

# 深度强化学习各种经典算法总结 2

## 目录

[一. A2C](#)

[二. PPO](#)

## A2C算法

### 策略与价值函数

- 策略:  $\pi_\theta(a|s)$ , 参数为 $\theta$
- 状态价值函数:  $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s]$
- 动作价值函数:  $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, a_t = a]$
- 优势函数:  $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$

### 目标函数

目标是最大化期望累积奖励:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$$

其中 $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots)$ 表示轨迹。

### 策略梯度定理

策略梯度定理告诉我们如何计算目标函数的梯度:

定理:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \right]$$

推导过程:

轨迹概率:

$$P(\tau|\theta) = P(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

轨迹概率的对数梯度:

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

**目标函数的梯度：**

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \nabla_{\theta} \log P(\tau | \theta)]$$

$$= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ R(\tau) \sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

**使用Q函数重写：**

由于  $\mathbb{E}[R(\tau) | s_t, a_t] = Q^{\pi_{\theta}}(s_t, a_t)$ , 我们得到：

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi_{\theta}}(s_t, a_t) \right]$$

**为什么使用优势函数？**

理论上的原始策略梯度(目的是最大化  $J = \mathbb{E} [\sum_{t=0}^{\infty} \gamma^t r_t]$ ):

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q(s_t, a_t) \right]$$

**问题：** Q函数的方差很大，导致梯度估计不稳定。

**解决方案：**引入基准函数  $b(s_t)$ :

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (Q(s_t, a_t) - b(s_t)) \right]$$

**最优基准：**可以证明，最优的基准函数是状态价值函数  $V(s_t)$ ，因此：

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A(s_t, a_t) \right]$$

其中  $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$  是优势函数。

A2C使用两个网络：

- **Actor网络：**参数  $\theta$ , 输出策略  $\pi_{\theta}(a | s)$
- **Critic网络：**参数  $\phi$ , 输出价值估计  $V_{\phi}(s)$

**优势函数估计**

在A2C中，我们使用**n-step TD误差**来估计优势函数：

$$A(s_t, a_t) = \left( \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) \right) - V(s_t)$$

**推导：**

真实的Q函数：

$$Q(s_t, a_t) = \mathbb{E} \left[ \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) \right]$$

首先回顾Q函数的定义：

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t \right]$$

把Q函数拆分为前n步和n步之后的部分：

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[ \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \sum_{k=n}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t \right]$$

对于第二部分，提取出公共因子 $\gamma^n$ ：

$$\sum_{k=n}^{\infty} \gamma^k r_{t+k} = \gamma^n \sum_{k=0}^{\infty} \gamma^k r_{t+n+k}$$

现在，观察这个表达式：

$$\sum_{k=0}^{\infty} \gamma^k r_{t+n+k} = r_{t+n} + \gamma r_{t+n+1} + \gamma^2 r_{t+n+2} + \dots = V(s_{t+n})$$

这正是从时刻 $t + n$ 开始的累积折扣回报！

令优势函数：

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

使用单样本估计：

$$A(s_t, a_t) \approx \left( \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) \right) - V(s_t)$$

**策略损失（Actor）** (要最小化)

$$L_{policy} = -\mathbb{E} [\log \pi_\theta(a_t|s_t) A(s_t, a_t)]$$

数学上,  $L_{policy}$  同时依赖于 $\theta$ 和 $\phi$ 。但是实践中, 这里的 $A(s, a)$ 被视为固定标量, 不是可微函数。目的是: 让两个网络独立更新, 避免相互干扰

$$\nabla_\theta L_{policy} = -\mathbb{E} [\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)]$$

即:

$$\nabla_\phi L_{policy} = -\mathbb{E} [A(s_t, a_t) \cdot \nabla_\phi \log \pi_\theta(a_t|s_t)]$$

**为什么是这个形式?**

- 我们希望最大化 $J(\theta)$ , 但优化器通常最小化损失
- 因此使用负号, 近似有:  $L = -\nabla_\theta J(\theta)$
- 在实际中, 我们使用样本估计:  $L_{policy} = -\log \pi_\theta(a_t|s_t) A(s_t, a_t)$

**价值损失 (Critic) (要最小化)**

$$L_{\phi,value} = \mathbb{E} \left[ \left( \left( \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n}) \right) - V_\phi(s_t) \right)^2 \right]$$

这是均方误差损失, 让Critic网络更好地估计状态价值。

**熵正则化 (要最小化)**

$$L_{\theta,entropy} = -\mathbb{E} [H(\pi_\theta(\cdot|s_t))] = \mathbb{E} \left[ \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \right]$$

- 鼓励探索, 防止策略过早收敛
- 提高训练稳定性

**总损失**

$$L_{\theta,\phi,total} = L_{\theta,policy} + c_1 L_{\phi,value} + c_2 L_{\theta,entropy}$$

