Solving CarRacing with Proximal Policy Optimisation

By Arnajak Tungchoksongchai st122458
Nutapol Thungpao st122148

Contents

- PPO algorithm
- Our CarRacing
- Result
- Run the best model !!
- Q&A

Proximal Policy Optimisation

PPO : Proximal Policy Optimisation

“Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune.”

Research in policy gradient methods has been prevalent in recent years, with algorithms such as TRPO, GAE, and A2C/A3C showing state-of-the-art performance over traditional methods such as Q-learning. One of the core algorithms in this policy gradient/actor-critic field is the Proximal Policy Optimization Algorithm implemented by OpenAI.

PPO : Motives

$$\underbrace{L^{PG}(\theta)}_{\text{Policy Loss}} = \underbrace{\overset{\text{Expected}}{\hat{\mathbb{E}}_t}}\Big[\underbrace{\log \pi_\theta(a_t \mid s_t)}_{\text{log(probabilities) from the output of the policy network}}\underbrace{\hat{A}_t}_{\text{Estimate of the relative value of selected action}}\Big].$$

- The policy pi is our neural network that takes the state observation from an environment as input and suggests actions to take as an output.The advantage is an estimation
- Multiplying log probabilities of policy’s output and advantage function gives us a clever optimization function (positive - actions the agent took in the sample trajectory resulted in a better than average return)
- Problem : destructively large policy updates which occur by often update the parameters so far outside of the range

PPO : Trust Region Policy Optimization

- Trust Region Policy can prevent destructive policy updates
- implemented an algorithm to limit the policy gradient step so it does not move too much away from the original policy

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1$$

Schulman et al., 2017

- rt probability ratio between the action under the current policy and the action under the previous policy

PPO : Trust Region Policy Optimization

the TRPO's objective in a more readable format:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \longrightarrow \begin{array}{ll} \underset{\theta}{\text{maximize}} & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{array}$$

(Schulman et al., 2017)

In this TRPO method, we notice that it is actually quite similar to the vanilla policy gradient method on the left. In fact, the only difference here is that the log operator is replaced with the probability of the action of current policy divided by the probability of the action under the previous policy. Optimizing this objective function is identical otherwise.

PPO : Clipped Surrogate Objective

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

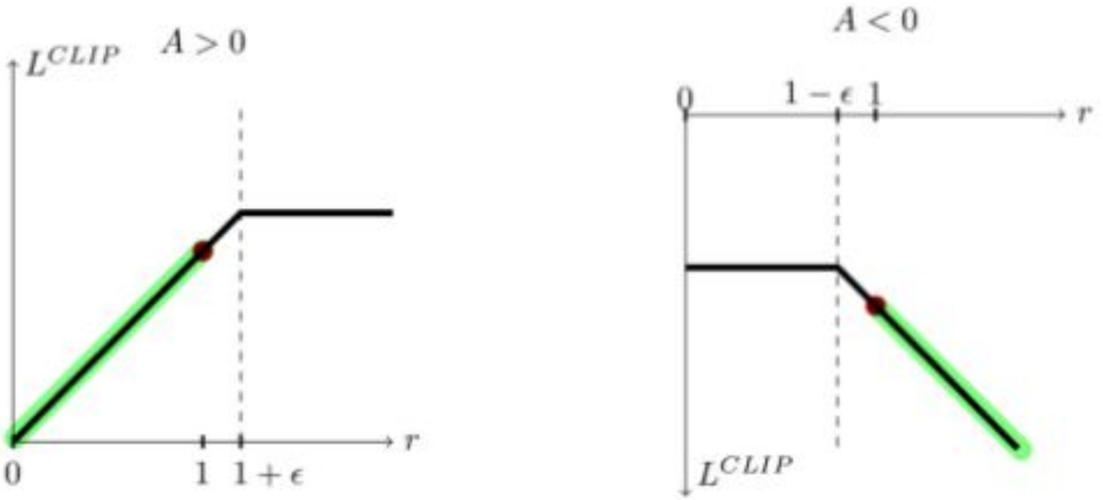


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

PPO : Clipped Surrogate Objective

As clever as this approach is, the clipping operation also helps us out for ‘undoing’ policy’s mistakes

our clipping operation will kindly tell the gradient to walk in the other direction in proportional to the amount we messed up. This is the only part where the first term inside min() is lower than the second term, acting as a backup plan. And the most beautiful part is that PPO does all of this without having to compute additional KL constraints.

