参考资料：Proximal Policy Optimization (PPO) 算法理解：从策略梯度开始

# 传统策略梯度算法

## 策略近似

设$\theta$为神经网络参数，基于策略的强化学习用参数化概率分布$\pi_\theta(a|s) = P(a|s;\theta)$确定策略，在返回的动作概率列表中对不同的动作进行抽样选择。

## 定义目标函数

目标就是找到那些可能获得更多奖励期望值的动作，使它们对应的概率更大，从而策略就更有可能选择这些动作。

定义的最大化目标函数：

$$\max_\theta J(\theta) = \max_\theta E_{\tau \sim \pi_\theta}(R(\tau)) = \max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

其中$\tau$是agent与环境交互产生的状态-动作轨迹$\tau = (s_1, a_1, \ldots, s_T, a_T)$。我们的目标是通过调整$\theta$，使得获得更大奖励期望的轨迹出现的概率更高。

其中，轨迹$\tau$在策略$\pi_\theta(a|s)$下发生的概率为：

$$P(\tau;\theta) = \left[\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t) \cdot \pi_\theta(a_t|s_t)\right]$$

实际枚举所有可能的轨迹是很困难的，基本都需要通过大量采样得到样本求期望值近似。

# 策略梯度

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau;\theta) \cdot R(\tau)$$

$$= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau)$$

$$= \sum_\tau P(\tau;\theta) \cdot \nabla_\theta \log P(\tau;\theta) \cdot R(\tau)$$

根据$P(\tau;\theta)$可得：

$$\nabla_\theta \log P(\tau;\theta) = \nabla_\theta \left[ \sum_{t=0}^{T} \log P(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) \right]$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

假设当前有$m$条轨迹的样本：

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) \cdot R(\tau^{(i)})$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[ \sum_{t^{(i)}=0}^{T^{(i)}} \nabla_\theta \log \pi_\theta(a_{t^{(i)}}|s_{t^{(i)}}) \right] \cdot R(\tau^{(i)})$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot R(\tau_i) \quad \left( n = \sum_{i=1}^{m} (T^{(i)} + 1) \right)$$

策略梯度的更新规则：

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$$

## Softmax策略

对于离散动作空间：

$$\pi_\theta(a|s) = \frac{e^{\phi(s,a)^T\theta}}{\sum_{a'\in A} e^{\phi(s,a')^T\theta}}$$

对应的策略梯度：

$$\nabla_\theta \log \pi_\theta(a|s)$$

$$= \nabla_\theta \left(\phi(s,a)^T\theta - \log \sum_{a'\in A} e^{\phi(s,a')^T\theta}\right)$$

$$= \phi(s,a) - \frac{\sum_{a'\in A} \phi(s,a') \cdot e^{\phi(s,a')^T\theta}}{\sum_{a'\in A} e^{\phi(s,a')^T\theta}}$$

$$= \phi(s,a) - \sum_{a'\in A} \phi(s,a') \cdot \pi_\theta(a'|s)$$

如果奖励信号很高并且观察到的向量与平均向量相差很大，就会有增加该动作概率的强烈趋势。

## 高斯策略

对于连续动作空间：

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi}\sigma_\theta} e^{-\frac{(a-\mu_\theta)^2}{2\sigma_\theta^2}}$$

其中正态分布的均值$\mu_\theta = \phi(s,a)^T\theta$。

对应的策略梯度：

$$\nabla_\theta \log \pi_\theta(a|s)$$

$$= \nabla_\theta \left( -\frac{1}{2} \cdot \log(2\pi\sigma_\theta^2) - \frac{(a-\mu_\theta)^2}{2\sigma_\theta^2} \right)$$

$$= \frac{(a-\mu_\theta)\phi(s,a)}{\sigma_\theta^2}$$

在高回报的情况下，远离均值的动作会触发强烈的更新信号。

在实际任务中，我们没有必要手动计算偏导数，使用深度学习框架的自动求导。定义损失函数：

$$\mathcal{L}(a,s,r) = -\log(\pi_\theta(a|s))r$$

即可让计算机自动求导。

## 算法实现

```
Network: theta := R^(|theta|)
for n = 1 to N:
    tau <- pi(theta)
    for t = 1 to T:
        R(tau | t) = R[t] + R[t + 1] + ... + R[T]
        theta <- theta - alpha * R(tau | t) * grad(theta, pi(theta, a | s))
```

