PPO 算法调研综述

SA23006171 吕奇澹

一、实验内容：

1. 回顾本学期所学知识，系统梳理强化学习的发展历程及其相关算法，以便对强化学习算法在实际应用中的基本运作有一个初步的了解。。
2. 对 PPO 算法进行调研，从数学原理、算法的局限性、现有的改进方案以及 value-based 和 policy-based 的对比对 PPO 算法进行分析，此外，通过运用与研究主题相一致的开源代码来实际验证 PPO 算法的有效性。

二、实验过程：

1. PPO 算法介绍：

强化学习可根据其学习策略的方法学特征，划分为基于价值的方法和基于策略的方法。在深度强化学习这一领域，深度学习技术与基于价值的 Q-Learning 算法的融合，催生了深度 Q 网络（DQN）算法。DQN 算法通过引入经验回放池和目标网络，成功地将深度学习算法应用于强化学习。其中，最具代表性的算法分别是 Q-Learning 和策略梯度(Policy Gradient)算法。深度 Q 网络是 Q-Learning 与深度学习结合的产物，而演员-评论家(Actor Critic)、确定性策略梯度(DPG)、深度确定性策略梯度（DDPG）、异步优势行动者-评论家（A3C）、信任域策略优化（TRPO）、近端策略优化（PPO）等算法，则是将这两种方法优势结合的更为先进的产物。本次实验关注的研究对象是目前表现较为出色的近端策略优化(PPO)算法。

PPO（近端策略优化）是一种 On Policy 的强化学习算法。由于其具有实施简便、易于理解、性能稳健、适用于处理离散与连续动作空间的问题，并且便于大规模训练等显著优势，因此近年来广受瞩目。PPO 算法本质上属于 PG（策略梯度）算法的范畴。自 1999 年 PG 算法由 Sutton 首次提出至今，其发展脉络可概述如下：

表 1. PPO 算法发展历程

算法	时间	论文
PG Policy Gradient	1999	Sutton R S, McAllester D, Singh S, et al. Policy gradient methods for reinforcement learning with function approximation[J]. Advances in neural information processing systems, 1999, 12.
CPI Conservative Policy Iteration	2002	Kakade S, Langford J. Approximately optimal approximate reinforcement learning[C]//Proceedings of the Nineteenth International Conference on Machine Learning. 2002: 267-274.
TRPO Trust Region Policy Optimization	2015	Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. PMLR, 2015: 1889-1897.
PPO Proximal Policy Optimization	2017	Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

PPO 算法是一种创新型的策略梯度算法。在策略梯度算法中，步长的选择对性能有着举足轻重的影响，然而合适的步长却难以确定。在训练过程中，若新旧策略之间的差异过大，将对学习产生不利影响。PPO 算法提出了一种新颖的目标函数，能够在多个训练步骤中实现小批量更新，从而有效解决了策略梯度算法中步长选择问题。虽然 TRPO 算法也旨在解决这一问题，但与之相比，PPO 算法更易于求解。

2. PPO 算法数学原理：

(1)首先我们给出一些强化学习算法中的基本概念的数学定义：

强化学习算法由马尔可夫决策过程发展而来。在一个马尔可夫过程中，S是一个有限的状态空间集合，A是一个有限的动作空间集合，$P: S \times A \times S \to \mathbb{R}$表示状态转移概率函数，例如$P(s'|s,a) = 0.6$ 表示的含义就是在状态s处执行动作a

到达的状态为s'的概率为 0.6。$r: S \to \mathbb{R}$是奖励函数，$\rho_0: S \to \mathbb{R}$是初始状态分布概率函数，$\gamma \in (0,1)$是折扣因子。

让π表示一个随机策略函数$\pi: S \times A \to [0,1]$, 例如$\pi(s,a) = 0.5$ 表示在状态s处选择动作a的概率为 0.5。

下面给出状态价值函数、状态动作价值函数、优势函数的定义：

(1) **状态动作价值函数**:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})]$$

表示的是在状态s_t处执行动作a_t后获得的长期期望折扣奖励。

(2) **状态价值函数**:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})] = \mathbb{E}_{a_t}[Q_\pi(s_t, a_t)]$$

表示从状态s_t开始获得的长期期望折扣奖励。

(3) **优势函数**:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s,a)$$

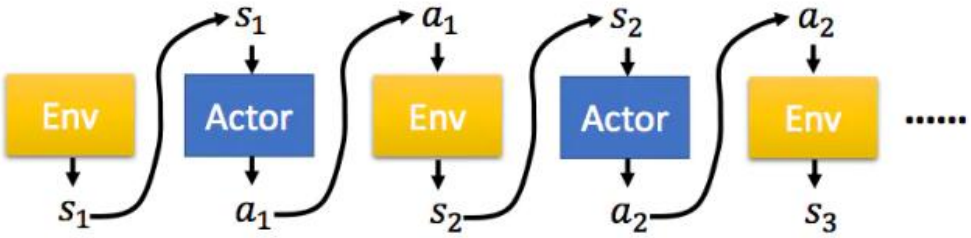
表示的是在状态s处，动作a相对于平均水平的高低。

强化学习的目标就是最大化长期期望折扣奖励:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$$

(2) Policy Gradient 算法介绍

在 PG 算法中，我们的 Agent 又被称为 Actor，Actor 对于一个特定的任务，都有自己的一个策略π，策略π通常用一个神经网络表示，其参数为θ。从一个特定的状态 state 出发，一直到任务的结束，被称为一个完整的 eposide，在每一步，我们都能获得一个奖励r，一个完整的任务所获得的最终奖励被称为R。这样，一个有T个时刻的 eposide，Actor 不断与环境交互，形成如下的序列τ：



Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$

这样一个序列τ是不确定的，因为 Actor 在不同 state 下所采取的 action 可能是不同的，一个序列τ发生的概率为：

$$
\begin{aligned}
& p_\theta(\tau) \\
&= p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)\cdots \\
&= p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)
\end{aligned}
$$

序列τ所获得的奖励为每个阶段所得到的奖励的和，称为$R(\tau)$。因此，在 Actor 的策略为π的情况下，所能获得的期望奖励为：

$$\bar{R}_\theta = \sum_\tau R(\tau)p_\theta(\tau) = E_{\tau\sim p_\theta(\tau)}[R(\tau)]$$

而我们的期望是调整 Actor 的策略π，使得期望奖励最大化，于是我们有了策略梯度的方法，既然我们的期望函数已经有了，我们只要使用梯度提升的方法更新我们的网络参数θ（即更新策略π）就好了，所以问题的重点变为了求参数的梯度。梯度的求解过程如下：

$$\nabla \bar{R}_\theta = \sum_\tau R(\tau)\nabla p_\theta(\tau) = \sum_\tau R(\tau) p_\theta(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)}$$

$R(\tau)$ do not have to be differentiable

It can even be a black box.

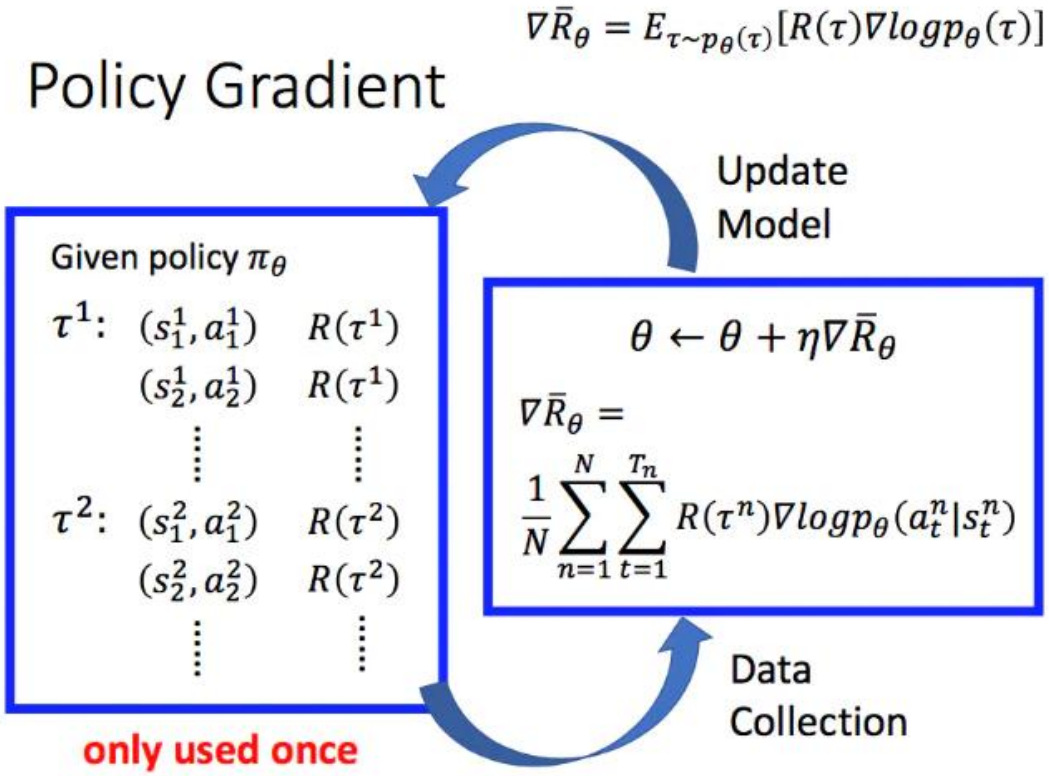
$$= \sum_\tau R(\tau) p_\theta(\tau) \nabla log p_\theta(\tau)$$

$$\nabla f(x) = f(x)\nabla log f(x)$$

$$= E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla log p_\theta(\tau)] \approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla log p_\theta(\tau^n)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla log p_\theta(a_t^n|s_t^n)$$

利用 log 函数的求导特点，我们可以将目标函数进行转化，随后用 N 次采样的平均值来近似期望，最后，我们将p_θ展开，将与θ无关的项去掉，即得到了最终的结果。



首先，我们进行数据的采集工作。接着，依据先前获得的梯度提升公式对参数进行更新。此后，基于更新后的策略再次进行数据采集，并据此更新参数。这

一过程循环进行。值得注意的是，图示中突出的大红字“only used once”指的是，在参数更新后，我们的策略已经发生改变，因此先前基于旧策略采集的数据将不再适用。

