

深度强化学习各种经典算法总结 1

目录

[一. DQN](#)

[二. DDQN](#)

[三. DDPG](#)

[四. TD3](#)

[五. SAC](#)

DQN算法

状态价值函数：

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right]$$

动作价值函数（Q函数）：

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a \right]$$

Bellman期望方程：

$$Q^\pi(s, a) = \mathbb{E}_\pi [r + \gamma Q^\pi(s', a') | s, a]$$

Bellman最优方程：

$$Q^*(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right]$$

Q-Learning算法

Q-Learning是基于时序差分(TD)学习的离策略算法。

TD目标：

$$y = r + \gamma \max_{a'} Q(s', a')$$

TD误差：

$$\delta = y - Q(s, a)$$

对于表格型方法，Q值更新为：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

当状态空间很大或连续时，使用函数逼近器（神经网络）来近似Q函数：

$$Q(s, a; \theta) \approx Q^*(s, a)$$

其中 θ 是神经网络的参数。目的：找到参数 θ 使得 $Q(s, a; \theta)$ 尽可能接近最优Q函数 $Q^*(s, a)$

根据Bellman最优方程，我们希望：

$$Q(s, a; \theta) \approx r + \gamma \max_{a'} Q(s', a'; \theta)$$

定义目标值：

$$y = r + \gamma \max_{a'} Q(s', a'; \theta)$$

损失函数（均方误差）：

$$L(\theta) = \mathbb{E}_{(s, a, r, s') \sim D} \left[(y - Q(s, a; \theta))^2 \right]$$

直接使用上述损失函数会导致训练不稳定（“移动目标”问题）。引入**目标网络** $Q(s, a; \theta^-)$ ：**改进的目标值：**

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

最终损失函数：

$$L(\theta) = \mathbb{E}_{(s, a, r, s') \sim D} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

使用随机梯度下降更新参数：

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(\theta)$$

计算梯度：

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D} [2(y - Q(s, a; \theta)) \nabla_{\theta} Q(s, a; \theta)]$$

其中：

- $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ 被视为常数（不计算梯度）
- 只对 $Q(s, a; \theta)$ 计算梯度

经验回放缓冲区D存储经验(s,a,r,s')，从中均匀采样小批量数据：

期望的近似：

$$\mathbb{E}_{(s,a,r,s') \sim D} [\cdot] \approx \frac{1}{N} \sum_{i=1}^N [\cdot]$$

这打破了数据间的时间相关性，使训练数据更接近独立同分布。

完整DQN算法流程

初始化:

当前Q网络 $Q(\theta)$ 随机初始化

目标Q网络 $Q(\theta^-)$ 参数 $\theta^- \leftarrow \theta$

经验回放缓冲区 $D \leftarrow \emptyset$

探索率 $\epsilon \leftarrow 1.0$

for episode = 1 to M **do:**

 初始化状态 s

for t = 1 to T **do:**

 # 动作选择 (ϵ -greedy策略)

 以概率 ϵ 选择随机动作 a

 否则选择 $a = \text{argmax}_a Q(s, a; \theta)$

 # 与环境交互

 执行动作 a, 观察奖励 r 和下一个状态 s'

 将经验 (s, a, r, s') 存储到 D

 # 状态转移

$s \leftarrow s'$

 # 训练网络

if $|D| \geq$ 批次大小 **then:**

 # 从D中随机采样批次 $\{(s_i, a_i, r_i, s'_i)\}$

 批次 \leftarrow 从D均匀采样N个经验

 # 计算目标Q值

for i = 1 to N **do:**

if s'_i 是终止状态 **then:**

$y_i \leftarrow r_i$

else:

$y_i \leftarrow r_i + \gamma \max_a' Q(s'_i, a'; \theta^-)$

 # 计算损失

$L(\theta) \leftarrow 1/N \sum_i (y_i - Q(s_i, a_i; \theta))^2$

 # 梯度下降

$\theta \leftarrow \theta - \alpha \nabla \theta L(\theta)$

 # 衰减探索率

$\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \times \epsilon_{\text{decay}})$

```
# 更新目标网络 (定期硬更新)
if t mod C == 0 then:
    θ⁻ ← θ
```

结束回合循环

结束所有回合

算法流程图

```

开始
↓
初始化Q网络、目标网络、经验池
↓
对于每个回合：
    初始化状态s
    ↓
    对于每个时间步：
        ↓
        使用 $\epsilon$ -greedy策略选择动作a
        ↓
        执行动作a，获得( $r, s'$ )
        ↓
        存储经验( $s, a, r, s'$ )到经验池
        ↓
        如果经验池足够大：
            ↓
            随机采样小批量经验
            ↓
            计算目标Q值： $y = r + \gamma \max Q(s', a'; \theta^-)$ 
            ↓
            计算损失： $L = (y - Q(s, a; \theta))^2$ 
            ↓
            梯度下降更新Q网络参数 $\theta$ 
            ↓
            定期更新目标网络参数 $\theta^-$ 
            ↓
            状态转移： $s \leftarrow s'$ 
            ↓
            直到回合结束
            ↓
            直到所有回合完成

```

DDQN算法

DQN的问题：Q值过高估计

在DQN的目标值计算中：

$$\max_{a'} Q(s', a'; \theta^-)$$

这个max操作会**系统性高估**真实的Q值。

假设真实Q值为 Q^* , 估计值为 $\hat{Q} = Q^* + \varepsilon$, 其中 ε 是零均值的估计误差。

那么:

$$\mathbb{E}[\max_a \hat{Q}(s, a)] = \mathbb{E}[\max_a (Q^*(s, a) + \varepsilon_a)]$$

由于max操作会选择那些 ε_a 为正的动作:

$$\mathbb{E}[\max_a \hat{Q}(s, a)] \geq \max_a \mathbb{E}[Q^*(s, a) + \varepsilon_a] = \max_a Q^*(s, a)$$