这里对于一条路径，将路径上每一个状态都进行了计算，对信息的利用最大化。

# 自然策略梯度算法

## 传统策略梯度算法的缺陷

在传统的策略梯度算法中，权重更新会遇到两个问题：

- 过冲（Overshooting）：更新错过了奖励峰值并落入了次优策略区域
- 下冲（Undershooting）：在梯度方向上采取过小的更新步长会导致收敛缓慢

在监督学习问题中，overshooting不是什么问题，因为数据是固定的，我们可以在下一个epoch中重新纠正；但在强化学习问题中，如果因为overshooting陷入了一个较差的策略区域，则未来的样本批次可

能不会提供太多有意义的信息，用较差的数据样本再去更新策略，从而陷入了糟糕的正反馈中无法恢复。较小的学习率可能会解决这个问题，但会导致收敛速度变慢的undershooting问题。

# 限制策略更新的差异

我们需要表示策略（分布）之间的差异，而不是参数本身的差异。计算两个概率分布之间的差异，最常见的是KL散度，也称为相对熵，描述了两个概率分布之间的距离：

$$\mathcal{D}_{KL}(\pi_\theta||\pi_{\theta+\Delta\theta}) = \sum_{x\in \mathrm{X}} \pi_\theta(x) \log\left(\frac{\pi_\theta(x)}{\pi_{\theta+\Delta\theta}(x)}\right)$$

调整后的策略更新限制为：

$$\Delta\theta^* = \underset{\Delta\theta, \mathcal{D}_{KL}(\pi_\theta||\pi_{\theta+\Delta\theta})\leq\epsilon}{\arg\max} J(\theta+\Delta\theta)$$

然而，计算KL散度需要遍历所有的状态-动作对，因此我们需要一些化简来处理现实的RL问题。

首先，我们使用拉格朗日松弛将原表达式的发散约束转化为惩罚项，得到一个更容易求解的表达式：

$$\Delta\theta^* = \underset{\Delta\theta}{\arg\max}\, J(\theta+\Delta\theta) - \lambda(\mathcal{D}_{KL}(\pi_\theta||\pi_{\theta+\Delta\theta}) - \epsilon)$$

用近似方法来化简。通过泰勒展开：

$$\Delta\theta^* \approx \underset{\Delta\theta}{\arg\max}\, J(\theta) + \nabla_\theta J(\theta)\cdot\Delta\theta - \frac{1}{2}\lambda(\Delta\theta^T F(\theta)\Delta\theta) + \lambda\epsilon$$

$$\approx \underset{\Delta\theta}{\arg\max}\, \nabla_\theta J(\theta)\cdot\Delta\theta - \frac{1}{2}\lambda(\Delta\theta^T F(\theta)\Delta\theta)$$

$$F(\theta) = \mathbb{E}_\theta[\nabla_\theta \log\pi_\theta(x)\nabla_\theta \log\pi_\theta(x)^T]$$

KL散度近似于二阶泰勒展开。用Fisher信息矩阵代替二阶导数，除了符号紧凑性外，还可以大大减少计算开销。

# 解决KL约束问题

对于近似简化后的表达式，可以通过将关于$\Delta\theta$的梯度设置为0，来找到最优的权重更新$\Delta\theta$：

$$0 = \frac{\partial}{\partial \Delta\theta}\left(\nabla_\theta J(\theta)\Delta\theta - \frac{1}{2}\lambda\Delta\theta^T F(\theta)\Delta\theta\right)$$

$$= \nabla_\theta J(\theta) - \frac{1}{2}\lambda F(\theta)\Delta\theta$$

$$\Delta\theta = -\frac{2}{\lambda}F(\theta)^{-1}\nabla_\theta J(\theta)$$

其中，$\lambda$是一个常数，可以吸收到学习率$\alpha$中。根据$\mathcal{D}_{KL}(\pi_\theta||\pi_{\theta+\Delta\theta}) \le \epsilon$，我们可以推出动态学习率：

$$\alpha = \sqrt{\frac{2\epsilon}{\nabla J(\theta)^T}F(\theta)^{-1}\nabla J(\theta)}$$

