计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(4):177−186 [doi: 10.15888/j.cnki.csa.009033]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

双裁切近端策略优化算法①

张　骏[1,2], 王红成[1]

[1](东莞理工学院 电子工程与智能化学院, 东莞 523808)
[2](东莞理工学院 计算机科学与技术学院, 东莞 523808)
通信作者: 王红成, E-mail: wanghc@dgut.edu.cn

摘　要: 近端策略优化 (proximal policy optimization, PPO) 是一种稳定的深度强化学习算法, 该算法的关键点之一是使用裁切后的代理目标限制更新步长. 实验发现当使用经验最优的裁切系数时, KL 散度 (Kullback-Leibler divergence) 无法被确立上界, 这有悖于置信域优化理论. 本文提出一种改进的双裁切近端策略优化算法 (proximal policy optimization with double clipping boundaries, PPO-DC). 该算法通过基于概率的两段裁切边界调整 KL 散度, 将参数限制在置信域内, 以保证样本数据得到充分利用. 在多个连续控制任务中, PPO-DC 算法取得了好于其他算法的性能.

关键词: 强化学习; 策略梯度; 近端策略优化; 裁切机制

引用格式: 张骏,王红成.双裁切近端策略优化算法.计算机系统应用,2023,32(4):177–186. http://www.c-s-a.org.cn/1003-3254/9033.html

Proximal Policy Optimization with Double Clipping Boundaries

ZHANG Jun[1,2], WANG Hong-Cheng[1]

[1](School of Electrical Engineering and Intelligentization, Dongguan University of Technology, Dongguan 523808, China)
[2](School of Computer Science and Technology, Dongguan University of Technology, Dongguan 523808, China)

Abstract: Proximal policy optimization (PPO) is a stable deep reinforcement learning algorithm. The key process of the algorithm is to use clipped surrogate targets to limit step size updates. Experiments have found that when a clipping coefficient with optimal experience is employed, the upper bound of Kullback-Leibler (KL) divergence cannot be determined. This phenomenon is against the optimization theory of trust region. In this study, an improved PPO with double clipping boundaries (PPO-DC) algorithm is proposed. The algorithm adjusts the KL divergence based on two probability-based clipping boundaries and limits parameters to the trust region, so as to ensure that the sample data are fully utilized. In several continuous control tasks, the PPO-DC algorithm achieves better performance than other algorithms.

Key words: reinforcement learning; policy gradient (PG); proximal policy optimization (PPO); clipping mechanism

近年来, 深度强化学习算法被广泛应用于解决各类复杂问题. 2014 年, DeepMind 团队推出深度 Q 学习算法[1], 在 2 600 个雅达利游戏上实现了人类水平的控制. 2016 年, DeepMind 团队结合卷积神经网络[2], 蒙特卡洛树搜索[3] 和强化学习理论, 推出人工智能机器人 AlphaGo[4], 并在围棋项目上以 4:1 的比分击败了人类世界冠军李世石. 此后, AlphaGo Zero[5] 在不使用任何先验知识的前提下以 100:0 的比分击败 AlphaGo, 宣告了人工智能在围棋项目上的全面胜利. 2018 年, OpenAI 团队推出了 OpenAI Five[6], 在 Dota2 游戏传统的 5v5 模式下击败了两届世界冠军 OG 战队. 2019 年 DeepMind 团队基于多智能体强化学习推出 AlphaStar[7], 在

① 基金项目: 广东省普通高校重点科研平台和项目 (2020ZDZX3075)
收稿时间: 2022-08-23; 修改时间: 2022-09-27; 采用时间: 2022-10-21; csa 在线出版时间: 2022-12-23
CNKI 网络首发时间: 2023-02-26

星际争霸 2 游戏中达到了人类大师级别的水平, 并于该游戏的官方排名中超越了 99.8% 的人类玩家. 在上述困难项目中的成功表明, 未来针对深度强化学习领域的研究有着广阔的前景.

一般而言, 深度强化学习算法可分为值函数算法和策略梯度算法 (policy gradient, PG)[8]. 值函数算法对回报建模, 在每次迭代中按当前最优策略更新. 策略梯度算法对策略建模, 在每次迭代中小幅修正策略使其收敛. 策略梯度算法相较于值函数算法更为稳定, 但受限于样本使用率低的困境. 策略梯度算法中, 一类解决样本效率问题的算法是确定性策略梯度 (deterministic policy gradient, DPG)[9], 这类算法将策略网络由状态的函数修改为状态-动作对的函数, 并通过双价值网络和延迟学习提升稳定性[10,11]. 另一类算法在随机策略的基础上增加数据复用次数, 最有代表性的包括置信域优化算法 (trust region policy optimization, TRPO)[12] 和近端策略优化算法 (proximal policy optimization, PPO)[13]. TRPO 算法提出一种代理目标, 通过优化该代理目标并限制更新前后策略分布的 KL 散度 (Kullback-Leibler divergence) 实现优化过程. PPO 算法使用近似点优化替代 TRPO 算法中的置信域优化, 降低了计算过程的复杂性[14]. PPO 的具体实现算法包括 PPO-Clip 和 PPO-Penalty. 其中 PPO-Clip 在代理目标中引入了裁切方法, 在实际应用中适用性更广. 一些研究通过细化裁切理论改进 PPO-Clip 算法.

PPO-λ算法[15] 提出一种适应性的裁切机制, 该机制根据样本重要性改变适应系数, 并在此基础上反复优化策略. 实验证明该算法能在保证效率的基础上预防破坏性较大的更新. 出于对更新稳定性的考虑, 基于回落的近端策略优化算法 (PPO with rollback, PPO-RB)[16] 提出在样本的动作似然比例到达边界后使用一种回落操作, 回落操作能帮助算法限制新旧策略的差异. 在同一篇论文中作者提出了基于置信域的近端策略优化算法 (trust region-based PPO, TR-PPO), 将裁切边界的决定条件由动作似然比例替换成 KL 散度. 平滑的近端策略优化算法 (PPO smooth, PPO-S)[17] 建立在 PPO-RB 的基础上, 该算法修正了裁切边界, 使用 tanh 函数处理回落项以避免极端采样影响算法稳定性.

早期停止的近端策略优化算法 (PPO with early stopping)[18] 和使用衰减裁切边界的近端策略优化算法 (PPO with decaying clipping range)[19] 在注重保留 PPO-Clip 算法简洁性的同时提升性能. 早期停止算法选择在 KL 散度首次到达边界的时候停止更新, 直接进入下一轮采样. 而边界衰减算法在整个更新过程中不断收窄裁切边界, 从而保证算法的稳定性.