其中 $c_1, c_2$ 是超参数, 通常 $c_1 = 0.5, c_2 = 0.01$ 。

梯度下降更新actor网络参数:

$$\theta = \theta - \eta \cdot \nabla_\theta L_{total}$$

即:

$$\theta = \theta - \eta \cdot (\nabla_\theta L_{\theta,policy} + \nabla_\theta L_{\theta,entropy})$$

梯度下降更新critic网络参数:

$$\phi = \phi - \alpha_{critic} \cdot \nabla_\phi L_{total}$$

即：

$$\phi = \phi - \alpha_{critic} \cdot \nabla_\phi L_{\phi,value}$$

**A2C算法完整流程**

初始化：

```

Actor网络参数 θ
Critic网络参数 ϕ
学习率 α_actor, α_critic
折扣因子 γ
熵系数 β
环境env

```

```

for episode = 1 to M do:
    初始化状态 s = env.reset()
    初始化经验缓冲区 buffers = []

```

```

for t = 0 to T-1 do:
    # 收集经验阶段
    根据π_θ(·|s)选择动作a
    执行动作a, 得到奖励r和下一个状态s'
    存储(s, a, r, s')到缓冲区

```

```
s = s'
```

```

if 缓冲区满 or episode终止:
    # 计算n-step回报和优势
    R = 0 if 终止 else V_ϕ(s')
    for i = len(buffers)-1 to 0:
        R = r_i + γR
        A = R - V_ϕ(s_i)

```

```

# 计算损失
策略损失 = -mean(log π_θ(a_i|s_i) * A_i)
价值损失 = mean((R_i - V_ϕ(s_i))^2)
熵损失 = -mean(H(π_θ(·|s_i)))
总损失 = 策略损失 + 0.5*价值损失 + β*熵损失

```

```

# 更新参数
θ = θ - α_actor * ∇_θ总损失
ϕ = ϕ - α_critic * ∇_ϕ总损失

```

```
清空缓冲区
```

**算法流程图**

开始训练  
↓  
初始化网络参数和环境

循环每个episode:

- |- 重置环境，获取初始状态
- |- 循环每个时间步:
  - |- 使用当前策略选择动作
  - |- 执行动作，收集经验( $s, a, r, s'$ )
  - |- 存储经验到缓冲区
- |- 如果缓冲区满或episode结束:
  - |- 计算n-step回报和优势
  - |- 计算策略、价值、熵损失
  - |- 计算总损失和梯度
  - |- 更新Actor和Critic网络

|- 直到达到最大episode数

↓  
结束训练

## PPO算法

强化学习的核心目标是找到最优策略  $\pi^*$  来最大化期望累积回报：

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

根据策略梯度定理，目标函数的梯度为：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta}}(s_t, a_t) \right]$$

其中  $A^{\pi_{\theta}}(s_t, a_t)$  是优势函数。

- $\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$  给出了增加动作  $a_t$  概率的方向
- $A^{\pi_{\theta}}(s_t, a_t)$  衡量了这个动作的相对好坏
- 乘积表示：好的动作应该增加概率，坏的动作应该减少概率

在强化学习中，策略  $\pi_{\theta}(a, s)$  的性能通常定义为从初始状态开始的期望累积回报：

$$J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \gamma^t r(s_t, a_t) \right]$$

其中：

$\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$  是一条轨迹 (trajectory)

轨迹的生成过程是：

$s_0 \sim p_0(s)$  (初始状态分布)

$a_t \sim \pi_{\theta}(\cdot | s_t)$

$s_{t+1} \sim P(\cdot | s_t, a_t)$  (环境动力学)

所以，整个轨迹的概率分布依赖于策略  $\pi_{\theta}$ ，记作  $p(\tau; \pi_{\theta})$ 。

### 轨迹概率 $p(\tau; \pi_{\theta})$

什么是“轨迹”(Trajectory)？

在强化学习中，智能体 (agent) 与环境 (environment) 交互的过程可以表示为一条 轨迹 (trajectory)：

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \dots, s_T)$$

$s_t$ : 第  $t$  步的状态 (state)

$a_t$ : 第  $t$  步采取的动作 (action)

$T$ : 终止时间 (可能是固定的，也可能是首次到达终止状态)

轨迹就是一次完整的“经历”或“episode”。

什么是“轨迹概率”  $p(\tau; \pi_{\theta})$ ？

在给定策略  $\pi_{\theta}$  的情况下，智能体生成这条特定轨迹  $\tau$  的概率是多少？

即：“用策略  $\pi_{\theta}$  去和环境互动，有多大概率会恰好走出这一串  $s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow \dots \rightarrow s_T$ ？”

### 轨迹概率的数学表达式

$$p(\tau; \pi_{\theta}) = p_0(s_0) \prod_{t=0}^T \pi_{\theta}(a_t | s_t) \cdot P(s_{t+1} | s_t, a_t)$$

推导：

这个概率由两部分共同决定：

1. 智能体的行为：由策略  $\pi_\theta(as)$  决定（我们控制的部分）
2. 环境的动态：由状态转移概率  $P(s's, a)$  决定（环境决定，我们无法控制）

