergence by reducing the correlation between samples, which is a problem with DRL.

In PPO, the idea is to constrain our policy updates with a new objective function called the Clipped surrogate objective function that will use clipping to limit policy changes to a small range, designed to avoid destructively large weight updates.

PPO's Clipped surrogate objective function:

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

The Ratio Function:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

It is the probability of acting *at* at state *st* in the current policy divided by the previous one. rt (θ) denotes the probability ratio between the current and old policy: If *rt*(*θ*) > 1, the action *at* at state *st* is more likely in the current policy than the old policy. If *rt*(*θ*) is between 0 and 1, the action is less likely for the current policy than for the old one.

The unclipped part of the Clipped Surrogate Objective function:

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

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

This ratio can replace the log probability used in the policy objective function. This gives the left part of the new objective function to multiply the ratio by the advantage. However, without a constraint, this would lead to a significant policy gradient step and, therefore, an excessive policy update.

The clipped Part of the Clipped Surrogate Objective function:

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

So, constrain this objective function by penalizing changes that lead to a ratio away from 1. By clipping the ratio, ensure that do not have a too large policy update because the current policy can't be too different from the older one.

RATIONALE AND JUSTIFICATION

The objective of this study was to explore the practical application of deep reinforcement learning. Since the 1990s, developments in the field of autonomous vehicles, animations, games, have been widely publicized and recognized. However, these advancements have not really met commercial and social needs. Through the progression of various sectors of applied artificial intelligence, such as machine learning, computer vision, reinforcement learning, and neural networks, autonomous vehicles can be produced to improve human society. This study uses these methods and will help to discuss developments in this field. Using positive reinforcement to incentivize a vehicle to stay on a desired path is similar to what is being developed for autonomous cars.

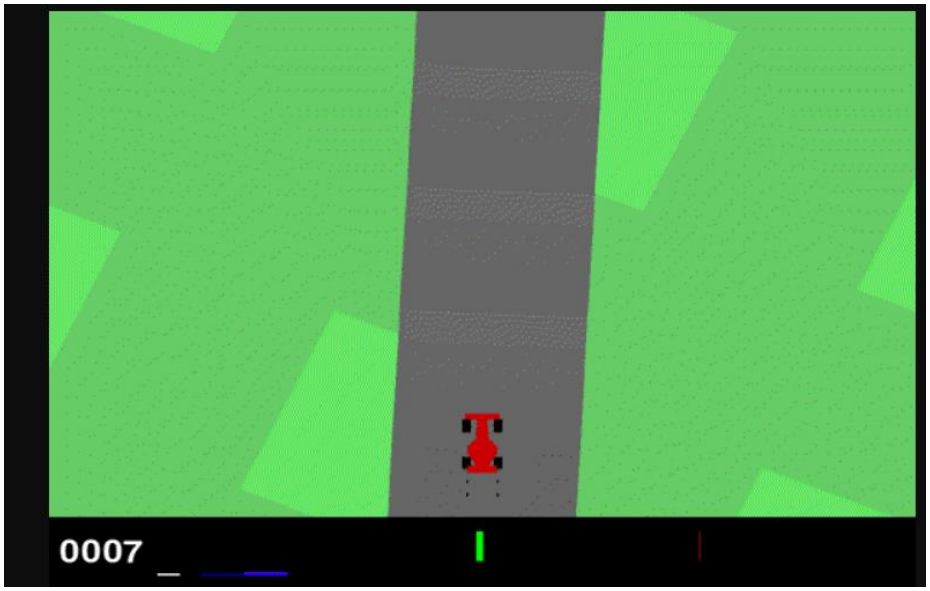
Likewise, this research paper also helps further the exploration of machine learning developments for diverse uses and other fields. For instance, this study's methods and outcomes can be applied in other fields such as natural language processing [18] and reinforcement learning for computer games.