DDQN的关键创新是将动作**选择和评估**两个步骤解耦。

步骤1：动作选择

使用**在线网络**选择下一个状态的最优动作:

$$a^* = \arg \max_{a'} Q(s', a'; \theta)$$

步骤2：动作评估

使用**目标网络**评估该动作的Q值:

$$Q(s', a^*; \theta^-)$$

步骤3：目标值计算

$$y^{DDQN} = r + \gamma Q(s', a^*; \theta^-)$$

完整形式:

$$y^{DDQN} = r + \gamma Q\left(s', \arg \max_{a'} Q(s', a'; \theta); \theta^-\right)$$

DDQN损失函数

$$L(\theta) = \mathbb{E}_{(s, a, r, s') \sim D} \left[(y^{DDQN} - Q(s, a; \theta))^2 \right]$$

其中:

$$y^{DDQN} = r + \gamma Q\left(s', \arg \max_{a'} Q(s', a'; \theta); \theta^-\right)$$

为什么DDQN能减少过高估计？

DDQN的目标值可以看作：

$$y^{DDQN} = r + \gamma Q(s', \pi(s'); \theta^-)$$

其中 $\pi(s') = \arg \max_{a'} Q(s', a'; \theta)$

由于选择网络 θ 和评估网络 θ^- 是独立的（虽然通过硬更新相关），如果：

- 在线网络 θ 因为噪声高估了差动作
- 目标网络 θ^- 很可能不会同样高估这个动作

这样就形成了**误差抵消**的效果。

DDQN算法流程

初始化:

在线网络 $Q(\theta)$ 随机初始化

目标网络 $Q(\theta^-) \leftarrow Q(\theta)$

经验回放缓冲区 $D \leftarrow \emptyset$

训练步数 $t \leftarrow 0$

for episode = 1 to M **do:**

 初始化状态 s

for step = 1 to T **do:**

 # 1. 动作选择 (ϵ -greedy)

 以概率 ϵ 选择随机动作 a

 否则 $a \leftarrow \text{argmax}_a Q(s, a; \theta)$

 # 2. 执行动作, 观察环境

 执行动作 a, 观察奖励 r 和下一个状态 s'

 将经验 (s, a, r, s') 存入 D

 # 3. 训练网络

if $|D| \geq \text{batch_size}$:

 # 3.1 从D中采样一批经验

 B $\leftarrow \text{sample_batch}(D, \text{batch_size})$

 # 3.2 计算目标值 (DDQN核心)

for each (s_i, a_i, r_i, s_i') in B:

$a_i^* \leftarrow \text{argmax}_a Q(s_i', a; \theta)$ # 在线网络选择

$y_i \leftarrow r_i + \gamma Q(s_i', a_i^*; \theta^-)$ # 目标网络评估

if s_i' 是终止状态: $y_i \leftarrow r_i$

 # 3.3 计算损失

$L(\theta) \leftarrow 1/N \sum_i (y_i - Q(s_i, a_i; \theta))^2$

 # 3.4 梯度下降更新在线网络

$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(\theta)$

 # 3.5 更新训练步数

$t \leftarrow t + 1$

 # 3.6 定期更新目标网络

if $t \% \text{target_update_frequency} == 0$:

$\theta^- \leftarrow \theta$ # 硬更新

```
# 或者使用软更新:  $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$ 
```

```
# 4. 状态转移
```

```
s ← s'
```

```
# 5. 衰减探索率
```

```
 $\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \times \epsilon_{\text{decay}})$ 
```

```
if 回合结束: break
```

对比

DQN:

$$y^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

DDQN:

$$y^{DDQN} = r + \gamma Q(s', \arg \max_{a'} Q(s', a'; \theta); \theta^-)$$

DQN目标值计算:

输入: s' (下一个状态)

输出: $\max_a Q(s', a; \theta^-)$ # 同一网络选择和评估

DDQN目标值计算:

输入: s' (下一个状态)

$a^* \leftarrow \operatorname{argmax}_a Q(s', a; \theta)$ # 在线网络选择

输出: $Q(s', a^*; \theta^-)$ # 目标网络评估

DDPG算法

$$J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

随机策略梯度 (REINFORCE):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)]$$

确定性策略: $a = \mu_\theta(s)$

确定性策略梯度定理 (Silver et al., 2014):

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s \sim \rho^\mu} [\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

Actor-Critic框架

Actor (策略网络):

- 输入: 状态 s
- 输出: 确定性动作 $a = \mu(s|\theta^\mu)$
- 目标: 学习最优策略

Critic (价值网络):

- 输入: 状态 s 和动作 a
- 输出: Q值 $Q(s, a|\theta^Q)$
- 目标: 准确估计状态-动作价值

为了解决训练不稳定性, 引入目标网络:

目标Actor: $\mu'(s|\theta^{\mu'})$

目标Critic: $Q'(s, a|\theta^{Q'})$

Critic网络更新

时序差分目标:

$$y = r + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'})$$

损失函数:

$$L(\theta^Q) = \mathbb{E}_{(s, a, r, s') \sim D} [(y - Q(s, a|\theta^Q))^2]$$

梯度更新:

$$\nabla_{\theta^Q} L(\theta^Q) = -\mathbb{E}_{(s, a, r, s') \sim D} [(y - Q(s, a|\theta^Q)) \nabla_{\theta^Q} Q(s, a|\theta^Q)]$$

梯度下降:

$$\theta^Q \leftarrow \theta^Q - \alpha \nabla_{\theta^Q} L(\theta^Q)$$

Actor网络更新

根据确定性策略梯度定理：

策略梯度：

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s \sim D} [\nabla_{\theta^\mu} \mu(s|\theta^\mu) \nabla_a Q(s, a|\theta^Q)|_{a=\mu(s|\theta^\mu)}]$$

即：要提升性能 $J(\mu_\theta)$ ，应该沿着 $\nabla_a Q^\mu(s, a)$ 的方向更新策略——>让策略产生的动作朝着Q值增长的方向移动

实际实现中，我们最小化负Q值：

$$L(\theta^\mu) = -\mathbb{E}_{s \sim D} [Q(s, \mu(s|\theta^\mu))|\theta^Q)]$$

梯度下降：

$$\theta^\mu \leftarrow \theta^\mu - \alpha \nabla_{\theta^\mu} L(\theta^\mu)$$

目标网络更新**软更新机制：**

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

其中 $\tau \ll 1$ (通常为0.001)