此外，还要考虑初始状态从哪里开始。

### 轨迹概率的数学表达式

$$p(\tau; \pi_\theta) = p_0(s_0) \prod_{t=0}^T \pi_\theta(a_t s_t) \cdot P(s_{t+1} | s_t, a_t)$$

- $p_0(s_0)$ : 初始状态分布，表示环境一开始处于状态  $s_0$  的概率。
- $\pi_\theta(a_t s_t)$ : 策略选择动作的概率  
在状态  $s_t$  下，策略  $\pi_\theta$  选择动作  $a_t$  的概率。这是我们要优化的部分（参数为  $\theta$ ）。
- $P(s_{t+1} | s_t, a_t)$ : 环境的状态转移概率  
给定当前状态  $s_t$  和动作  $a_t$ ，环境转移到下一个状态  $s_{t+1}$  的概率。这是环境的固有动力学，与策略无关。

为什么是“连乘”？

因为生成轨迹是一个**马尔可夫序列过程**，每一步都只依赖于前一步：

先从  $p_0$  抽出  $s_0$ ，然后根据  $\pi_\theta(\cdot | s_0)$  抽出  $a_0$ ，然后根据  $P(\cdot | s_0, a_0)$  抽出  $s_1$ ，然后根据  $\pi_\theta(\cdot | s_1)$  抽出  $a_1$ ……

特定轨迹概率 = 初始状态概率 × 所有“动作选择概率”× 所有“状态转移概率”

为什么这个概念重要？

- 策略梯度推导的基础  
如前所述， $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim p(\tau; \pi_\theta)} [\dots]$ ，所以必须知道轨迹如何依赖于  $\theta$ 。
- 重要性采样的核心  
当我们想用旧策略的数据估计新策略的性能时，需要计算：

$$\frac{p(\tau; \pi_\theta)}{p(\tau; \pi_{\text{old}})} = \prod_{t=0}^T \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}$$

注意：环境转移概率  $P$  和初始分布  $p_0$  被约掉了，因为它们与策略无关！

因此，目标函数本质上是：

$$J(\pi_\theta) = \int p(\tau; \pi_\theta) \cdot R(\tau) d\tau$$

其中  $R(\tau) = \sum_t \gamma^t r_t$

对  $J(\pi_\theta)$  关于  $\theta$  求导：

$$\nabla_{\theta} J(\pi_{\theta}) = \nabla_{\theta} \int p(\tau; \pi_{\theta}) R(\tau) d\tau = \int \nabla_{\theta} p(\tau; \pi_{\theta}) \cdot R(\tau) d\tau$$

利用对数导数技巧：

$$\nabla_{\theta} p(\tau; \pi_{\theta}) = p(\tau; \pi_{\theta}) \cdot \nabla_{\theta} \log p(\tau; \pi_{\theta})$$

代入得：

$$\nabla_{\theta} J(\pi_{\theta}) = \int p(\tau; \pi_{\theta}) \cdot \nabla_{\theta} \log p(\tau; \pi_{\theta}) \cdot R(\tau) d\tau = \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log p(\tau; \pi_{\theta}) \cdot R(\tau)]$$

轨迹概率  $p(\tau; \pi_{\theta})$  :

$$p(\tau; \pi_{\theta}) = p_0(s_0) \prod_{t=0}^T \pi_{\theta}(a_t | s_t) P(s_{t+1} | s_t, a_t)$$

取对数：

$$\log p(\tau; \pi_{\theta}) = \log p_0(s_0) + \sum_{t=0}^T \log \pi_{\theta}(a_t | s_t) + \sum_{t=0}^T \log P(s_{t+1} | s_t, a_t)$$

注意： $p_0(s_0)$  和  $P(s_{t+1} | s_t, a_t)$  与  $\theta$  无关（环境不是我们控制的）

所以对  $\theta$  求导时，只有  $\log \pi_{\theta}(a_t | s_t)$  项保留

于是：

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

代回梯度表达式：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \cdot R(\tau) \right]$$

但这里有个问题： $R(\tau)$  包含所有时间步的奖励，而我们在时间  $t$  做决策时，只应关心未来的回报。通过进一步推导（引入状态值函数  $V^{\pi}$ ），可以把  $R(\tau)$  替换为优势函数  $A^{\pi_{\theta}}(s_t, a_t)$ ，得到更高效的梯度估计：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \cdot A^{\pi_{\theta}}(s_t, a_t) \right]$$

证明：

在时间步  $t$ ，动作  $a_t$  只影响从  $t$  开始的未来回报，不应该对  $r_0, \dots, r_{t-1}$  负责。

但公式 (1) 中每个  $\nabla \log \pi_{\theta}(a_t | s_t)$  都乘上了包含过去奖励的  $R(\tau)$ ，这会引入不必要的噪声（高方差）。

**定义从时间  $t$  开始的（折扣）回报：**

$$R_t = \sum_{l=t}^T \gamma^{l-t} r_l$$

(为一般性, 我们加入折扣因子  $\gamma$ ; 若  $\gamma = 1$  则为无折扣情形)

以下等式成立:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot R_t \right]$$

每个动作只与它之后的回报相关。

为什么可以这样替换?

