# IERG 5350 HW5

Car Racing Game

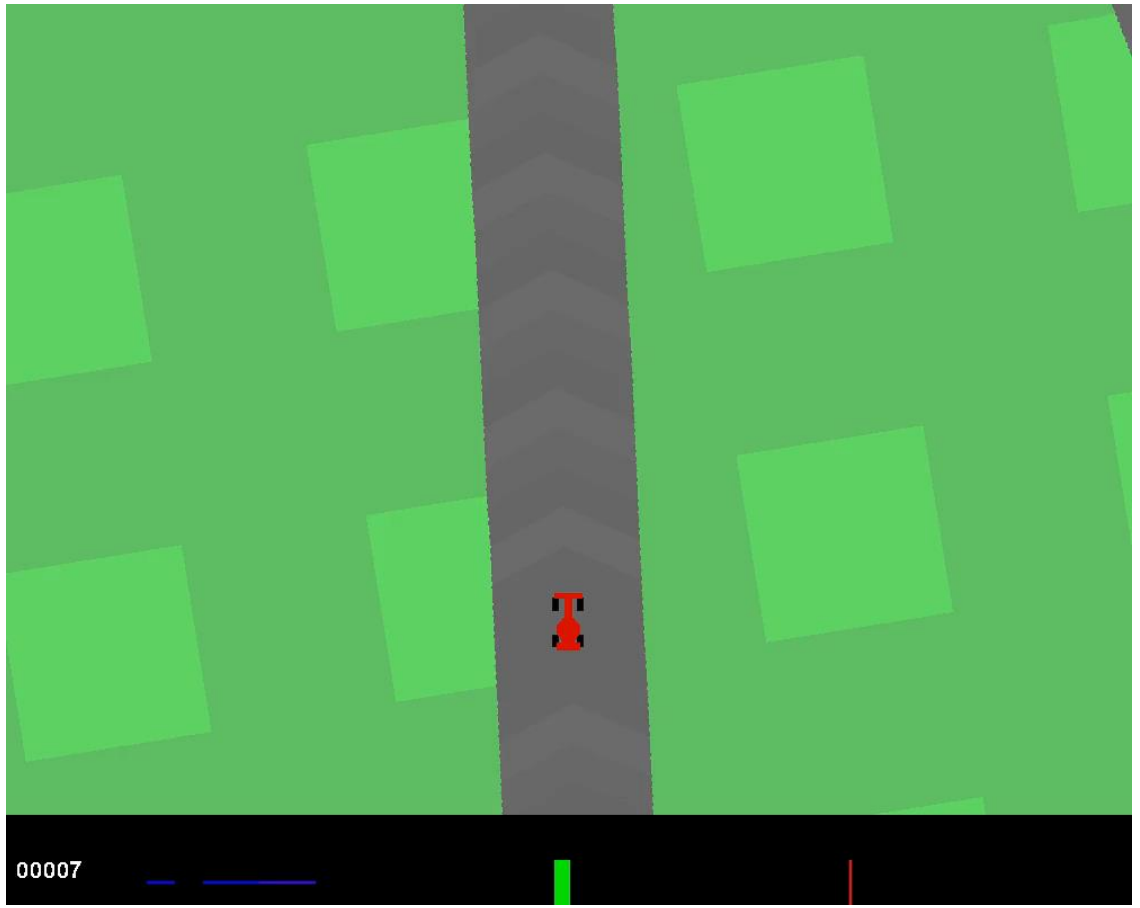
**Lixin LIU**

**2020.12.07**

# Results (Single Agent)

═══ Reward Matrix (row vs column) ═══

| | 1155136644 | alphacar | zhenghao | average |
|---|---|---|---|---|
| 1155136644 | 695.994 | 862.475 | 844.087 | 800.852 |
| alphacar | 715.542 | 601.989 | 599.341 | 638.957 |
| zhenghao | 625.051 | 599.529 | 556.458 | 593.679 |

