

- 多臂老虎机问题 (multi-arm bandit)
 - 多臂老虎机问题的基本概念
 - 探索与利用的权衡
 - 解决多臂老虎机问题的常见策略
 - 贪心策略
 - ϵ -贪婪策略 (ϵ -Greedy Strategy)
 - 上置信界策略 (Upper Confidence Bound, UCB)
 - 遗憾函数 (Regret Function)
 - 定义
 - 贪心策略 与 ϵ -greedy 策略
 - 马尔可夫决策过程 (Markov Decision Process, MDP)
- 马尔可夫过程 (Markov Process)
 - 随机过程
 - 马尔可夫性质
 - 马尔可夫奖励过程 (Markov Reward Process, MRP)
 - 定义
 - 价值函数 (Value Function)
 - 马尔可夫决策过程 (Markov Decision Process, MDP)
 - MDP的定义
 - MDP的动态
 - 价值函数
 - 贝尔曼方程
 - 最优策略
- 动态规划算法
 - 策略迭代 (Policy Iteration)
 - 策略评估 (Policy Evaluation)
 - 策略改进 (Policy Improvement)
 - 价值迭代 (Value Iteration)
 - 价值迭代的步骤
 - 同步 vs. 异步价值迭代
 - 同步价值迭代 (Synchronous Value Iteration)
 - 异步价值迭代 (Asynchronous Value Iteration)
 - 价值迭代与策略迭代的比较
- 基于值函数的无模型强化学习
 - 蒙特卡洛采样方法 (MC)
 - 第一回报蒙特卡洛方法 (First-Visit Monte Carlo)
 - 每回报蒙特卡洛方法 (Every-Visit Monte Carlo)

- 离线策略蒙特卡洛方法（Offline Policy Monte Carlo with Importance Sampling）
 - 目标策略（Target Policy）和行为策略（Behavior Policy）
 - 重要性采样（Importance Sampling）
 - 算法步骤
- 时序差分方法 (TD)
- SARSA (State-Action-Reward-State-Action)
- Q-learning
- 基于值函数的深度强化学习
 - 对状态/动作进行离散化或分桶
 - 基本概念
 - 离散化方法
 - 对大型马尔可夫决策过程分桶
 - 构建参数化的值函数近似估计
 - 蒙特卡洛状态值函数近似
 - 时序差分状态值函数近似
 - 深度Q网络（DQN, Deep Q-Network）
 - DQN算法流程
 - DQN的扩展算法
 - Q-learning中Q函数的过估计问题
 - Double DQN
 - 优先经验回放（Prioritized Experience Replay）
- 策略梯度算法
 - REINFORC
 - 核心思想
 - 主要步骤
 - Actor-Critic
 - 核心思想
 - 主要步骤
 - Trust Region
 - Trust Region Policy Optimization (TRPO)
 - 算法步骤
 - Proximal Policy Optimization (PPO)
 - PPO的核心思想
 - 剪辑版本的PPO
 - 自适应KL惩罚版本的PPO
 - 实现步骤
- 规划与学习

- 结合学习模型（Model-Based Learning）
 - Dyna架构：
 - Dyna-Q算法详细介绍
- 决策时规划（Planning at Decision Time）
 - 决策时规划的关键方法
 - 决策时规划的优点和缺点

多臂老虎机问题（multi-arm bandit）

多臂老虎机问题的基本概念

▣ 形式描述：多臂老虎机（multi-arm bandit）问题

动作集合： $a^i \in \mathcal{A}, i = 1, \dots, K$

奖励概率分布： $\mathcal{R}(r | a^i) = \mathbb{P}(r | a^i)$

▣ 目标：最大化累积时间的收益 $\max \sum_{t=1}^T r_t, r_t \sim \mathcal{R}(\cdot | a_t)$

▣ 问题：每次拉下摇臂，是否出钱与出多少钱都是不确定的，这个不确定的反馈函数该如何估计？

1. 臂（Arm）：

每个臂代表一个选择，每个选择有一个固定但未知的奖励分布。

2. 奖励（Reward）：

每次选择一个臂，都会从对应的奖励分布中抽取一个奖励。不同臂的奖励分布可能不同，目标是最大化累计奖励。

3. 策略（Policy）：

决策者根据已知的信息选择哪个臂，这种选择方式称为策略。策略需要在探索（exploration）和利用（exploitation）之间进行权衡。

探索与利用的权衡

- 探索（Exploration）：尝试不同的臂，收集关于各个臂的奖励信息，目的是了解各个臂的奖励分布。
- 利用（Exploitation）：选择已知收益最高的臂，以最大化当前的奖励。

平衡探索与利用是多臂老虎机问题的核心挑战，因为过多的探索会浪费时间在次优选择上，而过多的利用可能会错过更好的选择。

解决多臂老虎机问题的常见策略

贪心策略

- 定义: 在每个时间步，选择当前估计值最高的选项，即选择当前认为最优的臂。这种策略不进行探索，仅基于当前信息进行选择。
- 优点: 简单直接，计算和实现都很容易。在初期或者当奖励分布非常稳定且已知时，贪心策略可能表现良好。
- 缺点: 完全忽略探索，可能导致长期回报不理想。容易陷入局部最优，不能发现更优的选择。

