# 近端策略优化算法



## 摘要


我们提出了一系列用于强化学习的新的策略梯度方法，该方法在通过与环境交互来采样数据和使用随机梯度上升优化“替代”目标函数二者之间交替。尽管标准的策略梯度方法对每个数据样本执行一次梯度更新，但我们提出了一种新的目标函数，该函数可以实现多个时期的微批量更新。我们称之为近端策略优化（PPO）的新方法具有信任域策略优化（TRPO）的一些好处，但它们实现起来更简单，更通用，并且具有更好的样本复杂性。我们的实验在一系列基准任务上测试了 PPO，包括模拟机器人运动和雅达利游戏，我们表明 PPO 优于其他在线策略梯度方法，总体上在样本复杂性、简单性和墙时间之间取得了良好的平衡。


## 1 介绍

近年来，研究者们提出了许多不同的方法来利用神经网络函数逼近器进行强化学习。领先的竞争者是 DQN[Mni+15]、“Vanilla”策略梯度方法[Mni+16]和信任区域/自然策略梯度方法[Sch+15b]。然而，在开发一种可扩展（适用于大型模型和并行实现）、数据高效、稳健（即在不进行超参调整的情况下成功解决各种问题）的方法方面仍有提高的空间。Q 学习（使用函数逼近）在许多简单的问题上失败了，并且不被很好地理解，普通的策略梯度方法的数据效率和鲁棒性很差；而信任区域策略优化（TRPO）相对复

杂，并且与包含噪声（例如丢弃）或参数共享（在策略和值函数之间，或者与辅助任务一起）的架构不兼容。

本文旨在通过引入一种算法来改善当前的状况，该算法在仅使用一阶优化的情况下达到 TRPO 的数据效率和可靠性能。

我们提出了一个具有裁剪概率比的新目标，它形成了对策略性能的悲观估计（即下界）。为了优化策略，我们在从策略中采样数据和对采样数据进行多个阶段的优化。

我们的实验比较了各种不同版本的代理目标的性能，发现具有裁剪概率比的版本表现最好。我们还将 PPO 与文献中先前的几种算法进行了比较。在连续控制任务中，它的性能比我们比较的算法要好。在 Atari 游戏上，它的性能（就样本复杂性而言）明显好于 A2C，与 ACER 相似，但它要简单得多。

# 2 背景：策略优化

## 2.1 策略梯度方法

策略梯度方法通过计算策略梯度的估计值并将其插入随机梯度上升算法中来工作。最常用的梯度估计器具有以下形式：

$$\hat{g} = \hat{\mathbb{E}}_t \left[ \nabla_\theta \log \pi_\theta \left( a_t \mid s_t \right) \hat{A}_t \right]$$

其中 $\pi_\theta$ 是随机策略，而 $\hat{A}_t$ 是时刻 $t$ 的优势函数的估计值。在这里，期望 $\hat{\mathbb{E}}[\ldots]$ 表示在采用和优化之间交替的算法中，有限样本批次的经验平均值。对构造的目标函数使用自动求导软件进行微分，该目标函数的梯度即是策略梯度估计器；通过微分下式目标函数可获得估计值 $\hat{g}$：

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[ \log \pi_\theta \left( a_t \mid s_t \right) \hat{A}_t \right]$$

尽管使用相同的轨迹对该损失 $L^{PG}$ 进行多步优化很有吸引力，但是这样做并没有充分的理由，而且从经验上讲，它经常导致破坏性的大型策略更新（请参阅第6.1节；结果未显示，但与“无剪辑或惩罚”设置相似或更差）。

## 2.2 信任区域方法

在 TRPO 中，目标函数（“替代”目标函数）在策略更新的步长上受到限制。即：

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[ \mathrm{KL} \left[ \pi_{\theta_{\text{old}}} \left( \cdot \mid s_t \right), \pi_\theta \left( \cdot \mid s_t \right) \right] \right] \leq \delta. \end{aligned}$$

