作业 3：PPO 算法实践报告

张豪派

2024 年 7 月

1 问题描述

TRPO 算法计算过程复杂，且含有约束问题，因此提出了 PPO 算法用于简化计算过程。PPO 算法核心思想是将 TRPO 有约束最大化问题，利用拉格朗日函数，转换为无约束的最大化问题。主要分为两大类，即 PPO-惩罚和 PPO-Clip 算法。

PPO-惩罚算法利用拉格朗日函数将 KL 散度的约束限定在目标函数中，PPO-Clip 算法将目标函数限定在 $1-\epsilon$ 和 $1+\epsilon$ 范围内。

本次实验实现了 PPO-Clip 算法。

2 算法描述

对于策略网络 $\pi(a|s;\theta)$，TRPO 算法的目标是

$$\begin{aligned} &\theta : \arg\min_{\theta} L(\theta_{old}, \theta) \\ &s.t. \quad \overline{D}_{KL}(\theta||\theta_k) <= \delta \end{aligned} \tag{1}$$

其中，

$$L(\theta_{old}, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_\theta}(s,a)] \tag{2}$$

PPO 算法利用拉格朗日函数，将上述带有约束的最大化问题，转变为了无约束的最大化问题。PPO-Clip 算法的最大化目标转变为：

$$\mathbb{E}_{s,a\sim\pi_{\theta_k}}[min(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_\theta}(s,a), clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A_{\pi_\theta}(s,a))] \tag{3}$$

其中 $clip(a,b,c)$ 含义为将 a 限制在 $[b,c]$ 范围内，即 $b \le a \le c$

3 实验过程

3.1 策略和价值网络更新

策略和价值网络更新分为以下步骤：

1. 计算旧参数下的 log 概率分布
2. 计算当前优势函数
3. 循环 N 次
   (a) 计算新参数下的 log 概率分布
   (b) 做两次估计，估计 1 为 $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_\theta}(s,a)$
   (c) 估计 2 为 $clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A_{\pi_\theta}(s,a)$
   (d) 取估计 1 和估计 2 较小的作为策略损失，更新策略网络
   (e) 价值网络计算 mse 损失，更新价值网络

Listing 1: PPO-Clip 策略和价值网络更新

```
def update(self, trajectory):
state_list = torch.tensor(np.array(trajectory.state), dtype=
    torch.float)
action_list = torch.tensor(trajectory.action, dtype=torch.
    int64).view(-1, 1)
reward_list = torch.tensor(trajectory.reward, dtype=torch.
    float).view(-1, 1)
next_state_list = torch.tensor(np.array(trajectory.next_state
    ), dtype=torch.float)
done_list = torch.tensor(trajectory.done, dtype=torch.float).
    view(-1, 1)

td_target = reward_list + self.gamma * self.value_net(
    next_state_list) * (1 - done_list)
td_delta = td_target - self.value_net(state_list)
advantage_list = gae(self.gamma, self.lam, td_delta)
old_log_prob = torch.log(self.policy_net(state_list).gather
    (1, action_list)).detach()

for _ in range(10):
    # 计算新的log概率分布
    log_prob = torch.log(self.policy_net(state_list).gather
        (1, action_list))
    ratio = torch.exp(log_prob - old_log_prob)
    # 估计1
    surr1 = ratio * advantage_list
    # 估计2
    surr2 = torch.clamp(ratio, 1-eps, 1+eps)*advantage_list
    policy_loss = torch.mean(-torch.min(surr1, surr2))
    value_loss = torch.mean(F.mse_loss(self.value_net(
        state_list), td_target.detach()))
    self.policy_optimizer.zero_grad()
    self.value_optimizer.zero_grad()
    policy_loss.requires_grad_(True)
```

```
        policy_loss.backward()
        value_loss.requires_grad_(True)
        value_loss.backward()
        self.policy_optimizer.step()
        self.value_optimizer.step()
```