PPO-Clip 算法的理论部分基于 TRPO 算法, 但是基于近似点的优化并不能保证 TRPO 算法中提出的置信域. 反过来, 如果直接将 TRPO 优化目标中的置信域约束条件直接引入则会使 PPO-Clip 算法失去简洁性. 一个兼顾两者的方案是赋予裁切边界一定的灵活度, 并通过对裁切边界的调整将参数约束在置信域范围中. 基于以上观点, 本文提出双裁切近端策略优化算法 (proximal policy optimization with double clipping boundaries, PPO-DC). 该算法继承了 PPO-Clip 算法的主体, 不同的是 PPO 算法使用单一且固定的裁切边界, PPO-DC 算法使用双裁切边界. 在一次更新中, 每个样本依概率选择其中一个裁切边界, 该概率值是一个正态分布累计分布函数的输出. 而该累计分布函数的输入为采样簇上的平均 KL 散度的对数. PPO-DC 的理论部分基于 TRPO 论文提出的置信域优化目标, 其基本思路是将当前策略网络参数尽可能地置于置信域边界, 以保证样本数据得到充分利用. 而使用双裁切的一个优势在于, 算法可以通过选择任意一条边界自适应的调整 KL 散度, 并且依概率的决策方式可以保证以上调整的平滑性.

本文的第 1 节介绍了策略梯度算法的相关知识以及一些基于 PPO 裁切方法的研究. 第 2 节通过一些小规模实验探讨了裁切机制的作用原理, 并在此基础上提出 PPO-DC 算法. 第 3 节使用 OpenAI 的 Gym 作为测试环境[20], 在 Mujoco[21] 和 Box2d 的多个连续控制任务中对比了多个算法的性能. 实验结果表明, PPO-DC 算法拥有较好的性能.

1 相关工作

1.1 马尔可夫决策过程

强化学习中, 常使用有限马尔可夫决策过程对连续控制任务建模. 一个完整的马尔科夫决策过程包括状态空间s, 动作空间a, 折扣系数γ, 奖励函数$r(s)$, 以及策略$\pi(a|s)$. 其中状态是智能体收到的当前环境特征, 动作是智能体根据当前环境特征做出的决策, 奖励是环境在到达下一个状态后反馈的回报信号, 策略是智能体对环境状态的反馈过程. 在完全观测的马尔可夫决策过程下, 环境状态的转移只与当前状态与动作相关, 而与之前状态与动作无关.

1.2 策略梯度

深度强化学习算法一般分为值函数算法和策略梯度算法[8]. 值函数方法中, 智能体对累计回报建立神经网络模型, 并在决策中根据贪心参数选择是否试探. 在试探决策中选择随机动作, 在非试探决策中选择值函数最大动作, 最后根据反馈更新值函数模型. 在实际应用中, 值函数方法只能处理动作空间离散且低维的情况.

在策略梯度方法中, 智能体对决策方案建立神经网络模型, 根据输出概率分布选择动作, 在收到环境反馈的序列后, 通过贝尔曼方程计算各节点蒙特卡洛回报值, 最后使用策略梯度方法更新决策函数模型. 策略梯度的表达式如下:

$$\nabla J(\theta)=\sum Q_{s_t,a_t}\nabla_\theta \log \pi_\theta(a_t|s_t) \tag{1}$$

其中, J是优化目标, Q_{s_t,a_t}是t时刻状态-动作对下的期望回报, s是t时刻状态, a是t时刻动作, π_θ是当前策略, θ是策略网络参数. 在原始策略梯度算法中, Q_{s_t,a_t}通过蒙特卡洛法计算. 另一类算法使用神经网络预测状态-动作价值, 该框架称为行动器-评判器 (actor-critic, AC). 在 AC 框架中, 评判器负责对环境状态建立评价体系, 行动器根据当前评价体系调整策略, 两者参数交替更新, 层层递进, 从而实现回报最大化. 基于其实用性与稳定性, 目前的策略梯度方法都采用了该框架.

1.3 近端策略优化

策略梯度算法是单步更新的, 每一次迭代都会淘汰用过的样本, 这是因为策略梯度定理中的优化目标建立在目标策略的状态分布上. 而改进这一点需要把优化目标建立在当前或过去策略的状态分布上, 故有策略改进下界:

$$\begin{cases} J(\pi)-J(\pi_k)\geqslant \dfrac{1}{1-\gamma}\underset{(s,a)\sim d^{\pi_k}}{E}\left[\dfrac{\pi(a|s)}{\pi_k(a|s)}A^{\pi_k}(s,a)\right] \\ \qquad\qquad -\dfrac{2\gamma C^{\pi,\pi_k}}{(1-\gamma)^2}\underset{s\sim d^{\pi_k}}{E}\left[TV(\pi,\pi_k)(s)\right] \\ C^{\pi,\pi_k}=\max\limits_{s\in S}\left|E_{a\sim\pi(\cdot|s)}[A^{\pi_k}(s,a)]\right| \end{cases} \tag{2}$$

其中, π是目标策略, π_k是当前策略, d^{π_k}是当前策略下状态与动作遵循的轨迹, A^{π_k}是当前策略下的优势函数, $TV(\pi,\pi_k)$是当前策略分布与目标策略分布间的 TV 散度. 置信域优化算法 TRPO 根据式 (2) 推导出优化目标:

$$\begin{cases} \max\limits_\theta\ E_{s\sim\rho\theta_{\text{old}},a\sim q}\left[\dfrac{\pi_\theta(a|s)}{q(a|s)}Q_{\theta_{\text{old}}}(s,a)\right] \\ \text{subject to } E_{s\sim\rho\theta_{\text{old}}}\left[D_{\text{KL}}\left(\left(\pi_{\theta_{\text{old}}}\cdot|s\right)\|\left(\pi_\theta\cdot|s\right)\right)\right]\leqslant\delta \end{cases} \tag{3}$$

其中, ρ是平行环境, q是上一次采样遵循的策略, D_{KL}是 KL 散度, δ是控制 KL 散度的超参数. 该方法通过限制 KL 散度提高样本使用并保证了更新下界. 在训练时, TRPO 算法使用自然梯度下降[22], 通过计算参数空间的费雪信息矩阵优化目标. PPO-Clip 算法对此做出了改进, 通过裁切动作似然比例实现原式中对 KL 散度的约束:

$$L^{\text{CLIP}}(\theta)=E\left[\min\left(r_t(\theta)A_t,(r_t(\theta),1-\varepsilon,1+\varepsilon)A_t\right)\right] \tag{4}$$

其中, r_t是t时刻动作似然比例, A_t是t时刻优势函数, ε是裁切系数. 其中动作似然比例$r_t(\theta)$是指更新前后策略选择当前动作的概率$\pi_{\theta_{\text{old}}}(a|s)$与$\pi_\theta(a|s)$的比值. 计算优势函数时, 算法使用广义优势函数估计器 GAE[23]:

$$\begin{cases} \hat{A}_t=\delta_t+(\gamma\lambda)\delta_{t+1}+\cdots+(\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \delta_t=r_t+\gamma V(s_{t+1})-V(s_t) \end{cases} \tag{5}$$

其中, T是当前策略运行步数, $V(s_t)$是t时刻状态下的期望回报, γ是奖励折扣率, λ是优势函数折扣率. $\lambda=0$对应强化学习中的时序差分法, $\lambda=1$对应蒙特卡洛法.