因为对于  $l < t$  的奖励  $r_l$ , 它们与  $a_t$  无关, 而  $\mathbb{E}[\nabla_\theta \log \pi_\theta(a_t | s_t)] = 0$  (重要性质), 所以交叉项期望为零.

## 推导

$$\mathbb{E}_{a_t \sim \pi_\theta(\cdot | s_t)} [\nabla_\theta \log \pi_\theta(a_t | s_t)] = 0$$

直观理解:

$\pi_\theta(as)$  是一个概率分布 (对所有  $a$  求和/积分为 1)。

$\nabla_\theta \log \pi_\theta(as)$  衡量: 当参数  $\theta$  微小变化时, 某个动作  $a$  的 log-probability 如何变化。

但因为总概率必须始终为 1, 某些动作的概率增加, 必然有其他动作的概率减少。

所以, 在整个动作空间上“平均”来看, 这种变化的净效果为零。

就像一个守恒律: 你不能让所有动作的概率同时变大!

固定状态  $s$  (省略下标  $t$  简化记号), 考虑离散动作空间 (连续情形类似, 只需把求和换成积分)。

利用概率归一化, 对任意  $\theta$ , 策略是一个合法的概率分布:

$$\sum_a \pi_\theta(a, s) = 1$$

两边关于  $\theta$  求导:

$$\nabla_\theta \left( \sum_a \pi_\theta(a, s) \right) = \nabla_\theta(1) = 0$$

交换求和与梯度 (假设正则条件满足):

$$\sum_a \nabla_\theta \pi_\theta(a, s) = 0$$

用对数导数技巧重写

注意到:

$$\nabla_{\theta} \pi_{\theta}(a, s) = \pi_{\theta}(a, s) \cdot \nabla_{\theta} \log \pi_{\theta}(a, s)$$

代入：

$$\sum_a \pi_{\theta}(a, s) \cdot \nabla_{\theta} \log \pi_{\theta}(a, s) = 0$$

左边正是在分布  $a \sim \pi_{\theta}(\cdot | s)$  下， $\nabla_{\theta} \log \pi_{\theta}(as)$  的期望：

$$\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [\nabla_{\theta} \log \pi_{\theta}(as)] = 0$$

得证！

### 连续动作空间的情形

若动作空间连续，只需将求和换成积分：

$$\int \pi_{\theta}(a, s) da = 1 \Rightarrow \int \nabla_{\theta} \pi_{\theta}(a, s) da = 0 \Rightarrow \int \pi_{\theta}(a, s) \nabla_{\theta} \log \pi_{\theta}(a, s) da = 0$$

即：

$$\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [\nabla_{\theta} \log \pi_{\theta}(as)] = 0$$

成立。

### 进一步降方差：减去一个“基线”(Baseline)

一个关键观察是：对任意只依赖于  $s_t$  的函数  $b(s_t)$  (称为 baseline)，有：

$$\mathbb{E}_{a_t \sim \pi_{\theta}(\cdot | s_t)} [\nabla_{\theta} \log \pi_{\theta}(a_t s_t) \cdot b(s_t)] = b(s_t) \cdot \underbrace{\mathbb{E}[\nabla_{\theta} \log \pi_{\theta}(a_t s_t)]}_{=0} = 0$$

因此，我们可以从  $R_t$  中减去任意  $b(s_t)$ ，不改变梯度的期望值，但可能降低方差！

于是：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t s_t) \cdot (R_t - b(s_t)) \right] \quad (3)$$

最优的 baseline 是使方差最小的那个。可以证明，最优 baseline 就是状态值函数：

$$V^{\pi_{\theta}}(s_t) = \mathbb{E}_{a_t, s_{t+1}, \dots} [R_t | s_t]$$

即：在状态  $s_t$  下，按策略  $\pi_{\theta}$  继续执行所能获得的期望回报。

### 定义优势函数 (Advantage Function)

将  $b(s_t) = V^{\pi_{\theta}}(s_t)$  代入 (3)，得到：

$$A^{\pi_\theta}(s_t, a_t) := R_t - V^{\pi_\theta}(s_t)$$

这就是优势函数的定义：它衡量在状态  $s_t$  下，采取具体动作  $a_t$  比“平均水平”(即  $V^{\pi_\theta}(s_t)$ ) 好多少。

如果  $A > 0$ : 这个动作比平均好，应该增加其概率；

如果  $A < 0$ : 比平均差，应该减少其概率。

代入后，策略梯度变为：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot A^{\pi_\theta}(s_t, a_t) \right]$$

