PPO是怎么从Policy gradient演化而来的

Policy gradient策略梯度

之前介绍的Sarsa、Q-learning和DQN都是value-based方法，首先学习动作价值函数，然后根据动作价值函数采用贪心方法更新策略。policy gradient 是policy-based方法，直接对策略进行优化。

假设目标策略 π_θ 是一个随机的、处处可微的策略，用一个神经网络模型初始化，输入为一个状态，输出为动作的概率分布。训练目标是寻找一个最优策略并最大化这个策略在环境中的期望回报，将目标函数定义为：

$$J(\theta) = \mathbb{E}_{s_0}[V^{\pi_\theta}(s_0)]$$

基于目标函数对策略 θ 求导，然后利用**梯度上升**（因为是最大化，所以是梯度上升，如果是最小化损失函数那就是梯度下降）来最大化这个目标（也就是希望采样的都是价值高的状态），就能得到最优策略：

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \sum_{s\in S}\nu^{\pi_\theta}(s)\sum_{a\in A}Q^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a|s)\\ &= \sum_{s\in S}\nu^{\pi_\theta}(s)\sum_{a\in A}\pi_\theta(a|s)Q^{\pi_\theta}(s,a)\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_\theta(a|s)}\\ &= \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta\log\pi_\theta(a|s)]\end{aligned}$$

其中 ν^π 表示状态访问分布，$\nu^\pi(s)$ 表示状态 s 被访问的概率。上面公式证明依靠policy gradient theorem，参考《https://hrl.boyuai.com/chapter/2/%E7%AD%96%E7%95%A5%E6%A2%AF%E5%BA%A6%E7%AE%97%E6%B3%95#93-reinforce》，在此省略，只需要知道最终推导结果即可。

> 因为上式中期望的下标是 π_θ，所以策略梯度算法为on-policy算法，即必须使用当前策略采样得到的数据来计算梯度.

策略梯度的公式中有参数 $Q^{\pi_\theta}(s,a)$，在**REINFORCE 算法**中用**蒙特卡洛算法**来估计，策略梯度修改为下式，其中T是与环境交互的最大步数：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T}\left(\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}\right)\nabla_\theta\log\pi_\theta(a_t|s_t)\right]$$

REINFORCE伪代码：

- 初始化策略参数θ
- **for** 序列 $e = 1 \to E$ **do**：
- 用当前策略π_θ采样轨迹$\{s_1, a_1, r_1, s_2, a_2, r_2, \dots s_T, a_T, r_T\}$
- 计算当前轨迹每个时刻t往后的回报$\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$，记为$\psi_t$
- 对θ进行更新，$\theta = \theta + \alpha \sum_t^T \psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)$
- **end for**

面临问题: 跟蒙特卡洛算法类似，需要完整的episode，episode生成和策略更新不可同时进行，训练效率低，方差大。

> TRPO和PPO实际上是介于On-policy和off-policy之间的算法，PPO的每一轮迭代用 $\pi_{\theta_{old}}$ 与环境交互生成数据。尽管通过重要性采样可多次使用同一批数据（如多个epoch的梯度更新），但策略更新后必须重新采样，无法完全脱离当前策略，与完全的off-policy的完全数据复用不一样。PPO和TRPO是**On-policy 为主，兼具有限 off-policy 特性**。

Actor-Critic

REINFORCE算法用蒙特卡洛方法去估计公式中的 $Q^{\pi_\theta}(s, a)$ 函数，而Actor-Critic算法学习一个Q函数，结合了value-based和policy-based方法的优点，其中actor代表策略模型，critic表示价值函数。

📍 **如何理解Actor和Critic的关系**

- Actor 负责与环境交互采集数据，并在 Critic 价值函数的指导下用策略梯度学习一个更好的策略（类似于策略优化）。
- Critic 通过 Actor 与环境交互收集的数据学习一个价值函数（类似于策略评估），用于判断状态和动作的好坏。

Actor-Crtic包含了一系列的算法，首先将策略梯度写成一般的形式

$$g = \mathbb{E}\left[\sum_{t=0}^{T} \psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