在实际训练中, $V(s_t)$由价值网络预测, 该网络是实时更新的. 为提升训练速度, 算法同时运行 N 个平行环境, 在每个平行环境中运行 T 步, 收集到 NT 步的样本后通过 GAE 计算优势函数A_t. 样本打乱后, 每次抽取大小为 b 的批次, 再使用上式计算梯度并更新. 损失函数将两者联立, 并引入信息熵作为正则项:

$$L_t^{\text{CLIP+VF+}S}(\theta)=\hat{E}_t[L_t^{\text{CLIP}}(\theta)-c_1L_t^{\text{VF}}(\theta)+c_2S[\pi_\theta](s_t)] \tag{6}$$

其中, $L_t^{\text{CLIP+VF+}S}$是t时刻总损失函数, L_t^{CLIP}是t时刻策略网络损失函数, L_t^{VF}是t时刻价值网络损失函数, S是动作输出的平均信息熵, c_1和c_2是控制损失函数比例的超参数. L_t^{VF}由均方差计算:

$$\begin{cases} L_t^{\text{VF}}=(V_\theta(s_t)-V_t^{\text{targ}})^2 \\ V_t^{\text{targ}}=r_t+\gamma V_\theta(s_{t+1}) \end{cases} \tag{7}$$

其中, V_t^{targ}是价值网络目标, r_t是t时刻奖励. 在一轮更新内, PPO-Clip 算法会反复重用样本, 因此该算法的样本效率高于原始策略梯度下降. 但由于近端策略优化是一种在线算法, 当完成一轮更新后, 算法会丢弃所有用过的样本, 因此该算法的样本效率低于离线方法, 如 TD3[11]、SAC[24] 等.

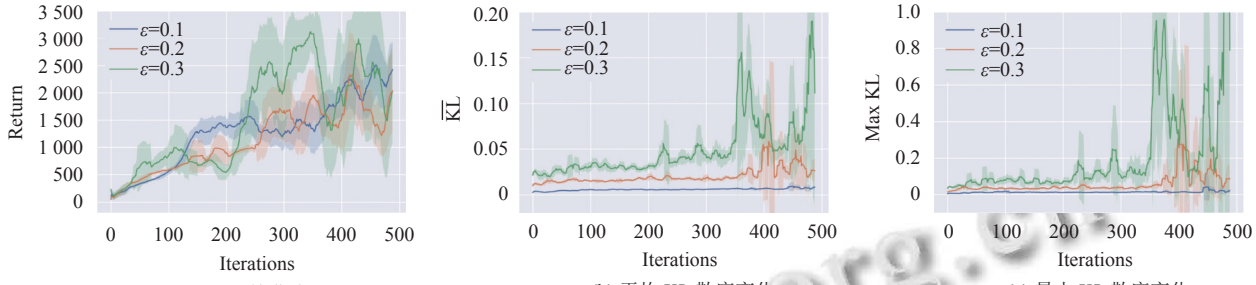
2 PPO-DC 算法

2.1 对裁切机制的研究

PPO-Clip 算法通过引入裁切机制完成 TRPO 的优

化目标中对 KL 散度的约束, 但是论文没有在理论上给出裁切的作用原理, 本节内容将从实验的角度对此进行探讨.

图 1(a) 对比了使用不同裁切系数的 PPO-Clip 算法在 Hopper 任务上的训练曲线. 随着裁切系数的增加, 训练曲线变得更不稳定. 从第 1.3 节中式 (2) 的策略改进下界可知保持训练稳定的前提是将更新前后的策略分布间距限制在一定范围内, 该间距可通过 KL 散度衡量.



(a) 训练曲线 (b) 平均 KL 散度变化 (c) 最大 KL 散度变化

图 1 使用不同裁切系数的 PPO-Clip 算法在 Hopper 任务上的训练结果

图 1(b) 和图 1(c) 分别对比了训练过程中的 KL 散度的平均值和最大值. 可以看到$\varepsilon=0.1$时的 KL 散度被严格限制, $\varepsilon=0.2$时的 KL 散度在训练的初期相对稳定, 但是在后期变得不可控, $\varepsilon=0.3$时的 KL 散度几乎是发散的. 注意到$\varepsilon=0.2$也是 PPO 论文中给出的经验最优值, 该取值被认为平衡了样本效率和训练稳定性.

以上实验得出的一个结论是, 使用经验较优的裁切系数$\varepsilon=0.2$的 PPO-Clip 算法 KL 散度波动较剧烈, 根据策略改进下界的定义, 该情况可能导致破坏性的迭代, 而破坏性的迭代是训练的不稳定因素之一. 第 3.4 节的多组对比实验同样验证了以上结论.

2.2 PPO-DC 算法思想

从第 2.1 节实验可知较宽松的裁切系数只对 KL 散度有限制性, 却不具备严格定义其上界的能力. 而式 (3) 明确了约束 KL 散度是稳定优化过程的前提, 这也是置信域优化算法的核心思想. 改进 PPO 算法的一个思路是使裁切方法实现对 KL 散度的严格约束, 同时尽可能保留算法对样本的重复利用率. 图 1(b) 表明收窄裁切边界会对 KL 散度进行更严格的限制, 因此一个直观的想法是动态调整裁切边界, 使 KL 散度保持在合理范围内. 该范围下的参数须满足两个条件: 一是落在置信域内, 二是尽量靠近置信域边界. 前者是置信域优化的核心, 后者是提升样本利用率的核心.

PPO-DC 算法基于以上思路被提出. 该算法确定两个裁切边界, 按概率选择其中一个使用. 上述概率随当前状态下的 KL 散度动态调整, 目的是将 KL 散度限制在一个合理范围内. 两段裁切系数虽然是超参数, 但是它们应具有以下特点: 内边界应当严格限制更新幅度, 侧重于保证置信域; 外边界应当放宽更新幅度, 侧重于提升样本利用率.