- 贪心策略

$$Q(a^i) = \frac{1}{N(a^i)} \sum_{t=1}^T r_t \cdot 1(a_t = a^i) \quad c$$



$$a^* = \arg \max_{a^i} Q(a^i)$$

r_t ：代表t时刻选择动作 a_t 获得的奖励

$1(a_t = a^i)$ ：代表当 $a_t = a^i$ 时返回1，其他情况返回0

ϵ -贪婪策略（ ϵ -Greedy Strategy）

以概率 ϵ 进行探索，即随机选择一个臂；以概率 $1-\epsilon$ 进行利用，即选择当前平均奖励最高的臂。

- ϵ -greedy 策略

$$a_t = \begin{cases} \arg \max_a Q(a) & \text{采样概率: } 1 - \epsilon \\ U(0, |\mathcal{A}|) & \text{采样概率: } \epsilon \end{cases}$$

\mathcal{A} : 代表动作空间

$|\mathcal{A}|$: 代表可选动作的个数

U : 代表均匀分布

上置信界策略（Upper Confidence Bound, UCB）

考虑选择的臂的平均奖励和选择频率，公式为：

$$UCB_i = \hat{\mu}_i + c \sqrt{\frac{\ln t}{n_i}}$$

其中 $\hat{\mu}_i$ 是臂 i 的平均奖励， t 是总选择次数， n_i 是臂 i 被选择的次数， c 是调节参数。

选择 UCB 值最大的臂。

遗憾函数（Regret Function）

遗憾（Regret）是衡量在选择过程中错失的潜在奖励，即实际获得的奖励与在最佳策略下可能获得的奖励之间的差距。通过计算和分析遗憾，我们可以评估不同策略在探索与利用之间的平衡效果。

定义

▣ 决策的期望收益: $Q(a^i) = \mathbb{E}_{r \sim \mathbb{P}(r|a^i)}[r]$

▣ 最优期望收益: $Q^* = \max_{a^i \in \mathcal{A}} Q(a^i)$

Regret

▣ Regret就是决策与最优决策的期望收益差:

$$R(a^i) = Q^* - Q(a^i)$$

▣ Total Regret 累计懊悔函数:

$$\sigma_R = \mathbb{E}_{a \sim \pi} \left[\sum_{t=1}^T R(a_t^i) \right]$$

等价性

贪心策略 与 ϵ -greedy 策略

▣ 贪心策略

$$Q(a^i) = \frac{1}{N(a^i)} \sum_{t=1}^T r_t \cdot 1(a_t = a^i)$$

↓

$$a^* = \arg \max_{a^i} Q(a^i)$$

$$\sigma_R \propto T \cdot [Q(a^i) - Q^*]$$

线性增长的 Total regret

▣ ϵ -greedy 策略

• ϵ 是一个比较小的数, 例如0.01

$$a_t = \begin{cases} \arg \max_{a^i} Q(a^i) & \text{采样概率: } 1 - \epsilon \\ U(0, |\mathcal{A}|) & \text{采样概率: } \epsilon \end{cases}$$

常量 ϵ 保证 total regret 满足

$$\sigma_R \geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Delta_a$$

Δ_a : 每个a的期望收益与最优期望收益差

马尔可夫决策过程 (Markov Decision Process, MDP)

较普遍的强化学习框架, 涉及状态转移和长期决策, 能够处理动态和复杂的环境

多臂老虎机问题扩展到 MDP，需要引入状态空间、状态转移概率和动作依赖的奖励结构。

马尔可夫过程（Markov Process）

马尔可夫过程是指满足马尔可夫性质的随机过程

随机过程

一个随机过程是一个由随机变量索引的集合，这些随机变量随时间或某个参数变化。例如，可以认为是一个系统在不同时间点上的状态的集合。

马尔可夫性质

□ 定义：

- 状态 S_t 是马尔可夫的，当且仅当

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$

□ 性质：

- 状态从历史（**history**）中捕获了所有相关信息
- 当前状态已知的时候，历史和未来是独立的
- 也就是说，**当前状态是未来的充分统计量**

马尔可夫奖励过程（Markov Reward Process, MRP）

在马尔可夫过程的基础上引入奖励机制的模型。它可以用一个四元组 (S, P, γ, R) 来表示

定义

1. 状态集合 S ，表示系统所有可能的状态

2. 状态转移概率矩阵 P ，定义了从一个状态转移到另一个状态的概率
3. 奖励函数 R ，定义了每个状态的即时奖励
4. 折扣因子 γ ，对未来奖励的折扣因子。 $[0,1]$ 内，当需要评估累积收益的时候，需要考虑与当前时间距离远近的奖励的权重

价值函数（Value Function）

为了评估一个状态下的长期收益，我们引入 价值函数（Value Function），通常记作 $V(s)$ ，它表示从状态 s 出发的累积期望奖励。价值函数可以通过贝尔曼方程（Bellman Equation）进行递归定义：