可以确保每次更新的KL散度（近似）等于$\epsilon$。

自然策略梯度：

$$\tilde{\nabla}J(\theta) = F(\theta)^{-1}\nabla J(\theta)$$

最终的权重更新方案为：

$$\Delta\theta = \sqrt{\frac{2\epsilon}{\nabla J(\theta)^T F(\theta)^{-1}\nabla J(\theta)}}\tilde{\nabla}J(\theta)$$

该方案的强大之处在于，无论分布的表示如何，它总是以相同的幅度改变策略。

# 信赖域策略优化算法（TRPO）

## 自然策略梯度算法的缺陷

- 近似值可能会违反KL约束，从而导致分析得出的步长过大，超出限制要求
- 矩阵$F$的计算时间太长，是$O(N^3)$复杂度的运算
- 我们没有检查更新是否真的改进了策略。由于存在大量的近似过程，策略可能并没有优化

# 算法理论

针对自然策略梯度算法的问题，我们希望可以对策略的优化进行量化，从而保证每次的更新一定是优化作用的。为此，我们需要计算两种策略之间预期回报的差异。这里采用的是原策略预期回报添加新策略预期优势的方式。该表达式在原策略下计算优势函数，无需重新采样：

$$J(\pi_{\theta+\Delta\theta}) = J(\pi_\theta) + \mathbb{E}_{\tau\sim\pi_{\theta+\Delta\theta}} \sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t)$$

其中优势函数的定义为：

$$A^{\pi_\theta}(s, a) = \mathbb{E}(Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s))$$

由于时间范围是无限的，引入状态的折扣分布：

$$\rho_\pi(s) = \sum_{k=0}^{\infty} \gamma^k P(s_k = s)$$

原差异表达式可重新表示为：

$$J(\pi_{\theta+\Delta\theta}) = J(\pi_\theta) + \sum_{s\in\mathcal{S}} \rho_{\pi_{\theta+\Delta\theta}}(s) \sum_{a\in\mathcal{A}} \pi_{\theta+\Delta\theta}(a|s) A^{\pi_\theta}(s, a)$$

引入近似误差，使用当前策略近似：

$$J(\pi_{\theta+\Delta\theta}) \approx J(\pi_\theta) + \sum_{s\in\mathcal{S}} \rho_{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_{\theta+\Delta\theta}(a|s) A^{\pi_\theta}(s, a)$$

将状态分布求和替换为期望，方便实际计算时使用蒙特卡洛模拟进行采样，同时将动作求和替换为重要性采样。通过重要性采样，可以有效利用当前策略的行动期望，并针对新策略下的概率进行了修正：

$$J(\pi_{\theta+\Delta\theta}) = J(\pi_\theta) + \sum_{s\in\mathcal{S}} \rho_{\pi_{\theta+\Delta\theta}}(s) \sum_{a\in\mathcal{A}} \pi_{\theta+\Delta\theta}(a|s) A^{\pi_\theta}(s, a)$$

$$= J(\pi_\theta) + \mathbb{E}_{s\sim\rho_{\pi_{\theta+\Delta\theta}}} \pi_{\theta+\Delta\theta}(a|s) A^{\pi_\theta}(s, a)$$

$$\approx J(\pi_\theta) + \mathbb{E}_{s\sim\rho_{\pi_\theta}} \frac{\pi_{\theta+\Delta\theta}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s, a)$$

描述更新策略相对于原策略的预期优势称为替代优势：

$$J(\pi_{\theta+\Delta\theta}) - J(\pi_\theta) \approx \mathbb{E}_{s\sim\rho_{\pi_\theta}} \frac{\pi_{\theta+\Delta\theta}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s,a) = \mathcal{L}_{\pi_\theta}(\pi_{\theta+\Delta\theta})$$

之前产生的近似误差可以用两种策略之间最坏情况的KL散度表示：

$$J(\pi_{\theta+\Delta\theta}) - J(\pi_\theta) \geq \mathcal{L}_{\pi_\theta}(\pi_{\theta+\Delta\theta}) - C\mathcal{D}_{KL}^{\max}(\pi_\theta||\pi_{\theta+\Delta\theta})$$