(3) PPO 算法数学原理

PG 方法的一个显著缺陷在于参数更新速度缓慢。这是因为每次更新参数都需要进行一次新的采样过程，实质上这是一种 on-policy 策略。所谓 on-policy 策略，是指进行训练的智能体与与环境互动的智能体为同一实体。相对应的，则是 off-policy 策略，其特点在于训练智能体与与环境互动的智能体并非同一实体。简言之，off-policy 策略类似于利用他人的经验进行自我训练。以下棋为例，若一个人通过自己下棋不断进步，那便是 on-policy 策略的体现；相反，如果通过观察他人下棋以提高自己的棋技，这便是 off-policy 策略的实践。

那为了加快我们的训练速度并使采样数据能够被重复利用，我们可以考虑将基于策略的(on-policy)方法转变为脱离策略的(off-policy)方法。具体来说，我们的训练数据可以通过另一个执行者（Actor）获得，其对应的网络参数为 θ 。要实现这一转变，我们可以遵循以下思路。

先给出 Importance Sampling 的概念：假设两个分布$p(x)$ 和$q(x)$，我们只知道$p(x)$满足某个分布，但是无法对$p(x)$进行积分；而且假设我们只能从$q(x)$中进行 sampling data。由以下公式：

$$E_{x\sim p}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

这样就把满足$p(x)$分布的均值转换成满足$q(x)$分布的均值了。

Importance Sampling

$$E_{x\sim p}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x^i)$$

x^i is sampled from $p(x)$

We only have x^i sampled from $q(x)$

$$= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$$

Importance weight

通过这种方式，我们必须确保 p(x)和 q(x)的分布之间的差异不宜过大。若两者分布相差甚远，则需执行大量采样操作方能获得近似结果。原因在于，尽管通过特定变换，这两个分布的均值可以保持一致，但它们的方差仍存在差异：

$$Var(X) = E(X^2) - E^2(X)$$

那么，有

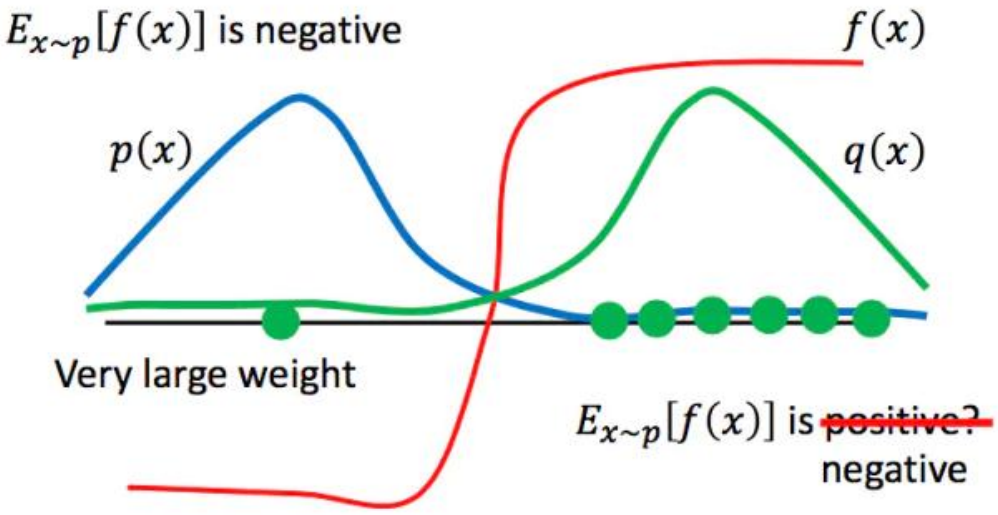
$$Var_{x\sim p}(x) = E_{x\sim p}\left[f^2(x)\right] - E^2_{x\sim p}[f(x)]$$

和

$$Var_{x\sim q}(x) = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$
$$= E_{x\sim q}\left[f^2(x)\left(\frac{p(x)}{q(x)}\right)^2\right] - E^2_{x\sim q}\left[f^2(x)\left(\frac{p(x)}{q(x)}\right)\right]$$
$$= E_{x\sim p}\left[f^2(x)\left(\frac{p(x)}{q(x)}\right)\right] - E^2_{x\sim p}[f(x)]$$

用一个简单的图示来说明：

$$E_{x\sim p}[f(x)] = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$$



如上图所示，显而易见，在变量 x 遵循 p(x)分布的情形下，函数 f(x)的期望值为负。因此，当我们从 q(x)分布中仅采样少量的 x 值时，所得样本很可能主要集中于数轴的右半部分。在这种情况下，f(x)的值大都大于 0，进而导致我们容易得出 f(x)的期望值为正的误判。这便是潜在的问题所在，为避免此问题，我们需要进行大量的采样以确保结果的准确性。

根据给出的公式，有：

$$\nabla \bar{R}_\theta = E_{\tau\sim p_{\theta'}(\tau)}\left[\frac{p_\theta(x)}{p_{\theta'}(x)} R(\tau)\nabla \log P_\theta(\tau)\right]$$