其中 ψ_t 替换了REINFORCE 方法中的 $Q^{\pi_\theta}(s,a)$，作为新的critic模型，用于衡量状态和动作的好坏。常见的形式有：

1. **轨迹的总回报** $\sum_{t'=0}^{T} \gamma^{t'} r_{t'}$：利用完整轨迹的总回报来指导每个时刻的策略更新，在实际中很少使用，因为它是整个轨迹的累积结果，需要整个轨迹计算完才能得到，缺乏即时反馈。此外，对所有时刻都采用同样的critic值，缺乏对时间维度的区分度。
2. **动作 a_t 之后的回报** $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$：这就是**REINFORCE算法**，但是因为每一步的状态转移都有不确定性，而每一步状态采取的动作所得到的不一样的奖励最终都会加起来，这会极大影响最终的价值估计方差较大。同样需要完整的episode，episode生成和策略更新不可同时进行，训练效率较低。
3. **加入基准线的REINFORCE** $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - b(s_t)$：引入基线函数 $b(s_t)$ 来减小方差，通常采用状态价值函数 $V(s_t)$。

✍️ **为什么能降低方差**

回忆在Dueling DQN中出现的优势函数， $A(s,a) = Q(s,a) - V(s)$，这里的 $V(s) = \sum_{a \in A} \pi(a|s) Q(s,a)$,因此优势函数的期望是0，相当于把所有数据做了平移，但这并不能直接降低方差。

之所以能降低方差，是因为将优势函数中心化能降低估计值的波动，因为 $V(s)$ 是一个只与状态相关的函数，减去它可以消除优势函数中的系统性偏差，使得估计值更加稳定。换句话说，如果不减去 $V(s)$，那么 $Q(s,a)$ 的价值会受到环境和动作的影响，环境的随机性会导致 $Q(s,a)$ 有很大的波动范围。而减去 $V(s)$ 后的优势函数，只关注动作之间的好坏关系，降低了波动和方差。

回到加入基准线的REINFORCE，与之同理，$E[\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}]=E(G_t)=V(s_t)$，也是将$\psi_t$的期望变成了0，$V(s_t)$只与状态相关，减去$V(s_t)$就消除了状态的影响，只关注动作之间的好坏关系，降低了波动和方差。

4. **动作价值函数** $Q^{\pi_\theta}(s_t,a_t)$：直接学习Q网络，也就是在状态s_t下采取动作a_t后能获得的期望回报。
5. **优势函数** $A^{\pi_\theta}(s_t,a_t)$：在状态s_t下采取动作a_t相对于当前策略的平均表现的优劣程度，能帮助Actor更高效地更新策略。
6. **时序差分误差** $r_t+\gamma V^{\pi_\theta}(s_{t+1})-V^{\pi_\theta}(s_t)$：优势函数需要同时维护Q和V两个网络，估计不稳定性会更强。时序差分误差只维护V网络（一般提到Actor-Crtic默认使用时序差分误差）。可以在每一步之后都进行更新，并且不对任务的步数做限制。时序差分误差是优势函数的有偏估计，$A^{\pi_\theta}(s_t,a_t)=E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}[r_t+\gamma V^{\pi_\theta}(s_{t+1})-V^{\pi_\theta}(s_t)]$

🥖 **时序差分误差本质就是优势函数用$r_t+\gamma V^{\pi_\theta}(s_{t+1})$替换了$Q(s_t,a_t)$**

根据马尔可夫决策过程的性质，我们有：

$$Q_\pi(s,a)=\sum_{s'}\sum_{r}p(s',r|s,a)[r+\gamma V_\pi(s')]$$

假设$p(s',r|s,a)$不随环境变化，而是固定的值，那么就有

$$Q(s_t,a_t)=r(s_t,a_t)+\gamma V(s_{t+1})$$

在$p(s',r|s,a)$固定的情况下，上面就是对$Q(s_t,a_t)$的无偏估计。但整体来看还是有偏的。

得到了策略梯度后，就可以用**梯度上升法**（因为是Actor的目标函数是最大化，所以是梯度上升）对Actor进行更新，Critic模型V_w的更新采用时序差分误差，损失函数为：