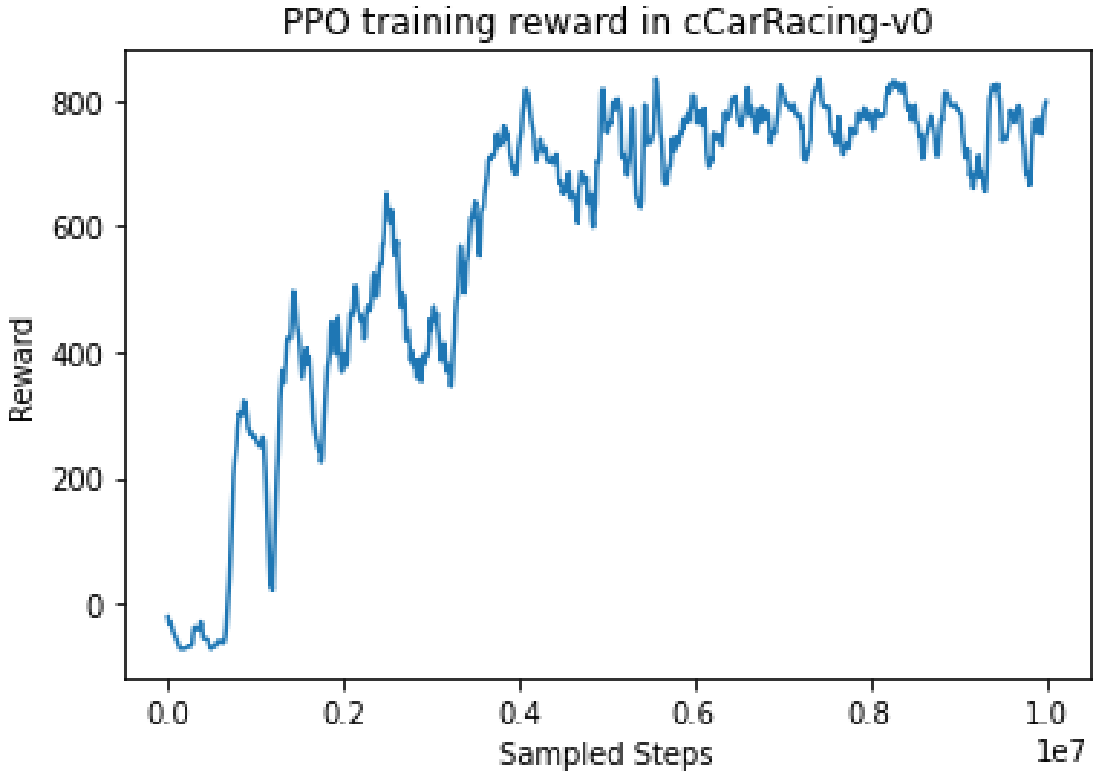


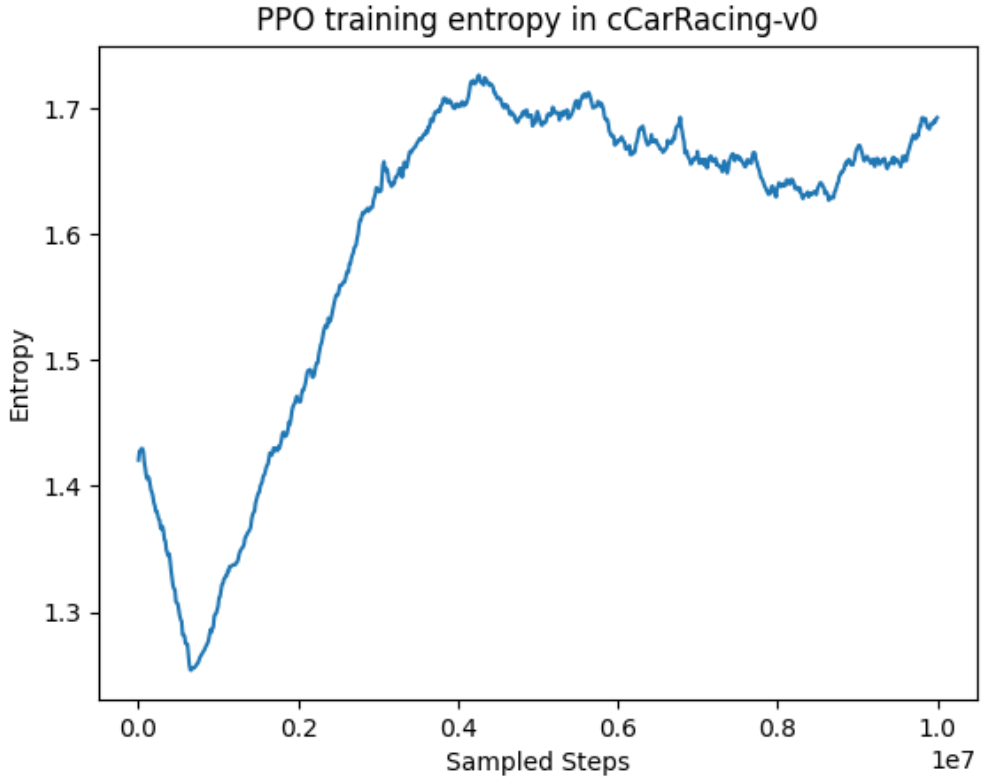
# Results (Multi Agents)