All of these ideas can be summarized in the final loss function by summing this clipped PPO objective and two additional terms:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[\underbrace{L_t^{CLIP}(\theta)}_{\text{clipped PG objective}} - c_1 \underbrace{L_t^{VF}(\theta)}_{\text{MSE of value function}} + c_2 \underbrace{S[\pi_\theta](s_t)}_{\text{entropy term}}\right],$$

(Schulman et al., 2017)

PPO : Multiple Epochs for Policy Updating

Finally, let's take a look at the algorithm altogether and its beauty of parallel actors:

Algorithm 1 PPO, Actor-Critic Style

for iteration=1, 2, . . . **do**
    for actor=1, 2, . . . , N **do**
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for

interacting w/ the environment & generating sequences for calculating advantage function

Run SGD on $L^{CLIP}(\theta)$ every so often

PPO Algorithm (Schulman et al., 2017)

PPO : To sum up

- PPO is an on-policy algorithm.
- PPO can be used for environments with either discrete or continuous action spaces.
- The Spinning Up implementation of PPO supports parallelization with MPI.
- There are two primary variants of PPO: PPO-Penalty and PPO-Clip.
    - **PPO-Penalty** approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint
    - **PPO-Clip** doesn’t have a KL-divergence term in the objective and doesn’t have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

Our CarRacing

CarRacing : CarRacing-v1

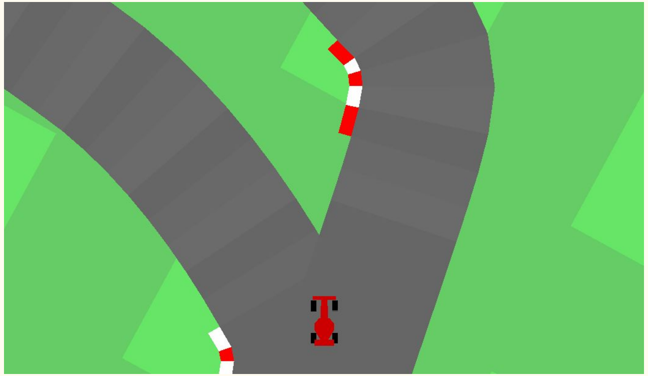
Mostly is like Caracing-v0 but has improvements on the complexity which are focus towards making Car-Racing env solvable, it is also intended to make the env complex enough to make it ideal to try new more complex tasks and RL problems.

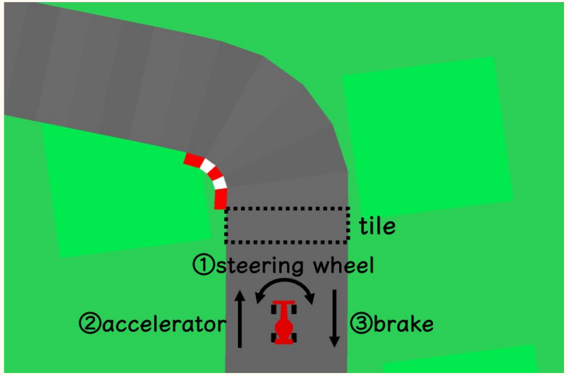
- **New Features in action_space**
- **New Features in observation_space**
- **New Features regarding the Map**
- **New Features regarding Reward Function**
- **New Features regarding Agent (i.e. car)**
- **Add some useful function ex. Position, set speed, etc.**

This develop by NotAnyMike : https://github.com/NotAnyMike/gym

CarRacing : Project Instruction

- Add random 50 obstacles
- Modifying the out of lane farer than 30% will stop the episode.
- Modify the continuous running to be stopped after 5400 steps.
- Calculate percentage of car running in each lap
- Count the number of frames which go out of the lane
- Set 30 frame per second.