$$L(w)=\frac{1}{2}(r+\gamma V_w(s_{t+1})-V_w(s_t))^2$$

采取类似于**DQN**的目标网络，上式中的$r+\gamma V_w(s_{t+1})$为目标网络，不会产生梯度来更新价值函数。因此损失函数的梯度为：

$$\nabla_w L(w)=-r(r+\gamma V_w(s_{t+1})-V_w(s_t))\nabla_w V_w(s_t)$$

然后使用**梯度下降方法**来更新 Critic 价值网络参数。当Actor模型接近最优策略时，$Q(s_t,a_t)=r+\gamma V_w(s_{t+1})$很接近$V_w(s_t)$，对应的Critic模型的损失很小。

伪代码为：

- 初始化策略网络参数θ，价值网络参数ω
- **for** 序列 $e = 1 \to E$ **do**：
- 用当前策略π_θ采样轨迹$\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots\}$
- 为每一步数据计算: $\delta_t = r_t + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)$
- 更新价值参数$w = w + \alpha_\omega \sum_t \delta_t \nabla_\omega V_\omega(s_t)$
- 更新策略参数$\theta = \theta + \alpha_\theta \sum_t \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$
- **end for**

TRPO

策略梯度算法沿着 $\nabla_\theta J(\theta)$ 策略梯度的方向更新策略模型 θ ，但是当策略网络是深度模型时，沿着策略梯度更新参数，很有可能由于步长太长，策略突然显著变差，进而影响训练效果。TRPO(trust region policy optimization)在更新时找到一块**信任区域**，在这个区域上更新策略就能够保证策略安全更新。

策略目标

给定策略 π_θ ，参数为 θ ，目标是找到一个更优的策略 θ' ，使得 $J(\theta') > J(\theta)$ 。由于初始状态 s_0 与策略无关，将策略梯度的优化目标 $J(\theta)$ 修改为在新策略 $\pi_{\theta'}$ 下的期望回报：

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{s_0}[V^{\pi_\theta}(s_0)] \\
&= \mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t V^{\pi_\theta}(s_t) - \sum_{t=1}^{\infty}\gamma^t V^{\pi_\theta}(s_t)\right] \\
&= -\mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t \left(\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right]
\end{aligned}
$$

新策略和旧策略之间的期望回报之差：

$$
\begin{aligned}
J(\theta') - J(\theta) &= \mathbb{E}_{s_0}\left[V^{\pi_{\theta'}}(s_0)\right] - \mathbb{E}_{s_0}\left[V^{\pi_\theta}(s_0)\right] \\
&= \mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right] + \mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t \left(\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t \left[r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right]\right]
\end{aligned}
$$

> 这里 $E_{s_0}[V^{\pi_{\theta'}}(s_0)] = E_{\pi_{\theta'}}[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)]$ ，因为 θ' 还没有找到，所以也无法得到 $V^{\pi_{\theta'}}$ ，只能通过采样数据得到的奖励来计算。

由优势函数和MDP的性质，$A(s,a)=Q(s,a)-V(s), Q(s,a)=r(s,a)+\gamma V(s')$，可以将时序差分误差定义为优势函数：

$$
\begin{aligned}
&= \mathbb{E}_{\pi_{\theta'}}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_\theta}(s_t,a_t)\right] \\
&= \sum_{t=0}^{\infty}\gamma^t \mathbb{E}_{s_t\sim P_t^{\pi_{\theta'}}}\mathbb{E}_{a_t\sim\pi_{\theta'}(\cdot|s_t)}\left[A^{\pi_\theta}(s_t,a_t)\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\nu^{\pi_{\theta'}}}\mathbb{E}_{a\sim\pi_{\theta'}(\cdot|s)}\left[A^{\pi_\theta}(s,a)\right]
\end{aligned}
$$