2.3 PPO-DC 算法内容

PPO-DC 算法使用双裁切系数, 生成两段裁切边界. 若 KL 散度偏高, 则增加样本被内边界裁切的概率, 减小被外边界裁切的概率, 若偏低则反之. 算法的总体优化目标如下:

$$L^{\text{Dclip}}(\theta)=E_{\substack{(s_t,a_t)\sim\tau,\\ \varepsilon_t\sim D_m}}[\min(r_t(\theta)A_t, clip(r_t(\theta),1-\varepsilon_t,1+\varepsilon_t)A_t)] \tag{8}$$

其中, L^{Dclip}是 PPO-DC 算法的代理目标, τ是采样轨迹, m 是样本簇上的训练轮数, ε_t是单样本的裁切边界. ε_t服从以下二元分布D_m, 该分布随训练轮数的迭代而变化:

$$\begin{cases} P\{\varepsilon_t=\varepsilon_L\}=p_m \\ P\{\varepsilon_t=\varepsilon_U\}=1-p_m \end{cases} \quad (0\leqslant p_m\leqslant 1) \tag{9}$$

其中, 超参数ε_L代表内边界, ε_U代表外边界. 参数p_m来自一个正态分布的累计分布函数的输出. 具体的, 设随机变量 X 服从均值为μ, 标准差为σ的正态分布 N:

$$X\sim N(\mu,\sigma) \tag{10}$$

其中, 超参数μ和σ决定了该正态分布的形态. 参数p_m即为该正态分布的累计分布函数的输出, 而输入是更新前后策略分布的 KL 散度在当前样本簇上的期望的对数形式:

$$\begin{aligned} p_m=&F(x_m)=P(X\leqslant x_m)=\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{x_m}\mathrm{e}^{-\frac{(t-\mu)^2}{2\sigma^2}}dt \\ &x_m\to\ln[E_{s\sim\tau}[D_{\text{KL}}(\pi_0,\pi_m)]+\omega] \end{aligned} \tag{11}$$

其中, π_0和π_m分别是初始时刻的策略和第 m 轮迭代后的策略. 输入采用对数形式是考虑到 KL 散度自身的

性质, 一方面 KL 散度的取值是非负数, 另一方面 KL 散度与分布离散程度并非呈线性关系. ω是一个略大于 0 的数, 它的作用是防止对数函数的输入取到零值. 图 2 为不同e^{μ}, σ取值下期望 KL 散度到二元分布参数 p 的映射图像.

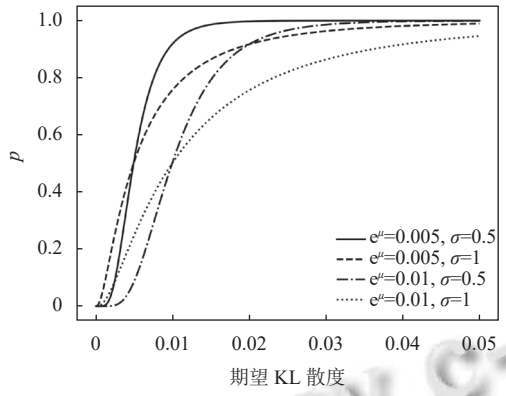


图 2 不同e^{μ}, σ取值下期望 KL 散度到二元分布参数 p 的映射图像

式 (11) 中期望 KL 散度的表达式如下:

$$E_{s\sim\tau}[D_{\mathrm{KL}}(\pi_0,\pi_m)]=E_{s\sim\tau,a\sim\pi_0}\left[\log\frac{\pi_0(a|s)}{\pi_m(a|s)}\right] \tag{12}$$

式 (12) 的计算有两种方法, 一种是在采样簇上对每个样本的动作分布积分后求均值:

$$E_{s\sim\tau,a\sim\pi_0}\left[\log\frac{\pi_0(a|s)}{\pi_m(a|s)}\right]=\frac{1}{T}\sum_{t=0}^{T-1}\int_{a_{\mathrm{low}}}^{a_{\mathrm{high}}}\log\frac{\pi_0(u|s_t)}{\pi_m(u|s_t)}du \tag{13}$$

其中, 超参数 T 为单轮更新采样步数, a_{high}和a_{low}是环境给出的动作边界. 该方法获得当前采样簇上的精确解. 另一种是蒙特卡洛法:

$$E_{s\sim\tau,a\sim\pi_0}\left[\log\frac{\pi_0(a|s)}{\pi_m(a|s)}\right]\approx\frac{1}{TN}\sum_{t=0}^{T-1}\sum_{\substack{n=0,\\ a_{t,n}\sim\pi_0(\cdot|s_t)}}^{N-1}\left[\log\frac{\pi_0(a_{t,n}|s_t)}{\pi_m(a_{t,n}|s_t)}\right] \tag{14}$$

其中, 超参数 N 是动作抽样次数, 通常根据计算资源决定. 蒙特卡洛法获得当前采样簇上的近似解, 使用该方法的好处在于能够通过超参数自由平衡计算资源与计算精度.

PPO-DC 算法的具体流程见算法 1. 该算法引入 4 个超参数, 分别是式 (9) 中的双裁切系数ε_L和ε_U, 以及式 (10) 中决定正态分布形态的μ和σ. μ通过e^{μ}决定了 KL 散度的预期值, 算法会使 KL 散度向该预期值收束, σ则决定了该收束过程的剧烈程度.

算法1. PPO-DC

1. 初始化策略网络π_θ, 价值网络ϕ_ω, 初始化采样长度T, 批次大小b, 训练轮数M;
2. while未触发终止条件:
3. 按策略π_θ从环境采样T步, 生成采样簇S;
4. for m=0 to MT/b:
5. 按式(13)或(14)计算KL散度D_{KL};
6. 按式(11)计算参数p_m;
7. 采样大小为b的小批次;
8. 按式(9)为小批次中每个样本s_t采样边界ε_t;
9. 按式(5)计算广义优势函数估计值$\hat{A}$;
10. 按式(8)计算策略梯度$\nabla L^{\mathrm{Dclip}}$;
11. 使用Adam优化器更新策略网络π_θ与价值网络ϕ_ω;
12. end
13. end

3 实验与结果分析

3.1 实验环境

本文使用 OpenAI 的强化学习环境库 Gym 作为实验平台, 在标准控制任务和 Box2d 环境中对比了几种算法的性能. 测试采用 PyTorch 作为深度学习框架, 并选择 ElegantRL[25] 作为强化学习基线.

第 3.4 节实验在 Mujoco 环境上进行, 包括 5 个控制任务 Walker2d, Swimmer, Ant, Hopper, Half-Cheetah. Mujoco 是 DeepMind 公司开发的物理仿真引擎. 以 HalfCheetah 任务为例, 智能体接收 17 维的状态输入, 并输出 6 维动作控制仿真机器人的不同关节. 算法在二维仿真环境内训练双足机器人前进, 训练得分与前进速度成正比. 图 3 为可视化的 Mujoco 环境.