这就是 Actor-Critic 方法的核心思想：Critic 学习  $V^\pi$  (或  $Q^\pi$ )，Actor 用优势函数更新策略。

## 总结

- 原始梯度  $\mathbb{E}[\sum_t \nabla \log \pi \cdot R(\tau)]$  正确但高方差
- 用未来回报  $\mathbb{E}[\sum_t \nabla \log \pi \cdot R_t]$  去掉无关历史奖励
- 减基线  $\mathbb{E}[\sum_t \nabla \log \pi \cdot (R_t - b(s_t))]$  不改变期望，降方差
- 选最优基线  $b = V^\pi \mathbb{E}[\sum_t \nabla \log \pi \cdot A^\pi(s_t, a_t)]$  最小化方差，得到优势函数形式

$$\boxed{\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot A^{\pi_\theta}(s_t, a_t) \right]}$$

这就是你提到的策略梯度定理的标准形式。

为什么必须是  $\tau \sim \pi_\theta$ ?

因为：

- 目标函数  $J(\pi_\theta)$  的定义本身就依赖于  $\pi_\theta$  生成的轨迹分布。
- 梯度是对这个特定期望求导的结果，数学推导中每一步都假设轨迹由  $\pi_\theta$  生成。
- 如果你用别的策略（比如  $\pi_{old}$ ）生成的轨迹来计算这个期望，那么：

你实际上是在估计  $\mathbb{E}_{\tau \sim \pi_{old}} [\dots]$  而这不是  $\nabla_\theta J(\pi_\theta)$ ，而是有偏的估计！

即：

**策略梯度公式成立的前提是：数据（轨迹）来自当前被优化的策略  $\pi_\theta$ 。**

这就是所谓的 **on-policy** 方法的本质：你的数据必须和当前策略一致。

## 重要性采样

为了重用旧策略的数据，我们使用重要性采样：

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[ \frac{p(x)}{q(x)} f(x) \right]$$

应用到策略梯度中：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} \left[ \sum_{t=0}^T \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta_{old}}}(s_t, a_t) \right]$$

## 推导

我们想估计一个期望，但不能直接从目标分布采样。

假设有一个函数  $f(x)$ ，想计算它在某个概率分布  $p(x)$  下的期望：

$$\mathbb{E}_{x \sim p}[f(x)] = \int f(x)p(x) dx$$

但在实际中，无法从  $p(x)$  中采样（比如因为  $p$  太复杂，或者采样成本太高），但你可以从另一个“容易采样”的分布  $q(x)$  中获取样本。

这时候怎么办？—— 重要性采样 就是解决这个问题的数学工具。

可以把上面的积分改写为：

$$\int f(x)p(x) dx = \int f(x) \cdot \frac{p(x)}{q(x)} \cdot q(x) dx = \mathbb{E}_{x \sim q} \left[ \frac{p(x)}{q(x)} f(x) \right]$$

只要  $q(x) > 0$  在  $p(x) > 0$  的地方成立（即  $q$  能覆盖  $p$  的支撑集），这个等式就成立。

虽然我们不能从  $p$  采样，但可以用从  $q$  采样的数据，通过加权（权重 =  $p(x)/q(x)$ ）来无偏地估计  $p$  下的期望。

这个权重  $\frac{p(x)}{q(x)}$  就叫 重要性权重 (importance weight)。

回到强化学习：为什么需要重要性采样？

在策略梯度 (Policy Gradient) 方法中，我们的目标是优化策略  $\pi_{\theta}$ ，其性能指标通常是：

$$J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r_t \right]$$

对应的梯度（根据策略梯度定理）为：

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta}}(s_t, a_t) \right]$$

问题来了：这个期望要求轨迹  $\tau$  是从当前策略  $\pi_{\theta}$  中采样得到的。但如果我们每更新一次参数  $\theta$  就重新采样新数据，效率极低。

解决方案：重用旧策略  $\pi_{\theta_{old}}$  采集的数据！

但期望是对  $\pi_{\theta}$  的，而数据来自  $\pi_{\theta_{old}}$  —— 重要性采样

## 重要性采样到策略梯度

考虑单个时间步的项（忽略轨迹依赖简化理解）：

想估计：

$$\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [\nabla_\theta \log \pi_\theta(as) A(s, a)]$$

但我们只有从  $\pi_{\theta_{\text{old}}}(\cdot|s)$  采样的动作  $a$ 。

于是用重要性采样：

$$= \mathbb{E}_{a \sim \pi_{\theta_{\text{old}}}(\cdot|s)} \left[ \frac{\pi_\theta(a, s)}{\pi_{\theta_{\text{old}}}(a, s)} \cdot \nabla_\theta \log \pi_\theta(a, s) A(s, a) \right]$$

扩展到整条轨迹（假设状态转移与策略无关），就得到以下公式：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}} \left[ \sum_{t=0}^T \underbrace{\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}}_{r_t(\theta)} \nabla_\theta \log \pi_\theta(a_t | s_t) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right]$$