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s' | s) V(s')$$

这个方程表明状态 s 的价值是当前状态的奖励加上折扣后的预期未来价值。可以将这个方程展开为

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s' | s) [R(s') + \gamma \sum_{s'' \in S} P(s'' | s') V(s'')]$$

递归展开，可以看到价值函数考虑了当前奖励以及所有未来可能状态的累积奖励

马尔可夫决策过程（Markov Decision Process, MDP）

MDP的定义

一个MDP通常由以下五个元素组成：

1. 状态集合 S ：描述环境可能的所有状态。
2. 动作集合 A ：描述在每个状态下决策者可以采取的所有可能动作。
3. 状态转移概率 $P(s' | s, a)$ ：描述从状态 s 采取动作 a 后转移到状态 s' 的概率。
4. 奖励函数 $R(s, a, s')$ ：描述从状态 s 采取动作 a 并转移到状态 s' 所得到的即时奖励。
5. 折扣因子 γ ：介于 0 和 1 之间，用于平衡当前奖励与未来奖励的重要性。

MDP的动态

□ MDP的动态如下所示:

- 从状态 s_0 开始
- 智能体选择某个动作 $a_0 \in A$
- MDP随机转移到下一个状态 $s_1 \sim P_{s_0 a_0}$
- 智能体得到奖励 $R(s_0, a_0)$
- 以上过程不断进行, 直到终止

$$s_0 \xrightarrow{a_0, R(s_0, a_0)} s_1 \xrightarrow{a_1, R(s_1, a_1)} s_2 \xrightarrow{a_2, R(s_2, a_2)} s_3 \cdots s_T$$

- 直到终止状态 s_T 出现为止, 或者无止尽地进行下去
- 智能体的回报Return为

$$G = R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots \gamma^T R(s_T, a_T)$$

折扣因子 γ 在累积奖励公式中越往后乘得越小的原因在于:

1. 平衡当前和未来的奖励: 让决策既考虑当前的奖励也考虑未来的奖励, 但对未来奖励的重视程度逐渐减小。
2. 防止无限累积奖励: 确保累积奖励在无限时间步下收敛到一个有限值。
3. 模拟现实中的决策偏好: 符合人们对近期收益更为重视的现实决策倾向。
4. 数学性质和收敛性: 保证累积奖励的几何级数求和具有良好的数学性质, 便于分析和计算。

价值函数

为了在MDP中找到最佳策略, 我们引入两个关键的价值函数:

1. **状态价值函数** $V^\pi(s)$: 表示在状态 s 下按照策略 π 行动的期望累积奖励。
2. **状态-动作价值函数** $Q^\pi(s, a)$: 表示在状态 s 下采取动作 a 并随后按照策略 π 行动的期望累积奖励。

1. 状态价值函数 ($V^\pi(s)$):

- 解释: 状态价值函数 ($V^\pi(s)$) 表示在给定状态 (s) 下, 按照策略 (π) 行动所能获得的期望累积奖励。换句话说, 它评估了在特定状态 (s) 下按照策

略 (π) 行动的好坏程度。这个函数给出了从状态 (s) 开始, 按照策略 (π) 行动所能期望累积的总奖励的估计值。

2. 状态-动作价值函数 ($Q^{\pi}(s, a)$):。

- 解释: 状态-动作价值函数 ($Q^{\pi}(s, a)$) 表示在给定状态 (s) 下, 采取动作 (a) 并随后按照策略 (π) 行动所能获得的期望累积奖励。它评估了在状态 (s) 下采取特定动作 (a) 并随后按照策略 (π) 行动的好坏程度。实际上, 它提供了从状态 (s) 开始, 采取动作 (a), 然后按照策略 (π) 行动所能期望累积的总奖励的估计值。

• 区别

1. 定义域的不同:

- 状态价值函数 ($V^{\pi}(s)$):
 - 只与状态 (s) 相关。
 - 表示在状态 (s) 下按照策略 (π) 行动的期望累积奖励。
- 状态-动作价值函数 ($Q^{\pi}(s, a)$):
 - 与状态 (s) 和动作 (a) 相关。
 - 表示在状态 (s) 下采取动作 (a) 并随后按照策略 (π) 行动的期望累积奖励。

2. 用途和含义的不同:

- 状态价值函数 ($V^{\pi}(s)$):
 - 用于评估在某一特定状态下, 按照策略 (π) 行动的好坏程度。
 - 仅提供了处于状态 (s) 时的期望累积奖励, 不涉及具体的动作选择。
- 状态-动作价值函数 ($Q^{\pi}(s, a)$):
 - 用于评估在某一特定状态下采取某一特定动作后的好坏程度, 然后再按照策略 (π) 继续行动。
 - 详细提供了在状态 (s) 下采取动作 (a) 后的期望累积奖励, 因此可以帮助决定在特定状态下应该采取哪个动作。