此处，$\theta_{old}$ 是更新之前策略参数的向量。在对目标函数进行线性近似并且对约束项进行二次近似之后，可以使用共轭梯度算法有效地解决该问题。

证明TRPO合理的理论实际上建议使用惩罚项而不是约束项，即解决无约束的优化问题：

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t - \beta \mathrm{KL}\left[\pi_{\theta_{\text{old}}}\left(\cdot \mid s_t\right), \pi_\theta\left(\cdot \mid s_t\right)\right] \right]$$

对于某些系数 $\beta$。这是基于这样的事实，即某个替代目标函数（它计算状态上的最大KL而不是其均值）在策略 $\pi$ 的性能上形成了一个下界（即一个悲观的界限）。TRPO使用约束项而不是惩罚项，因为很难选择一个在不同问题上甚至在特征随学习过程变化的单个问题内表现良好的 $\beta$ 值。因此，为了实现我们的模拟TRPO单调改进的一阶算法的目标，实验表明，仅仅选择固定惩罚系数 $\beta$ 并使用SGD优化目标函数（5）是不够的。需要其他修改。

# 3 裁剪替代目标

令 $r_t(\theta)$ 表示概率比 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\,old}(a_t|s_t)}$，所以 $r(\theta_{old}) = 1$。TRPO最大化如下代替目标函数：

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[ r_t(\theta) \hat{A}_t \right]$$

上标 $CPI$ 指的是约束策略迭代，其论文中提出了该目标函数。在没有约束项的情况下，$L^{CPI}$ 的最大化将导致策略更新过大。因此，我们现在考虑如何修改该目标函数，以惩罚 $r_t(\theta)$ 远离 1 的策略更新。

我们提出的主要目标函数如下：

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

其中，$\epsilon$ 是超参数，通常，$\epsilon = 0.2$。目标函数提出的动机如下：位于 $min()$ 中的第一项是 $L^{CPI}$。位于 $min()$ 中的第二项，$clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$，通过裁剪概率比来修改替代目标函数，从而消除了将 $r_t$ 移到区间 $[1-\epsilon, 1+\epsilon]$ 之外的动机。最后，我们取裁剪和未裁剪目标函数中的最小值，因此最终的目标函数是在未裁剪目标函数中的一个下界（即悲观界限）。通过这种方案，我们仅在概率率提高目标函数时才忽略概率率的变化，而在概率率使目标函数减少时将其包括在内。注意，在 $\theta_{old}$ 附近（即，$r = 1$）处于一阶时，$L^{CLIP}(\theta) = L^{CPI}(\theta)$，但是，当 $\theta$ 远离 $\theta_{old}$ 时，它们会变得不同。图1绘制了 $L^{CLIP}$ 中的一个单项（即单个 $t$）；注意，根据优势值是正的还是负的，概率比 $r$ 被 $1+\epsilon$ 或 $1-\epsilon$ 裁剪。