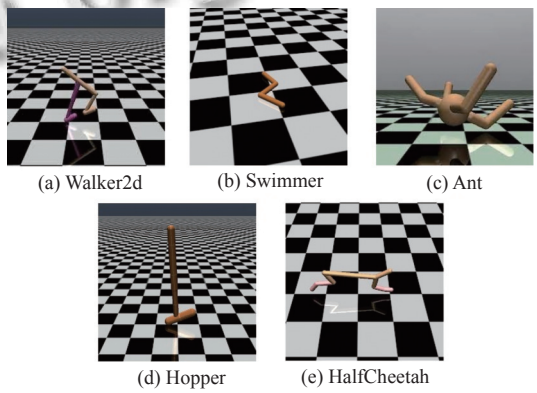


图 3 Mujoco 引擎上的连续控制任务

第 3.5 节的实验在 Box2d 环境上进行, 环境包括 2 个连续控制任务 BipedalWalker 和 LunarLander. 在 BipedalWalker 任务中, 智能体接收 24 维状态并输出 6 维动作, 以此控制一个双足机器人前进, 并被要求能

够行走得更为快速和稳健. 该环境被认为是连续控制任务里较难的一个. LunarLander 任务则模拟了宇宙飞船在月球的着陆过程. 智能体接收 8 维状态并输出 2 维动作, 分别控制两个不同位置的推进器. 训练的总体目标为调整宇宙飞船双足平稳着陆在目标区域内, 并在接触地面的瞬间速度小于一个阈值. 若飞船顺利着陆环境将反馈一个+100 的奖励, 反之若撞毁则反馈一个−100 的奖励. 图 4 为可视化的 Box2d 环境.

(a) BipedalWalker (b) LunarLander

图 4 Box2d 环境上的连续控制任务

本节实验的评价方法如下: 在 Mujoco 和 BipedalWalker 任务中, 通过计算所有幕的平均回报评价算法的训练速度, 计算最后 100 幕的平均回报评价算法的最终性能. 在 LunarLander 任务中, 由于飞船完成着陆后的回报基本恒定, 故通过计算首次成功着陆的采样轮数评价训练速度.

3.2 超参数

PPO-DC 算法主干部分与 PPO-Clip 算法一致, 因此继承了原算法的主要超参数. 表 1 列出了通用超参数的取值, 这些取值同时被应用于对比算法中. 除此以外 PPO-DC 算法引入了 4 个超参数, 其中ε_L和ε_U控制着内外裁切边界, μ和σ是正态分布的期望和标准差.

表 2 通用超参数取值表

超参数名称	超参数值	
	Mujoco环境	Box2D环境
优化器	Adam	Adam
学习率	3×10^{-4}	3×10^{-4}
折扣率	0.99	0.99
采样步数	2048	4096
批次大小	64	256
训练轮数	8	8
GAE系数	0.95	0.95
信息熵系数	0.01	0.01

两段裁切系数ε_L和ε_U应具有以下特点: 内边界ε_L应当严格限制 KL 散度, 侧重于保证置信域, 外边界ε_U应当放宽更新幅度, 侧重于提升样本利用率. 第 2.1 节实验表明$\varepsilon=0.1$的情形下 KL 散度是被严格约束的, 但是$\varepsilon=0.2$的情形下 KL 散度的上界无法被确立. 考虑到$\varepsilon=0.2$是 PPO-Clip 算法的经验最优值, 初步认为使用$\varepsilon_L=0.1$作为内边界, $\varepsilon_U=0.2$作为外边界是合理的选择. μ通过e^{μ}决定了 KL 散度的预期值, 算法会使 KL 散度向该预期值收束, 该参数通过小规模对比实验确定. σ调整了收束的剧烈程度, 该参数被固定为 0. 8.

表 2 对比了超参数e^{μ}的不同取值, 评价分数来自标准化后的 4 个 Gym 测试环境的得分总和. 综合评价最好的结果来自于e^{μ}取 0.005 的情况, 因此后文实验将在该组参数下运行.

表 3 超参数e^{μ}评价表

超参数名称				综合评价
ε_L	ε_U	e^{μ}	σ	
0.1	0.2	0.005	0.8	**3.54**
0.1	0.2	0.01	0.8	3.24
0.1	0.2	0.02	0.8	3.25
0.1	0.2	0.03	0.8	3.05

3.3 参与对比的算法

参与对比的算法除 PPO-DC 与 PPO-Clip[9] 外, 还包括 PPO-Penalty[9], PPO-S[13] 以及 PPO-Decay[15]. 一些算法引入了新的超参数, 本节内容将介绍这些算法的核心内容并给出超参数的取值.

3.3.1 PPO-Penalty

PPO-Penalty 是将 KL 散度作为罚项引入优化目标的方法, 优化目标如下:

$$L^{\text{penalty}}(\theta)=E\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t-\beta KL\left[\pi_{\theta_{\text{old}}}(\cdot|s_t),\pi_\theta(\cdot|s_t)\right]\right] \tag{15}$$

其中, 超参数β控制了罚项的权重, 实验中取$\beta=3$.

3.3.2 PPO-S

PPO-S 是 PPO-RB 的一种改进算法, 其裁切方法如下:

$$ratio^{\text{PPOS}}=\begin{cases}-\alpha\tanh(r_{s,a}(\pi)-1)+1+\varepsilon+\alpha\tanh(\varepsilon), & r_{s,a}(\pi)\leqslant 1-\varepsilon\\ -\alpha\tanh(r_{s,a}(\pi)-1)+1-\varepsilon+\alpha\tanh(\varepsilon), & r_{s,a}(\pi)\geqslant 1+\varepsilon\\ r_{s,a}(\pi), & \text{otherwise}\end{cases} \tag{16}$$

其中, α是决定回落项权重的超参数, 实验中取$\alpha=0.3$.

3.3.3 PPO-Decay

PPO-Decay 则是一种在单轮训练中不断衰减裁切边界的算法. 其边界衰减方式有两种, 分别为线性衰减与指数衰减. 对比实验选择线性衰减方式, 其表达式如下:

$$\varepsilon_t = \frac{T-t}{T}\varepsilon_0 \tag{17}$$

其中, T 是单轮更新采样步数, t 是当前更新步数, ε_0是初始边界, 实验中取$\varepsilon_0 = 0.3$.

3.4 不同算法在 Mujoco 环境上的对比实验

本节实验对比了不同算法在 Mujoco 环境上的运行结果, 环境包括 5 个标准控制任务 Walker2d, Swimmer, Ant, Hopper, HalfCheetah. 由于 Ant 是三维控制环境, 接收 111 维状态并输出 8 维动作, 学习难度较高, 故训练在 Ant 任务上运行 2E+6 步, 在其余任务上运行 1E+6 步. 评价在完成一轮更新后进行, 选择策略网络输出分布的均值作为每步动作, 在完成一整幕后计算总回报, 取 5 次实验的平均值作为结果. 训练曲线见图 5. 通过计算所有幕的平均回报评价算法的训练速度, 计算最后 100 幕的平均回报评价算法的最终性能. 评价数据见表 3.