总之, 状态价值函数 ($V^{\pi}(s)$) 是对状态的评估, 而状态-动作价值函数 ($Q^{\pi}(s, a)$) 是对状态和动作组合的评估。

贝尔曼方程

贝尔曼方程为价值函数提供了递归定义，反映了当前决策的长期价值：

状态价值函数的贝尔曼方程：

$$V^{\pi}(s) = \sum_{a \in A} \pi(a | s) \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V^{\pi}(s')]$$

状态-动作价值函数的贝尔曼方程：

$$Q^{\pi}(s, a) = \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma \sum_{a' \in A} \pi(a' | s') Q^{\pi}(s', a')]$$

$$V^{\pi}(s) = \mathbb{E}[R(s_0, \pi(s_0)) + \underbrace{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots}_{\text{future rewards}} | s_0 = s, \pi]$$

$$V^{\pi}(s) = \mathbb{E}[R(s_0, \pi(s_0)) + \gamma V^{\pi}(s_1) | s_0 = s, \pi]$$

$$V^{\pi}(s) = \sum_{a \in A} \pi(a | s) [\underset{\uparrow}{R(s, a)} + \gamma \sum_{s' \in S} \underset{\uparrow}{P_{sa}(s')} \underset{\uparrow}{V^{\pi}(s')}]$$

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

$$Q^{\pi}(s, a) = \sum_{s' \in S} P_{sa}(s') \sum_{a' \in A} \pi(a' | s') [R(s, a) + \gamma Q^{\pi}(s', a')]$$

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') \sum_{a' \in A} \pi(a' | s') Q^{\pi}(s', a')$$

最优策略

最优状态价值函数 $V^*(s)$ ：表示在状态 s 下按照最优策略行动的期望累积奖励。

最优状态-动作价值函数 $Q^*(s, a)$ ：表示在状态 s 下采取动作 a 并随后按照最优策略行动的期望累积奖励。

最优价值函数满足以下贝尔曼最优方程：

- **最优状态价值函数的贝尔曼方程:**

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V^*(s')]$$

- **最优状态-动作价值函数的贝尔曼方程:**

$$Q^*(s, a) = \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a')]$$

□ 贝尔曼期望方程

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) [R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')]$$

□ 贝尔曼最优方程

$$V^*(s) = \max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')]$$

□ 最优价值与最优策略

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

□ 状态价值的贝尔曼最优方程

$$V^*(s) = \max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')]$$

□ 状态动作价值的贝尔曼最优方程

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') \max_{a' \in A} Q^*(s', a')$$

□ 状态动作价值的贝尔曼期望方程

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s', a')$$

动态规划算法

基于动态规划的强化学习算法主要包括策略迭代（Policy Iteration）和价值迭代（Value Iteration）两种方法。这些算法在解决马尔可夫决策过程（MDP）的问题时非常有效，

特别是在状态和动作空间相对较小且已知转移概率的情况下

策略迭代（Policy Iteration）

策略迭代是一种迭代方法，用于找到最优策略 π^* 。它分为两个主要步骤：策略评估和策略改进

策略评估（Policy Evaluation）

在策略评估步骤中，我们给定一个策略 π ，并计算该策略下的状态价值函数 V 。策略评估通过以下迭代过程来进行，直到价值函数收敛

$$V^\pi(s) = \sum_{a \in A} \pi(a | s) \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

这个过程重复进行，直到价值函数的变化小于某个阈值 θ

策略改进（Policy Improvement）

在策略改进步骤中，我们使用当前的状态价值函数 V^π 来改进策略。改进策略的方法是选择在每个状态下能带来最高预期价值的动作

$$\pi'(s) = \arg \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

策略改进生成了一个新的策略 π' 。然后，我们用这个新策略重新进行策略评估。

□ 策略迭代过程

1. 随机初始化策略 π

2. 重复以下过程直到收敛{

- a) 当前策略的价值评估：迭代更新策略的价值函数至收敛
- b) 对每个状态，更新策略

$$\pi(s) = \arg \max_{a \in A} (R(s, a) + \gamma \sum_{s' \in S} P_{sa}(s') V(s'))$$

}

价值迭代（Value Iteration）

价值迭代直接利用贝尔曼最优方程来计算最优状态价值函数 V^* 。价值迭代的目标是通过迭代更新价值函数，逐步逼近最优价值函数。

价值迭代的步骤

1. **初始化**：选择一个初始状态价值函数 V ，通常初始化为零或其他任意值。
2. **更新价值函数**：使用贝尔曼最优方程更新每个状态的价值函数，直到价值函数收敛：

$$V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$$

3. **提取最优策略**：在价值函数收敛后，根据最优状态价值函数提取最优策略：

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$$

同步 vs. 异步价值迭代

在价值迭代算法中，状态的价值函数通过贝尔曼最优方程不断更新，直到价值函数收敛。根据更新的方式不同，价值迭代可以分为同步价值迭代和异步价值迭代

同步价值迭代（Synchronous Value Iteration）

在同步价值迭代中，所有状态的价值函数在每次迭代中同时更新。也就是说，在每一轮迭代中，新的价值函数是基于前一轮迭代中的所有状态的价值函数计算出来的。这种方式保证了所有状态的更新都是基于同一个时间步的价值函数