> 第二个公式是因为期望里边的值与s和a有关，所以对 $\pi_{\theta'}$ 求期望实际就是在算对s和a的期望。最后一个公式用到状态访问分布的性质。

状态访问分布与占用度量：

定义MDP初始状态下的分布为 $\nu_0(s)$，表示agent的初始状态在状态s的概率，用 $P_t^\pi(s)$ 表示采用策略 π 时agent在t时候状态为s的概率，有 $P_0^\pi(s)=\nu_0(s)$。策略的**状态访问分布**为，表示agent与环境交互时会访问到的状态的分布（这一交互还没有进行，要预测访问到每个状态的概率）：

$$
\nu^\pi(s)=(1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_t^\pi(s)
$$

前边乘以 $1-\gamma$ 是为了保障所有 $\nu^\pi(s)$ 之和为1的归一化因子。理论上在计算该分布时需要交互到无穷步之后，但实际上agent和环境的交互次数有限的，可以用下式来计算状态访问概率：

$$
\nu^\pi(s')=(1-\gamma)\nu_0(s')+\gamma\int P(s'|s,a)\pi(a|s)\nu^\pi(s)ds\,da
$$

类似的定义策略的**占用度量**，表示动作状态对 (s,a) 被访问的概率：

$$
\rho^\pi(s,a)=(1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_t^\pi(s)\pi(a|s)
$$

两者之间的关系为：

$$
\rho^\pi(s,a)=\nu^\pi(s)\pi(a|s)
$$

以上公式表明只要新策略满足 $E_{s\sim\nu^{\pi_{\theta'}}}E_{(a\sim\pi_{\theta'}(\cdot|s))}[A^{\pi_\theta}(s,a)]\geq 0$，就能保证策略性能单调递增，即 $J(\theta')\geq J(\theta)$。但是这里即需要用 $\pi_{\theta'}$ 进行数据采样，又要求解 $\pi_{\theta'}$，不可能把所有的新策略都用来收集数据，再基于收集到的数据判断是否满足上述条件，采样到的数据利用率低下。TRPO采用了近似的方法，当新旧策略非常接近时，状态访问分布变化很小，因此采用旧策略 π_θ 的状态访问分布，这样的 $J(\theta')$ 就变成了：

$$L_\theta(\theta') = J(\theta) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim\nu^{\pi_\theta}}\mathbb{E}_{a\sim\pi_{\theta'}(\cdot|s)}\left[A^{\pi_\theta}(s,a)\right]$$

但是，动作仍然用新策略采样得到。可以用**重要性采样**对动作分布进行处理：

$$L_\theta(\theta') = J(\theta) + \mathbb{E}_{s\sim\nu^{\pi_\theta}}\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}A^{\pi_\theta}(s,a)\right]$$

这样就可以基于旧策略 π_θ 进行数据采样，来估计并优化新策略 $\pi_{\theta'}$ 。为了保证新旧策略足够接近，为优化目标添加了KL散度限制，优化目标为：

$$\begin{aligned}&\max_{\theta'}\ L_\theta(\theta')\\&\text{s.t. }\ \mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}}[D_{KL}(\pi_{\theta_k}(\cdot|s),\pi_{\theta'}(\cdot|s))]\le\delta\end{aligned}$$

这里的限制条件划定了一个**信任区域**，在这个区域中，可以认为新策略和环境交互的状态访问分布与旧策略最后采样的状态访问分布一致，进而可以基于一步行动的重要性采样方法使当前学习策略稳定提升。

🥇 **重要性采样：**

假设我们需要计算 **目标分布** $\pi(a \mid s)$ **下的期望值**，但只能从 **行为分布** $\mu(a \mid s)$ **中采样**。重要性采样通过引入权重（重要性比率）来修正偏差：