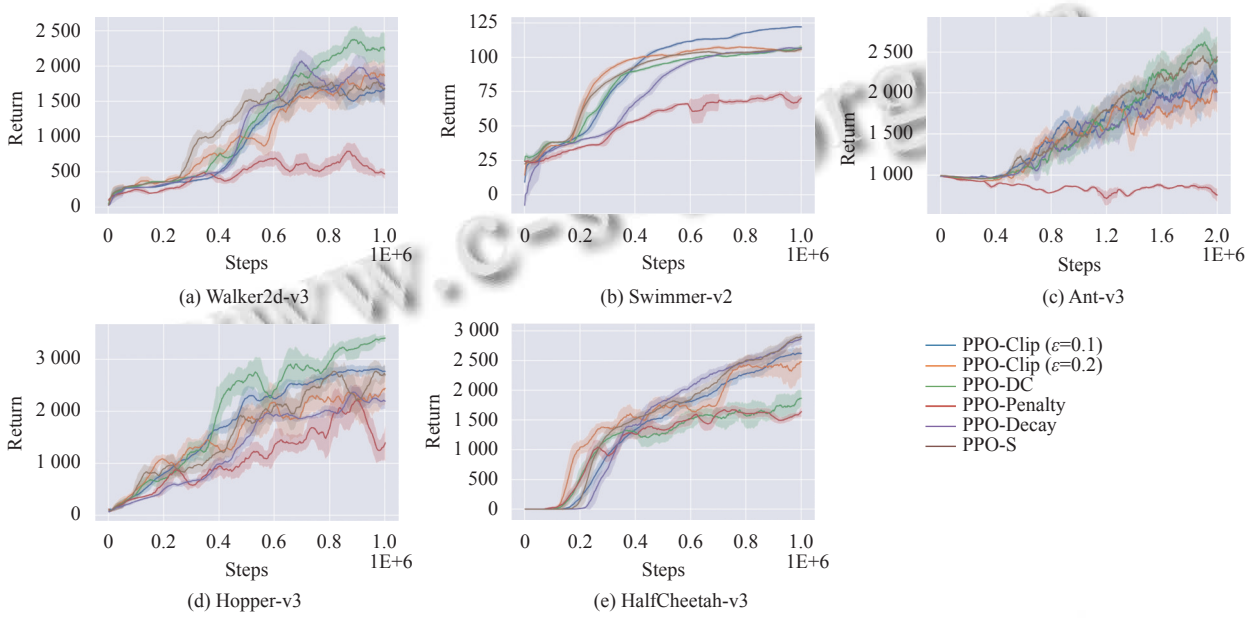


图 5 不同算法在 5 个 Mujoco 任务上的训练曲线

首先, 无论采用两种指标的哪一种评价, PPO-DC 算法在 Walker2d 和 Hopper 两个任务下的分数都是 6 个对比算法中最高的. 尤其是在 Hopper 任务中, 该算法的得分远超其他算法. 而在 Ant 任务中, PPO-DC 算法展现了优秀的最终性能, 并在 180 万步后超越了其余所有算法. 在 Swimmer 任务中, 最好的结果来自$\varepsilon = 0.1$时的 PPO 算法. 这可能是因为大多数对比算法默认采用$\varepsilon = 0.2$作为裁切系数, 而该任务更适合严格的裁切系数. 在 HalfCheetah 任务中, PPO-DC 算法没有展示出好的性能, 得分仅优于 PPO-Penalty 算法, 该任务下得分较高的算法是 PPO-Decay 和 PPO-S. 综合所有环境下的表现, PPO-DC 算法的性能是相对最优的.

其次, PPO-DC 与 PPO-Clip 的对比能帮助我们判断双裁切是否真正有效, 因为 PPO-DC 依概率采用$\varepsilon_L = 0.1$, $\varepsilon_U = 0.2$作为双裁切系数, 而 PPO-Clip 分别采用$\varepsilon = 0.1$和$\varepsilon = 0.2$作为单裁切系数, 所以前后二者的比较可以说明双裁切机制是否真正有助于智能体训练. 从实验结果来看, PPO-DC 在 Walker2d, Ant, Hopper 这 3 类任务下的表现明显优于两个使用单裁切系数的 PPO-Clip, 仅在 HalfCheetah 任务下明显劣势, 因此双裁切机制是有效的.

另一个观察角度是置信域. 图 6 对比了训练过程中平均 KL 散度的变化, 参与对比的算法包括 PPO-Clip 和 PPO-DC. PPO-Clip 算法采用$\varepsilon = 0.1$和$\varepsilon = 0.2$作为裁切系数, 而 PPO-DC 算法采用$\varepsilon_L = 0.1$, $\varepsilon_U = 0.2$作为内外裁切边界, 并对比不同超参数e^{μ}的取值. 可以看到 PPO-Clip 算法在$\varepsilon = 0.2$时的置信域是无法得到保证的, 尤其是在训练的末期, KL 散度的上界几乎无法判断, 这与第 2.1 节的实验相对应. 而$\varepsilon = 0.1$时 KL 散度的波动范围较小, 置信域得到保证, 但是该情况下更新幅度又过于保守. 相较而言, PPO-DC 算法在超参数$e^{\mu} \leqslant 0.01$时 KL 散度的波动范围大致确定, 样本依然有概率得到宽松的裁切, 而 PPO-Clip 无法做到这一点.

Walker2d, Hopper, HalfCheetah 这 3 个任务的图像较好的说明了以上性质. 可以看到e^{μ} = 0.005 (绿线) 的曲线几乎没有出现波峰, e^{μ} = 0.01 (红线) 的曲线仅在 HalfCheetah 任务中出现一段波峰, 而二者都位于ε = 0.1 (蓝线) 的曲线之上. 这说明当e^{μ}设置合理, PPO-DC 算法将比 PPO-Clip 算法更有效率地使用样本.

