# 近端策略优化算法(Proximal Policy Optimization Algorithms, PPO)

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

这篇博文是Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.的阅读笔记，用来介绍PPO优化方法及其一些公式的推导。文中给出了三种优化方法，其中第三种是第一种的拓展，这两种使用广泛，第二种实验验证效果不好，但也是一个小技巧。阅读本文，需要事先了解信赖域策略优化(Trust Region Policy Optimization, TRPO)，从Proximal这个词汇中，可以联想到一类涉及矩阵范数的优化问题中的软阈值算子(soft thresholding/shrinkage operator)以及图Lasso求逆协方差矩阵(Graphical Lasso for inverse covariance matrix)中使用近端梯度下降(Proximal Gradient Descent, PGD)求解Lasso问题。更多强化学习内容，请看：随笔分类 - Reinforcement Learning。

## 1. 前提知识

策略梯度法(Policy Gradient Methods)与信赖域策略优化(Trust Region Policy Optimization, TRPO)

## *Proximal Policy Optimization Algorithms*

### ➢ 前提知识

- 策略梯度法(Policy Gradient Methods)

目标函数：

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

目标函数关于策略参数θ的导数：

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

其中$\pi_\theta$为一随机策略，

$\hat{A}_t$为在$t$时刻时优势函数的估计值.

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

- 信赖域策略优化(Trust Region Policy Optimization, TRPO)

目标函数：

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

$$s.t.\ \hat{\mathbb{E}}_t\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot \mid s_t) \,\|\, \pi_\theta(\cdot \mid s_t)\right)\right] \le \delta$$

问题等价为：

$$\max_{\theta} J^{TRPO}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t - \beta D_{KL}\left(\pi_{\theta_{old}}(\cdot \mid s_t) \,\|\, \pi_\theta(\cdot \mid s_t)\right)\right]$$

其中$\theta_{old}$为更新前的策略参数向量，$\beta$为超参数.

- 策略梯度法的工作原理是计算策略梯度的一个估计量，并将其代入随机梯度上升算法中。其缺点是，每次更新参数后，都需要重新与环境互动，计算新的策略的优势函数，再进行参数更新。即一条轨迹只能更新参数一次，导致大部分时间浪费在与环境互动上。
- TRPO算法便使得一条轨迹可以多次更新参数，从而节省时间。但TRPO将新旧策略差异不能太大作为约束项，在实际操作中时比较困难，即使将约束性作为惩罚项放入目标函数中，也需要自适应调整超参数$\beta$以获得最佳性能，在不同的问题上选择合适的$\beta$值是非常困难的，甚至在单个问题上，不同的特征也会随着学习而变化。所以，仅仅简单地设置一个固定的参数来用SGD优化TRPO目标函数，是不够的。

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.



由于TRPO使用了一个硬约束来计算策略梯度，因此很难选择一个在不同问题中都表现良好的单一约束值。

## 2. 方法一：Clipped Surrogate Objective

## ➢ 方法一：Clipped Surrogate Objective

- TRPO的目标函数项进一步改写为(不考虑约束项)

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

其中 $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$，$r_t(\theta_{old}) = 1.$

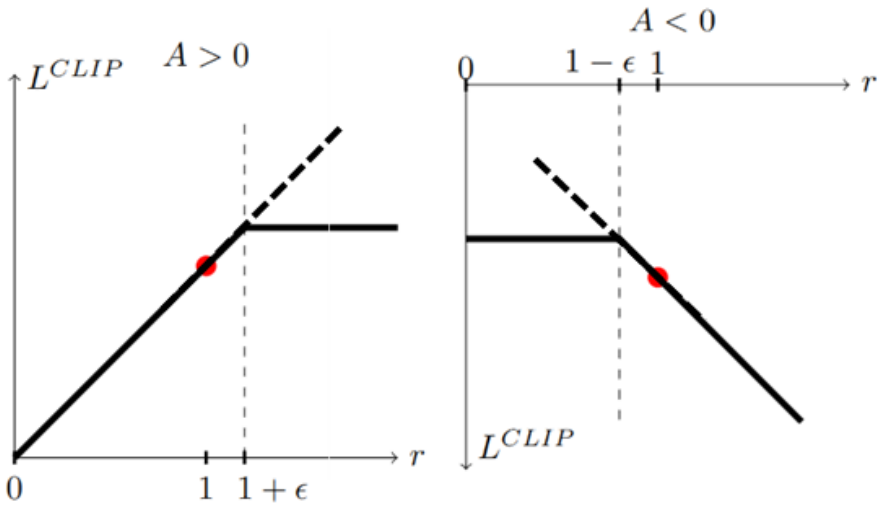
CPI: Conservative Policy Iteration.

