TRPO

基于策略的优化目标为：

$$J(\theta) = E_{s_0}[V^{\pi_\theta}(s_0)] = E_{\pi_\theta}[\sum_{t=0}^{\inf} \gamma^t r(s_t, a_t)]$$

第一个E可以理解为在不同初始状态分布下使用策略π_θ的价值期望，即初始状态不同得到的期望价值。

第二个E是将V展开成$E_{\pi_\theta}[G_t|s_0]$的形式，最后的结果省略了s0，但其实还是隐含着初始状态分布。

TRPO的想法是设立一个信任区域，使得在这个信任区域上进行策略更新时能够保证策略是越来越好的。

公式推导

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

这里其实下标无所谓的，沿用s0也可以，因为期望内的部分跟$\pi_{\theta'}$五关。重点是将里面用两项差的形式展开了。

基于以上等式，我们可以推导新旧策略的目标函数之间的差距：

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

这里就是构造出了时序差分的形式，将其定义为优势函数A，如果优势函数A>0，表示当前动作比之前更优，反之更劣，因为优势函数前两项是由Gt变换来的，V（st）则是表示之前在该状态产生的价值。

将时序差分残差定义为优势函数A：

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

最后一个等号的成立运用到了状态访问分布的定义：$\nu^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_t^\pi(s)$，所以只要我们能找到一个新策略，使得$\mathbb{E}_{s \sim \nu^{\pi_{\theta'}}} \mathbb{E}_{a \sim \pi_{\theta'}(\cdot|s)}\left[A^{\pi_\theta}(s, a)\right] \geq 0$，就能保证策略性能单调递增，即$J(\theta') \geq J(\theta)$。

状态访问分布表示在策略π下到达状态s的概率。

但是直接求解该式是非常困难的，因为$\pi_{\theta'}$是我们需要求解的策略，但我们又要用它来收集样本。把所有可能的新策略都拿来收集数据，然后判断哪个策略满足上述条件的做法显然是不现实的。于是 TRPO 做了一步近似操作，对状态访问分布进行了相应处理。具体而言，忽略两个策略之间的状态访问分布变化，直接采用旧的策略π_θ的状态分布，定义如下替代优化目标：

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

使用当前策略的状态访问分布替代新策略的状态访问分布，即在每个状态访问的概率。

当新旧策略非常接近时，状态访问分布变化很小，这么近似是合理的。其中，动作仍然用新策略$\pi_{\theta'}$采样得到，我们可以用重要性采样对动作分布进行处理：

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

重要性采样可以表示两个策略的差异，这里通过引入这个将动作a也变为从当前策略π_θ下选取。

这样，我们就可以基于旧策略π_θ已经采样出的数据来估计并优化新策略$\pi_{\theta'}$了。为了保证新旧策略足够接近，TRPO 使用了**库尔贝克-莱布勒**（Kullback-Leibler，KL）散度来衡量策略之间的距离，并给出了整体的优化公式：

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

KL散度就是评估两个分布之间的变化量的，这里要求两个分布变化量在一定区间内，即“信任区域”。因为如果变化太大的话新旧策略之间的状态访问分布就不能近似看成一样的了。

求解

最终的更新方程其实是共轭梯度那步关于x的式子，而要求x则要用到g和H，也就需要求优势函数和KL散度。

11.3 近似求解

直接求解上式带约束的优化问题比较麻烦，TRPO 在其具体实现中做了一步近似操作来快速求解。为方便起见，我们在接下来的式子中用θ_k代替之前的θ，表示这是第k次迭代之后的策略。首先对目标函数和约束在θ_k进行泰勒展开，分别用 1 阶、2 阶进行近似：

$$\mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}}\mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)}\left[\frac{\pi_{\theta'}(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \approx g^T(\theta'-\theta_k)$$

$$\mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}}[D_{KL}(\pi_{\theta_k}(\cdot|s),\pi_{\theta'}(\cdot|s))] \approx \frac{1}{2}(\theta'-\theta_k)^T H(\theta'-\theta_k)$$