$$E_{a\sim\mu}[\frac{\pi(a|s)}{\mu(a|s)}f(a)] = E_{a\sim\pi}[f(a)]$$

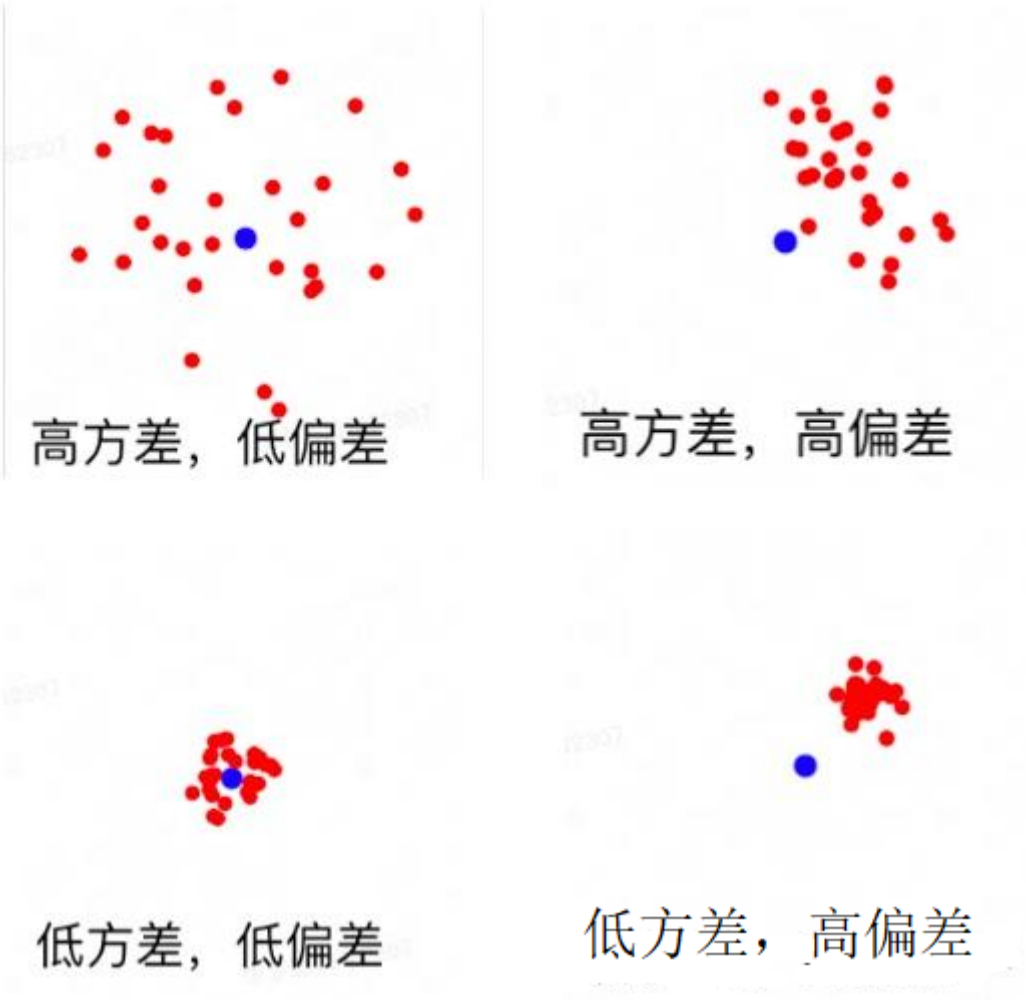
其中 $\frac{\pi(a|s)}{\mu(a|s)}$ 为重要性权重，用于补偿两个分布之间的差异。 $f(a)$ 是待估计的函数。

采用重要性采样可以提升样本效率，因为从 θ 采样的数据可以复用。但要求 π和μ 的差异不能太大，否则方差可能会变得很大，影响估计的准确性（例如PPO中采用截断 $clip(\rho_t, 1-\epsilon, 1+\epsilon)$ 来解决）。

广义优势估计GAE

首先介绍一下方差和偏差：

- **低方差，低偏差**：`E(射击点) = 靶心`，且射击点密集分布在靶心周围。此时我们随机选一个射击点就能很好代表靶心
- **高方差，低偏差**：`E(射击点) = 靶心`，但射击点们离靶心的平均距离较远。此时随机一个射击点不能很好代表靶心，我们必须使用足够多的射击点才能估计靶心的坐标
- **高/低方差，高偏差**：`E(射击点)!=靶心`，无论你做多少次射击，你都估计不准靶心的位置。