在没有约束的情况下，最大化CPI目标函数将导致过大的策略更新。因此，我们现在考虑当$r_t(\theta)$远离1时该如何修改目标函数能够惩罚策略的更新。

- 所提出的方法一的目标函数

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

同样为确保新旧策略差异不能太大，方法一采用一个裁剪函数来限制新旧策略之间的变化。$L^{CLIP}$(实线)是$L^{CPI}$(虚线)的一个下界，惩罚过大的策略更新。




- 当$A$>0时，最大化目标函数L相当于最大化$r$，当$r$过于大时，即$r$>1+ε，L为常数，不再继续上升，此时L对θ的导数为0，策略不再更新，迫使新旧策略差异不那么大。
- 当$A$<0时，最大化目标函数L相当于最小化$r$，当$r$过于小时，即$r$<1+ε，L为常数，不再随$r$改变而改变，对θ的导数为0，因此策略不再更新，迫使新旧策略差异小。
- $r$=1为优化起点，表示新旧策略一致无差异。
- ε经验地取值为0.2。

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.

➢ 方法一： Clipped Surrogate Objective

**Algorithm 2:** PPO, modified from [Sch+17b]

**for** *iteration*=1, 2, . . . **do**
  run policy $\pi_{\theta_{old}}$ in environment for $T$ timesteps
  to obtain trajectory $\{s_0, a_0, \ldots s_{T-1}, a_{T-1}, s_T\}$
  with rewards $\{r_1, \ldots r_T\}$
  **for** $t$=1, . . . $T$ **do**
    compute performance measure $G_t$
  **end**
  compute objective function $\mathcal{L}^{CLIP}$ by summing trajectories and averaging time-steps
  **for** *epoch in* 1, . . . $K$ **do**
    optimize surrogate $\mathcal{L}^{CLIP}(\theta)$ w.r.t. $\theta$ using mini-batches
    obtain $\theta$ by Gradient Ascent
  **end**
  $\theta_{old} \leftarrow \theta$
**end**

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.
[2] Proximal Policy Optimization Algorithms, slides, https://dvl.in.tum.de/slides/automl-ss19/01_stadler_ppo.pdf



**3. 方法二： Adaptive KL Penalty Coefficient**

## ➢ 方法二：Adaptive KL Penalty Coefficient

- 另一种方法，对KL散度施加惩罚，并自适应调整惩罚系数，以便每次策略更新时我们都能达到KL散度的一些目标值$d_{\text{targ}}$。在实验中发现方法二的效果不如方法一，但在这里也列出来，用作一个基准算法进行对比。
- Using several epochs of minibatch SGD, optimize the KL-penalized objective:

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

- Compute $d = \hat{\mathbb{E}}_t\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot \mid s_t) \,\|\, \pi_\theta(\cdot \mid s_t)\right)\right]$

$$if\ d < d_{\text{targ}}/1.5,\ \beta \leftarrow \beta/2$$

$$if\ d > d_{\text{targ}} \times 1.5,\ \beta \leftarrow \beta \times 2$$

- 总体思想还是使得新旧策略之间差异不能太大，但也不能完全没变化。
  - 当差异大时，让惩罚系数也变大，增大这部分惩罚，这样目标函数J的变化会放缓，J对参数θ的梯度会变小，迫使新旧策略差异小；
  - 当差异小时，让惩罚系数也变小，减少这部分惩罚，这样目标函数J的变化幅度适当大些，J对参数θ的梯度会适当变大，使得新旧策略存在一定差异，而非完全一致。

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.