论文中推导出$C$的值以及目标函数改进的下限。如果我们改进右侧，可以保证左侧也得到改进。本质上，如果替代优势$\mathcal{L}_{\pi_\theta}(\pi_{\theta+\Delta\theta})$超过最坏情况下的近似误差$C\mathcal{D}_{KL}^{\max}(\pi_\theta||\pi_{\theta+\Delta\theta})$，我们一定会改进目标。

这就是**单调改进定理**。相应的过程是**最小化最大化算法（MM）**。即如果我们改进下限，我们也会将目标改进至少相同的量。

## 算法实现

在实际的算法实现方面，TRPO和自然策略梯度算法没有太大的区别。TRPO的核心是利用单调改进定理，验证更新是否真正改进了我们的策略。

咕咕咕

# 近端策略优化算法（PPO）

## TRPO算法的缺陷

- 无法处理大参数矩阵
- 二阶优化很慢
- TRPO 很复杂

## PPO Penalty

TRPO在理论分析上推导出与KL散度相乘的惩罚项，但在实践中，这种惩罚往往过于严格，只产生非常小的更新。因此，问题是如何可靠地确定缩放参数$\beta$，同时避免overshooting：

$$\Delta\theta^* = \arg\max_{\Delta\theta} \mathcal{L}_{\theta+\Delta\theta}(\theta+\Delta\theta) - \beta\mathcal{D}_{KL}(\pi_\theta||\pi_{\theta+\Delta\theta})$$

PPO通过设置目标散度$\delta$的方式解决了这个问题，希望我们的每次更新都位于目标散度附近的某个地方。目标散度应该大到足以显著改变策略，但又应该小到足以使更新稳定。

每次更新后，PPO都会检查更新的大小。如果最终更新的散度超过目标散度的1.5倍，则下一次迭代我们将加倍$\beta$来更加重惩罚。相反，如果更新太小，我们将$\beta$减半，从而有效地扩大信任区域。迭代更新的思路与TRPO线搜索有一些相似之处，但PPO搜索是在两个方向上都有效的，而TRPO是单向减小的。

只是基于启发式确定的。根据经验，PPO对数值设置是非常不敏感的。总之，我们牺牲了一些数学上的严谨性来使实际的效果更好。

```
Input: initial policy parameters theta_0, initial KL penalty beta_0, target KL-divergence delta
for k = 0, 1, 2, ... do
    Collect set of partial trajectories D_k on policy pi_k = pi_{theta_k}
    Estimate advantages A_t^{pi_k} using any advantage estimation algorithm
    Compute policy update:
        theta_{k + 1} = argmax(theta, L_{theta_k}(theta) - beta_k * D_{KL}(theta || theta_k))
    by taking K steps of minibatch SGD (via Adam)
    if D_{KL}(theta_{k + 1} || theta_k) >= 1.5 delta then
        beta_{k + 1} = beta_{k} * 2
    else if D_{KL}(theta_{k + 1} || theta_k) <= delta / 1.5 then
        beta_{k + 1} = beta_{k} / 2
    else
        beta_{k + 1} = beta_k
```

# PPO Clip

与其费心随着时间的推移改变惩罚，PPO Clip直接限制策略可以改变的范围。我们重新定义了替代优势：

$$\mathcal{L}_{\pi_\theta}^{CLIP}(\pi_{\theta_k}) \\ = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}[\min\left(\rho_t(\pi_\theta,\pi_{\theta_k})A_t^{\pi_{\theta_k}}, clip(\rho_t(\pi_\theta,\pi_{\theta_k}),1-\epsilon,1+\epsilon)A_t^{\pi_{\theta_k}}\right)]\right]$$

$\rho_t$ 为重要性采样：

$$\rho_t(\theta)=\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$$

为了实现想要达到的效果，我们应该调整$\epsilon$，作为对KL散度的隐式限制。根据经验，$\epsilon=0.1 or 0.2$是实际效果较好的值。

# PPO2

PPO2是Open AI发布的算法更新版本，是矢量化环境的PPO算法实现，针对 GPU 进行了优化，更好地支持并行训练。它与PPO也有许多实际实现的差异，例如优势被自动归一化、价值函数被裁剪等，但与本文概述的PPO具有相同的数学基础。如果需要直接使用OpenAI实现的PPO算法，则应该使用PPO2。