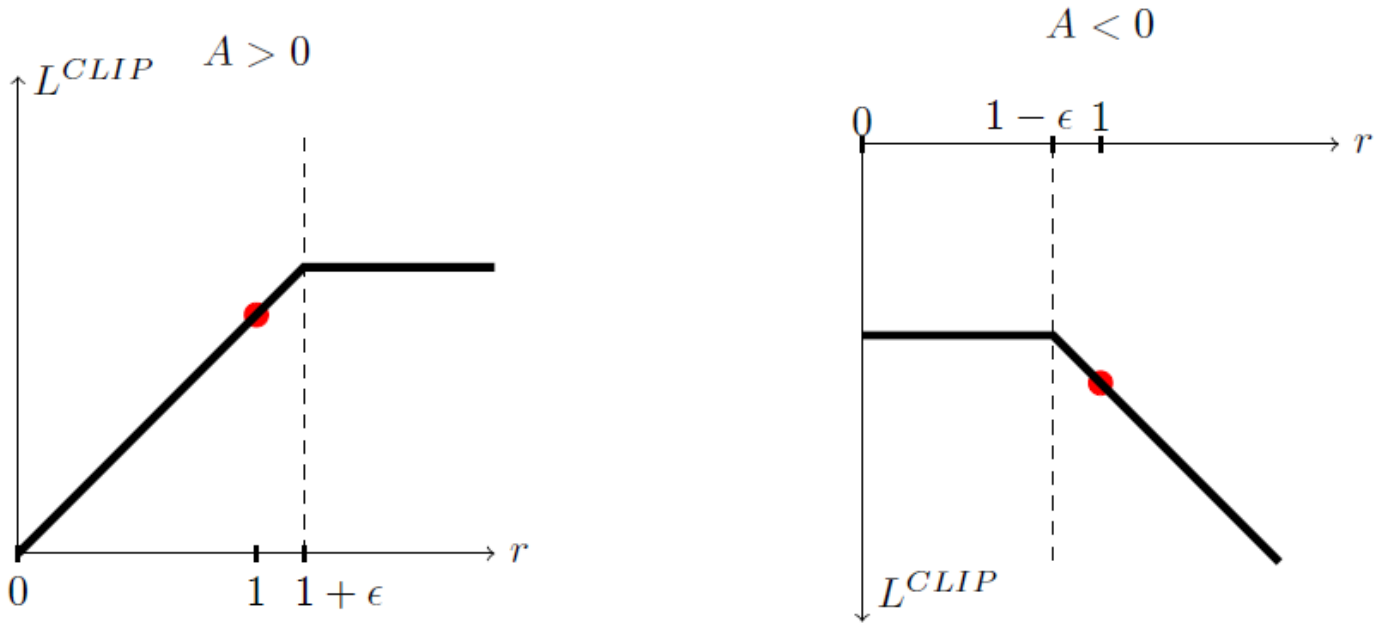


图1：显示代用函数 LCLIP 的一个项（即单个时间步长）与概率比r的函数的图，正优势（左）和负优势（右）。每个图上的红圈表示优化的起点，即 r=1。请注意，LCLIP 对其中许多项进行了求和。

图2提供了关于替代目标函数 $L^{CLIP}$ 的另一种直观来源。它显示了当我们沿着策略更新方向进行插值时，几个目标函数是如何变化的，这是通过对连续控制问题进行近端策略优化（稍后将介绍的算法）获得的。我们可以看到 $L^{CLIP}$ 是 $L^{CPI}$ 的下界，这是因为策略更新太大而带来的惩罚。