同时令$A^\theta(s_t, a_t) = \sum_\tau (R(\tau) - b)$，那么

$$\begin{aligned} & E_{(s_t,a_t)\sim\pi_\theta}\left[A^\theta(s_t, a_t)\nabla \log P_\theta(a_t^{(n)}|s_t^{(n)})\right] \\ = & E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(s_t, a_t)}{p_{\theta'}(s_t, a_t)} A^{\theta'}(s_t, a_t)\nabla \log P_\theta(a_t^{(n)}|s_t^{(n)})\right] \\ = & E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}\frac{p_\theta(s_t)}{p_{\theta'}(s_t)} A^{\theta'}(s_t, a_t)\nabla \log P_\theta(a_t|s_t)\right] \end{aligned}$$

在公式1的最后，因为状态出现的概率一般与Actor无关，所以有$p_\theta(s_t) = p'_\theta(s_t)$，所以最后有公式：

$$E_{(s_t,a_t)\sim\pi_\theta}\left[A^\theta(s_t, a_t)\nabla \log P_\theta(a_t^{(n)}|s_t^{(n)})\right] = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t)\nabla \log P_\theta(a_t|s_t)\right]$$

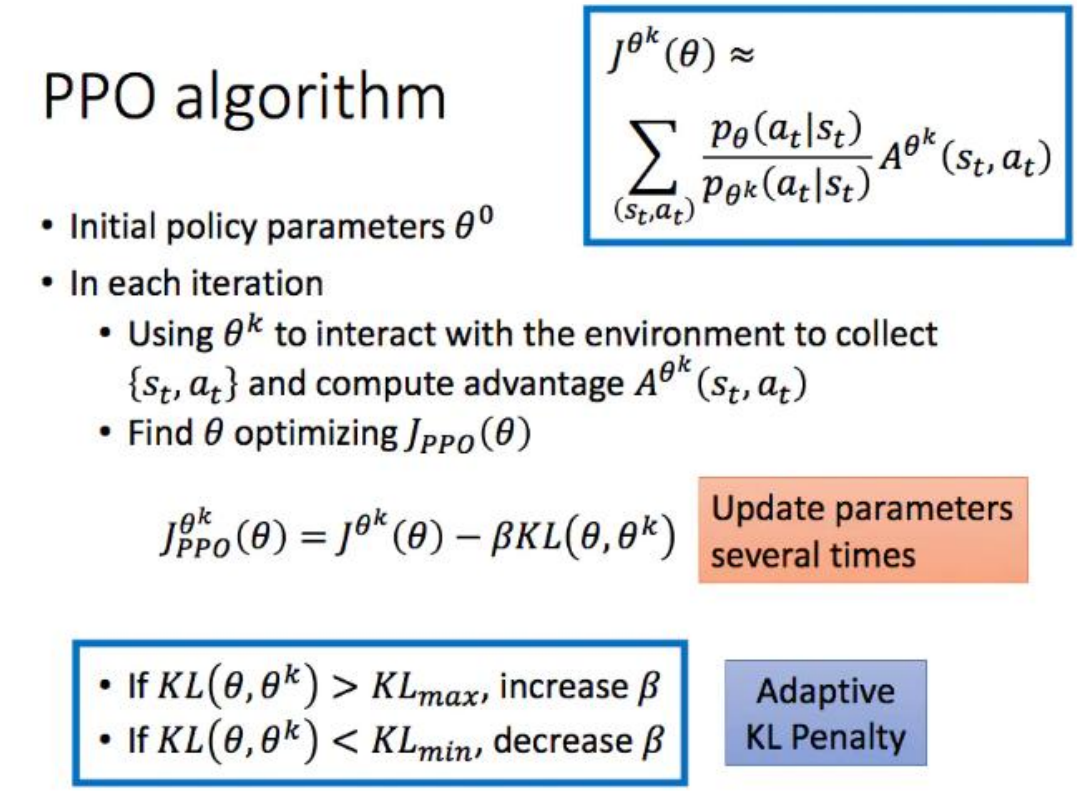
又因为有公式：

$$\nabla f(x) = f(x)\nabla \log f(x)$$

所以上式化简为：

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

由此，我们得到了 PPO 算法的整体过程：



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

3. PPO 算法局限性及改进

局限性：PPO 算法是当前工业界广泛应用于工程领域的一种算法，它不仅继承了

TRPO 算法的优势，同时也改善了 TRPO 的不足之处。PPO 算法本身较为成熟完善，没有显著的局限性。作为一种 on-policy 算法，PPO 在每次迭代之后都需要一批新的数据。因此，为了有效学习，它需要通过对成功数据的轮次探索来获得这些新数据。

改进：现有的对 PPO 算法的改进主要有两种，包括 PPO-penalty 和 PPO-clip.