1. 初始化所有状态的价值函数 $V(s)$ 为零或其他任意值。
2. 对每个状态 s ，计算新的价值 $V_{new}(s)$ ：
$$V_{new}(s) = \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$$
3. 将所有状态的价值函数同时更新：
$$V(s) \leftarrow V_{new}(s)$$
4. 重复步骤2和3，直到价值函数收敛（即，所有状态的价值函数变化小于某个阈值）。

异步价值迭代（Asynchronous Value Iteration）

在异步价值迭代中，每次迭代只更新一个或部分状态的价值函数，而不是所有状态。这种方法允许更灵活和不规则的更新顺序，甚至可以在一些状态的更新尚未完成之前就开始下一步的更新。

1. 初始化所有状态的价值函数 $V(s)$ 为零或其他任意值。
2. 随机选择一个状态 s 或按照某种顺序选择状态，计算新的价值 $V_{new}(s)$:
$$V_{new}(s) = \max_{a \in A} \sum_{s' \in S} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$$
3. 更新该状态的价值函数:
$$V(s) \leftarrow V_{new}(s)$$
4. 重复步骤2和3，直到所有状态的价值函数收敛。

适合于并行计算和分布式系统中使用。对于某些问题，可能更快收敛，因为可以频繁地更新那些对整体价值函数影响大的状态。

价值迭代与策略迭代的比较

1. 算法复杂度：

策略迭代在每次策略评估时需要多次迭代计算状态价值函数，策略改进相对简单。价值迭代每次更新价值函数时都计算最优动作，因此每一步的计算量较大，但整体迭代次数较少。

2. 收敛速度：

策略迭代在每次策略评估和策略改进之间可能需要多次迭代，但每次策略改进后收敛速度快。价值迭代每次更新都接近最优值，但每次迭代的计算量较大。

3. 实现复杂度：

策略迭代的步骤分明，实现相对简单。价值迭代实现更为直接，但每一步计算较复杂

异步动态规划

异步动态规划是一类就地迭代的 DP 算法，这类算法使用任意可用的状态值，以任意顺序来更新状态值。在某些状态的值更新一次之前，某些状态值可能已经更新了好几次。但为了保证收敛，每个状态都必须被更新，不能忽略某些状态。这种就地迭代算法一般都是收敛的。

广义策略迭代

策略迭代包括两个同时进行的相互作用的流程：策略评估与策略改进，这两个过程交替进行。我们用**广义策略迭代**来表示策略评估与策略改进相互作用的思路。**几乎所有强化学习方法都可以被表述位广义策略迭代（GPI）**。几乎所有方法都包含明确定义的策略和价值函数，策略总是基于特定的价值函数进行改进，价值函数也始终会向对应特定策略的真实价值函数收敛。

动态规划的效率

动态规划方法对于一些大规模的问题并不是非常适用（维度灾难），但是和其他解决马尔可夫决策过程的方法相比，DP 的算法效率是相当高的。对于状态空间很大的问题，一般可以采用异步动态规划方法，问题中最优解可能只涉及状态空间中部分的状态。当前强化学习算法的大体思路本质上应该还是动态规划，只是进行了一些近似。

基于值函数的无模型强化学习

RL的目标是什么？ 最大化累积奖励的期望

在给定模型的时候我们怎么做？ 在模型的基础上按照贝尔曼期望方程不断迭代计算状态价值

面临的问题：在未给定模型或者模型规模太大的时候我们怎么办？

模型无关的强化学习中，可以通过与环境交互采集一系列可以用来估计值函数的经验

蒙特卡洛采样方法（MC）

蒙特卡洛（Monte Carlo）采样方法是一种统计学技术，用于通过随机抽样来估计数值结果或概率分布。在强化学习中，蒙特卡洛方法通过直接与环境交互，使用实际的样本数据来估计策略的价值函数。这与基于模型的方法（如动态规划）不同，蒙特卡洛依赖于已知的状态转移概率和奖励函数。

□ 目标：从策略 π 下的经验片段学习 V^π

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \dots s_T^{(i)} \sim \pi$$

□ 回顾：累计奖励 (**return**) 是总折扣奖励

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots \gamma^{T-1} R_T$$

□ 回顾：值函数 (**value function**) 是期望累计奖励

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi] \\ &= \mathbb{E}[G_t | s_t = s, \pi] \end{aligned}$$

$$\simeq \frac{1}{N} \sum_{i=1}^N G_t^{(i)}$$

- 使用策略 π 从状态 s 采样 N 个回合的轨迹数据
- 计算平均累计奖励

- 蒙特卡洛策略评估使用**经验累计奖励均值**而不是**期望累计奖励**

第一回报蒙特卡洛方法 (**First-Visit Monte Carlo**)

第一回报蒙特卡洛方法估计的是每个状态的平均回报，基于第一次访问该状态时的回报。

每回报蒙特卡洛方法 (**Every-Visit Monte Carlo**)