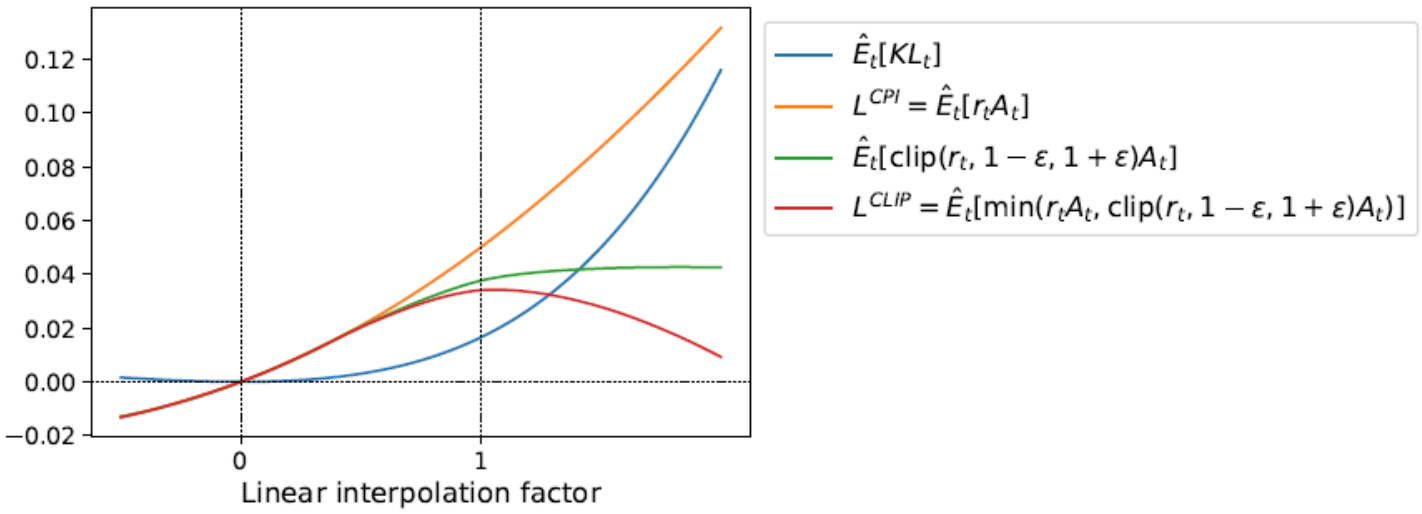


图2：代理目标，我们在初始策略参数 θ 和更新的旧策略参数之间进行插值，我们在 PPO 的一次迭代后计算出这一参数。更新后的策略与初始策略的KL散度约为0.02，这就是 LCLIP 最大的点。该图对应于使用第6.1节提供的超参数对 Hopper-v1 问题进行的首先策略更新。

# 4 自适应 KL 惩罚系数

另一种方法，可以作为裁剪代理目标的替代品，或者作为它的补充，是使用对KL散度的惩罚，并调整惩罚系数，使我们在每次策略更新时达到KL散度的一些目标值。在我们的实验中，我们发现KL惩罚的表现比剪切代理目标要差，然而，我们在这里包括它，因为它是一个重要的基线。

在这个算法的最简单的实例中，我们在每次策略更新中执行以下步骤。

- 使用 minibitch SGD 的几个 epoch，优化 KL 惩罚目标

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi(\cdot|s_t)]]$$

- 计算 $d = \hat{\mathbb{E}}[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$
  - 如果 $d < d_{targ}/1.5, \beta \leftarrow \beta/2$
  - 如果 $d > d_{targ} * 1.5, \beta \leftarrow \beta * 2$

更新后的 $\beta$ 被用于下一次策略更新。有了这个方案，我们偶尔会看到 KL 散度与 d 有很大不同的策略更新，然而，这些都是罕见的，而且 $\beta$ 很快就会调整。上面的参数1.5和2是启发式选择的，但该算法对它们并不十分敏感。$\beta$ 的初始值是另一个超参数，但在实践中并不重要，因为算法会迅速调整它。

# 5 算法

前面几节的替代损失可以通过对典型的策略梯度实现的微小改变来计算和区分。对于使用自动微分的实现方式，可以简单地构建损失 $L^{CLIP}$ 或 $L^{KLPEN}$ 而不是 $L^{PG}$，并在此目标上执行多个随机梯度上升步骤。

大多数计算降低方差的优势函数估计的技术都使用了学习到的状态-价值函数 $V(s)$；例如，广义优势估计[Sch+15a]，或[Mni+16]中的有限跨度估计。如果使用一个在策略和价值函数之间共享参数的神经网络架构，我们必须使用一个结合策略代理和价值函数误差项的损失函数。这个目标可以通过增加熵奖励来进一步增强，以确保充分的探索，正如过去的工作[Wil92; Mni+16]中所提议的那样。

结合这些项，我们得到以下目标，该目标在每次迭代中都是（近似）最大化的：

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

其中 $c_1, c_2$ 为系数，S 表示熵的奖励，$L_t^{VF}$ 是平方差损失。

一种策略梯度实现方式，在[Mni+16]中得到推广，非常适合用于循环神经网络，在 T 个时间步中运行策略（其中 T 远小于回合长度），并使用收集的样本进行更新。这种方式需要一个优势估计器，它不会超越时间步长T。[Mni+16]所使用的估计器是：

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

其中 t 指定了[0, T]中的时间索引，在一个给定的长度为T的轨迹段中。泛化这一选择，我们可以使用广义优势估计的截断版本，当 λ=1 时，它可简化为公式（10）：

$$\hat{A}_t = \delta + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + \gamma\lambda^{T-t+1}\delta_{T-1}$$

其中 $\delta_t = r_t + \gamma V(s_{t+1} - V(s_t))$

一个使用固定长度轨迹段的近似策略优化（PPO）算法如下所示。每次迭代，N 个（平行的）actor 中的每一个都收集 T 个时间段的数据。然后，我们在这 N 个 T 个时间段的数据上构建代用损失，并用 minibatch SGD（或者通常为了更好的性能，用 Adam [KB14]）来优化它，持续 K 个 epoch。