算法流程

初始化:

在线Critic网络 $Q(s, a | \theta^Q)$ 和在线Actor网络 $\mu(s | \theta^\mu)$

目标网络 Q' 和 μ' , 参数 $\theta^Q' \leftarrow \theta^Q$, $\theta^\mu' \leftarrow \theta^\mu$

经验回放缓冲区 R

for episode = 1 to M **do**

 初始化随机过程 N 用于动作探索

 获取初始状态 s_1

for t = 1 to T **do**

 选择动作 $a_t = \mu(s_t | \theta^\mu) + N_t$

 执行动作 a_t , 观察奖励 r_t 和新状态 s_{t+1}

 存储转移样本 (s_t, a_t, r_t, s_{t+1}) 到 R

 从 R 中随机采样 N 个样本 (s_i, a_i, r_i, s_{i+1})

 设置 $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^\mu') | \theta^Q')$

 更新Critic: $\theta^Q \leftarrow \theta^Q - \alpha_Q \nabla_{\theta^Q} (1/N \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2)$

 更新Actor: $\theta^\mu \leftarrow \theta^\mu + \alpha_\mu \nabla_{\theta^\mu} (1/N \sum_i Q(s_i, \mu(s_i | \theta^\mu) | \theta^Q))$

 更新目标网络:

$\theta^Q' \leftarrow \tau \theta^Q + (1-\tau) \theta^Q$

$\theta^\mu' \leftarrow \tau \theta^\mu + (1-\tau) \theta^\mu$

end for

end for

```

开始训练
↓
初始化网络和经验回放缓冲区
↓
循环每个episode
↓
重置环境，获取初始状态
↓
循环每个时间步
|—— 选择动作（Actor网络 + 探索噪声）
|—— 执行动作，与环境交互
|—— 存储经验到回放缓冲区
|—— 从缓冲区采样小批量经验
|—— 更新Critic网络（最小化TD误差）
|—— 更新Actor网络（最大化Q值）
|—— 软更新目标网络
↓
直达到达最大episode数
↓
结束训练

```

为什么使用确定性策略梯度？

优势：

1. **样本效率**：不需要对动作空间积分
2. **计算效率**：梯度计算更简单
3. **理论保证**：确定性策略梯度是无偏估计

随机策略需要估计：

$$\mathbb{E}_{a \sim \pi} [\nabla_{\theta} \log \pi(a|s) Q(s, a)]$$

确定性策略只需要：

$$\nabla_{\theta} \mu(s) \nabla_a Q(s, a)|_{a=\mu(s)}$$

TD3算法

DDPG的目标Q值：

$$y = r + \gamma Q_{\theta'}(s', \mu_{\phi'}(s'))$$

问题1：函数近似误差

由于神经网络是函数近似器，存在近似误差：

$$\hat{Q}(s, a) = Q^*(s, a) + \epsilon(s, a)$$

问题2：最大化偏差

当使用同一个网络选择和评估动作时，会产生正向偏差：

$$\max_a \hat{Q}(s, a) \geq \max_a Q^*(s, a)$$

双Q网络 (Twin Q-Networks)

假设两个Q网络的误差 ϵ_1 和 ϵ_2 是独立同分布的，那么：

$$\mathbb{E}[\min(\hat{Q}_1, \hat{Q}_2)] = \mathbb{E}[\min(Q^* + \epsilon_1, Q^* + \epsilon_2)] = Q^* + \mathbb{E}[\min(\epsilon_1, \epsilon_2)]$$

由于 $\mathbb{E}[\min(\epsilon_1, \epsilon_2)] \leq \min(\mathbb{E}[\epsilon_1], \mathbb{E}[\epsilon_2]) = 0$ ，因此取最小值可以减少过估计。

目标Q值计算：

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a}')$$

其中 \tilde{a}' 是添加噪声后的目标动作。

目标策略平滑 (Target Policy Smoothing)

在目标动作上添加裁剪的噪声，相当于对Q函数进行正则化：

$$\begin{aligned}\tilde{a}'(s') &= \text{clip}(\mu_{\phi'}(s') + \epsilon, a_{low}, a_{high}) \\ \epsilon &\sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)\end{aligned}$$

这相当于对目标Q值进行平滑处理：

$$\mathbb{E}_{\epsilon}[Q(s', \mu_{\phi'}(s') + \epsilon)] \approx Q(s', \mu_{\phi'}(s')) + \frac{\sigma^2}{2} \nabla_a^2 Q(s', a)|_{a=\mu_{\phi'}(s')}$$

延迟策略更新 (Delayed Policy Updates)

如果Q函数估计不准确时更新策略，会导致策略朝着错误的方向更新。延迟更新确保在Q函数相对准确后再更新策略。

策略更新条件：

$$\text{if } t \bmod d = 0 : \text{update policy}$$

Critic损失函数

目标Q值计算：

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \text{clip}(\mu_{\phi'}(s') + \epsilon, -a_{max}, a_{max}))$$

Critic损失函数：

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} [(Q_{\theta_i}(s, a) - y)^2], \quad i = 1, 2$$

梯度下降：

$$\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}(\theta_i), i = 1, 2$$

Actor损失函数：

策略目标函数：

$$J(\phi) = -\mathbb{E}_{s \sim \mathcal{D}} [Q_{\theta_1}(s, \mu_{\phi}(s))]$$

策略梯度：

$$\nabla_{\phi} J(\phi) = -\mathbb{E}_{s \sim \mathcal{D}} [\nabla_a Q_{\theta_1}(s, a)|_{a=\mu_{\phi}(s)} \nabla_{\phi} \mu_{\phi}(s)]$$

注意： 关于使用 Q_{θ_1} 还是 $\min(Q_{\theta_1}, Q_{\theta_2})$ 存在讨论，原论文使用 Q_{θ_1} ，但实践中使用最小值可能更稳定。

梯度下降：

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} J(\phi)$$

目标网络更新

使用软更新

$$\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i, \quad i = 1, 2$$

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$

其中 $\tau \ll 1$, 通常取0.005。

算法流程

初始化:

Actor网络 μ_ϕ , Critic网络 $Q_{\theta 1}, Q_{\theta 2}$

目标网络 $\phi' \leftarrow \phi, \theta 1' \leftarrow \theta 1, \theta 2' \leftarrow \theta 2$

经验回放缓冲区 D

for 回合 = 1 to M do:

 初始化环境, 获得初始状态 s

 for 时间步 = 1 to T do:

 # 选择动作 (带探索噪声)

 a = clip($\mu_\phi(s) + \varepsilon, -a_{\max}, a_{\max}$), $\varepsilon \sim N(0, \sigma_{\text{explore}})$

 # 执行动作, 观察环境

 执行 a, 观察奖励 r, 下一个状态 s', 终止标志 done

 # 存储经验

 存储 (s, a, r, s', done) 到 D

 # 更新状态

 s \leftarrow s'

 # 如果缓冲区中有足够样本, 开始训练

 if |D| > N_min:

 # 从D中采样小批量 {(s, a, r, s', done)}

 # 目标策略平滑

$\varepsilon \sim \text{clip}(N(0, \sigma_{\text{target}}), -c, c)$

$\tilde{a} = \text{clip}(\mu_{\phi'}(s') + \varepsilon, -a_{\max}, a_{\max})$

 # 计算目标Q值 (双Q网络)

 y = r + $\gamma * (1 - \text{done}) * \min(Q_{\theta 1'}(s', \tilde{a}), Q_{\theta 2'}(s', \tilde{a}))$

 # 更新Critic网络

 计算 Critic 损失: $L_{\text{critic}} = \sum[(Q_{\theta i}(s, a) - y)^2] \text{ for } i=1, 2$

 梯度下降更新 $\theta 1, \theta 2$

 # 延迟策略更新

 if 时间步 mod d == 0:

 # 更新Actor网络

 计算 Actor 损失: $L_{\text{actor}} = -\sum[Q_{\theta 1}(s, \mu_{\phi}(s))]$

梯度下降更新 ϕ

```
# 软更新目标网络  
 $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$  for i=1,2  
 $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$ 
```

```
end for  
end for
```

3.2 算法流程图解

```
开始
  |
  └── 初始化网络和经验回放缓冲区
  |
  └── 对于每个回合:
    |
    └── 初始化环境状态
    |
    └── 对于每个时间步:
      |
      └── 选择动作 (Actor网络 + 探索噪声)
      |
      └── 执行动作, 获得经验
      |
      └── 存储经验到回放缓冲区
      |
      └── 如果缓冲区足够大:
        |
        └── 采样小批量经验
        |
        └── 计算目标Q值 (目标策略平滑 + 双Q网络)
        |
        └── 更新Critic网络
        |
        └── 如果满足延迟更新条件:
          |
          └── 更新Actor网络
          |
          └── 软更新目标网络
        |
        └── 更新状态
      |
      └── 回合结束
    |
    └── 训练完成
```

SAC 算法

最大熵目标

SAC引入了最大熵目标，在奖励基础上增加了策略的熵：

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

其中：

- $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a \sim \pi}[\log \pi(a|s_t)]$ 是策略的熵
- $\alpha > 0$ 是温度参数，平衡奖励和熵的重要性

为什么这样设计？

- **鼓励探索：**高熵策略更随机，避免过早陷入局部最优
- **鲁棒性：**学会多个等效行为，对环境变化更鲁棒
- **改善学习稳定性：**熵项作为正则化，防止策略过早收敛

软状态值函数

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$$

推导过程：

根据值函数的定义和最大熵目标：

$$\begin{aligned} V(s_t) &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} (r(s_k, a_k) + \alpha \mathcal{H}(\pi(\cdot|s_k))) \middle| s_t \right] \\ &= \mathbb{E}_{a_t \sim \pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) + \gamma \mathbb{E}_{s_{t+1}} [V(s_{t+1})]] \\ &= \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)] \end{aligned}$$

软动作值函数

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} [V(s_{t+1})]$$

软贝尔曼方程：

将软值函数代入Q函数：

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V(s_{t+1})] \\ &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} [\mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1})]] \end{aligned}$$

最优软策略

在最大熵框架下，最优策略具有封闭形式：

$$\pi^*(a|s) = \frac{\exp\left(\frac{1}{\alpha}Q^*(s, a)\right)}{Z(s)}$$

其中 $Z(s) = \sum_a \exp\left(\frac{1}{\alpha}Q^*(s, a)\right)$ 是配分函数。

推导：

通过求解带约束的优化问题：

$$\begin{aligned} \max_{\pi} \quad & \mathbb{E}_{a \sim \pi}[Q(s, a)] + \alpha \mathcal{H}(\pi(\cdot|s)) \\ \text{s.t.} \quad & \sum_a \pi(a|s) = 1 \end{aligned}$$

其中：

- $\pi(a|s)$ 是策略，表示在状态 s 下选择动作 a 的概率
- $Q(s, a)$ 是动作价值函数
- $\mathcal{H}(\pi(\cdot|s)) = -\sum_a \pi(a|s) \log \pi(a|s)$ 是策略的熵
- $\alpha > 0$ 是温度参数

引入拉格朗日乘子 λ 来处理概率归一化约束，构造拉格朗日函数：

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{a \sim \pi}[Q(s, a)] + \alpha \mathcal{H}(\pi(\cdot|s)) + \lambda \left(1 - \sum_a \pi(a|s)\right)$$

将期望和熵展开为显式求和形式：

$$\mathcal{L}(\pi, \lambda) = \sum_a \pi(a|s)Q(s, a) - \alpha \sum_a \pi(a|s) \log \pi(a|s) + \lambda \left(1 - \sum_a \pi(a|s)\right)$$

对于每个动作 a ，我们对 $\pi(a|s)$ 求偏导：

$$\frac{\partial \mathcal{L}}{\partial \pi(a|s)} = Q(s, a) - \alpha(\log \pi(a|s) + 1) - \lambda$$