优势函数 $A^{\pi_\theta}(s_t,a_t)=E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}[r_t+\gamma V^{\pi_\theta}(s_{t+1})-V^{\pi_\theta}(s_t)]$，$A^{\pi_\theta}(s_t,a_t)$ 就是蓝色的靶心，红色的点就是 (s_t,a_t) 的多次采样，每次采样得到 $r_t+\gamma V^{\pi_\theta}(s_{t+1})-V^{\pi_\theta}(s_t)$。由于 $V^{\pi_\theta}(s_{t+1})$ 用下一个状态的价值来估计真实的价值，是有偏差的，对应上图右下。可以将其进行递归展开得到：

$$-V^{\pi_\theta}(s_t)+\sum_{l=0}^{\infty}\gamma^l r_{t+l}$$

其中r都是通过采样得到的即时奖励（也就是**蒙特卡洛算法**），这些数据没有偏差。但是这些r都是随机变量，引入更大的随机性导致方差变大，对应上图左上。

为了平衡偏差和方差，采用**广义优势估计**来估计优势函数，用 $\delta=r_t+\gamma V(s_{t+1})-V(s_t)$ 表示时序差分误差，V是一个已经学习的状态价值函数，采用**多步Sarsa**的思想：

> ❤️ **回忆下多步Sarsa**:
>
> 传统优势函数估计基于**时序差分**或者**蒙特卡洛算法**，但是蒙特卡洛方法具有比较大的方差，时序差分算法则是有偏的。**多步Sarsa**采用两者折中的算法，同时减少了方差和偏差，提高了估计的准确度。

$$
\begin{aligned}
A_t^{(1)} &= \delta_t &&= -V(s_t) + r_t + \gamma V(s_{t+1}) \\
A_t^{(2)} &= \delta_t + \gamma\delta_{t+1} &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
A_t^{(3)} &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \\
&\vdots &&\vdots \\
A_t^{(k)} &= \sum_{l=0}^{k-1} \gamma^l \delta_{t+l} &&= -V(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})
\end{aligned}
$$

然后，GAE 将这些不同步数的优势估计进行指数加权平均：

$$
\begin{aligned}
A_t^{GAE} &= (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots) \\
&= (1-\lambda)(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2}) + \cdots) \\
&= (1-\lambda)(\delta(1 + \lambda + \lambda^2 + \cdots) + \gamma\delta_{t+1}(\lambda + \lambda^2 + \lambda^3 + \cdots) + \gamma^2\delta_{t+2}(\lambda^2 + \lambda^3 + \lambda^4 + \ldots) + \cdots) \\
&= (1-\lambda)\left(\delta_t \frac{1}{1-\lambda} + \gamma\delta_{t+1}\frac{\lambda}{1-\lambda} + \gamma^2\delta_{t+2}\frac{\lambda^2}{1-\lambda} + \cdots\right) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}
\end{aligned}
$$

其中 $\lambda \in [0, 1]$ 控制每一步的差分的重要性。当 $\lambda = 0$ 时表示只看一步差分的优势，$A_t^{GAE} = \delta_t = r_t + \gamma V(s_{t+1} - V(s_t))$，对应高偏差-低方差；$\lambda = 1$ 表示每一步差分的重要性都一样，$A_t^{GAE} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t)$，对应高方差-低偏差情况。

TRPO面临问题：存在限制条件，要求参考策略和目标策略距离接近，难以优化。PPO消除了限制条件

2.2.9 PPO

传统的PPO

> **传统的PPO**
>
> 这里的传统PPO指的是agent和环境进行交互（例如机器人），因此奖励也是环境即时给的，也就是每一个动作都会有一个奖励，不需要额外训练一个奖励模型。而之后的RLHF-PPO的奖励信号来源于人类反馈（例如对话生成任务），需要额外设计奖励模型（无论是基于规则的奖励还是用深度学习模拟），奖励往往是在序列结束时给出，并且在奖励中引入了额外的KL-散度限制。
>
> 传统的PPO与Actor-Critic方法一样，只有**Actor**和**Critic**两个模型，而RLHF-PPO额外引入了**Reference**模型（用于约束策略模型输出，防止偏差）和**Reward**模型（对完整序列（如生成的文本）进行评分）。