CarRacing : Action-space

- **5 actions (Accelerate, Brake, Left, Right, Do-Nothing)**
- **Continuous to discrete action**

Discrete action		Continous action
`Turn_left`	→	`[ −1.0, 0.0, 0.0 ]`
`Turn_right`	→	`[ +1.0, 0.0, 0.0 ]`
`Brake`	→	`[ 0.0, 0.0, 0.8 ]`
`Accelerate`	→	`[ 0.0, 1.0, 0.8 ]`
`Do-Nothing`	→	`[ 0.0, 0.0, 0.0 ]`

CarRacing : Observation space

- The default observation space is an RGB 96x96 pixel game frame
- This used a grayscale frame to reduce computation
- This used stacked four consecutive frames together
    - Stack of 4 frames is simply to catch information like velocity of objects.
    - The Max wrapper is because in Atari, sometimes the screen gets buggy and has stray pixels getting turned on/off in some frames which messed up the training. For e.g., in Pong, the DQN could get confused where the ball actually is if there are stray white pixels on the screen.

CarRacing : Rewards

- Reward is -0.1 every frame and +1000/N for every track tile visited, where N is the total number of tiles in track.
- For example, if you have finished in 732 frames.
  your reward is 1000 - 0.1*732 = 926.8 points.
- Hit obstacle -10
- -100 when game done ex. Out lane more than 30%, more than 5400 step

CarRacing : Rewards

- Hit obstacle -10
- -100 when game done ex. Out lane more than 30%, more than 5400 step

```
re_p,sum_obc_touch = env.check_obstacles_touched()
reward += re_p
```

```
x, y = env.car.hull.position
if not done and abs(x) > PLAYFIELD or abs(y) > PLAYFIELD:
    done = True
    reward -= HARD_NEG_REWARD
```

CarRacing : Project Instruction

- **Add random 50 obstacles**

```python
def _set_config(self,
        num_tracks=1,
        num_lanes=1,
        num_lanes_changes=0,
        num_obstacles=50,
```

- **Set 30 frame per second.**

```python
SCALE       = 6.0        # Track scale
TRACK_RAD   = 900/SCALE  # Track is heavily morphed circle with this radius
PLAYFIELD   = 2000/SCALE # Game over boundary
TRACK_30   =   92/SCALE
FPS         = 30
ZOOM        = 2.7        # Camera zoom, 0.25 to take screenshots, default 2.7
ZOOM_FOLLOW = True       # Set to False for fixed view (don't use zoom)
```

- **Modify the continuous running to be stopped after 5400 steps.**

```python
def check_timeout(self,reward,done):
    if self._steps_in_episode  >= 5400:
        # if too many seconds outside the track
        done = True
        if self.verbose > 0:
            print("done by time")
        reward -= HARD_NEG_REWARD
    return reward,done
```

CarRacing : Project Instruction

- **Calculate percentage of car running in each lap**

```
lap_complete_percent = env.tile_visited_count / len(env.track)
```

```
if env.tile_visited_count == len(env.track):
    lap_count +=1
    reward += 100
```

- **Count the number of frames which go out of the lane**

```
def check_outside(self,reward,done,count_out):
    right = self.info['count_right']
    left = self.info['count_left']
    #print('right  : {0} \n left : {1}'.format(right.sum(), left.sum()))
    if self._is_outside():
        # In case it is outside the track
   #    done = True
        count_out +=1
        reward -=1
    return reward,done,count_out
```

