Proximal Policy Optimization Algorithms

Introduction：

我们提出了一个新的目标函数，可以多回合的批量更新。我们叫它 PPO。

信赖域策略优化相对比较复杂，和包含噪音的架构(比如 dropout)和参数共享(策略和价值函数间，或者辅助任务)的架构不兼容。

PPO 采用一阶优化。算法使用一个新的带有裁切概率比技巧目标函数形成了一种对于策略效用的悲观估计，(比如使用下界进行估计)。带有裁切概率比的 PPO 效果是最佳的。

裁切概率比 the clipped probability ratios

Policy Gradient Methods：

使用随机梯度上升进行策略梯度下降，常见优化函数如下：

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

A 为优势函数，参数为 θ 梯度下降函数如下：

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

对于同一个 trajectory 使用 PG 大步更新参数是对结果有破坏性的，最终结果差不多或者差于不带裁剪或惩罚的设定。

Trust Region Methods:

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

从 TRPO 的 paper 可以知道，TRPO 方法优化的是上述带不等式的目标函数。经过对目标进行线性逼近，对约束进行二次逼近后，容易使用梯度下降计算。

TRPO 实用性表明使用惩罚代替约束效果更好，即使用以下优化目标：

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

之后用 KL 散度平均值代替最大值，但即使化上述形式后，参数 β 也很难选择可以使得结果很好，因此需要对 TRPO 进行改进，使得惩罚系数 β 容易选择。

以下用两种方法对系数 β 进行改进：

Clipped Surrogate Objective 裁剪目标函数：

令 $r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$，则 TRPO 表示为：

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

上式没有约束容易出现更新步长过大，因此增加一个裁剪操作，惩罚 r 远离 1 的策略，因此，裁剪后的优化函数如下所示：

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

将 r 控制在 1 附近。最后，我们取修剪目标和未修剪目标的最小值，因此最终目标是一个下界。如果原来的目标函数可以优化，则忽略 r 的裁剪；若目标函数会变差，则考虑进 r 的裁剪。

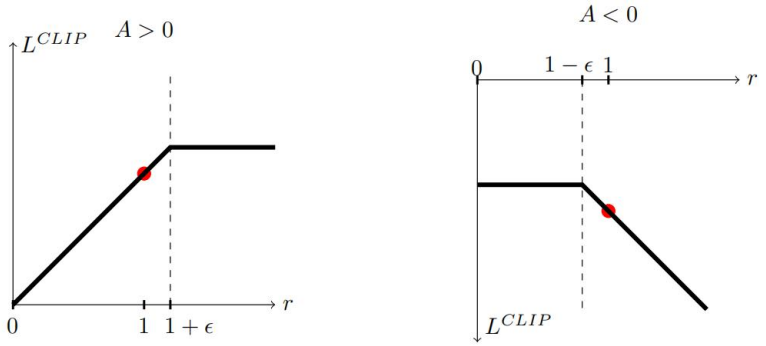

Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

注意到裁切比率 r 被限制成 1- ε 或者 1+ ε ，取什么值依赖于优势函数是正的还是负的。可以看到 LCLIP 是 LCPI 的下界，裁剪是为了惩罚更新步长过大。

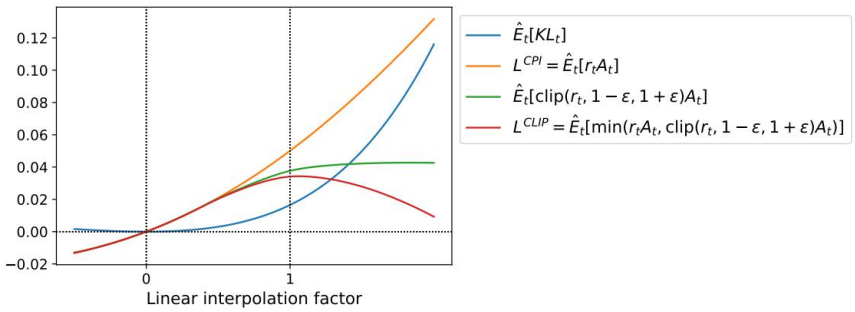

Figure 2: Surrogate objectives, as we interpolate between the initial policy parameter θ_{old}, and the updated policy parameter, which we compute after one iteration of PPO. The updated policy has a KL divergence of about 0.02 from the initial policy, and this is the point at which L^{CLIP} is maximal. This plot corresponds to the first policy update on the Hopper-v1 problem, using hyperparameters provided in Section 6.1.