ENVIRONMENTS

Classic

Car navigation can be simulated in various environments. The classic and straightforward one is CarRacing-v0 environment in gym. The default observation space is a RGB 96x96 pixel frame of the game, which includes all the control bars at the bottom of the screen. The car has to take actions (turn, accelerate, and brake certain magnitudes) to follow a grey-marked path and complete the randomly generated track, all while doing so as quickly as possible to get higher scores. The reward is equal to +1000/N for each track tile visited, where N is

represented by the total number of tiles in the entire track and -0.1 for each frame. To be considered a successful episode, the agent must always receive a reward of 900, which means the agent must be on the track for a maximum of 1000 frames. Additionally, there is a barrier outside the track that, if crossed, will result in a -100 penalty and end the episode immediately. Outside, the track consists of grass, which does not give rewards , but the friction of the environment makes it difficult for the vehicle to get back on the track.



[1] (Classic environment)

Custom

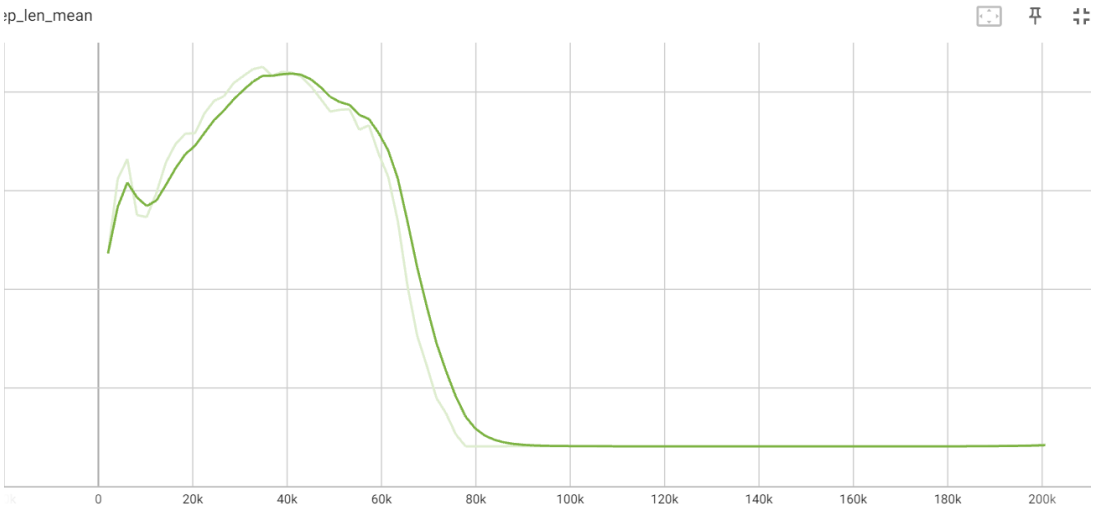
Timeout

Timeout refers to the case when the car goes out of the track and stays outside longer than Tmax(5s in our case). This avoids wasting computing time in scenarios where the car is already in a not desired position, Tmax should be big enough to allow the car to recover when it goes out, but short enough to terminate the episode when reward is no longer to increase. Also this function leads to another issue which is that our agent drops into timeout states and

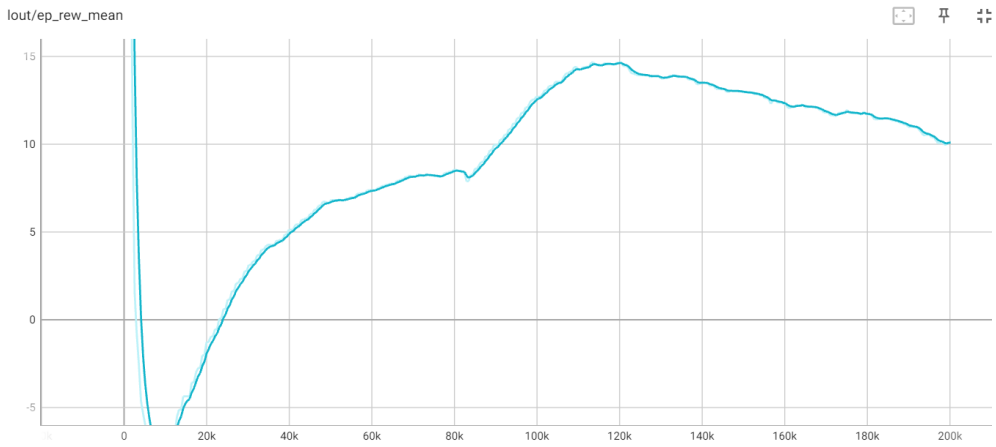
stops searching for better rewards[1] and it can be solved by normalizing rewards which will be discussed next.