表 4 Mujoco 环境下的对比实验结果

评价指标	算法	Mujoco环境				
		Walker2d	Swimmer	Ant	Hopper	HalfCheetah
所有幕平均回报	PPO-Clip (0.1)	995	**90**	1 543	1854	1 457
	PPO-Clip (0.2)	1 053	88	1 451	1 614	1 587
	PPO-DC	**1 232**	83	1 576	**2 103**	1 196
	PPO-Penalty	467	54	853	1 123	1 170
	PPO-Decay	1 115	75	1 476	1 358	1 566
	PPO-S	1 182	86	**1 609**	1 672	**1 604**
最后100幕平均回报	PPO-Clip (0.1)	1 620	**120**	2 099	2 772	2 504
	PPO-Clip (0.2)	1803	105	1926	2 296	2 423
	PPO-DC	**2 257**	104	**2 504**	**3 287**	1 719
	PPO-Penalty	599	70	827	1 798	1 585
	PPO-Decay	1812	105	2 112	2 189	2 704
	PPO-S	1 660	104	2 386	2 542	**2 712**

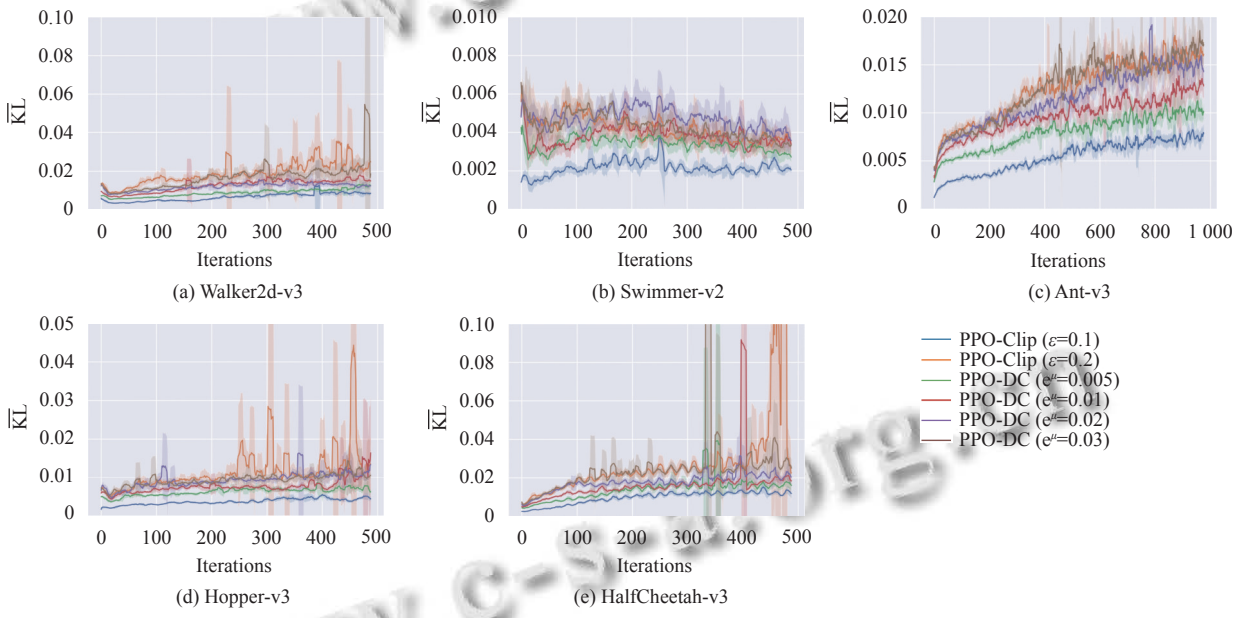


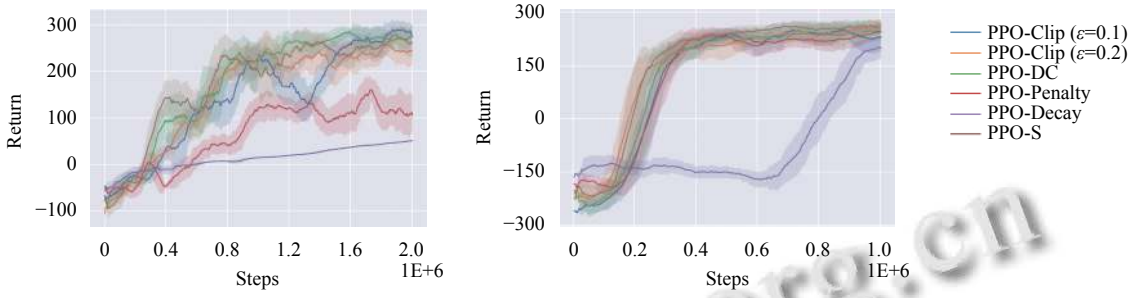
图 6 PPO-Clip 和 PPO-DC 算法在 5 个 Mujoco 任务上的平均 KL 散度

3.5 不同算法在 Box2d 任务上的对比实验

本节实验对比了不同算法在 Box2d 任务上的运行结果, 环境包括 2 个连续控制任务 BipedalWalker 和 LunarLander. 训练在 BipedalWalker 任务上运行 2E+6 步, 在 LunarLander 上运行 1E+6 步. 评价在完成一轮更新后进行, 选择策略网络输出分布的均值作为每步动作, 在完成一整幕后计算总回报, 取 5 次实验的平均值作为结果. 训练曲线见图 7. 由于 LunarLander 任务目标为宇宙飞船稳定着陆, 故使用首次成功着陆时的采样轮数作为评价指标. 评价数据见表 4.

BipedalWalker 任务中表现较好的算法包括 PPO-DC 和 PPO-S. 虽然 PPO-Clip 在ε = 0.1的取值下获得了最优的最终性能, 但在整个训练过程中表现的极不稳定, 而 PPO-DC 算法在该任务上的训练速度和最终性能都是最好的. 在 LunarLander 任务中, 表现较好的算法包括 PPO-S 和ε = 0.2时的 PPO-Clip, PPO-DC 的曲线上升速度在对比算法中处于平均水平. 另一个发现是 PPO-Decay 算法在 Box2d 环境中的表现远不如它在 Mujoco 环境的表现, 分析产生这种现象的原因是两个环境使用了不同的 batch size 和 target step, 因此判断裁切边界衰减算法对于超参数较敏感.

图 8 对比了两类算法在训练过程中的平均 KL 散度变化. 由于使用了较大的更新批次, KL 散度的波动相对前一节实验稳定, 而各曲线随超参数变化的排列顺序较为明显. LunarLander 任务的 KL 散度曲线在初始阶段上升并在其后缓慢下降, 这是因为算法在早期已达成了训练目标.