（1）PPO-penalty （PPO-惩罚）

PPO-penalty 是 PPO 的一种变体，该算法可以近似地解决类似 TRPO 的 KL 约束更新，但在目标函数中惩罚 KL 发散，而不是使其成为一个硬约束，并在训练过程中自动调整惩罚系数，使其适当缩放。

首先初始化一个参数θ^0，在每一个迭代里面，我们要用前一个训练的迭代得到的演员的参数θ^k去跟环境做互动，采样到一大堆状态-动作的对。 根据θ^k互动的结果，估测$A^{\theta^k}(s_t,a_t)$。如下式所示。

$$J_{\mathrm{PPO}}^{\theta^k}(\theta)=J^{\theta^k}(\theta)-\beta \mathrm{KL}\left(\theta,\theta^k\right)$$

上述 KL 散度前需要乘一个权重β，需要一个方法来动态调整β。 这个方法就是自适应 KL 惩罚：如果$KL(\theta,\theta^k)>KL_{max}$，增加$\beta$；如果 $KL(\theta,\theta^k)<KL_{min}$，减少$\beta$。简单来说就是 KL 散度的项大于自己设置的 KL 散度最大值，说明后面这个惩罚的项没有发挥作用，就把β调大。同理，如果 KL 散度比最小值还要小，这代表后面这一项的效果太强了，所以要减少β。近端策略优化惩罚公式如下。

$$J_{PPO}^{\theta^k}(\theta)=J^{\theta^k}(\theta)-\beta KL\left(\theta,\theta^k\right)$$

$$J^{\theta^k}(\theta)\approx\sum_{(s_t,a_t)}\frac{p_\theta\left(a_t\mid s_t\right)}{p_{\theta^k}\left(a_t\mid s_t\right)}A^{\theta^k}\left(s_t,a_t\right)$$

但 PPO-penalty 仍然存在较为显著的局限性。目前，如 OpenAI 等公司广泛采用的主流变种是接下来将要介绍的 PPO-clip。

（2）PPO-clip （PPO-截断）

KL 散度的计算较为复杂，而 PPO-clip 近端策略优化裁剪则不需要计算 KL

散度。它通过如下方式更新策略：

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L\left(s,a,\theta_k,\theta\right)\right]$$

通常，我们采用多个步骤执行小批量的随机梯度下降（SGD）算法，以便最大化目标的实现。在此过程中，通过以下方式进行计算得到结果：

$$L\left(s,a,\theta_k,\theta\right) = \min\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\pi_{\theta_k}}(s,a),\right.$$
$$\left.\operatorname{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)},1-\epsilon,1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)$$

其中ε是一个（小的）超参数，它粗略地新策略与旧策略之间允许的距离。这是一种颇为复杂的表述方式，初看时难以判断其具体作用，或者它如何促进新旧策略的协调一致。实际上，该目标存在一个较为简化的版本，相对容易处理，如下所述：

$$L\left(s,a,\theta_k,\theta\right) = \min\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\pi_{\theta_k}}(s,a),\quad g\left(\epsilon,A^{\pi_{\theta_k}}(s,a)\right)\right)$$

$$g(\epsilon,A) = \begin{cases}(1+\epsilon)A & A\geq 0\\ (1-\epsilon)A & A<0\end{cases}$$

为了找出哪些直觉可以从中得到启发，让我们看看一个单一的状态行为对(s,a)，然后想想案例。

Advantage is positive：当状态-行为对的优势呈现出积极特征时，其对目标的贡献相应减少。在此情形下，我们可以观察到：

$$L\left(s,a,\theta_k,\theta\right) = \min\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)},(1+\epsilon)\right)A^{\pi_{\theta_k}}(s,a)$$

因为优势是正的，如果行动变得更有可能（也就是说，如果$\pi_\theta(a|s)$增加），

目标就会增加。但是这一项的最小值限制了目标可以增加多少。一旦$\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$，最小值发挥作用，这一项达到了$(1+\epsilon)A^{\pi_{\theta_k}(s,a)}$。因此，新策略与旧策略背道而驰，并不能从中受益。

Advantage is negative：假设状态-行为对的优势是负的，在这种情况下，它对目标的贡献减少到

$$L(s,a,\theta_k,\theta) = \max\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s,a)$$

因为优势是负的，如果行动变得更不可能（也就是说，如果$\pi_\theta(a|s)$减小），目标就会增加。但是这一项的最大值限制了目标可以增加多少。一旦$\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$，最大值发挥作用，这一项达到了$(1-\epsilon)A^{\pi_{\theta_k}(s,a)}$。因此，新策略再次与旧策略背道而驰，并不能从中受益。

到目前为止，我们所看到的是，通过消除策略急剧变化的激励因素，clipping作为一种规范化手段，其超参数ϵ对应于新策略与旧策略之间的距离有多远，同时仍然有利于实现目标。

虽然这种裁剪方法对于确保合理的策略更新具有重要作用，但仍存在一种可能，即最终形成的新策略与旧策略相差过大。不同的 PPO（比例策略优化）实现采用了多种技巧来避免这一情况。其中，有一种特别简便的方法是：提前终止。如果新策略与旧策略的平均 KL 散度超过了预设的阈值，我们便会停止进一步的梯度步骤。

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

4. Value-based learning 和 Policy-based learning 的比较

Value-based learning 方法和 Policy-based learning 方法主要在以下方面存在较为明显的差距：

1. 生成 policy 上的差异：**一个随机，一个确定**

在价值基础学习（Value-Based Learning）中，动作价值（action-value）的估计值最终会趋于收敛到真实值（true values），这些真实值通常是不同的有限数，且可转化为 0 到 1 之间的概率值。因此，这种学习方法通常会导致确定性策略（deterministic policy）的形成。而基于策略的学习（Policy-Based Learning）则不会收敛到确定性的值。相反，它们倾向于生成最优随机策略（optimal stochastic policy）。如果最优策略是确定性的，那么最优行动所对应的性能函数值将显著高于次优行动所对应的性能函数值。这里的性能函数值大小代表着相应行动的概率大小。