[2] (Episodes terminated by timeout eventually and no longer optimize)

Normalize Reward



Due to the timeout penalty, training results did not reach our expected performance, normalizing reward techniques improves the training process very obviously. Our wrapped function normalizes the rewards with the running mean rewards and their variance. This aims to avoid the exploiting gradient problem as well as taking too big steps which can result in non-optimal step-sizes. Basically, when the agent gets relatively low rewards, it stabilizes the training process and does not get stuck in low-reward episode. In contrast, when the reward is already satisfied our agent still searches for a better track.



[3] (Normalized_reward in training)

Grayscale and Resize observation

Color of pixels has no importance in this research, therefore reducing color tunnel helps deep neural networks have an easier time learning with binary inputs. Also resize observation space from 96x96 pixels to 64 pixels suits better for neural network which is

[4] (Custom environments)

STRUCTURES

Model

The parameters of PPO are mainly referenced from RL Baselines3 Zoo which is a training framework for stable Baselines3 and locates the best optimized parameters that we can utilize. Another mechanism in our experiment is applying the Learning Rate Schedule for learning rate[5]. which have the benefit of making large changes at the beginning of the training procedure when larger learning rate values are used and decreasing the learning rate so that a smaller rate and, therefore, smaller training updates are made to weights later in the training procedure(1). In our experiment point of view, this stabilized changes of weights at the beginning episodes.

```
def linear_schedule(initial_value: float) -> Callable[[float], float]:
    """
    Linear learning rate schedule.

    :param initial_value: Initial learning rate.
    :return: schedule that computes
      current learning rate depending on remaining progress
    """
    def func(progress_remaining: float) -> float:
        """
        Progress will decrease from 1 (beginning) to 0.

        :param progress_remaining:
        :return: current learning rate
        """
        return progress_remaining * initial_value

    return func
```

[5] (Linear_schedule function)

Eval_callback

Callback function during training process is simply run the current model in evaluate environment and stores the better weights in every 5000epsiode

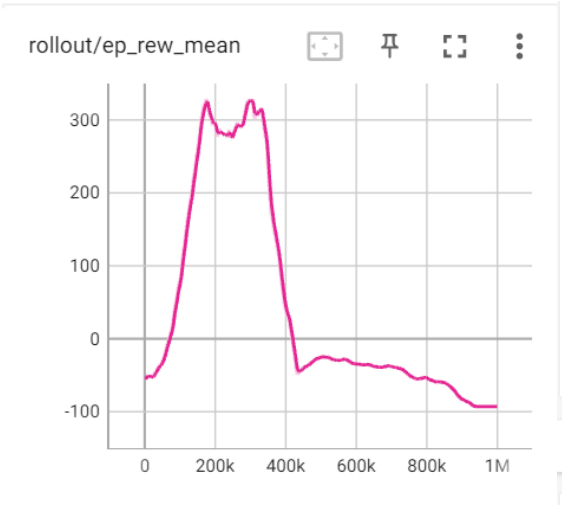
Stablebaselines3

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch. An open-source framework implementing seven commonly used model-free deep RL algorithms. Adhere to software engineering best practices to achieve high quality implementations that match prior results. Each algorithm has been benchmarked on common environments and compared to prior implementations.

EXPERIMENTS

Process

At the beginning only using default PPO parameters and classic-no-change environment did not work well. Even after 1M episode training(5 hours) the performance is getting worse[6]. Then tuned parameters, custom environment like adding time_out function and continuous learning applied and got an obvious improvement in those experiments. After 30mins training the best average reward is around 376 which is even better than previous 1M training results[7]. A funny phenomenon in this experiment is braking. Boosting rapidly inside the track and stopped instantly in case of out of the track and terminated due to timeout.