每回报蒙特卡洛方法估计的是每个状态的平均回报，基于每次访问该状态时的回报。

离线策略蒙特卡洛方法 (**Offline Policy Monte Carlo with Importance Sampling**)

使用重要性采样的离线策略蒙特卡洛方法 (**Offline Policy Monte Carlo with Importance Sampling**) 是一种估计目标策略 (**target policy**) 价值函数的技术。它可以利用不同于目标策略的行为策略 (**behavior policy**) 生成的数据，进行策略评估和优化。

目标策略 (**Target Policy**) 和行为策略 (**Behavior Policy**)

目标策略 π ：我们希望评估或优化的策略。行为策略 b ：用于生成经验数据的策略，可能与目标策略不同。

重要性采样 (**Importance Sampling**)

重要性采样是一种统计方法，用于通过调整样本的权重，将来自一个分布的样本用于估计另一个分布的期望值。在强化学习中，它用于修正行为策略和目标策略之间的差异。

1. 生成经验片段：使用行为策略**b** 生成若干个从起始状态到终止状态的完整序列（episode）
2. 计算权重：对于每个经验片段，根据行为策略和目标策略的概率计算权重。权重反映了行为策略生成该片的概率与目标策略生成该片的概率之比
3. 计算回报：对于每个经验片段，计算从每个状态开始的累计回报 G_t
4. 更新价值函数：使用加权回报更新目标策略下的状态价值函数 $V^\pi(s)$

□ 更新值函数以逼近修正的累计奖励值

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(G_t^\pi - V^\pi(s_t)) \quad \leftarrow \quad s_0^{(i)} \xrightarrow[r_0^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[r_1^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[r_2^{(i)}]{a_2^{(i)}} s_3^{(i)} \dots s_T^{(i)} \sim \pi$$

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(\underline{G_t^{\pi/\mu}} - V^\pi(s_t)) \quad \leftarrow \quad s_0^{(i)} \xrightarrow[r_0^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[r_1^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[r_2^{(i)}]{a_2^{(i)}} s_3^{(i)} \dots s_T^{(i)} \sim \mu$$

$$G_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \dots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} G_t$$

无法在 π 非零而 μ 为零时使用

重要性采样将显著增大方差 (variance)

时序差分方法 (TD)

时序差分（Temporal-Difference, TD）方法是一种在强化学习中用于估计价值函数的关键技术。它结合了蒙特卡洛方法和动态规划的优点，通过从经验数据中逐步更新价值估计，从而避免了动态规划所需的完整模型和蒙特卡洛方法对长期回报的依赖。

核心思想：时序差分方法的核心思想是利用当前的估计来改进未来的估计。它通过“引导误差”来更新价值函数，这个误差称为TD误差（TD Error）。

TD误差是当前估计和新的估计之间的差异。对于状态价值函数 $V(s)$ ，TD误差的定义为：

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

其中：

- δ_t 是在时间步 t 的TD误差。
- R_{t+1} 是从状态 S_t 到 S_{t+1} 所获得的即时奖励。
- γ 是折扣因子，决定了未来奖励的重要性。
- $V(S_t)$ 是当前状态 S_t 的价值估计。
- $V(S_{t+1})$ 是下一个状态 (S_{t+1}) 的价值估计。

时序差分更新规则

使用TD误差，可以更新当前的价值估计。对于状态价值函数 $V(s)$ ，更新规则为：

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

其中：

- α 是学习率，控制每次更新的步长。

对于状态-动作价值函数 $Q(s, a)$ ，TD误差和更新规则类似：

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \delta_t$$

SARSA (State-Action-Reward-State-Action)

SARSA是一个基于状态-动作对的时序差分控制算法。它的名字来源于五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ ，用于更新Q值。

更新规则：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-learning

Q-learning是另一种时序差分控制算法，但它是基于最优动作的更新，因此是一个离策略算法。

更新规则：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

基于值函数的深度强化学习

之前的做法都是基于创建一个查询表，在表中维护状态值函数 $V(s)$ 或状态-动作值函数 $Q(s, a)$ ，这种方法有什么缺点呢？

面对大规模马尔可夫决策过程（MDP）时难以处理，即：

状态或者状态-动作空间非常大 / 连续的状态或动作空间

解决思路：

1. 对状态/动作进行离散化或分桶
2. 构建参数化的值函数近似估计

对状态/动作进行离散化或分桶

- 例如，如果用2维连续值 (s_1, s_2) 表示状态，可以使用网格对状态空间进行切分从而转化为离散的状态值

- 记离散的状态值为 \bar{s}

S

- 离散化的马尔可夫决策过程可以表示为：

$$(\bar{S}, A, \{P_{\bar{s}a}\}, \gamma, R)$$

- 这样一来，就能够使用前述方法求解马尔可夫决策过程

基本概念

状态空间离散化：将连续的状态空间划分为若干个离散的状态区域，每个区域代表一个离散状态。动作空间离散化：将连续的动作空间划分为若干个离散的动作，每个动作代表一个连续动作区域内的一个离散动作。