**Algorithm 1** PPO, Actor-Critic Style

**for** iteration=$1, 2, \ldots$ **do**
  **for** actor=$1, 2, \ldots, N$ **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 $\theta$, with $K$ epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
**end for**

# 6 实验

## 6.1 目标函数的比较

首先，我们在不同的超参数下比较几个不同的代用目标。在这里，我们将代用目标 $L^{CLIP}$ 与几个自然变化和消融的版本进行比较。

对于 KL 惩罚，我们可以使用一个固定的惩罚系数 $\beta$ 或者一个自适应系数，如第4节所述，使用目标 KL 值 $d_{targ}$。请注意，我们也尝试过在对数空间中进行修剪，但发现其性能并没有提高。

因为我们正在搜索每个算法变体的超参数，所以我们选择了一个计算量小的基准来测试算法。也就是说，我们使用了 在OpenAI Gym [Bro+16]中实现的7个模拟机器人任务，它们使用了 MuJoCo [TET12] 物理引擎。我们在每个任务上做一百万次的训练。除了用于剪切（$\epsilon$）和KL惩罚（$\beta, d_{targ}$）的超参数（我们在上面搜索），其他的超参数都在文件中提供。

为了表示策略，我们使用了一个全连接的 MLP，它有两个由64个单元组成的隐藏层，以及 tanh 非线性，按照[Sch+15b; Dua+16]，输出高斯分布的平均值，标准偏差可变。我们不在策略和价值函数之间共享参数（所以系数 $c_1$ 是不相关的），我们也不使用熵奖励。

每个算法都在所有7个环境中运行，每个环境中都有3个随机种子。我们通过计算最后100回合的平均总奖励对算法的每次运行进行评分。我们对每个环境的分数进行移位和缩放，使随机策略给出的分数为0，最好的结果被设置为1，并对21次运行进行平均，为每个算法设置产生一个单一的标量。

结果显示在表1中。请注意，在没有剪切或惩罚的情况下，得分是负的，因为对于一个环境（half cheetah），它导致了一个非常负的分数，比最初的随机策略更差。

| algorithm | avg. normalized score |
|---|---|
| No clipping or penalty | -0.39 |
| Clipping, $\epsilon = 0.1$ | 0.76 |
| **Clipping, $\epsilon = 0.2$** | **0.82** |
| Clipping, $\epsilon = 0.3$ | 0.70 |
| Adaptive KL $d_{\text{targ}} = 0.003$ | 0.68 |
| Adaptive KL $d_{\text{targ}} = 0.01$ | 0.74 |
| Adaptive KL $d_{\text{targ}} = 0.03$ | 0.71 |
| Fixed KL, $\beta = 0.3$ | 0.62 |
| Fixed KL, $\beta = 1.$ | 0.71 |
| Fixed KL, $\beta = 3.$ | 0.72 |
| Fixed KL, $\beta = 10.$ | 0.69 |

## 6.2 在连续域中与其他算法的比较

接下来，我们将 PPO（与第3节中的 "剪切 "代理目标）与文献中的其他几种方法进行比较，这些方法被认为对连续问题有效。我们与以下算法的调整实现进行了比较：信任区域策略优化[Sch+15b]，交叉熵法（CEM）[SL06]，带有自适应步长的最原始的策略梯度，A2C [Mni+16]，带有信任区域的 A2C [Wan+16]。A2C 代表优势 actor-critic，是 A3C 的一个同步版本，我们发现它的性能与异步版本相同或更好。对于 PPO，我们使用上一节的超参数，$\epsilon = 0.2$。我们看到，PPO 在几乎所有的连续控制环境上都优于以前的方法。