CarRacing : Rewards

- Train

```
from stable_baselines3.common.callbacks import CheckpointCallback
env = gym.make(environment_name)
env = DummyVecEnv([lambda: env])

log_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Logs')
model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=log_path)
ppo_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Saved_Models/PPO_car_best_Model')
checkpoint_callback = CheckpointCallback(save_freq=1000, save_path='/content/drive/MyDrive/RL_project/logs/',name_prefix='rl_model_new')

eval_env = model.get_env()

eval_callback = EvalCallback(eval_env=eval_env, best_model_save_path=ppo_path,
                             n_eval_episodes=5,
                             eval_freq=5000,verbose=1,
                             deterministic=True, render=False)

model.learn(total_timesteps=1000000,callback=eval_callback)
ppo_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Saved_Models/PPO_2m_Model_final')
model.save(ppo_path)
```

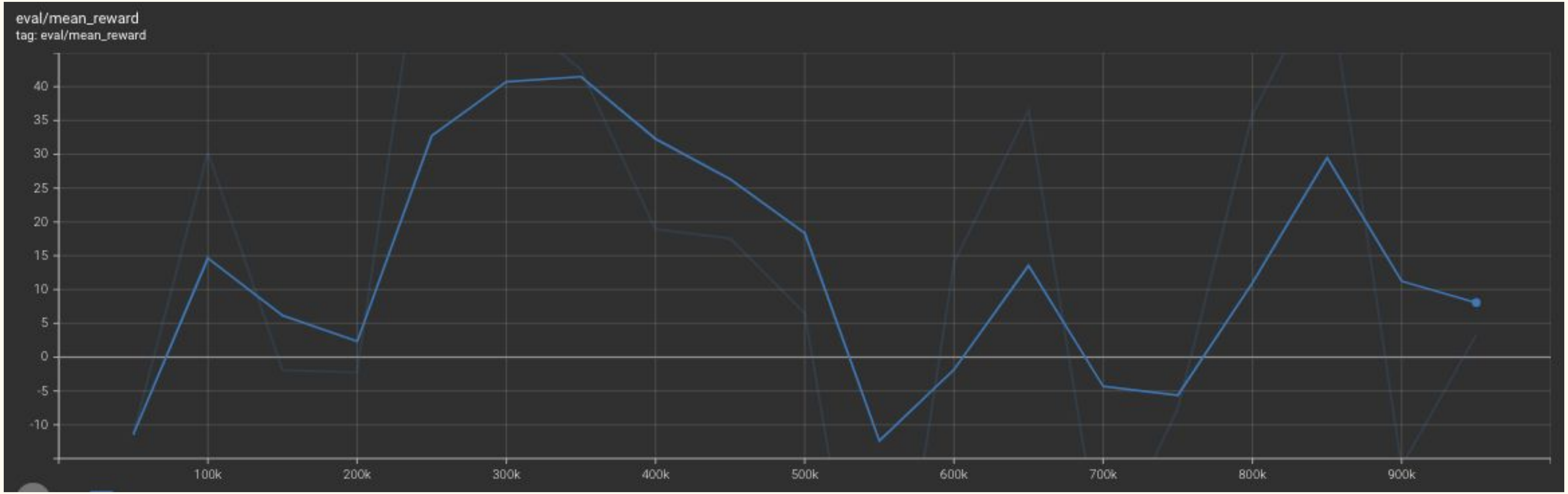
CarRacing : Rewards

- Train

```
from stable_baselines3.common.callbacks import CheckpointCallback
env = gym.make(environment_name)
env = DummyVecEnv([lambda: env])

log_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Logs')
ppo_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Saved_Models/PPO_car_best_Model/70สำเนาของ best_model.zip')
model = PPO.load(ppo_path, env=env, verbose=1, tensorboard_log=log_path)

ppo_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Saved_Models/PPO_car_best_Model')
checkpoint_callback = CheckpointCallback(save_freq=1000, save_path='/content/drive/MyDrive/RL_project/logs/',name_prefix='rl_model_new')

eval_env = model.get_env()
eval_callback = EvalCallback(eval_env=eval_env, best_model_save_path=ppo_path,
                             n_eval_episodes=5,
                             eval_freq=5000,verbose=1,
                             deterministic=True, render=False)

model.learn(total_timesteps=10000000,callback=eval_callback)
ppo_path = os.path.join('/content/drive/MyDrive/RL_project/Training/Saved_Models/PPO_2m_Model_final')
model.save(ppo_path)
```