其中 $g = \nabla_{\theta'}\mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}}\mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)}\left[\frac{\pi_{\theta'}(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right]$，表示目标函数的梯度，$H = \mathbf{H}[\mathbb{E}_{s\sim\nu^{\pi_{\theta_k}}}[D_{KL}(\pi_{\theta_k}(\cdot|s),\pi_{\theta'}(\cdot|s))]]$表示策略之间平均 KL 距离的**黑塞矩阵**（Hessian matrix）。

于是我们的优化目标变成了：

$$\theta_{k+1} = \arg\max_{\theta'} g^T(\theta'-\theta_k) \quad \text{s.t.} \quad \frac{1}{2}(\theta'-\theta_k)^T H(\theta'-\theta_k) \le \delta$$

此时，我们可以用**卡罗需-库恩-塔克**（Karush-Kuhn-Tucker，KKT）条件直接导出上述问题的解：

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}}H^{-1}g$$

11.4 共轭梯度

一般来说，用神经网络表示的策略函数的参数数量都是成千上万的，计算和存储黑塞矩阵H的逆矩阵会耗费大量的内存资源和时间。TRPO 通过**共轭梯度法**（conjugate gradient method）回避了这个问题，它的核心思想是直接计算 $x = H^{-1}g$，x即参数更新方向。假设满足 KL 距离约束的参数更新时的最大步长为β，于是，根据 KL 距离约束条件，有$\frac{1}{2}(\beta x)^T H(\beta x) = \delta$。求解$\beta$，得到$\beta = \sqrt{\frac{2\delta}{x^T H x}}$。因此，此时参数更新方式为

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{x^T H x}}x$$

11.5 线性搜索

由于 TRPO 算法用到了泰勒展开的 1 阶和 2 阶近似，这并非精准求解，因此，θ'可能未必比θ_k好，或未必能满足 KL 散度限制。TRPO 在每次迭代的最后进行一次**线性搜索**（Line Search），以确保找到满足条件。具体来说，就是找到一个最小的非负整数i，使得按照

$$\theta_{k+1} = \theta_k + \alpha^i \sqrt{\frac{2\delta}{x^T H x}} x$$

求出的θ_{k+1}依然满足最初的 KL 散度限制，并且确实能够提升目标函数L_{θ_k}，这其中$\alpha \in (0,1)$是一个决定线性搜索长度的超参数。

至此，我们已经基本上清楚了 TRPO 算法的大致过程，它具体的算法流程如下：

- 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$\{s_1, a_1, r_1, s_2, a_2, r_2, \dots\}$
- 根据收集到的数据和价值网络估计每个状态动作对的优势$A(s_t, a_t)$
- 计算策略目标函数的梯度g
- 用共轭梯度法计算$x = H^{-1} g$
- 用线性搜索找到一个i值，并更新策略网络参数$\theta_{k+1} = \theta_k + \alpha^i \sqrt{\frac{2\delta}{x^T H x}} x$，其中$i \in \{1, 2, \dots, K\}$为能提升策略并满足 KL 距离限制的最小整数
- 更新价值网络参数（与 Actor-Critic 中的更新方法相同）
- **end for**

优势函数求解

11.6 广义优势估计

从 11.5 节中，我们尚未得知如何估计优势函数A。目前比较常用的一种方法为**广义优势估计**（Generalized Advantage Estimation，GAE），接下来我们简单介绍一下 GAE 的做法。首先，用$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$表示时序差分误差，其中$V$是一个已经学习的状态价值函数。于是，根据多步时序差分的思想，有：

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

然后，GAE 将这些不同步数的优势估计进行指数加权平均：

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

其中，$\lambda \in [0,1]$是在 GAE 中额外引入的一个超参数。当$\lambda = 0$时，$A_t^{GAE} = \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$，也即是仅仅只看一步差分得到的优势；当$\lambda = 1$时，$A_t^{GAE} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t)$，则是看每一步差分得到优势的完全平均值。

```
def compute_advantage(gamma, lmbda, td_delta):
    td_delta = td_delta.detach().numpy()
    advantage_list = []
    advantage = 0.0
    for delta in td_delta[::-1]:
        advantage = gamma * lmbda * advantage + delta
        advantage_list.append(advantage)
    advantage_list.reverse()
    return torch.tensor(advantage_list, dtype=torch.float)
```

这段代码是在根据GAE公式逆向求解，即

$$A_t = \delta_t + \gamma\lambda A_{t+1}$$