令偏导数为零（极值条件）：

$$Q(s, a) - \alpha(\log \pi(a|s) + 1) - \lambda = 0$$

从上述方程解出 $\pi(a|s)$ ：

$$\log \pi(a|s) = \frac{Q(s, a) - \alpha - \lambda}{\alpha}$$

$$\pi(a|s) = \exp\left(\frac{Q(s, a) - \alpha - \lambda}{\alpha}\right)$$

$$\pi(a|s) = \exp\left(\frac{Q(s, a)}{\alpha}\right) \cdot \exp\left(-\frac{\alpha + \lambda}{\alpha}\right)$$

利用概率归一化约束 $\sum_a \pi(a|s) = 1$ ：

$$\sum_a \exp\left(\frac{Q(s, a)}{\alpha}\right) \cdot \exp\left(-\frac{\alpha + \lambda}{\alpha}\right) = 1$$

$$\exp\left(-\frac{\alpha + \lambda}{\alpha}\right) \cdot \sum_a \exp\left(\frac{Q(s, a)}{\alpha}\right) = 1$$

$$\exp\left(-\frac{\alpha + \lambda}{\alpha}\right) = \frac{1}{\sum_a \exp\left(\frac{Q(s, a)}{\alpha}\right)}$$

定义配分函数 $Z(s) = \sum_a \exp\left(\frac{Q(s, a)}{\alpha}\right)$, 则：

$$\exp\left(-\frac{\alpha + \lambda}{\alpha}\right) = \frac{1}{Z(s)}$$

将结果代回 $\pi(a|s)$ 的表达式：

$$\pi(a|s) = \exp\left(\frac{Q(s, a)}{\alpha}\right) \cdot \frac{1}{Z(s)}$$

因此，最优策略为：

$$\pi^*(a|s) = \frac{\exp\left(\frac{1}{\alpha} Q(s, a)\right)}{Z(s)}$$

其中 $Z(s) = \sum_a \exp\left(\frac{1}{\alpha} Q(s, a)\right)$ 是配分函数，确保概率归一化。

对于连续动作空间，求和变为积分：

$$\pi^*(a|s) = \frac{\exp\left(\frac{1}{\alpha}Q(s, a)\right)}{\int \exp\left(\frac{1}{\alpha}Q(s, a)\right) da}$$

在实际情况中，**无法直接使用**这个理论最优策略，因为：

1. 我们不知道真实的 Q^* 函数
2. 配分函数 $Z(s)$ 很难计算（需要对所有动作积分）
3. 需要一个可以参数化、可以微分的策略

KL散度最小化

既然无法直接得到理论最优策略，我们就让参数化策略 π_ϕ **尽可能接近**理论最优策略。

通过最小化KL散度来实现这一点：

$$\pi_{\text{new}} = \arg \min_{\pi} D_{\text{KL}} \left(\pi(\cdot|s_t) \middle\| \frac{\exp\left(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t, \cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)} \right)$$

等价于最小化策略的函数：

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_\phi} \left(\alpha \log \pi_\phi(a_t|s_t) - \min_{i=1,2} Q_{\theta_i}(s_t, a_t) \right) \right]$$

梯度下降：

$$\phi \leftarrow \phi - \eta_\pi \nabla_\phi J_\pi(\phi)$$

KL散度 (Kullback-Leibler Divergence) 是衡量**两个概率分布差异**的指标。可以把它想象成“概率分布之间的距离”，不过它不是一个真正的距离度量（不满足对称性和三角不等式）。

定义

对于两个离散概率分布 P 和 Q ，KL散度定义为：

$$D_{\text{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

对于连续分布：

$$D_{\text{KL}}(P\|Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$$

性质**非负性**

$$D_{\text{KL}}(P\|Q) \geq 0$$

当且仅当 $P = Q$ 时，等号成立

不对称性

$$D_{\text{KL}}(P\|Q) \neq D_{\text{KL}}(Q\|P)$$

这意味着KL散度不是对称的， P 对 Q 的KL散度与 Q 对 P 的KL散度不同。

信息论视角：

- $-\log Q(x)$ 表示用分布 Q 来编码事件 x 时所需的信息量（惊讶程度）
- $-\log P(x)$ 表示用真实分布 P 来编码事件 x 时所需的信息量
- KL散度衡量了用错误分布 Q 编码相比用真实分布 P 编码多花费的平均信息量

为什么在SAC中使用KL散度？

让参数化策略 π_ϕ 接近理论最优策略 π^*

为什么选择KL散度？

1. **直接衡量分布差异**: KL散度专门设计用来比较概率分布
2. **有良好的数学性质**: 非负性确保我们有明确的最小值 (0)
3. **信息论解释**: 最小化KL散度相当于让我们的策略用最少的额外"惊讶"来模拟最优策略
4. **可优化性**: KL散度对于策略参数通常是可微的

在SAC中，我们使用的是**前向KL散度**：

$$D_{\text{KL}}(\pi_\phi\|\pi^*)$$

前向KL的特性：

- **模式覆盖** (Mode Covering): 倾向于让 π_ϕ 覆盖 π^* 的所有模式
- **避免零概率**: 如果 $\pi^*(a|s) > 0$, 前向KL会惩罚 $\pi_\phi(a|s) = 0$ 的情况

反向KL的特性：

$$D_{\text{KL}}(\pi^* \parallel \pi_\phi)$$

- **模式寻求** (Mode Seeking)：只覆盖 π^* 的主要模式
- **允许零概率**：即使 $\pi^*(a|s) > 0$, 也允许 $\pi_\phi(a|s) = 0$

在SAC中，选择前向KL，希望策略能够探索所有可能的好动作，而不仅仅是当前认为最好的动作。

重参数化技巧 (Reparameterization Trick) :

为了计算梯度，我们使用：

$$a = f_\phi(s, \epsilon), \quad \epsilon \sim \mathcal{N}(0, I)$$

这样梯度可以通过采样过程回传。

梯度计算：

$$\nabla_\phi J_\pi(\phi) = \nabla_\phi \mathbb{E}_{a \sim \pi_\phi} [\alpha \log \pi_\phi(a|s) - Q(s, a)]$$