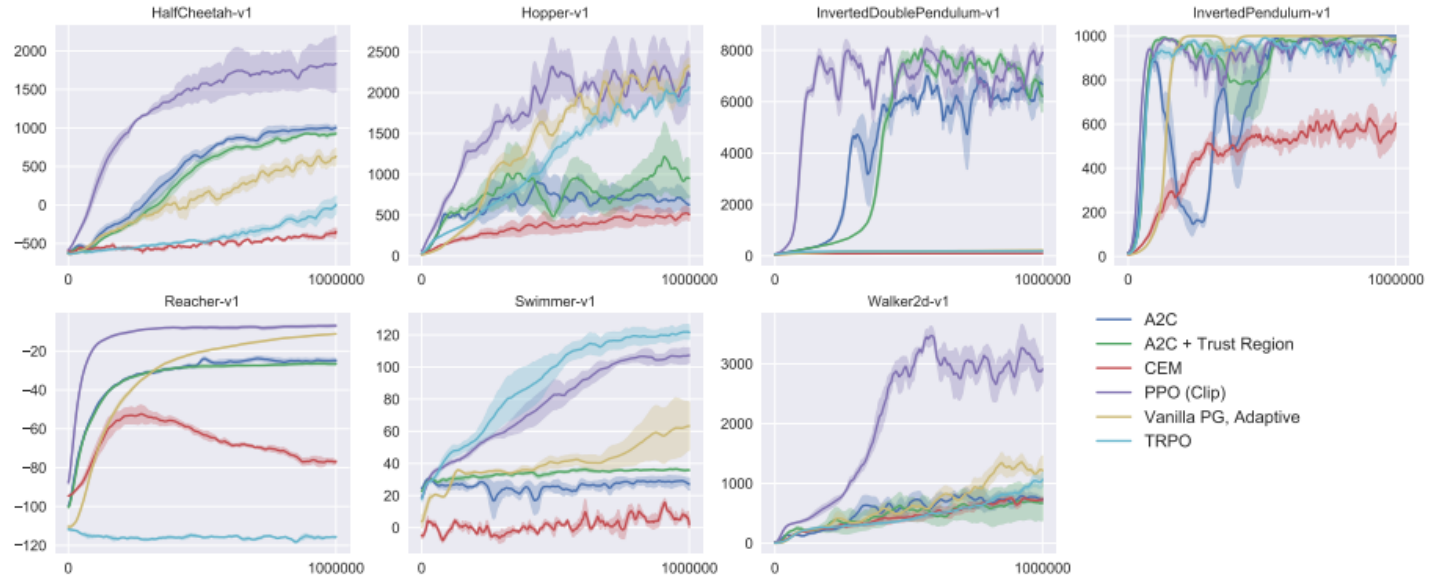


图3：几种算法在 MuJoCo 环境下的比较，训练了100万个时间步长。

## 6.3 连续域示例：人形机器人奔跑和转向

为了展示 PPO 在高维连续控制问题上的性能，我们对一组涉及三维人形机器人的问题进行了训练，在这些问题中，机器人必须跑步、转向，并从地面上站起来，同时可能被方块击中。我们测试的三个任务是：（1）RoboschoolHumanoid：仅向前运动；（2）RoboschoolHumanoidFlagrun：目标的位置每200个时间步数或只要达到目标就随机变化；（3）RoboschoolHumanoidFlagrunHarder，机器人被方块砸中，需要从地上站起来。图5是学习策略的静止画面，图4是这三个任务的学习曲线。表4中提供了超参数。在同期的工作中，Heess 等人[Hee+17]使用 PPO 的自适应 KL 变体（第4节）来学习三维机器人的运动策略。