PPO直接对策略更新的幅度进行限制，为了消除限制条件有两种形式，PPO-penalty和 PPO-clip：

- **PPO-penalty**：用**拉格朗日乘数法**直接将 **KL 散度**的限制放进了目标函数中，这就变成了一个无约束的优化问题，在迭代的过程中不断更新 KL 散度前的系数 β ，定义KL-散度的阈值 KL_{max}, KL_{min}
- 当 $KL \geq KL_{max}$ 时，说明当前策略已经偏离old策略较远了，这时我们应该增大 β ，把分布拉回来。
- 当 $KL \leq KL_{min}$ 时，说明当前策略很可能找到了一条捷径，即它只优化KL散度一项，让自己和old更相近，而不去优化前面优势相关的项，所以这时我们应该减小 β ，降低KL散度一项的影响

$$\arg\max_{\theta} \mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}} \mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) - \beta D_{KL}[\pi_{\theta_k}(\cdot|s), \pi_\theta(\cdot|s)] \right]$$

- **PPO-clip（PPO默认都采用这种方式）：**

在目标函数中进行限制，以保证新的参数和旧的参数的差距不会太大，优化目标为：

$$\arg\max_{\theta} \mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}} \mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)} \left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right) \right]$$

其中 π_θ 表示Actor模型，优势函数作为Critic模型，由Critic模型确定Actor模型优化的方向，π_{θ_k} 表示旧的策略模型，由 π_{θ_k} 负责数据采样。 $clip(x,l,r) = max(min(x,r),l)$ ，将 x 限制在区间 $[l,r]$ 之内， ϵ 控制截断的范围。

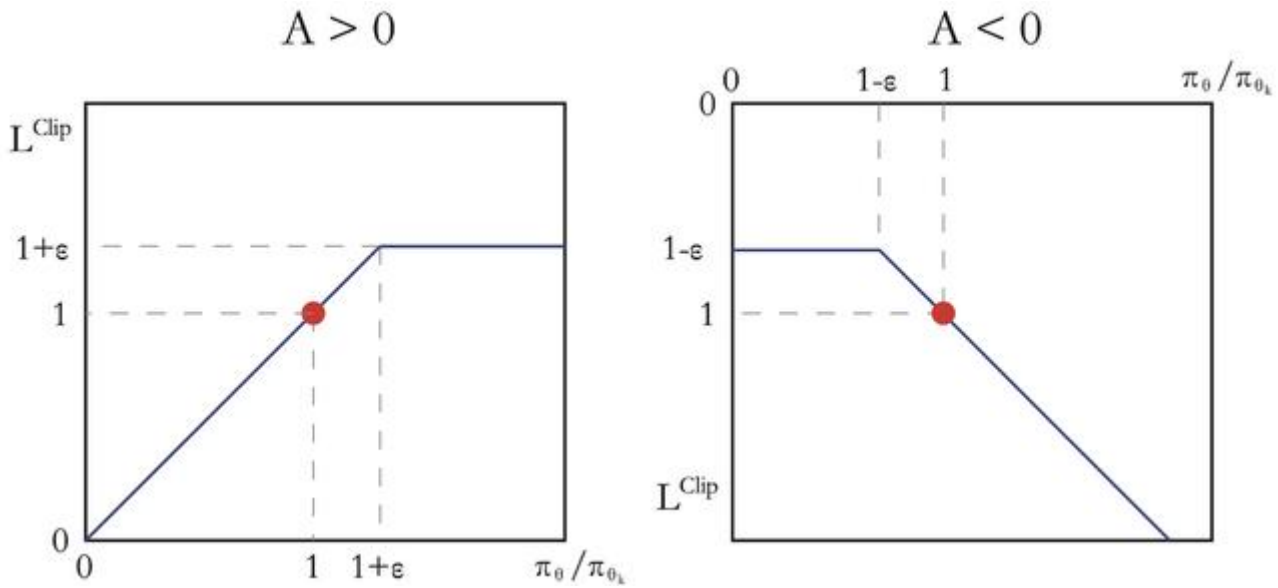
- 如果 $A^{\pi_{\theta_k}(s,a)} > 0$ ，说明这个动作的价值高于平均，最大化这个式子会增大 $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ ，也就是加大在状态s下采取行动a的概率，但不会让其超过 $1+\epsilon$ 。
- 反之，如果 $A^{\pi_{\theta_k}(s,a)} < 0$ ，最大化这个式子会减小 $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ ，也就是减小在状态s下采取行动a的概率，但不会让其小于 $1-\epsilon$ 。