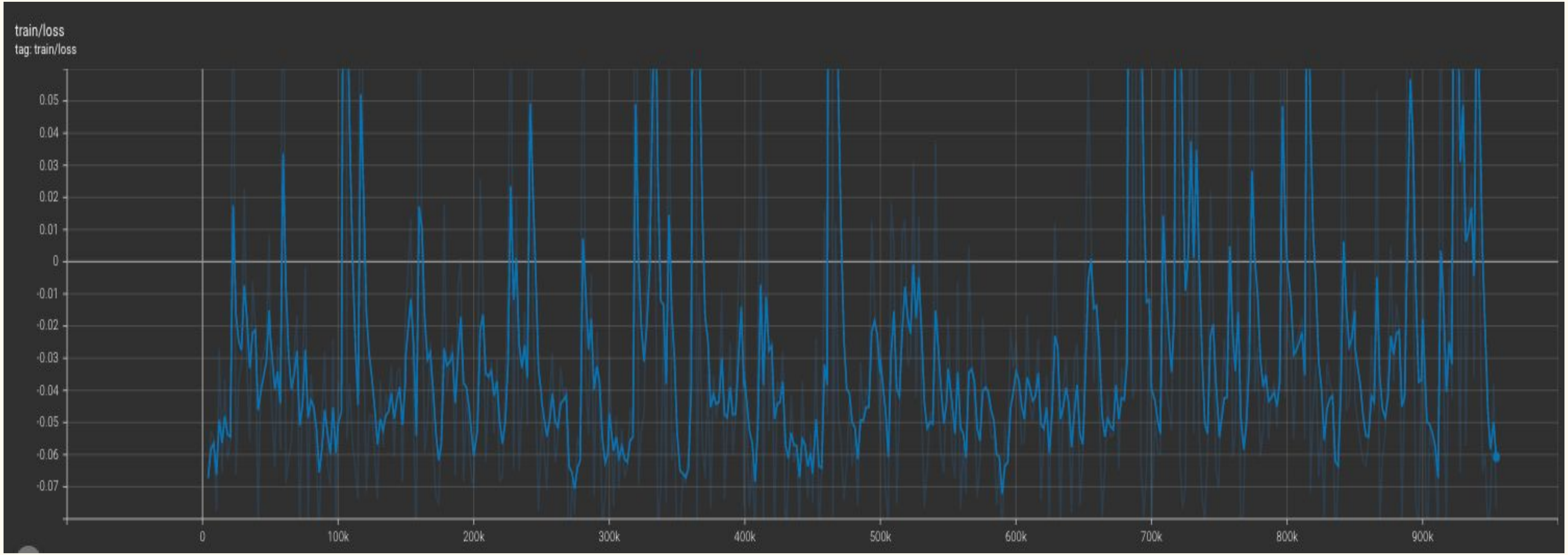
Result

Result : Rewards



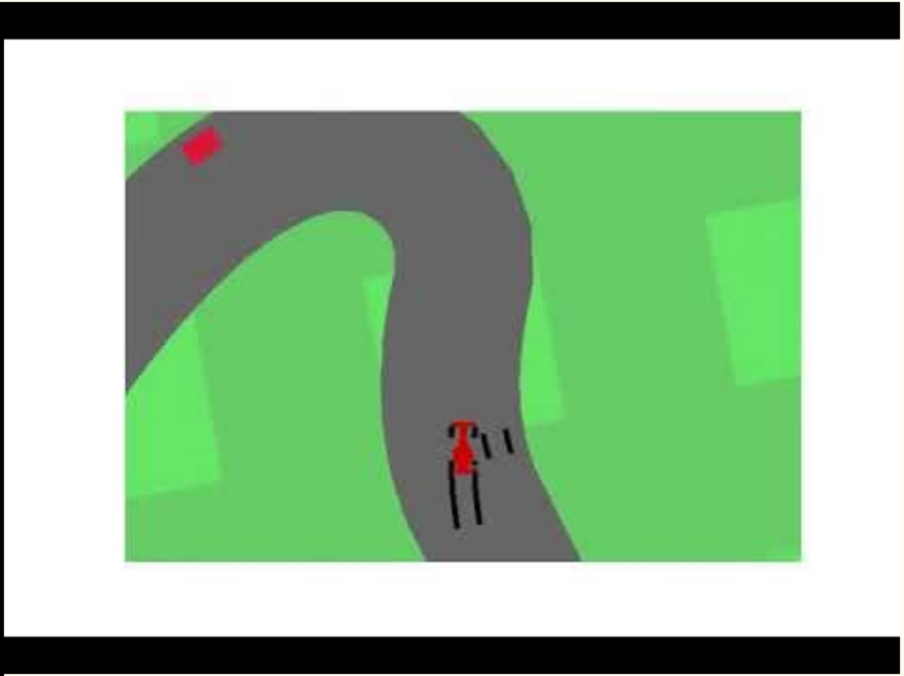
Rewards significantly increase after 200k iterations and peak at 300 k iterations then. rewards drop to -100 before slight increase again.

Result : Rewards

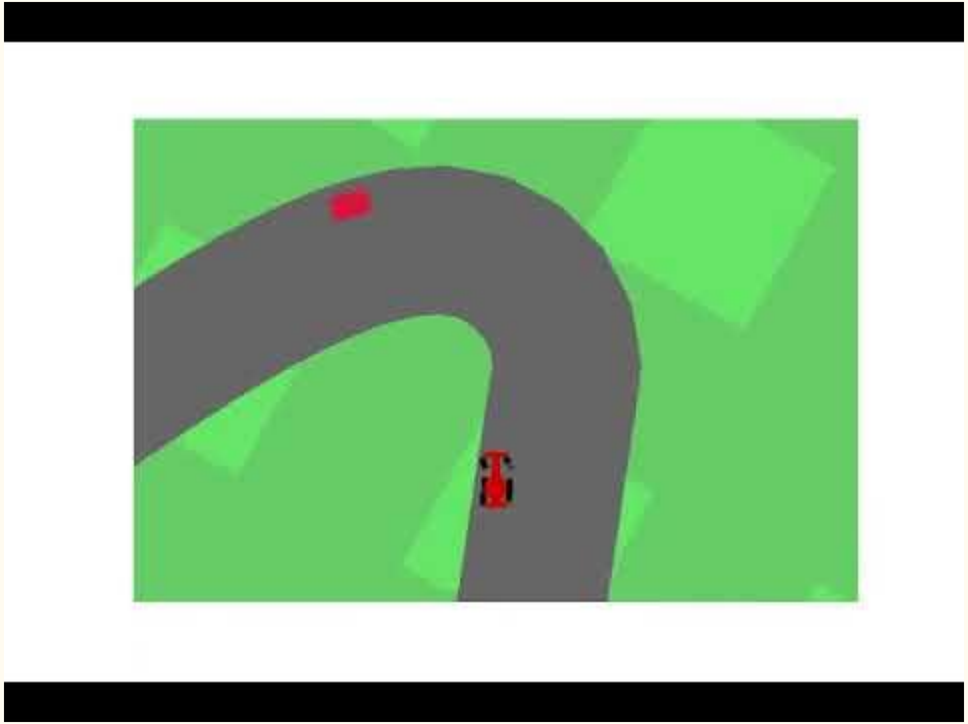


Losses decrease after 100k iterations. However, in some iterations losses fluctuate

Result : Video



Result : Video



Code

Env : nutapol97/Deep_RL_Projec_Car_Racing: This repository for Deep reinforcement learning subject (github.com)

Train :
https://colab.research.google.com/drive/1o-pJ796n83TkYHdUe6dlxg2F8R-tn_Ta?usp=sharing

Q & A