此外，值得一提的是随机策略的优势：在许多问题中，最优策略往往是随机策略。例如在石头剪刀布的游戏中，如果一个确定性策略总是选择石头，而随机策略则在石头、剪刀和布之间随机选择，那么相比之下，随机策略将有更大的获胜机会。

2. **一个连续，一个离散**

在探讨基于价值的学习（Value-Based Learning）时，针对连续动作空间问题的处理存在一定挑战。虽然可以通过将动作空间进行离散化来处理，但是如何恰当选择离散间距并非易事。若离散间距设置过大，则可能导致算法无法达到最优的行动（action）选择，而只能在最佳行动附近徘徊；若离散间距设置过小，则会使行动的维度增加，从而引发与高维度动作空间相同的维度灾难问题，进而影响算法的运行速度。与此相对，基于策略的学习（Policy-Based Learning）则更适用于连续的动作空间。在连续动作空间中，无需计算每个行动的概率，而是可以通过高斯分布（Gaussian distribution）来选择行动，从而在处理连续动作空间问题时展现出其优势。

3. 在 Value-Base 中，value function 的微小变化对策略的影响很大，可能直接决定了这个 action 是否被选取，而 Policy-Based 避免了此缺点。

三、代码验证：

在本次实验中，选用 pytorch 实现的 PPO-clip 算法进行代码验证。在本次实验中，我们使用了多个简单的强化学习环境以对算法进行验证，主要包括 OpenAI 给出的一些强化学习环境

1. Continuous RoboschoolHalfCheetah-v1
2. Continuous RoboschoolHopper-v1
3. Continuous RoboschoolWalker2d-v1

前三项都是 OpenAI 的机器人模拟库。机器人模拟库的作用，是在计算机虚拟环境中，利用力学原理，模拟现实中多关节物体的物理运动。在这样的虚拟环境里学习运动，可将得到的模型结果直接运用于现实中的机器人、机械臂等设备。好的虚拟环境，可以大大加快机器人的开发进度。

4. Continuous BipedalWalker-v2
5. Discrete CartPole-v1
6. Discrete LunarLander-v2

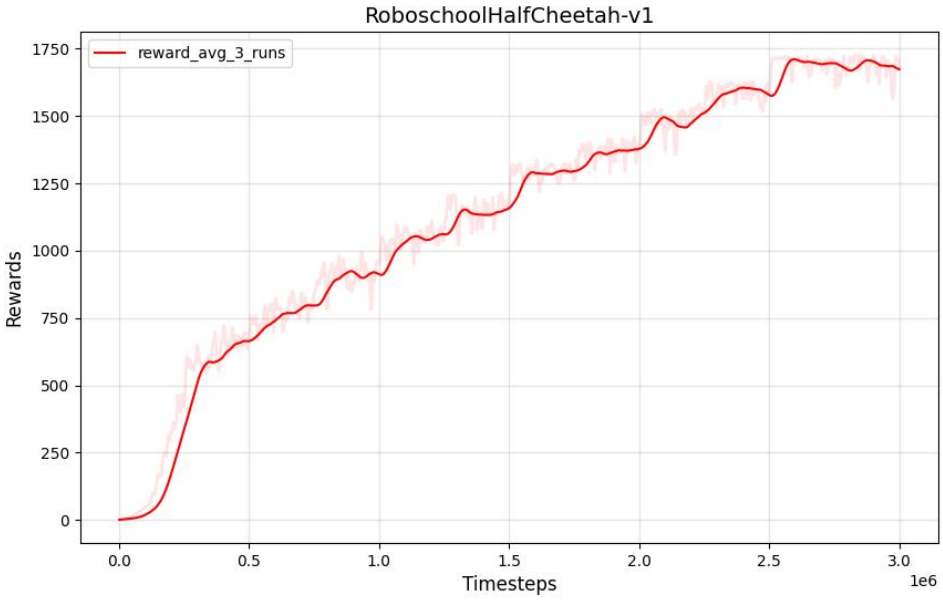
以上环境都可以在 OpenAI 官方网站中找到详细解释，环境预处理、动作空间、状态空间、奖励等都已经得到了良好的定义，这里就不进行重复阐述。在本实验