接下来讲解PPO中的一些具体细节，如果不采用这些技术，那么PPO的最基础版本会是如下格式：

$$arg\max_{\theta} E_{s\sim\nu^{\pi_\theta}} E_{a\sim\pi_\theta(\cdot|s)}[\pi_\theta(a|s)A^{\pi_\theta}(s,a))]$$

- **重要性采样**：最基础版本的PPO是**on-policy**的，既要 π_θ 采样也要更新，为了提升数据利用率，同TRPO中同样的**近似方法**，优化目标修改为 $arg\max_{\theta} E_{s\sim\nu^{\pi_{\theta_k}}} E_{a\sim\pi_\theta(\cdot|s)}[\pi_\theta(a|s)A^{\pi_\theta}(s,a))]$ 。再采用TRPO中**重要性采样**，利用 π_{θ_k} 产生的数据来更新 π_θ ，优化目标修改为 $arg\max_{\theta} E_{s\sim\nu^{\pi_{\theta_k}}} E_{a\sim\pi_{\theta_k}(\cdot|s)}[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_\theta}(s,a))]$ ，PPO也就可以重复利用之前策略生成的数据了。

- **策略截断**：为了消除TRPO中的限制条件（实际上，TRPO的求解比较复杂，感兴趣的可以参考《https://hrl.boyuai.com/chapter/2/trpo%E7%AE%97%E6%B3%95#113-%E8%BF%91%E4%BC%BC%E6%B1%82%E8%A7%A3》中的11.3-11.5），PPO首先定义了 $r(\theta)=\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ ，并对 $r(\theta)$ 采取了**clip**操作，控制其更新幅度再 $[1-\epsilon,1+\epsilon]$ 之间，确保更新不会偏离参考模型过远。变成了最终的优化目标。



- **GAE**：采用TRPO中同样的GAE来估计优势函数

> 注意， π_{θ_k} 与 π_{ref} 不一样， π_{ref} 是参考模型，是为了避免策略模型偏离太远的。 π_{θ_k} 是上一轮的策略模型，是用来采样数据的，用来更新当前轮的策略模型 π 。

PPO的训练过程与Actor-Crtic一样，也是先更新策略模型，再更新评估模型。以上更新完策略模型后(也就是用一个batch中的trajectory都更新完策略模型)，定义评估模型的损失。设置待更新的评估模型为 V_t^{old} ，由GAE的定义，递归展开有：

$$A_t^{GAE}=\delta_t+\gamma A_{t+1}^{GAE}=(r_t+\gamma V_{t+1}^{old}-V_t^{old})+\gamma\lambda A_{t+1}^{GAE}$$

$$R_t=A_t^{GAE}+V_t^{old}=(r_t+\gamma V_{t+1}^{old})+\gamma\lambda A_{t+1}^{GAE}$$

优势可以从后往前推导出每个动作的优势。评估模型的损失函数为：

$$L_{loss}=((V_t^{new}-R_t)^2=[(V_t^{new}-(r_t+\gamma V_{t+1}^{old}))-\gamma\lambda A_{t+1}^{GAE}]^2$$

与Actor-Critic模型的评估模型损失一致，只不过这里引入了GAE对优势做了指数平滑而已。在此基础上还加了**clip**操作 $V_t^{clip}=clip(V_t^{new},V_t^{old}+\epsilon,V_t^{old}-\epsilon)$ ，类似于策略模型中的clip操作。