使用重参数化：

$$= \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} [\nabla_\phi (\alpha \log \pi_\phi(f_\phi(s, \epsilon)|s) - Q(s, f_\phi(s, \epsilon)))]$$

KL散度定义

$$D_{\text{KL}}(P \parallel Q) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right]$$

应用到我们的情况：

$$D_{\text{KL}} \left(\pi \middle\| \frac{\exp(Q/\alpha)}{Z} \right) = \mathbb{E}_{a \sim \pi} \left[\log \frac{\pi(a|s)}{\frac{\exp(Q(s,a)/\alpha)}{Z(s)}} \right]$$

展开对数项

$$\begin{aligned}
& \text{base_DRL_model_1} \\
& = \mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) - \log \left(\frac{\exp(Q(s, a)/\alpha)}{Z(s)} \right) \right] \\
& = \mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) - \left(\frac{Q(s, a)}{\alpha} - \log Z(s) \right) \right] \\
& = \mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) - \frac{Q(s, a)}{\alpha} + \log Z(s) \right]
\end{aligned}$$

分离常数项

$$= \mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) - \frac{Q(s, a)}{\alpha} \right] + \log Z(s)$$

$\log Z(s)$ 与策略 π 无关，所以在优化时可以忽略。最小化KL散度等价于最小化：

$$\mathbb{E}_{a \sim \pi} \left[\log \pi(a|s) - \frac{Q(s, a)}{\alpha} \right]$$

乘以 α , 不影响最优解：

$$\min_{\pi} \mathbb{E}_{a \sim \pi} [-Q(s, a) + \alpha \log \pi(a|s)]$$

最终策略目标函数：

$$J_{\pi}(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} [\mathbb{E}_{a_t \sim \pi_{\phi}} [\alpha \log \pi_{\phi}(a_t|s_t) - Q_{\theta}(s_t, a_t)]]$$

使用两个Q网络，取最小值：

$$J_{\pi}(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_{\phi}} \left(\alpha \log \pi_{\phi}(a_t|s_t) - \min_{i=1,2} Q_{\theta_i}(s_t, a_t) \right) \right]$$

梯度下降：

$$\phi \leftarrow \phi - \eta_{\pi} \nabla_{\phi} J_{\pi}(\phi)$$

Q函数学习

使用软贝尔曼方程作为目标，最小化贝尔曼误差：

$$J_Q(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} (Q_{\theta_i}(s_t, a_t) - y(s_t, a_t))^2 \right], i = 1, 2$$

梯度下降：

$$\theta_{i,t+1} = \theta_{i,t} - \alpha \nabla J_Q(\theta_{i,t})$$

其中目标Q值(y)为：

$$y(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V_{\bar{\psi}}(s_{t+1})]$$

为了减少Q值的高估，SAC使用两个Q网络并取最小值：

$$y(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[\min_{i=1,2} Q_{\bar{\theta}_i}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1}) \right]$$

即：

$$y(s_t, a_t) = r(s_t, a_t) + \gamma \left[\min_{i=1,2} Q_{\bar{\theta}_i}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1}) \right]$$

自动熵调整

将熵项视为约束，优化问题变为：

$$\begin{aligned} & \max_{\pi} \mathbb{E} \left[\sum_t \gamma^t r(s_t, a_t) \right] \\ \text{s.t. } & \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi}} [-\log \pi(a_t | s_t)] \geq \mathcal{H}_0 \end{aligned}$$

其中：

- ρ_{π} 是策略 π 下的状态-动作访问分布
- \mathcal{H}_0 是目标熵值
- 约束条件表示策略的平均熵应该至少为 \mathcal{H}_0

温度参数的优化目标为：

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t} [-\alpha \log \pi_t(a_t | s_t) - \alpha \mathcal{H}_0]$$

梯度下降：

$$\alpha_{t+1} = \alpha_t - \eta \nabla J(\alpha_t)$$

引入拉格朗日乘子 $\alpha \geq 0$ (因为是不等式约束)，构造拉格朗日函数：

$$\mathcal{L}(\pi, \alpha) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \alpha (\mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [-\log \pi(a_t | s_t)] - \mathcal{H}_0)$$

对偶问题：

$$\min_{\alpha \geq 0} \max_{\pi} \mathcal{L}(\pi, \alpha)$$

要：

1. 对于固定的 α , 找到最优策略 π^* 最大化 $\mathcal{L}(\pi, \alpha)$
2. 然后调整 α 来最小化这个最大值

内层最大化问题 (固定 α) :对于固定的 α , 求解:

$$\max_{\pi} \left\{ \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \alpha (\mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [-\log \pi(a_t | s_t)] - \mathcal{H}_0) \right\}$$

由于 $\alpha \mathcal{H}_0$ 是常数 (对于固定的 α), 可以忽略不影响优化:

$$\max_{\pi} \left\{ \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \alpha \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [-\log \pi(a_t | s_t)] \right\}$$

这正好是标准的最大熵强化学习目标!

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) - \alpha \log \pi(a_t | s_t)) \right]$$

外层最小化问题 (优化 α)

现在考虑外层对 α 的优化。对于最优策略 $\pi^*(\alpha)$ (依赖于 α), 对偶函数为:

$$D(\alpha) = \max_{\pi} \mathcal{L}(\pi, \alpha) = \mathcal{L}(\pi^*(\alpha), \alpha)$$

需要求解:

$$\min_{\alpha \geq 0} D(\alpha) = \min_{\alpha \geq 0} \mathcal{L}(\pi^*(\alpha), \alpha)$$

为了使用梯度下降优化 α , 我们需要计算 $\nabla_{\alpha} D(\alpha)$ 。

根据包络定理 (Envelope Theorem), 当 π 是最优时, 有:

$$\nabla_{\alpha} D(\alpha) = \nabla_{\alpha} \mathcal{L}(\pi^*(\alpha), \alpha)$$

计算梯度：

$$\nabla_{\alpha} \mathcal{L}(\pi, \alpha) = \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi}} [-\log \pi(a_t | s_t)] - \mathcal{H}_0$$

因此：

$$\nabla_{\alpha} D(\alpha) = \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi^*(\alpha)}} [-\log \pi^*(a_t | s_t)] - \mathcal{H}_0$$