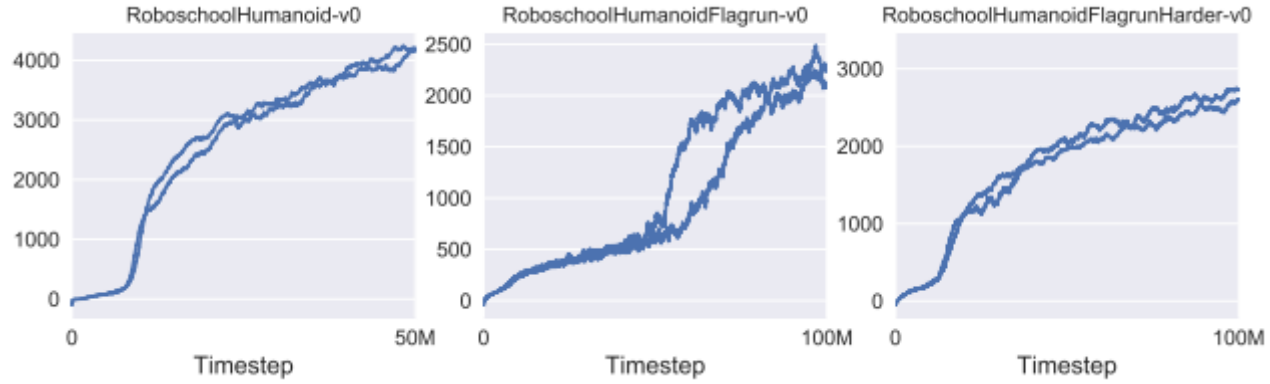


图4：使用Roboschool在3D人形控制任务上的PPO学习曲线。

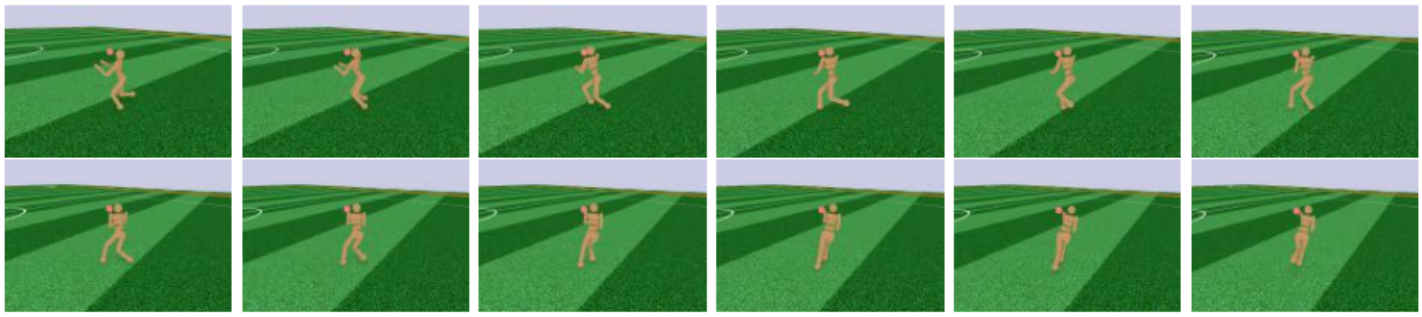

图5：从 RoboschoolHumanoidFlagrun 学到的策略的静态帧。在前六帧中，机器人向一个目标跑去。然后位置被随机改变，机器人转向并向新的目标跑去。

## 6.4 在 Atari 游戏中与其他算法的比较

我们还在 Arcade Learning Environment[Bel+15]基准上运行 PPO，并与 A2C [Mni+16]和 ACER [Wan+16]的良好调整实现进行比较。对于这三种算法，我们使用了与[Mni+16]相同的策略网络架构。表5中提供了 PPO 的超参数。对于其他两种算法，我们使用了经过调整的超参数，以使该基准的性能最大化。

附录B中提供了所有49场比赛的结果和学习曲线表。我们考虑以下两个评分指标。（1）整个训练期间每个回合的平均奖励（有利于快速学习），以及（2）训练的最后100回合中每个回合的平均奖励（有利于最终表现）。表2显示了每种算法 "赢得 "的游戏数量，我们通过对三次试验的平均得分指标来计算胜利者。

| | A2C | ACER | PPO | Tie |
|---|---|---|---|---|
| (1) avg. episode reward over all of training | 1 | 18 | **30** | 0 |
| (2) avg. episode reward over last 100 episodes | 1 | **28** | 19 | 1 |

表2：每种算法“赢得”的游戏数量，其中得分指标是三次试验的平均数。

# 7 结论

我们介绍了近似策略优化，这是一个策略优化方法系列，使用随机梯度上升的多个epoch来执行每个策略更新。这些方法具有信任区域方法的稳定性和可靠性，但实现起来要简单得多，只需要对普通的策略梯度实现进行几行代码的修改，适用于更普遍的环境（例如，当使用策略和价值函数的联合架构时），并且具有更好的整体性能。