其中： $r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$  就是重要性权重。

它修正了“用旧策略数据估计新策略梯度”带来的偏差，使得估计无偏（在理想条件下）。

### 需要注意的问题

- 方差可能很大

如果  $\pi_\theta$  和  $\pi_{\theta_{\text{old}}}$  差别很大，那么  $r_t(\theta)$  可能非常大或非常小。导致梯度估计噪声极大，训练不稳定。

这正是 PPO 引入裁剪（clip）机制的原因：限制  $r_t(\theta)$  的变化范围，牺牲一点无偏性，换取低方差和稳定性。

- 只适用于 on-policy 方法的离线重用

**严格来说，策略梯度是 on-policy 的（依赖当前策略的数据）。重要性采样让我们能“近似 off-policy”，但不能无限期重用旧数据。**

重要性采样允许我们用从分布  $q$  采样的数据，去无偏估计在另一个分布  $p$  下的期望，只需给每个样本乘上权重  $p(x)/q(x)$ 。

在强化学习中，它让我们能重用旧策略采集的数据来更新新策略，极大提升样本效率。

我们可以构造一个替代目标函数，在当前策略点  $\theta = \theta_{\text{old}}$  处，其梯度与原始目标函数相同：

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

其中 CPI 表示 Conservative Policy Iteration。

### 推导

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

的梯度与原始策略梯度目标函数在  $\theta = \theta_{\text{old}}$  处是完全一致的。

根据策略梯度定理 (Policy Gradient Theorem)，原始策略梯度目标函数的梯度为：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) A^{\pi_\theta}(s_t, a_t) \right]$$

但在实际中，我们往往用旧策略  $\pi_{\theta_{\text{old}}}$  的数据来估计这个梯度（为了样本效率）。于是引入重要性采样，得到一个替代目标函数  $L^{\text{CPI}}(\theta)$ ，其定义如上。

即：在当前策略点  $\theta = \theta_{\text{old}}$  处，两个目标函数的梯度相同。

$$L^{\text{CPI}}(\theta) = \mathbb{E}_{(s_t, a_t) \sim \pi_{\theta_{\text{old}}}} [r_\theta(s_t, a_t) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)]$$

注意：这里没有对轨迹求期望，而是对单步  $(s_t, a_t)$  求期望，这是合理的近似（尤其在 actor-critic 框架中）。

对  $L^{\text{CPI}}(\theta)$  关于  $\theta$  求梯度：

$$\nabla_\theta L^{\text{CPI}}(\theta) = \nabla_\theta \mathbb{E}_{(s, a) \sim \pi_{\theta_{\text{old}}}} \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right]$$

由于期望是对固定分布  $\pi_{\theta_{\text{old}}}$ （与  $\theta$  无关）取的，可以把梯度移入期望内：

$$= \mathbb{E}_{(s, a) \sim \pi_{\theta_{\text{old}}}} \left[ \nabla_\theta \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \cdot A^{\pi_{\theta_{\text{old}}}}(s, a) \right]$$

现在计算  $\nabla_\theta \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$ ：

$$\nabla_\theta \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} = \frac{1}{\pi_{\theta_{\text{old}}}(a_t, s_t)} \nabla_\theta \pi_\theta(a_t, s_t)$$

又因为：

$$\nabla_\theta \pi_\theta(a_t, s_t) = \pi_\theta(a_t, s_t) \cdot \nabla_\theta \log \pi_\theta(a_t, s_t)$$

所以：

$$\nabla_\theta \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{\text{old}}}(a_t, s_t)} \cdot \nabla_\theta \log \pi_\theta(a_t, s_t)$$

代入梯度表达式：

$$\nabla_\theta L^{\text{CPI}}(\theta) = \mathbb{E}_{(s, a) \sim \pi_{\theta_{\text{old}}}} \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \cdot \nabla_\theta \log \pi_\theta(a_t, s_t) \cdot A^{\pi_{\theta_{\text{old}}}}(s, a) \right]$$

在  $\theta = \theta_{\text{old}}$  处求值

当  $\theta = \theta_{\text{old}}$  时：

$$\begin{aligned}\pi_\theta &= \pi_{\theta_{\text{old}}} \\ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} &= 1\end{aligned}$$

同时

$$\nabla_\theta \log \pi_\theta(a, s)_{\theta=\theta_{\text{old}}} = \nabla_\theta \log \pi_{\theta_{\text{old}}}(a, s)$$

因此：

$$\nabla_\theta L^{\text{CPI}}(\theta)_{\theta=\theta_{\text{old}}} = \mathbb{E}_{(s,a) \sim \pi_{\theta_{\text{old}}}} [1 \cdot \nabla_\theta \log \pi_{\theta_{\text{old}}}(a, s) \cdot A^{\pi_{\theta_{\text{old}}}}(s, a)]$$