# Hyperparameters

**I use the default hyperparameters…**

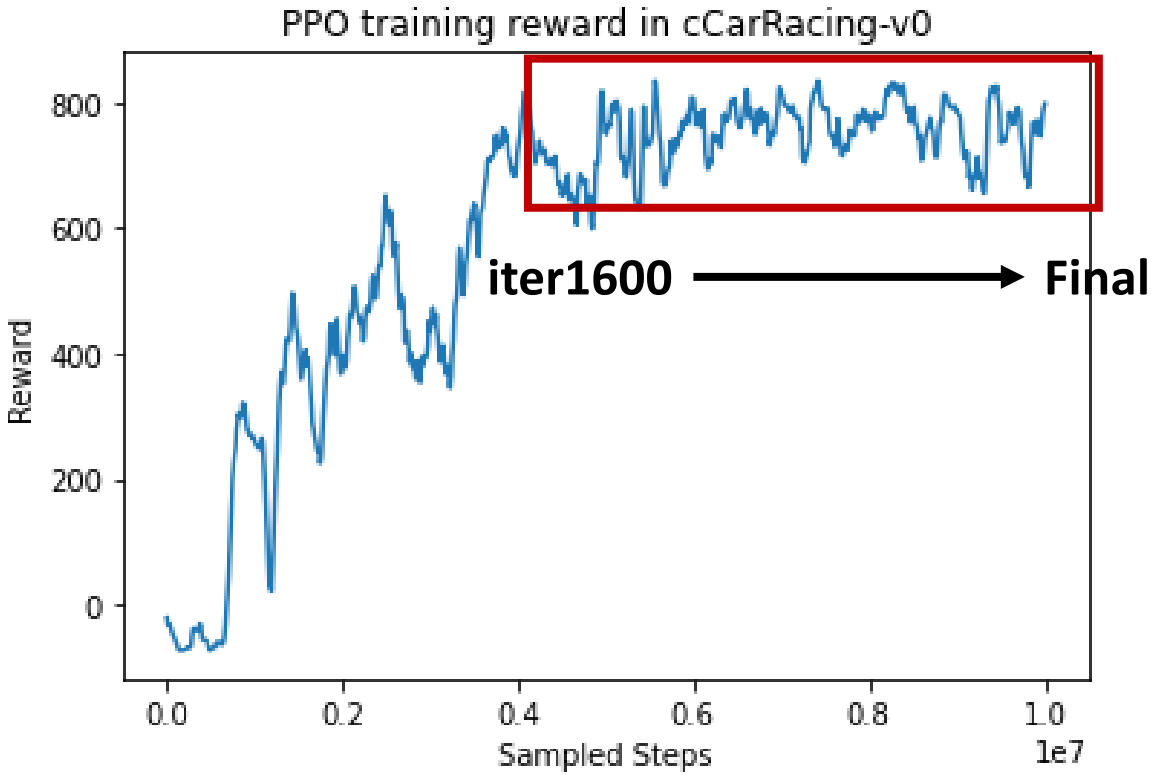
train.py --algo PPO --env-id cCarRacing-v0 --num-envs 12 --lr 1e-4 --action-repeat 1 --entropy 0.0