**4. 方法三：Actor-Critic-Style Algorithm**

## ➢ 方法三：Actor-Critic-Style Algorithm

- 上述算法均是基于策略梯度的算法。但此类方法解空间大，难以充分采样，导致方差大。为减少优势函数的方差，一般会引入状态值函数V(s)。如果使用神经网络结构，在策略和值函数之间共享参数，我们必须使用一个结合策略目标和值函数误差项的损失函数。在此基础上，目标函数可以通过添加Entropy Bonus来进一步确保足够的探索，因此，得到最优化问题：

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

方法一的目标函数　值函数损失项　Entropy Bonus

其中$L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2$

$$L_t^{CLIP}(\theta) = \min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)$$

$$S[\pi_\theta](s_t) = -\sum_i \pi_\theta(a_i \mid s_t)\log \pi_\theta(a_i \mid s_t)$$

- 熵表示分布的混乱程度。在强化学习中，熵(entropy)代表一个动作的不可预测性。熵越大，分布越混乱，动作的随机性越强，模型的探索能力会越强，熵越小，分布越均匀，策略变为确定性，模型不去探索新方案，从而失去更多选择最优方案的机会。一般强化学习的方法是通过优化期望折扣奖励来学习一系列的动作，但这种学习过程容易使策略熵减，模型过早收敛于局部最优，一种解决方案是在每次迭代时引入一项Entropy Bonus正则项，鼓励策略熵增，从而确保模型有足够的探索，逃离局部最优。

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.

## ➢ 方法三：Actor-Critic-Style Algorithm

- 下面确定优势函数$A$的估计量。A3C算法[2]中给出的方法是：

$$\hat{A}_t^{A3C} = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t)$$

令$k = T - t$，则$\hat{A}_t^{A3C} = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i} + \gamma^{T-t} V(s_T) - V(s_t) = -V(s_t) + r_t + \cdots + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T)$

- 将上述公式进行一般化，我们可以使用广义优势估计的截短版本，

$$\hat{A}_t^{PPO} = \sum_{i=0}^{T-t-1} (\gamma\lambda)^i \left( r_{t+i} + \gamma V(s_{t+1+i}) - V(s_{t+i}) \right)$$

当$\lambda = 1$时，

$$\begin{aligned}
\hat{A}_{t,\lambda=1}^{PPO} &= r_t + \gamma V(s_{t+1}) - V(s_t) + \gamma \left( r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1}) \right) + \cdots + \gamma^{T-t-1} \left( r_{T-1} + \gamma V(s_T) - V(s_{T-1}) \right) \\
&= r_t + \cancel{\gamma V(s_{t+1})} - V(s_t) + \gamma r_{t+1} + \cancel{\gamma^2 V(s_{t+2})} - \cancel{\gamma V(s_{t+1})} + \cdots + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T) - \cancel{\gamma^{T-t-1} V(s_{T-1})} \\
&= -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T) \\
&= \hat{A}_t^{A3C}
\end{aligned}$$

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.
[2] V. Mnih, A.Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. ICML, 2016.

➢ 方法三：Actor-Critic-Style Algorithm

**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**

- 上述是一种使用固定长度轨迹段的近端策略优化(PPO)算法。每次迭代，每N个(并行)演员收集T个时间步长的数据。然后，我们在这些N*T时间步上构造替代损失，并使用minibatch SGD(或Adam)对K个epochs进行优化。

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.



## 5. 参考文献

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. Advances in Neural Information Processing Systems, 2017.
[2] Proximal Policy Optimization — Spinning Up documentation https://spinningup.openai.com/en/latest/algorithms/ppo.html

[3] V. Mnih, A.Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. ICML, 2016.
[4] Proximal Policy Optimization Algorithms, slides, https://dvl.in.tum.de/slides/automl-ss19/01_stadler_ppo.pdf