(a) BipedalWalker-v3 (b) LunarLanderContinuous-v2

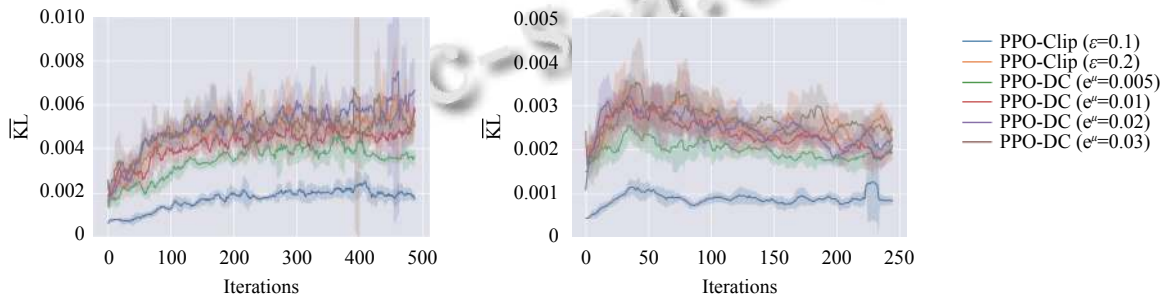
图 7 不同算法在 2 个 Box2d 任务上的训练曲线

表 5 Box2d 环境下的对比实验结果

算法	Box2d环境			
	BipedalWalker		LunarLanderContinuous	
	所有幕平均回报	最后100幕平均回报	所有幕平均回报	稳定着陆的采样轮数
PPO-Clip (0.1)	152	**273**	140	64
PPO-Clip (0.2)	161	242	165	**46**
PPO-DC	**184**	270	149	61
PPO-Penalty	64	120	140	68
PPO-Decay	15	45	−66	217
PPO-S	182	261	**175**	52

4 结论

虽然 PPO 算法本身的设计理念是使用近似点替代置信域, 并不严格要求参数落在置信域内, 但是更新后的参数也应当被限制在合理的范围内. PPO-Clip 算法无法保证这一点, 实验表明当使用较宽松裁切系数时, KL 散度的上界无法被确立, 这有悖于置信域优化目标. 基于以上缺陷, 本文对经典 PPO-Clip 算法改进, 提出了 PPO-DC 算法. 该算法使用两段裁切的代理目标, 每段边界被选择的概率随训练轮数的迭代而变化. 其中内边界被选择的概率是一个正态分布累计分布函数的输出, 该函数的输入为采样簇上平均 KL 散度的对数, 外边界被选择的概率与之相反. 这样做的优势在于, 当参数越过置信域边界, PPO-DC 算法能通过提升内边界的概率严格约束 KL 散度. 反之, 当置信域得到保证, 外边界被选择的概率提升, 样本更有可能得到宽幅的更新.



(a) BipedalWalker-v3 (b) LunarLanderContinuous-v2

图 8 PPO-Clip 算法和 PPO-DC 算法在 2 个 Box2d 任务上的平均 KL 散度

在 Mujoco 和 Box2d 环境的多个连续控制任务上, 本文对比了不同算法的训练曲线, PPO-DC 算法的性能是所有算法中是最优的. 同时, 本文也对比了改进前后算法在训练过程中的 KL 散度变化, 结果表明 PPO-DC 算法可以在使用较宽松裁切系数的情况下完成对 KL 散度的约束.

参考文献

1 Mnih V, Kavukcuoglu K, Silver D, *et al*. Human-level control through deep reinforcement learning. Nature, 2015,

518(7540): 529–533. [doi: 10.1038/nature14236]
2 He KM, Zhang XY, Ren SQ, *et al*. Deep residual learning for image recognition. Proceedings of the 2016 IEEE conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 770–778.
3 Browne CB, Powley E, Whitehouse D, *et al*. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games, 2012, 4(1): 1–43. [doi: 10.1109/TCIAIG.2012.2186810]
4 Silver D, Huang A, Maddison CJ, *et al*. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529(7587): 484–489. [doi: 10.1038/nature16961]
5 Silver D, Schrittwieser J, Simonyan K, *et al*. Mastering the game of go without human knowledge. Nature, 2017, 550(7676): 354–359. [doi: 10.1038/nature24270]
6 Berner C, Brockman G, Chan B, *et al*. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680v1, 2019.
7 Vinyals O, Babuschkin I, Czarnecki WM, *et al*. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 2019, 575(7782): 350–354. [doi: 10.1038/s41586-019-1724-z]
8 Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Boston: MIT Press, 2018. 216–320.
9 Silver D, Lever G, Heess N, *et al*. Deterministic policy gradient algorithms. Proceedings of the 31st International Conference on International Conference on Machine Learning. Beijing: JMLR.org, 2014. I-387–I-395.
10 Lillicrap TP, Hunt JJ, Pritzel A, *et al*. Continuous control with deep reinforcement learning. arXiv:1509.02971, 2015.
11 Fujimoto S, van Hoof H, Meger D. Addressing function approximation error in actor-critic methods. Proceedings of the 35th International Conference on Machine Learning. Stockholmsmässan: PMLR, 2018. 1582–1591.
12 Schulman J, Levine S, Abbeel P, *et al*. Trust region policy optimization. Proceedings of the 32nd International Conference on Machine Learning. Lille: PMLR, 2015. 1889–1897.
13 Schulman J, Wolski F, Dhariwal P, *et al*. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.
14 Engstrom L, Ilyas A, Santurkar S, *et al*. Implementation matters in deep RL: A case study on PPO and TRPO. 8th International Conference on Learning Representations. Addis Ababa: OpenReview.net, 2020. 1–14.
15 Chen G, Peng YM, Zhang MJ. An adaptive clipping approach for proximal policy optimization. arXiv:1804.06461, 2018.
16 Wang YH, He H, Tan XY. Truly proximal policy optimization. Proceedings of the 35th Uncertainty in Artificial Intelligence Conference. Tel Aviv: PMLR, 2020. 113–122.
17 Zhu WS, Rosendo A. A functional clipping approach for policy optimization algorithms. IEEE Access, 2021, 9: 96056–96063. [doi: 10.1109/ACCESS.2021.3094566]
18 Dossa RFJ, Huang SY, Ontañón S, *et al*. An empirical investigation of early stopping optimizations in proximal policy optimization. IEEE Access, 2021, 9: 117981–117992. [doi: 10.1109/ACCESS.2021.3106662]
19 Farsang M, Szegletes L. Decaying clipping range in proximal policy optimization. Proceedings of the 2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI). Timisoara: IEEE, 2021. 521–526.
20 Brockman G, Cheung V, Pettersson L, *et al*. OpenAI gym. arXiv:1606.01540v1, 2016.
21 Todorov E, Erez T, Tassa Y. MuJoCo: A physics engine for model-based control. Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. Vilamoura-Algarve: IEEE, 2012. 5026–5033.
22 Kakade SM. A natural policy gradient. Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. Vancouver: MIT Press, 2001. 1531–1538.
23 Schulman J, Moritz P, Levine S, *et al*. High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438, 2015.
24 Haarnoja T, Zhou A, Abbeel P, *et al*. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. Proceedings of the 35th International Conference on Machine Learning. Stockholmsmässan: PMLR, 2018. 1861–1870.
25 Liu XY, Li ZC, Yang ZR, *et al*. ElegantRL-Podracer: Scalable and elastic library for cloud-native deep reinforcement learning. arXiv:2112.05923, 2021.

(校对责编: 牛欣悦)