离散化方法

1. 状态空间离散化 常见的状态空间离散化方法包括：

- 等间距离散化（Uniform Discretization）：将每个维度的连续区间划分为等长度的小区间，每个小区间代表一个离散状态。
- 聚类离散化（Clustering Discretization）：使用聚类算法（如k-means）将连续状态空间中的样本点分成若干个簇，每个簇的中心代表一个离散状态。
- 自适应离散化（Adaptive Discretization）：根据经验数据动态调整划分策略，以更精细地离散化高访问频率的区域。

2. 动作空间离散化 动作空间离散化的方法与状态空间类似：

- 等间距离散化：将动作空间划分为等长度的区间，每个区间代表一个离散动作。

- 自适应离散化： 动态调整离散化策略，针对特定区域进行更细致的划分。

对大型马尔可夫决策过程分桶

□ 对于一个大型的离散状态马尔可夫决策过程，我们可以对状态值进一步分桶以进行采样聚合

- 使用先验知识将相似的离散状态归类到一起
 - 例如，利用根据先验知识抽取出来的状态特征对状态进行聚类

构建参数化的值函数近似估计

1. 常见的模型 线性模型 多项式模型 神经网络
2. 定义值函数形式 值函数可以是状态值函数 $V(s)$ 或状态-动作值函数 $Q(s,a)$ 。
3. 参数化值函数 假设使用参数 θ 表示模型参数，值函数可以表示为：

对于状态值函数： $V(s;\theta)$ 对于状态-动作值函数： $Q(s,a;\theta)$ 4. 初始化参数 参数 θ 通常初始化为小的随机值，或者根据某些启发式方法初始化。

5. 选择损失函数 常用的损失函数包括：

均方误差（MSE）： 用于监督学习中的回归任务

时间差分误差（TD error）： 用于时序差分学习

6. 优化方法 选择优化方法来更新参数 θ ：

梯度下降，随机梯度下降（SGD），动量方法，Adam优化器等。

蒙特卡洛状态值函数近似

$$\theta_{k+1} \leftarrow \theta_k + \alpha(V^\pi(s) - V_\theta(s))x_k(s)$$

□ 我们用 $V^\pi(s)$ 表示真实的目标价值函数，真实值我们知道吗？

□ 第一种方法：利用蒙特卡洛采样估计

- 在“训练数据”上运用监督学习对价值函数进行预测

$$\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \dots, \langle s_T, G_T \rangle$$

- 对于每个数据样本 $\langle s_t, G_t \rangle$

$$\theta_k \leftarrow \theta_k + \alpha(G_t - V_\theta(s))x_k(s_t)$$

□ 蒙特卡洛预测至少能收敛到一个局部最优解

- 在价值函数为线性的情况下可以收敛到全局最优

时序差分状态值函数近似

$$\theta_k \leftarrow \theta_k + \alpha(V^\pi(s_t) - V_\theta(s_t))x_k(s_t)$$



□ 第二种方法：利用时序差分估计

$$r_{t+1} + \gamma V^\pi(s_{t+1})$$

- 时序差分算法的目标 $r_{t+1} + \gamma V_\theta(s_{t+1})$ 是真实目标价值 $V_\pi(s_t)$ 的有偏采样

- 在“训练数据”上运用监督学习

$$\langle s_1, r_2 + \gamma V_\theta(s_2) \rangle, \langle s_2, r_3 + \gamma V_\theta(s_3) \rangle, \dots, \langle s_T, r_T \rangle$$

- 对于每个数据样本 $\langle s_t, r_{t+1} + \gamma V_\theta(s_{t+1}) \rangle$

$$\theta_k \leftarrow \theta_k + \alpha(\underbrace{r_{t+1} + \gamma V_\theta(s_{t+1})}_{\text{Target}} - V_\theta(s_t))x_k(s_t)$$


Target

□ 线性情况下时序差分学习（接近）收敛到全局最优解

深度Q网络（DQN, Deep Q-Network）

深度Q网络（DQN，Deep Q-Network）是结合深度学习和Q学习的一种强化学习算法。
DQN利用神经网络来逼近Q值函数

DQN算法流程

- 
1. 收集数据：使用 ϵ -greedy 策略进行探索，将得到的状态动作组 (s_t, a_t, s_{t+1}, r_t) 放入经验池D (replay-buffer)
 2. 采样：从经验池D中均匀随机采样 k 个动作状态组
 3. 更新网络
 - 用采样得到的数据计算 $Loss$ 。
 - 更新 Q 函数网络 θ 。
 - 每 C 次迭代（更新 Q 函数网络 C 次）更新一次目标网络 θ^- 。


DQN的扩展算法

Q-learning中Q函数的过估计问题

$$L_i(\theta_i) = \mathbb{E}_{s_t, a_t, s_{t+1}, r_t, p_t \sim D} \left[\frac{1}{2} \underbrace{\left((r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1}, a')) - Q_{\theta_i}(s_t, a_t) \right)^2}_{target} \right]$$