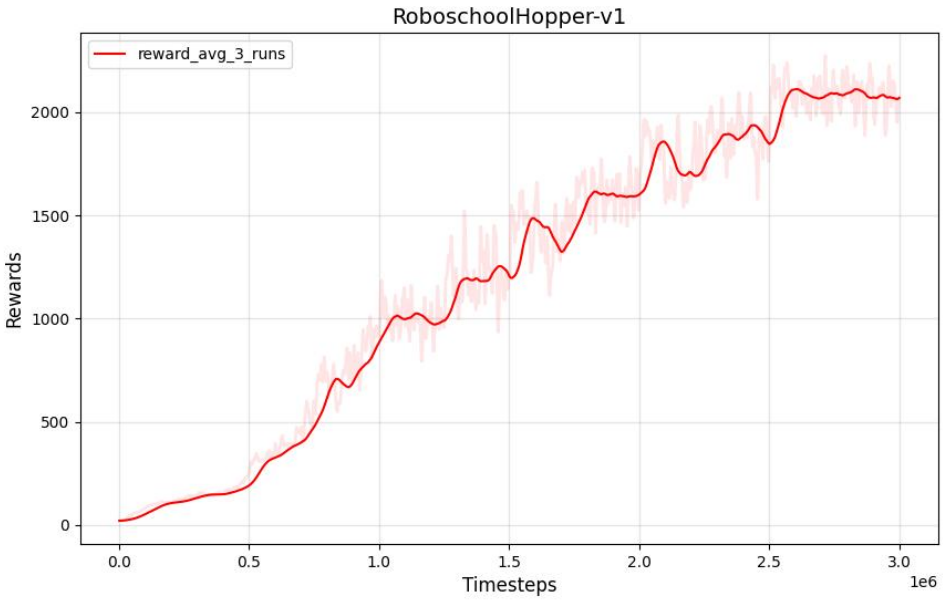
中也保存了每个游戏环境运行的 Gif 动图，可以在代码目录的 PPO_gifs 目录下查看每个游戏环境的 Gif 动图。

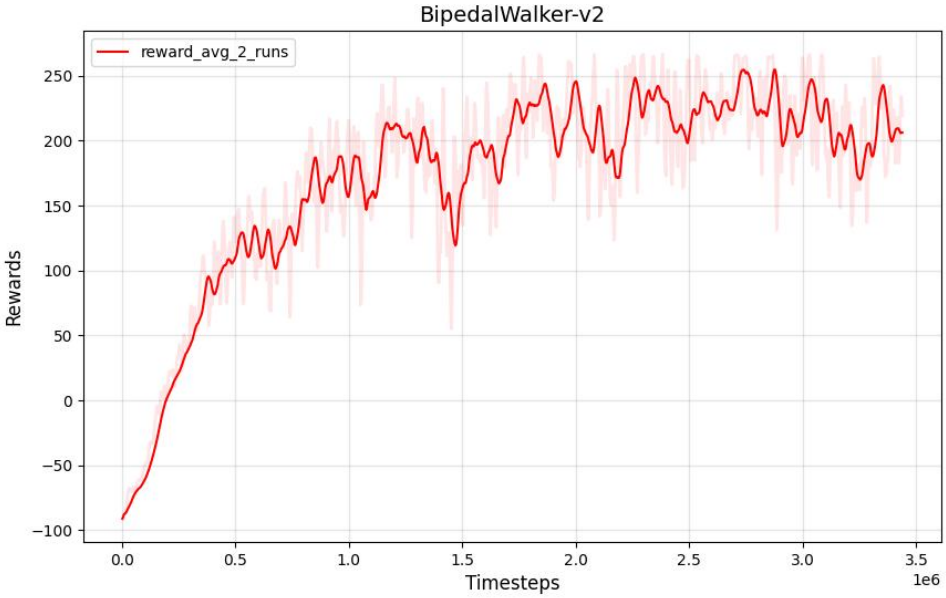
下面对代码进行一些说明。为了使训练过程简单：

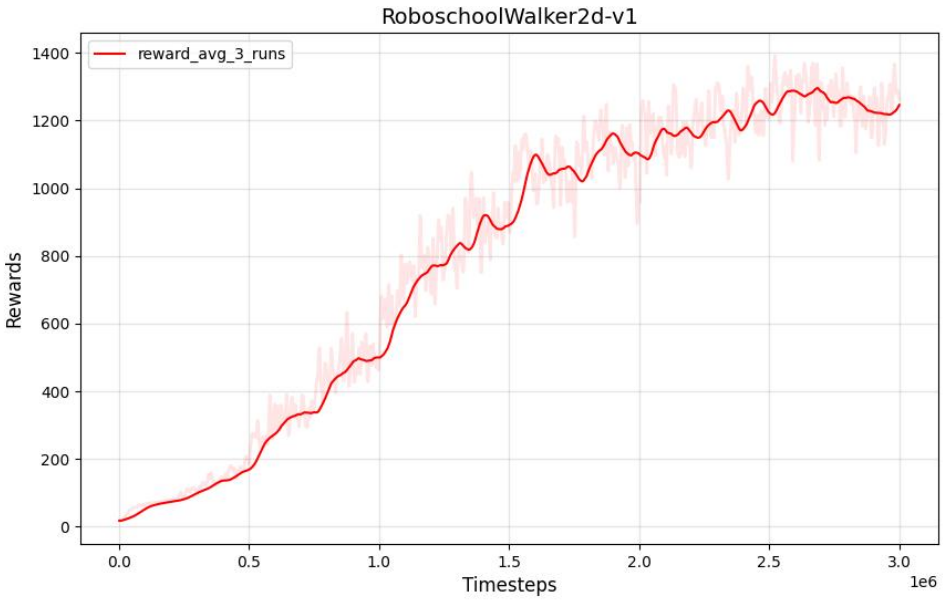
1. 代码中对于连续环境的输出动作分布（具有对角协方差矩阵的多元正态分布）具有恒定的标准偏差，即它是超参数而不是可训练参数。然而，它是线性衰减的。（action_std 显着影响性能）
2. 代码中使用简单的蒙特卡洛估计来计算优势，而不是广义优势估计（查看 OpenAI 旋转实现）。
3. 代码中是一个单线程实现，即只有一个 worker 收集 experience。