而这就是标准策略梯度在  $\theta = \theta_{\text{old}}$  处的形式。

原始策略梯度为：

$$\nabla_\theta J(\pi_\theta)_{\theta=\theta_{\text{old}}} = \mathbb{E}_{(s,a) \sim \pi_{\theta_{\text{old}}}} [\nabla_\theta \log \pi_{\theta_{\text{old}}}(a, s) \cdot A^{\pi_{\theta_{\text{old}}}}(s, a)]$$

两者完全相等！

结论：

$$\nabla_\theta L^{\text{CPI}}(\theta)_{\theta=\theta_{\text{old}}} = \nabla_\theta J(\pi_\theta)_{\theta=\theta_{\text{old}}}$$

也就是说， $L^{\text{CPI}}(\theta)$  是原始目标函数  $J(\pi_\theta)$  在  $\theta_{\text{old}}$  附近的一阶泰勒展开（线性近似）。

意义：

$L^{\text{CPI}}(\theta)$  是一个可计算的代理目标函数：它只依赖于旧策略采集的数据  $((s, a)$  和  $A^{\pi_{\theta_{\text{old}}}})$ ，却能在当前点给出正确的梯度方向。

这使得我们可以重用旧数据进行多次策略更新（sample-efficient）。

但直接优化  $L^{\text{CPI}}$  会导致策略更新过大（因为它是局部近似），所以 PPO 引入了 clip 机制来限制更新幅度，形成更稳健的  $L^{\text{CLIP}}$ 。

定义概率比：

$$r(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$$

则：

$$L^{\text{CPI}}(\theta) = \mathbb{E}_t [r(\theta) A_t]$$

### PPO-Clip

直接优化  $L^{\text{CPI}}$  会导致策略更新过大，PPO 通过裁剪来约束更新：

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t [\min(r(\theta) A_t, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) A_t)]$$

## 为什么需要这个复杂的min和clip结构?

让我们分情况分析:

情况1:  $A_t > 0$  (好的动作)

我们希望增加这个动作的概率, 但要防止过度增加:

- 如果  $r(\theta) < 1 + \epsilon$ , 使用  $r(\theta)A_t$
- 如果  $r(\theta) > 1 + \epsilon$ , 使用  $(1 + \epsilon)A_t$

情况2:  $A_t < 0$  (坏的动作)

我们希望减少这个动作的概率, 但要防止过度减少:

- 如果  $r(\theta) > 1 - \epsilon$ , 使用  $r(\theta)A_t$
- 如果  $r(\theta) < 1 - \epsilon$ , 使用  $(1 - \epsilon)A_t$

**数学表达的统一形式:**

$$\min(r(\theta)A_t, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)A_t)$$

这个设计确保了:

1. 策略改进: 好的动作概率增加, 坏的动作概率减少
2. 信任区域: 更新幅度不超过  $\epsilon$
3. 单调改进: 类似于TRPO的保证, 但实现更简单

完整的PPO目标函数包含三个部分:

$$L^{TOTAL}(\theta, \phi) = \mathbb{E}_t [L^{CLIP}(\theta) - c_1 L^{VF}(\phi) + c_2 S[\pi_\theta](s_t)]$$

其中:

**价值函数损失** (要最小化)

$$L^{VF}(\phi) = (V_\phi(s_t) - V_{target})^2$$

$V_\phi(s_t)$ : 当前Critic网络的参数  $\phi$  对状态  $s_t$  的价值预测, 这个  $V_\phi(s_t)$  需要计算梯度, 通过反向传播更新网络参数。

$V_{target}$  通常是通过回报或GAE计算的目标价值。

$$V_{target} = A_t^{GAE} + V_{\phi_{old}}(s_t)$$

$V_{\phi_{old}}(s_t)$ : 旧Critic网络 (数据收集时的网络) 对状态  $s_t$  的价值预测, 这个  $V_{\phi_{old}}(s_t)$  应该被固定, 不计算梯度。

**广义优势估计 (GAE)**

优势函数  $A_t$  通过GAE计算:

$$A_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

其中 TD误差  $\delta_t = r_t + \gamma V_{\phi_{old}}(s_{t+1}) - V_{\phi_{old}}(s_t)$  是TD误差。

在TD误差  $\delta_t$  的计算中，使用的也是旧Critic网络  $V_{\phi_{old}}$

$V_{\phi_{old}}$  会软更新/硬更新。

- $\lambda = 0$ :  $A_t = \delta_t$  (高偏差, 低方差)
- $\lambda = 1$ :  $A_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V_{\phi_{old}}(s_t)$  (低偏差, 高方差)
- $\lambda \in (0, 1)$ : 在偏差和方差之间取得平衡