在实际算法中，我们使用随机梯度下降来优化 α ：

$$\alpha \leftarrow \alpha - \eta \nabla_{\alpha} D(\alpha)$$

其中 η 是学习率。

代入梯度表达式：

$$\alpha \leftarrow \alpha - \eta \left(\mathbb{E}_{(s_t, a_t) \sim \rho_{\pi^*(\alpha)}} [-\log \pi^*(a_t | s_t)] - \mathcal{H}_0 \right)$$

现在，我们想要构造一个目标函数 $J(\alpha)$ ，使得最小化 $J(\alpha)$ 等价于执行上述梯度下降。

观察梯度表达式：

$$\nabla_{\alpha} D(\alpha) = \mathbb{E} [-\log \pi(a_t | s_t)] - \mathcal{H}_0$$

希望找到一个函数 $J(\alpha)$ 满足：

$$\nabla_{\alpha} J(\alpha) = \mathbb{E} [-\log \pi(a_t | s_t)] - \mathcal{H}_0$$

一个自然的选择是：

$$J(\alpha) = \mathbb{E} [-\alpha \log \pi(a_t | s_t) - \alpha \mathcal{H}_0]$$

因此，温度参数 α 的优化目标为：

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t} [-\alpha \log \pi_t(a_t | s_t) - \alpha \mathcal{H}_0]$$

在实际实现中，我们使用经验估计：

$$J(\alpha) \approx \frac{1}{N} \sum_{i=1}^N [-\alpha \log \pi(a_i | s_i) - \alpha \mathcal{H}_0]$$

在SAC算法中，完整的自动熵调整流程为：

1. **初始化**: $\alpha > 0$, 目标熵 \mathcal{H}_0 (通常设为 $-dim(\mathcal{A})$)
2. **策略优化** (固定 α):

$$\max_{\pi} \mathbb{E} \left[\sum_t \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))) \right]$$

3. **温度参数优化**:

$$\min_{\alpha} \mathbb{E} [-\alpha \log \pi(a_t|s_t) - \alpha \mathcal{H}_0]$$

4. **梯度下降更新**:

$$\alpha \leftarrow \alpha - \eta (\mathbb{E}[-\log \pi(a_t|s_t)] - \mathcal{H}_0)$$

完整SAC算法流程

初始化：

策略网络 π_ϕ , Q网络 $Q_{\theta 1}, Q_{\theta 2}$
 目标Q网络 $Q_{\bar{\theta}1}, Q_{\bar{\theta}2}$ (权重 $\theta_i \leftarrow \bar{\theta}_i$)
 经验回放缓冲区 D
 可学习的温度参数 α (如果使用自动调整)

for 每个迭代步骤 do:

数据收集
 从环境中采样动作 $a_t \sim \pi_\phi(\cdot | s_t)$
 执行动作 a_t , 观测奖励 r_t 和下一个状态 s_{t+1}
 存储转移 (s_t, a_t, r_t, s_{t+1}) 到 D

模型更新

for 每次梯度更新 do:
 # 从D中采样批次 $B = \{(s, a, r, s')\}$

Q函数更新
 对每个 $(s, a, r, s') \in B$:
 $a' \sim \pi_\phi(\cdot | s')$
 $y = r + \gamma * [\min(Q_{\bar{\theta}1}(s', a'), Q_{\bar{\theta}2}(s', a')) - \alpha * \log \pi_\phi(a' | s')]$

更新 $Q_{\theta 1}$ 最小化 $(Q_{\theta 1}(s, a) - y)^2$
 更新 $Q_{\theta 2}$ 最小化 $(Q_{\theta 2}(s, a) - y)^2$

策略更新

采样新动作 $\tilde{a} \sim \pi_\phi(\cdot | s)$ (使用重参数化)
 更新 π_ϕ 最小化 $[\alpha * \log \pi_\phi(\tilde{a} | s) - \min(Q_{\theta 1}(s, \tilde{a}), Q_{\theta 2}(s, \tilde{a}))]$

温度参数更新 (如果使用自动调整)

更新 α 最小化 $\alpha * [-\log \pi_\phi(\tilde{a} | s) - H_\alpha]$

目标网络更新

$\bar{\theta}_i \leftarrow \tau \theta_i + (1-\tau) \bar{\theta}_i$ for $i = 1, 2$

end for

end for

算法步骤

步骤1：网络初始化

- 策略网络 π_ϕ
- 两个Q网络 $Q_{\theta_1}, Q_{\theta_2}$
- 对应的目标网络 $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$
- 经验回放缓冲区 \mathcal{D}

步骤2：数据收集

- 对于每个时间步 t :
 - 根据当前策略选择动作 $a_t \sim \pi_\phi(\cdot|s_t)$
 - 执行动作，观测奖励 r_t 和下一状态 s_{t+1}
 - 存储转移 (s_t, a_t, r_t, s_{t+1}) 到 \mathcal{D}

步骤3：Q函数更新

对于采样的批次 $(s, a, r, s') \sim \mathcal{D}$:

1. 采样下一动作 $a' \sim \pi_\phi(\cdot|s')$
2. 计算目标Q值：

$$y = r + \gamma \left(\min_{i=1,2} Q_{\bar{\theta}_i}(s', a') - \alpha \log \pi_\phi(a'|s') \right)$$

3. 更新Q网络参数：

$$\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\theta_i}(s, a) - y)^2, i = 1, 2$$

步骤4：策略更新

1. 使用重参数化采样新动作 $\tilde{a} \sim \pi_\phi(\cdot|s)$
2. 更新策略参数：

$$\phi \leftarrow \phi - \lambda_\pi \nabla_\phi \frac{1}{|B|} \sum_{s \in B} \left(\alpha \log \pi_\phi(\tilde{a}|s) - \min_{i=1,2} Q_{\theta_i}(s, \tilde{a}) \right)$$

步骤5：温度参数更新（如果使用自动调整）

$$\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \frac{1}{|B|} \sum_{s \in B} (-\alpha (\log \pi_\phi(\tilde{a}|s) - \mathcal{H}_0))$$