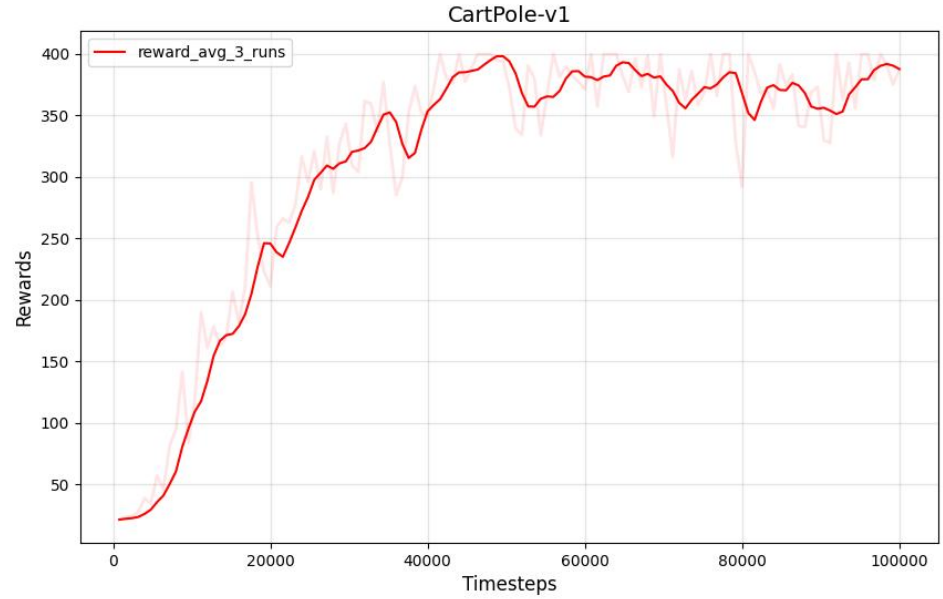
下面依次给出每个环境中的 PPO-clip 训练曲线：

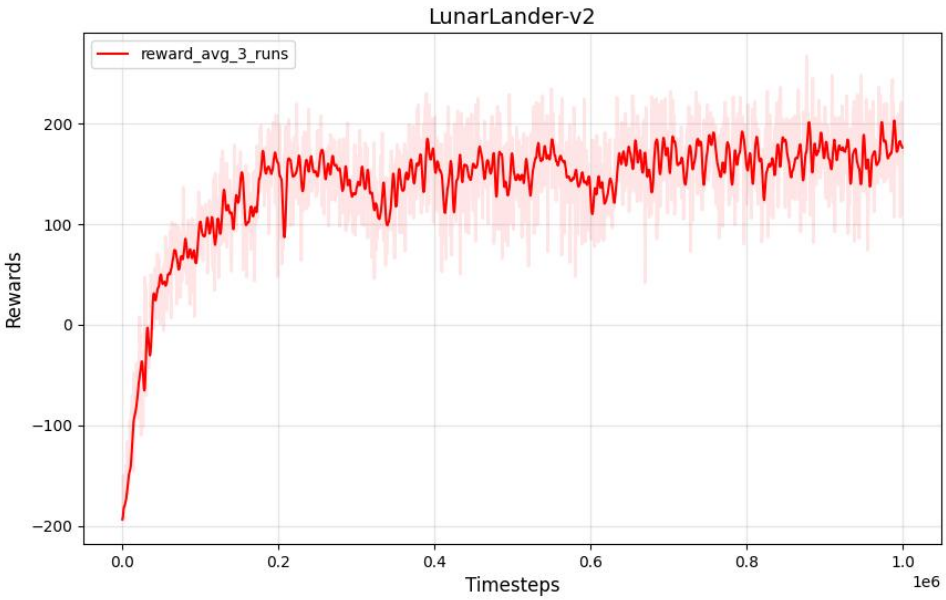



RoboschoolHopper-v1
reward_avg_3_runs
Rewards
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Timesteps
1e6


BipedalWalker-v2
reward_avg_2_runs
Rewards
250
200
150
100
50
0
−50
−100
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Timesteps
1e6


RoboschoolWalker2d-v1
reward_avg_3_runs
Rewards
Timesteps
1e6
0
200
400
600
800
1000
1200
1400
0.0
0.5
1.0
1.5
2.0
2.5
3.0


CartPole-v1
reward_avg_3_runs
Rewards
Timesteps
50
100
150
200
250
300
350
400
0
20000
40000
60000
80000
100000



结果分析：

在分析 PPO-clip 算法的训练曲线时，我们可以观察到其一些显著特性：

1. 相比于 DQN 等一些经典的 value-based 方法，PPO-clip 的表现更稳定，并没有出现训练过程中 reward 突然降到很低的情况，而这也正对应了 PPO 的一个显著优势：训练稳定，效果较好，这也是其成为工业界最广泛应用的强化学习算法之一的原因。
2. PPO-clip 对于 discrete 环境的学习能力相比于 continuous 环境表现受到了限制。在 LunarLander-v2 环境中，PPO 的性能曲线出现了较大的波动，这也代表着其较大的方差。