熵奖励 (要最小化)

$$S[\pi_{\theta}](s_t) = - \sum_a \pi_{\theta}(a|s_t) \log \pi_{\theta}(a|s_t)$$

熵奖励鼓励探索，防止策略过早收敛到局部最优。

算法完整流程

```

初始化策略参数  $\theta$ , 价值函数参数  $\phi$ 
for iteration = 1, 2, ... do
    # 数据收集阶段
    清空经验缓冲区
    for episode = 1, 2, ..., N do
        重置环境, 获得初始状态  $s_{\theta}$ 
        for t = 0, 1, ..., T do
            根据当前策略  $\pi_{\theta}(\cdot | s_t)$  选择动作  $a_t$ 
            执行动作  $a_t$ , 获得奖励  $r_t$  和下一状态  $s_{t+1}$ 
            存储转移 ( $s_t, a_t, r_t, V_{\phi}(s_t)$ , done)
        end for
    end for

    # 优势估计阶段
    计算优势估计  $A_t$  和目标价值  $V_{target}$  使用GAE

    # 策略更新阶段
    for epoch = 1, 2, ..., K do
        随机打乱经验数据
        for minibatch in 数据 do
            计算概率比  $r(\theta) = \pi_{\theta}(a|s) / \pi_{\theta\_old}(a|s)$ 
            计算裁剪目标  $L^{CLIP}(\theta)$ 
            计算价值损失  $L^{VF}(\phi)$ 
            计算熵奖励  $S[\pi_{\theta}]$ 
            计算总损失  $L^{TOTAL} = L^{CLIP} + c1 \cdot L^{VF} + c2 \cdot S$ 
            使用梯度下降更新  $\theta, \phi$ 
        end for
    end for

    # 更新旧策略
     $\theta_{old} \leftarrow \theta$ 
end for

```

## 算法步骤

阶段1：数据收集

```

def collect_experience():
    states, actions, rewards, values, dones = [], [], [], [], []

    state = env.reset()
    for t in range(max_steps):
        # 使用当前策略选择动作
        action, log_prob, entropy = policy_network.get_action(state)
        value = value_network.get_value(state)

        # 执行动作
        next_state, reward, done, _ = env.step(action)

        # 存储经验
        states.append(state)
        actions.append(action)
        rewards.append(reward)
        values.append(value)
        dones.append(done)

        state = next_state
        if done: break

    return states, actions, rewards, values, dones

```

阶段2：优势计算

```

def compute_advantages(rewards, values, dones, next_value, gamma=0.99, lambda_=0.95):
    advantages = []
    returns = []

    # 计算TD误差
    deltas = []
    for t in range(len(rewards)):
        td_target = rewards[t] + gamma * (1 - dones[t]) * values[t+1] if t < len(rewards)-1 else next_value
        delta = td_target - values[t]
        deltas.append(delta)

    # 计算GAE
    advantage = 0
    for t in reversed(range(len(deltas))):
        advantage = deltas[t] + gamma * lambda_* advantage
        advantages.insert(0, advantage)
        returns.insert(0, advantage + values[t])

    return advantages, returns

```

2025/11/26 13:37

阶段3：PPO更新

base\_DRL\_model\_2

```

def ppo_update(states, actions, old_log_probs, advantages, returns, clip_epsilon=0.2):
    # 转换为tensor
    states = torch.tensor(states)
    actions = torch.tensor(actions)
    old_log_probs = torch.tensor(old_log_probs)
    advantages = torch.tensor(advantages)
    returns = torch.tensor(returns)

    # 多轮更新
    for epoch in range(ppo_epochs):
        # 随机打乱
        indices = torch.randperm(len(states))

        # 小批量更新
        for start in range(0, len(indices), batch_size):
            end = start + batch_size
            batch_indices = indices[start:end]

            batch_states = states[batch_indices]
            batch_actions = actions[batch_indices]
            batch_old_log_probs = old_log_probs[batch_indices]
            batch_advantages = advantages[batch_indices]
            batch_returns = returns[batch_indices]

            # 计算新策略的概率
            new_log_probs, entropy, values = policy_network.evaluate_actions(
                batch_states, batch_actions)

            # 概率比
            ratios = torch.exp(new_log_probs - batch_old_log_probs)

            # PPO-Clip 目标
            surr1 = ratios * batch_advantages
            surr2 = torch.clamp(ratios, 1-clip_epsilon, 1+clip_epsilon) * batch_advantages
            actor_loss = -torch.min(surr1, surr2).mean()

            # Critic损失
            critic_loss = 0.5 * (values.squeeze() - batch_returns).pow(2).mean()

            # 熵奖励
            entropy_bonus = -entropy.mean()

            # 总损失
            total_loss = actor_loss + 0.5 * critic_loss + 0.01 * entropy_bonus

            # 梯度更新

```

```
optimizer.zero_grad()
total_loss.backward()
torch.nn.utils.clip_grad_norm_(policy_network.parameters(), 0.5)
optimizer.step()
```

TRPO使用约束优化：

$$\max_{\theta} \mathbb{E}[r(\theta)A] \quad \text{s.t.} \quad \mathbb{E}[KL(\pi_{old} || \pi_{\theta})] \leq \delta$$

PPO通过裁剪隐式地实现了类似的约束，但计算更简单。