Adaptive KL Penalty Coefficient 自适应 KL 惩罚系数：

另一种被用来替代裁剪方法的技巧是自适应 KL 惩罚系数，是在 KL 散度的系数上面进

行限制而不是惩罚，以便确定策略更新时 KL 散度的一些目标值。

因为以下不等式方法对 β 进行限制，因此 β 使用自适应的方法很容易调整，因此对于一些目标值是容易确定的。

- Using several epochs of minibatch SGD, optimize the KL-penalized objective

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

- Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
  - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

  β 可以随机初始化，因为自适应方式能很快调整 β 。

Algorithm：

为了实施算法以便于可以自动求导，需要两个技巧。一是使用损失 LCLIP or LKLPEN 代替 LPG，二是对同一个目标多步使用随机梯度上升，n-step 更新。

如果使用在策略和值函数之间共享参数的神经网络体系结构，则必须使用组合了策略和价值函数错误项的损失函数。

目标函数可以通过增加熵来进一步增强，以确保足够的探索。

结合以上技巧，得到如下的优化函数：

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

其中 c1, c2 是系数，同时 S 表示熵增，LV Ft 是一个均方误差项(V θ (s)- Vtarg)^2。

有一种风格的策略梯度实现方式，在[MNI+16]中推广，非常适合与递归神经网络一起使用，先运行策略 T 步（小于一个片段的长度），然后使用 T 步的样本进行一次更新。这里需要一个优势函数估计器可以将 T 步样本都考虑进去。

因此可以考虑如下函数：

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

推广上式，我们可以使用缩减版本进行替换，如下式，其中 lamda = 1，

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$
$$\text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

下图是 PPO 使用固定长轨迹段，即 T 步轨迹，每次迭代，有 N 个 agent（并行）分别使用各自的固定长轨迹段。

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 θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Experiments：

1. Comparison of Surrogate Objectives

首先我们比较 L 的各种变种和简介版本，比较版本如下：

$$
\begin{aligned}
&\text{No clipping or penalty:} && L_t(\theta) = r_t(\theta)\hat{A}_t \\
&\text{Clipping:} && L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t \\
&\text{KL penalty (fixed or adaptive)} && L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]
\end{aligned}
$$

我们发现裁剪使用 log 并没有对结果造成影响，也没有更好。1M 训练，表格为不同的参数对于训练结果的影响。

Hyperparameter	Value
Horizon (T)	2048
Adam stepsize	3×10^{-4}
Num. epochs	10
Minibatch size	64
Discount (γ)	0.99
GAE parameter (λ)	0.95

Table 3: PPO hyperparameters used for the Mujoco 1 million timestep benchmark.

我们使用 MLP 全连接多层感知机连接 2 个带有 64 个神经元的隐藏层，用 tanh-non-linearities 函数，输出具有可变标准偏差的高斯分布的均值。我们在策略和价值估计函数之间不使用参数共享，因此系数不相关。同时我们也不使用熵对策略进行加噪声。

各种形式的 L 作用下的 PPO 结果如下：

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

Table 1: Results from continuous control benchmark. Average normalized scores (over 21 runs of the algorithm, on 7 environments) for each algorithm / hyperparameter setting . β was initialized at 1.

2. 在连续空间中和其他算法进行对比

我们比较了以下算法的优化实现：TRPO、CEM、带有自适应步长的 VPG、A2C、带有信赖域的 A2C。

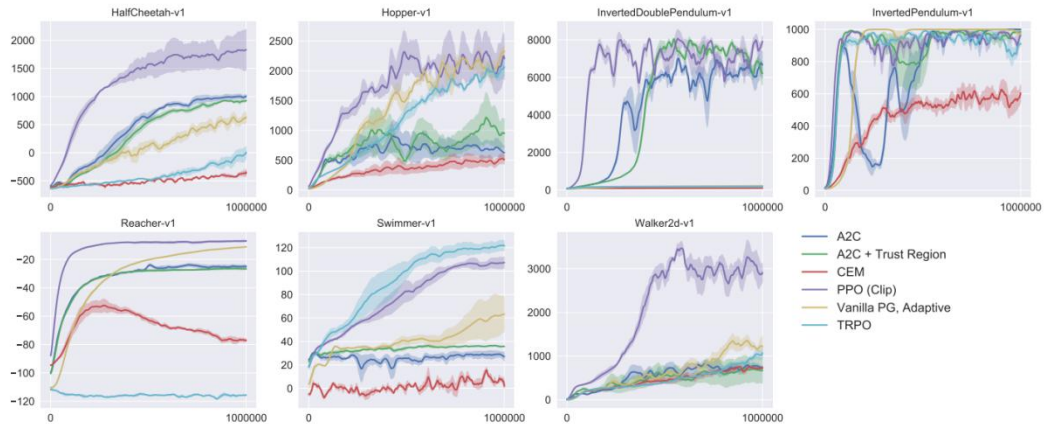

Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

3. Showcase in the Continuous Domain: Humanoid Running and Steering

4. 在 Atari 和其他算法比较：

我们考虑两种指标：(1) 整个训练期间每回合平均奖励(训练速度)(2) 超过 100 回合以后，每回合的平均奖励(算法上限)。

	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

Table 2: Number of games "won" by each algorithm, where the scoring metric is averaged across three trials.

Conclusions：

一组使用多个阶段的随机梯度上升来执行每个策略更新的策略优化方法。

改善版 TRPO，Improved TRPO