4 实验问题分析

4.1 梯度消失问题

在训练过程中，会出现梯度消失问题，导致网络参数出现 nan。我采取了以下措施：

1. 将激活函数 relu 修改为 leaky_relu
2. 调小学习率

5 实验运行结果

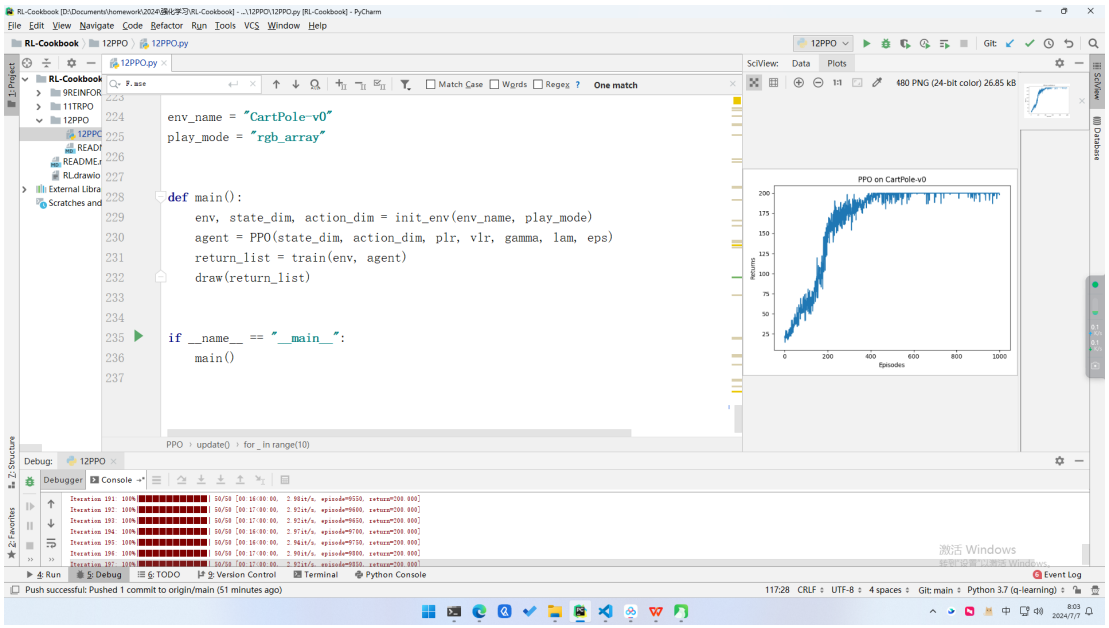

Figure 1: 带时间的实验结果截图

根据 Figure2, 我们发现智能体能够长期达到游戏得分的最高值，即 200 分。

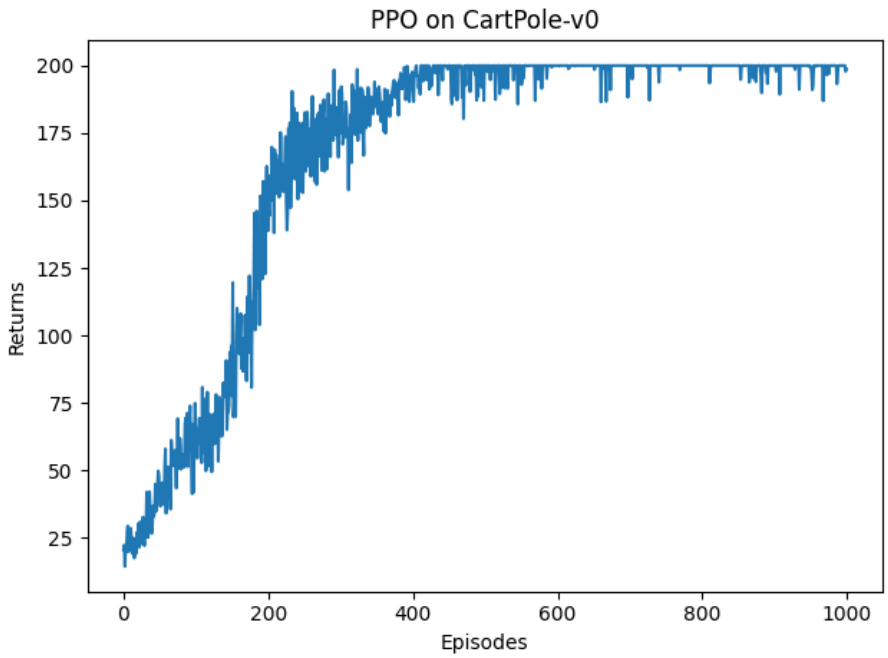

Figure 2: reward 随着 episodes 的变化情况

6 附录：实验运行环境设置

请使用 python 3.7，进入 PPO 项目目录执行以下命令：

```
pip install -r requirements.txt
```

安装依赖后运行

```
python PPO.py
```