**Nothing special…**

# Checkpoint Selection




Find the best checkpoint with a **simple** scripts

```
best_ckpt = OneGoodCkpt
ckpt_list = []
ckpt_list.append(best_ckpt)
for cur_ckpt in all_ckpts:
    markdown_res_path = tournament(cur_ckpt, best_ckpt)
    cur_avg_score = parse_md(markdown_res_path)
    if cur_avg_score > best_avg_score:
        best_avg_score = cur_avg_score
        best_ckpt = cur_ckpt
        ckpt_list.append(best_ckpt)

markdown_res_path = tournament(*ckpt_list)
pick_best_ckpt_from_markdown_res
```

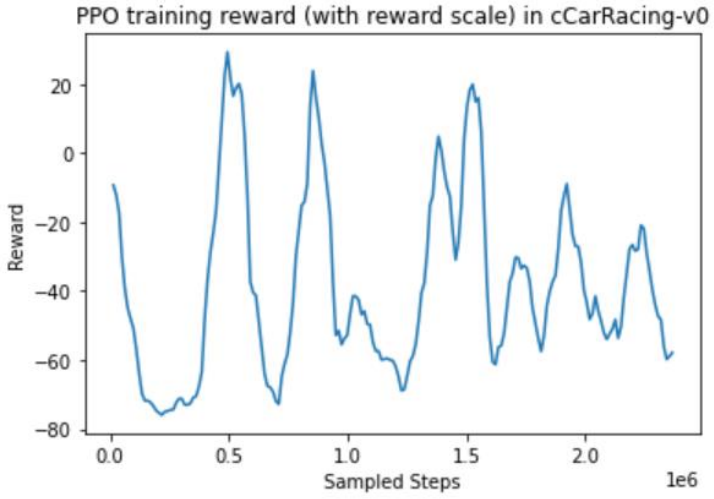
Best: iter4100

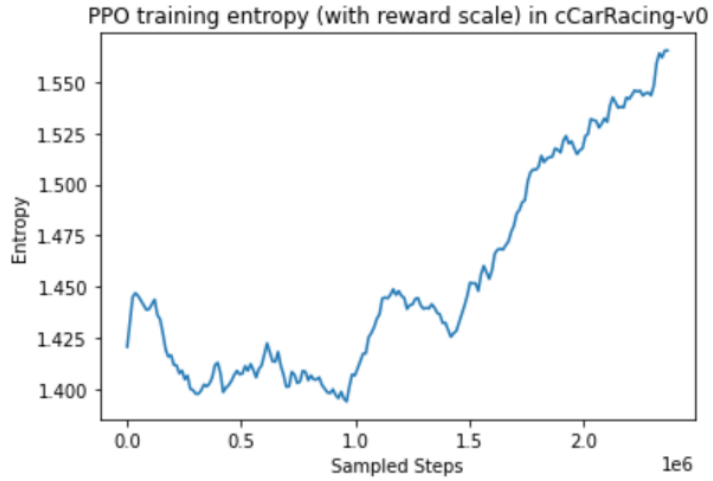
# Why using the default params

## Something I tried…

- Reward scale
  - Reward scale doesn't work in car-racing game
- Value clip
  - It seems that value clip cannot bring significant improvements in car-racing game
- Different value of entropy loss weight
  - Value: 0, 1e-2, 1e-3, 1e-4
  - Better to train without entropy loss... (entropy_loss_weight == 0)
- Opponent training
  - Not working...

# Reward Scale




The training is early terminated around 2.4e6 step. Reward scale may not be useful in car-racing game.

A.2 PPO CODE-LEVEL OPTIMIZATIONS

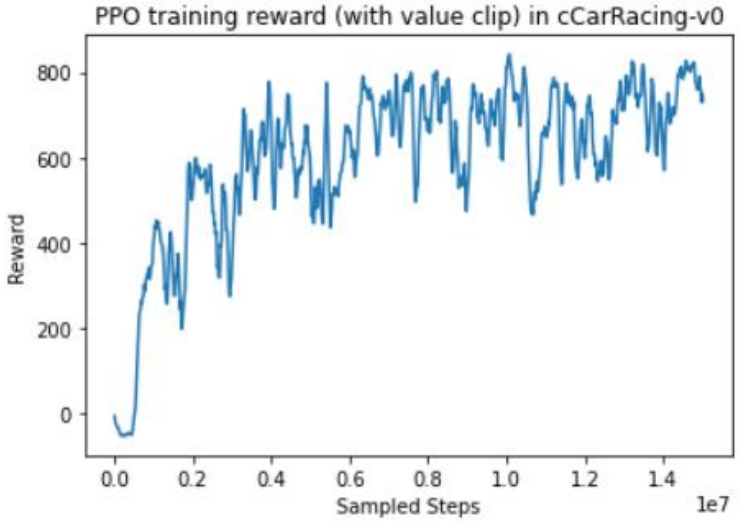
**Algorithm 1** PPO scaling optimization.