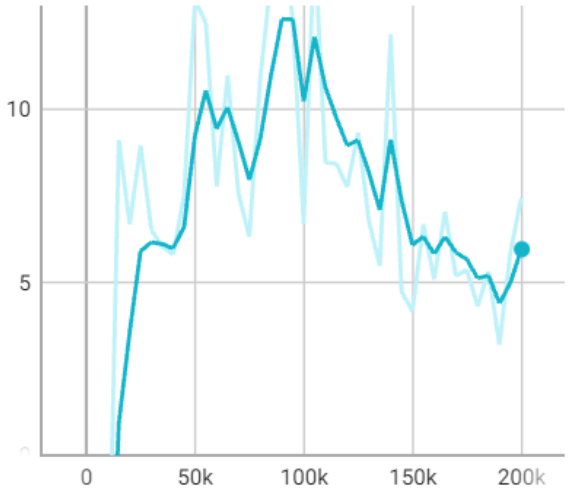


[6]

[7]

Results

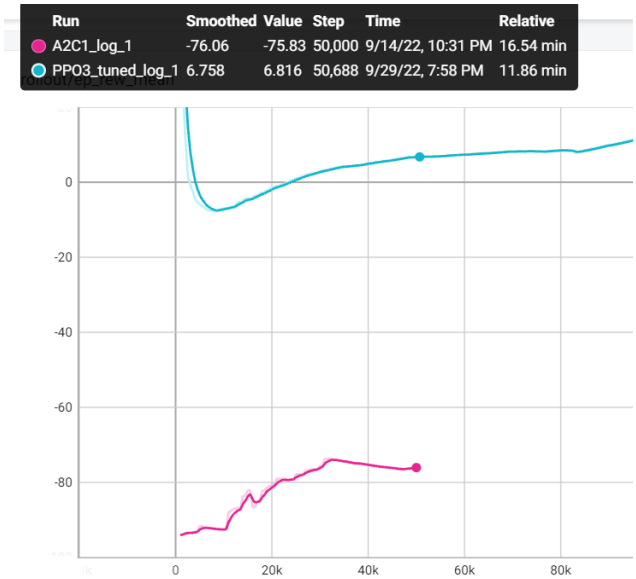
After all, test experiments and try on a different custom environment. Finally a significant results came out by using all mechanisms which can almost complete all track(700 score) and only trained in 40 mins[8].



[8] (The reward is normalized)

Comparison

Comparing other reinforcement algorithms and finding out the best model is also part of this project and PPO is obviously the optimal way to solve this simulated autonomous vehicle[9]. And the other experiments(2) with different parameters and custom environment only got 600 scores with 1M training[10].



[9] compare with A2C

[10]

CONCLUSION

In this report, we use PPO to solve the car racing problem in gym. We have some findings after acquiring the results and do some analysis. To make training faster and make the score better we experiment a lot in different ways of customizing environments and the model itself. It turns out that compared with other reinforcement algorithms like A2C, DQN PPO is clearly the optimal one to solve this dynamic simulated autonomous vehicle. And There is various methods to handle stabilization in the training process for big changes in weights we tested. Normalization in reward plays a significant part in this experiment.

REFERENCES

(1) Jason Brownlee. (2022) Using Learning Rate Schedules for Deep Learning Models in Python with Keras:

https://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/

(2) Jonghyun Ho. (2022) Learning CarRacing environment with stable-baselines3:

https://jonghyunho.github.io/reinforcement/learning/learning-carracing-env-with-stable-baselines3.html

(3) Mike.W. Solving Car Racing with Proximal Policy Optimisation:

https://notanymike.github.io/Solving-CarRacing/

(4) Ali Fakhry. (2020) Applying a Deep Q Network for OpenAI's Car Racing Game:

https://towardsdatascience.com/applying-a-deep-q-network-for-openais-car-racing-game-a642daf58fc9

(5) Thomas Simonini. (2022) Proximal Policy Optimization (PPO):

https://huggingface.co/blog/deep-rl-ppo

(6) Stable-baselines3: https://github.com/DLR-RM/stable-baselines3
(7) Yash Maniyar, Nidhi Manoj. Racetrack Navigation on OpenAIGym with Deep Reinforcement Learning.
(8) Changmao Li. Challenging On Car Racing Problem from OpenAI gym, 2019.