步骤6：目标网络更新

$$\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i \quad \text{for } i = 1, 2$$

重参数化技巧 (Reparameterization Trick)

对于连续动作空间，策略通常输出高斯分布的参数：

$$a = \tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \epsilon), \quad \epsilon \sim \mathcal{N}(0, I)$$

这使得梯度可以通过随机性回传到策略网络。

随机采样的不可微性

在强化学习中，当我们从策略分布中采样动作时：

$$a \sim \pi_\phi(a|s)$$

这个采样操作是**不可微的**，意味着梯度无法通过采样操作反向传播。

- 策略网络输出分布参数（如高斯分布的均值和方差）
- 我们从该分布中随机采样得到动作
- 使用动作计算损失函数
- 但梯度无法通过随机采样回传到策略网络参数

将随机采样分解为：

1. 从固定分布中采样随机噪声
2. 通过确定性变换得到最终样本

原始不可微形式：

$$a \sim \mathcal{N}(\mu_\phi(s), \sigma_\phi(s)^2)$$

这里 a 是随机变量，与 ϕ 的关系不是函数关系。

重参数化形式：

$$\epsilon \sim \mathcal{N}(0, 1)$$

$$a = \mu_\phi(s) + \sigma_\phi(s) \cdot \epsilon$$

现在 a 是 ϕ 和 ϵ 的确定性函数。

目标：计算 $\nabla_\phi \mathbb{E}_{a \sim \pi_\phi}[f(a)]$

不可微形式（困难）：

$$\nabla_{\phi} \mathbb{E}_{a \sim \pi_{\phi}}[f(a)] = \nabla_{\phi} \int f(a) \pi_{\phi}(a) da$$

重参数化形式（可微）：

设 $a = g_{\phi}(\epsilon)$, 其中 $\epsilon \sim p(\epsilon)$

$$\begin{aligned}\nabla_{\phi} \mathbb{E}_{a \sim \pi_{\phi}}[f(a)] &= \nabla_{\phi} \mathbb{E}_{\epsilon \sim p(\epsilon)}[f(g_{\phi}(\epsilon))] \\ &= \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_{\phi} f(g_{\phi}(\epsilon))]\end{aligned}$$

算法流程

对于高斯策略，完整流程如下：

步骤1：策略网络输出分布参数

```
def forward(self, state):
    # 策略网络输出均值和标准差
    mean = self.mean_layer(x)
    log_std = self.log_std_layer(x)
    log_std = torch.clamp(log_std, self.log_std_min, self.log_std_max)
    std = torch.exp(log_std)
    return mean, std
```

步骤2：重参数化采样

```

def sample(self, state, epsilon=1e-6):
    mean, std = self.forward(state)

    # 从标准正态分布采样噪声
    noise = torch.randn_like(mean)

    # 重参数化：确定性变换
    action = mean + std * noise

    # 计算对数概率（用于策略梯度）
    log_prob = self.log_prob(mean, std, noise)

    # Tanh变换和概率修正
    action_tanh = torch.tanh(action)
    log_prob -= torch.log(1 - action_tanh.pow(2) + epsilon)
    log_prob = log_prob.sum(1, keepdim=True)

    return action_tanh, log_prob

```

步骤3：对数概率计算

```

def log_prob(self, mean, std, noise):
    """计算给定噪声下的对数概率"""
    return (-0.5 * noise.pow(2) - torch.log(std) - 0.5 * np.log(2 * np.pi)).sum(1, keepdim=True)

```

完整的前向-反向传播流程

前向传播：

```

输入状态 s
↓
策略网络 → (μ, σ)
↓
采样 ε ~ N(0,1)
↓
动作 a = μ + σ ⊕ ε
↓
Tanh变换 a_tanh = tanh(a)
↓
计算损失 L

```

反向传播：

```

梯度 ∂L/∂a_tanh
↓
梯度 ∂L/∂a = ∂L/∂a_tanh · (1 - tanh²(a))
↓
梯度 ∂L/∂μ = ∂L/∂a · 1
梯度 ∂L/∂σ = ∂L/∂a · ε
↓
更新策略网络参数 φ

```

Tanh变换的概率修正

为什么需要概率修正？

当我们使用Tanh将动作限制在[-1, 1]范围内时，概率密度发生了变化：

原始分布： $a \sim \mathcal{N}(\mu, \sigma^2)$

变换后： $a_{\text{tanh}} = \tanh(a)$

根据概率论中的变量变换公式：

$$p(a_{\text{tanh}}) = p(a) \cdot \left| \frac{da}{da_{\text{tanh}}} \right|$$

对数概率修正：

$$\log \pi(a_{\text{tanh}}|s) = \log \pi(a|s) - \log(1 - \tanh^2(a) + \epsilon)$$

当 $|a|$ 很大时, $\tanh(a)$ 接近 ± 1 , 因此 $1 - \tanh^2(a)$ 接近 0。为了数值稳定性问题, 需要添加一个很小的常数项 ϵ 。

代码实现:

```
action_tanh = torch.tanh(action)
log_prob -= torch.log(1 - action_tanh.pow(2) + epsilon)
```

在SAC中的具体应用

```
def update_policy(self, state_batch):
    # 重参数化采样新动作
    new_action, log_prob = self.policy_net.sample(state_batch)

    # 评估新动作的Q值
    q1_new = self.q_net1(state_batch, new_action)
    q2_new = self.q_net2(state_batch, new_action)
    q_new = torch.min(q1_new, q2_new)

    # 策略损失 (梯度可以通过重参数化回传)
    policy_loss = (self.alpha * log_prob - q_new).mean()

    # 反向传播
    self.policy_optimizer.zero_grad()
    policy_loss.backward()
    self.policy_optimizer.step()
```

梯度流分析

```
policy_loss
↓
q_new ← Q网络 ← new_action
↑
log_prob ← 概率修正 ← action_tanh
↑
action = μ + σ·ε ← 策略网络参数(μ,σ)
↑
状态 s
```

关键: 梯度可以通过确定性的路径 $s \rightarrow \mu, \sigma \rightarrow a \rightarrow \text{loss}$ 流动, 而随机性被隔离在 ϵ 中。