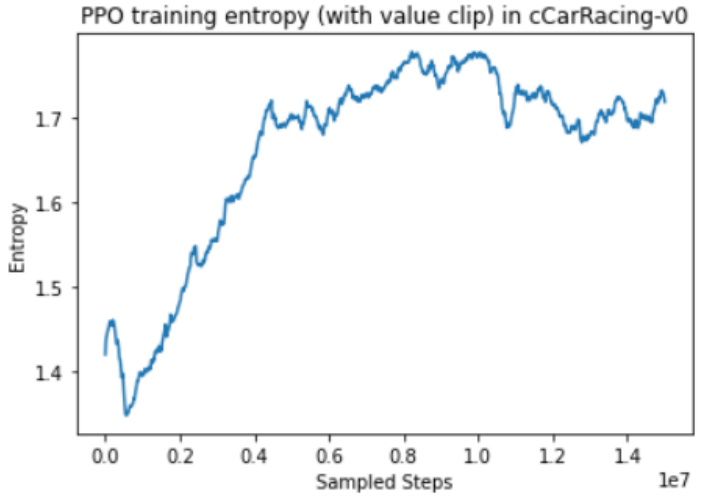
**procedure** INITIALIZE-SCALING()
  $R_0 \leftarrow 0$
  $RS = \text{RUNNINGSTATISTICS}()$ ▷ New running stats class that tracks mean, standard deviation
**procedure** SCALE-OBSERVATION($r_t$) ▷ Input: a reward $r_t$
  $R_t \leftarrow \gamma R_{t-1} + r_t$ ▷ $\gamma$ is the reward discount
  ADD($RS, R_t$)
  **return** $r_t / \text{STANDARD-DEVIATION}(RS)$ ▷ Returns scaled reward

1. Rewards are too random in the early stage of training.
2. Bug?

Engstrom, Logan, et al. "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO." In ICLR (2020).
https://github.com/thu-ml/tianshou

# Value Clip




There is no significant difference between enabling value clip and disabling value clip... So, I disable the value clip.

```
if self.use_value_clip:
    v_clip = old_value_batch + (values - old_value_batch).clamp(-self.clip_param, self.clip_param)
    value_term1 = (return_batch - values).pow(2)
    value_term2 = (return_batch - v_clip).pow(2)
    value_loss = 0.5 * torch.max(value_term1, value_term2).mean()
else:
    value_loss = 0.5 * (return_batch - values).pow(2).mean()
```

1. **Value function clipping:** Schulman et al. (2017) originally suggest fitting the value network via regression to target values:

$$L^V = (V_{\theta_t} - V_{targ})^2,$$

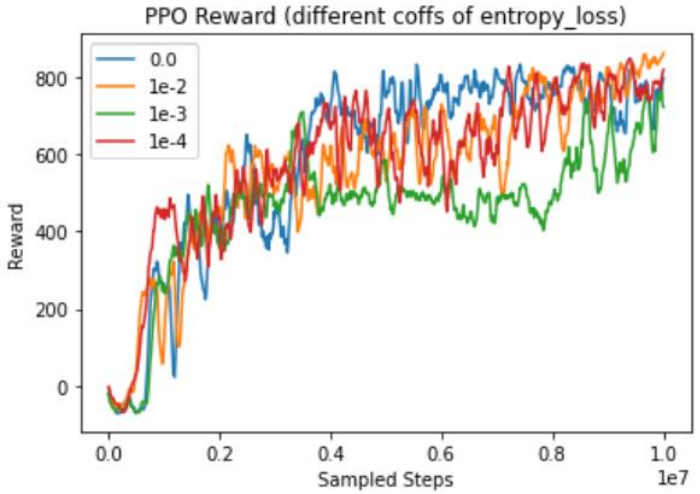
but the standard implementation instead fits the value network with a PPO-like objective:

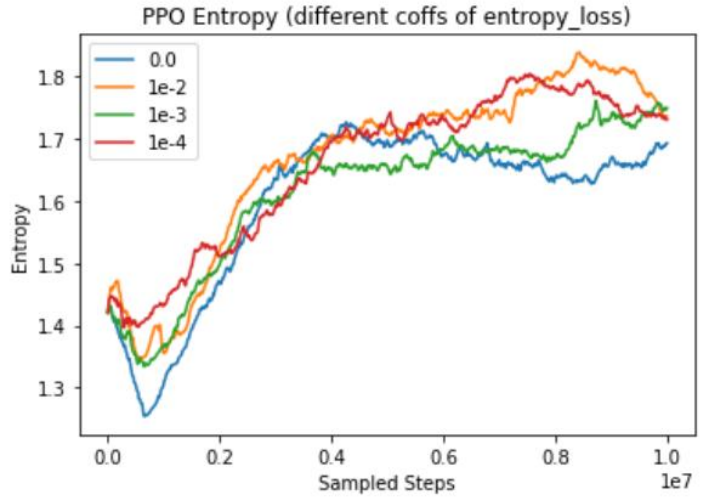
$$L^V = \min\left[(V_{\theta_t} - V_{targ})^2, \left(\text{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{targ}\right)^2\right],$$

where $V_\theta$ is clipped around the previous value estimates (and $\varepsilon$ is fixed to the same value as the value used in (2) to clip the probability ratios).

Engstrom, Logan, et al. "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO." In ICLR (2020).
https://github.com/thu-ml/tianshou

# Entropy Loss Weight




After evaluating the performance of these agents, I choose the agent which is trained without entropy_loss.

**Entropy loss need not to be involved.**

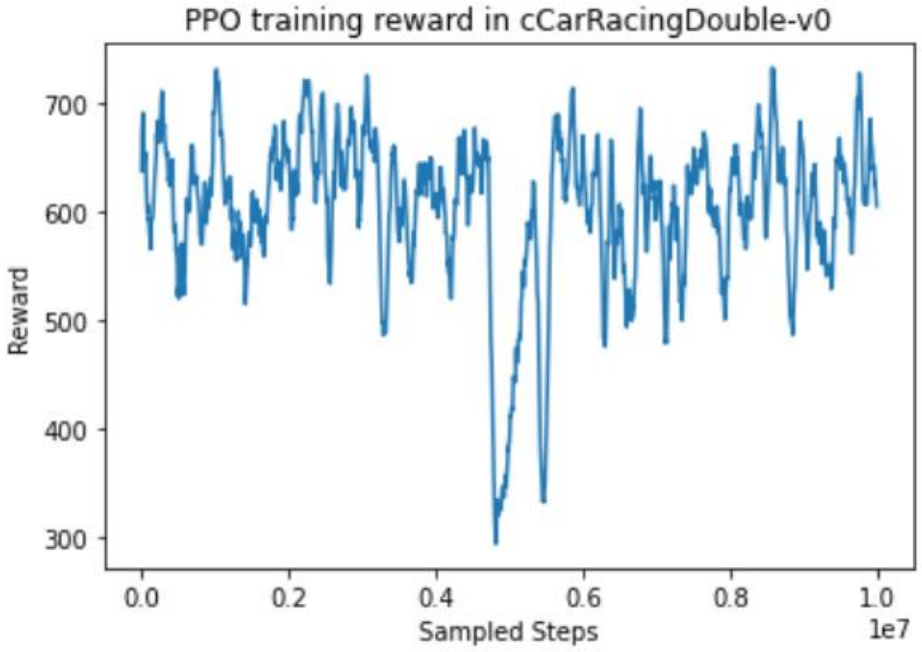
Table 6: Hyperparameters for all algorithms for Hopper-v2.

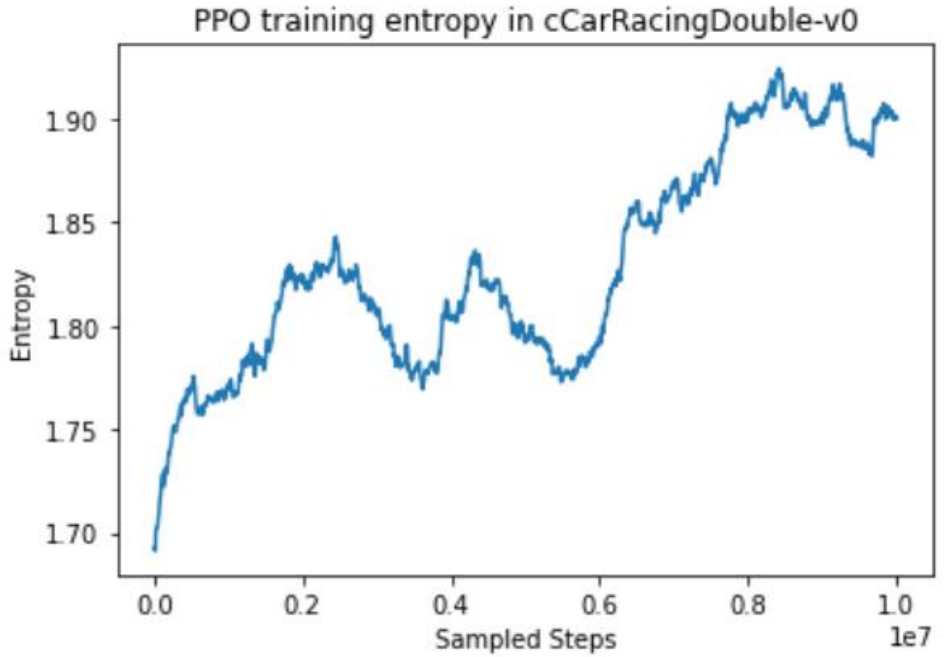
| | PPO | TRPO | PPO-NoClip | PPO-M | TRPO+ |
|---|---|---|---|---|---|
| Timesteps per iteration | 2048 | 2048 | 2048 | 2048 | 2048 |
| Discount factor ($\gamma$) | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| GAE discount ($\lambda$) | 0.95 | 0.95 | 0.925 | 0.95 | 0.95 |
| Value network LR | 0.00025 | 0.0002 | 0.0004 | 0.0004 | 0.0002 |
| Value network num. epochs | 10 | 10 | 10 | 10 | 10 |
| Policy network hidden layers | [64, 64] | [64, 64] | [64, 64] | [64, 64] | [64, 64] |
| Value network hidden layers | [64, 64] | [64, 64] | [64, 64] | [64, 64] | [64, 64] |
| KL constraint ($\delta$) | N/A | 0.13 | N/A | N/A | 0.04 |
| Fisher estimation fraction | N/A | 0.1 | N/A | N/A | 0.1 |
| Conjugate gradient steps | N/A | 10 | N/A | N/A | 10 |
| Conjugate gradient damping | N/A | 0.1 | N/A | N/A | 0.1 |
| Backtracking steps | N/A | 10 | N/A | N/A | 10 |
| Policy LR (Adam) | 0.0003 | N/A | 6e-05 | 0.00017 | N/A |
| Policy epochs | 10 | N/A | 10 | 10 | N/A |
| PPO Clipping $\varepsilon$ | 0.2 | N/A | 1e+32 | 0.2 | N/A |
| Entropy coeff. | 0 | 0 | -0.005 | 0 | 0 |
| Reward clipping | [-10.0, 10.0] | – | [-2.5, 2.5] | – | [-10.0, 10.0] |
| Gradient clipping ($\ell_2$ norm) | -1 | -1 | 4 | -1 | 1 |
| Reward normalization | returns | none | rewards | none | returns |
| State clipping | [-10.0, 10.0] | – | [-2.5, 2.5] | – | [-10.0, 10.0] |

Engstrom, Logan, et al. "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO." In ICLR (2020).
https://github.com/thu-ml/tianshou

# Opponent training

Python train.py --algo PPO --env-id cCarRacing-v0 --num-envs 6 --lr 1e-4 --entropy 0.0 --max-steps 1e7 --opponent true --restore data/alphacar/checkpoint-1155136644.pkl




Not working. There are neither penalties nor rewards for collision.

# Simple is better...

# Q&A

# Thanks!

**Lixin LIU**

**2020.12.07**