□ Q函数的过高估计

- Target值 $y_t = r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1}, a')$



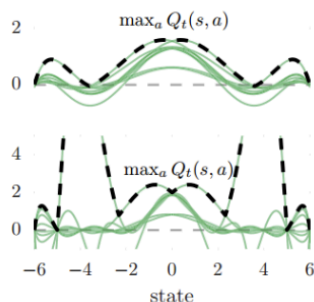
\max 操作使得 当 Q 函数的近似值偏大时
Target值也相应的偏大，甚至远高于真实值

Double DQN

□ 使用不同的网络来估值和决策

$$\text{DQN} \quad y_t = r_t + \gamma Q_{\theta}(s_{t+1}, \arg \max_{a'} Q_{\theta}(s_{t+1}, a'))$$

$$\text{Double DQN} \quad y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg \max_{a'} Q_{\theta}(s_{t+1}, a'))$$



假设上面对应 $Q_{\theta'}$ ，下面对应 Q_{θ} ，这样哪怕使用下面的选择动作，用上面取值，也可以避免Q值的过估计

核心思想

- 使用两个Q网络，一个用来选动作a，另一个用来计算最终的Q值
- 因为两个Q网络不同，在同一个位置上两个网络同时过估计发生的概率得到抑制

优先经验回放（Prioritized Experience Replay）

问题动机：当经验池中存在大量无效经验的时候，均匀采样好吗？

□ 衡量标准

- 以 Q 函数的值与 Target 值的差异来衡量学习的价值，即

$$p_t = |r_t + \gamma \max_{a'} Q_{\theta}(s_{t+1}, a') - Q_{\theta}(s_t, a_t)|$$

- 为了使各样本都有机会被采样，存储 $e_t = (s_t, a_t, s_{t+1}, r_t, p_t + \epsilon)$ 。

□ 选中的概率

- 样本 e_t 被选中的概率为 $P(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}$ 。

□ 重要性采样（Importance Sampling）：使用优先级改变了随机采样的概率分布，因此需要对所得的参数更新量按如下权重比例做出相应调整

$$\text{权重为} \quad w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta \quad \Delta \leftarrow \Delta + \underbrace{w_j}_{\text{重要性采样权重}} \cdot \delta_j \cdot \nabla_{\theta} Q(S_{j-1}, A_{j-1})$$

重要性采样权重

策略梯度算法

通过直接优化策略（即行为策略）来最大化累积回报。不同于基于价值函数的方法（如 Q-learning），策略梯度方法不需要构建值函数，而是直接优化策略，使得策略本身逐渐改进。策略梯度方法在处理高维连续动作空间和策略优化方面具有显著优势。

REINFORC

它通过直接优化策略参数来最大化累积回报。

核心思想

REINFORCE算法的核心思想是通过蒙特卡罗采样方法来估计策略梯度，并使用梯度上升法（或下降法）来更新策略参数，从而优化策略。

主要步骤

1. 策略参数化:

- 策略 $\pi_{\theta}(a|s)$ 参数化为 θ ，表示在状态 s 下采取动作 a 的概率。

2. 生成轨迹:

- 从当前策略 π_{θ} 生成一条完整的轨迹 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$ ，直到达到终止状态。

3. 计算回报:

- 对于每个时间步 t ，计算从 t 时刻开始的累积回报 R_t :

$$R_t = \sum_{k=t}^T \gamma^{k-t} r_k$$

- 其中 γ 是折扣因子。

4. 更新策略参数:

- 通过梯度上升法更新策略参数:

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

- 其中 α 是学习率。

• 缺点:

- 高方差：由于每个回报都是基于整个轨迹的累积回报，导致梯度估计的方差较高。

- 样本效率低：需要大量样本来获得良好的策略。
- 每个更新步骤需要完整的一条轨迹，计算开销较大。

为更复杂的策略梯度算法（如Actor-Critic方法和PPO）奠定了基础。

Actor-Critic

Actor-Critic是一类结合了策略梯度（Actor）和价值函数（Critic）的强化学习算法。它们通过将策略和价值函数的优点结合起来，旨在提高训练的效率和稳定性。

核心思想

Actor-Critic算法同时学习两个函数：

1. 策略（Actor）：直接输出动作或动作分布，参数化为 $\pi_{\theta}(a|s)$ 。
2. 价值函数（Critic）：估计当前策略下的状态值函数 $V_{\phi}(s)$ 或动作值函数 $Q_{\phi}(s, a)$ ，参数化为 ϕ 。

Critic的作用是减少策略梯度的方差，从而使策略更新更稳定和有效。

主要步骤

1. 初始化参数：
 - 初始化策略参数 θ 和价值函数参数 ϕ 。
2. 生成轨迹：
 - 从当前策略 π_{θ} 生成一条轨迹 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$ 。
3. 计算优势函数：
 - 优势函数 A_t 用于评价在状态 s_t 采取动作 a_t 相对于其他动作的优越程度。
 - 常见的计算方式为： $A_t = R_t - V_{\phi}(s_t)$
 - 其中 R_t 是从 t 时刻开始的累积回报， $V_{\phi}(s_t)$ 是Critic估计的状态值。
4. 更新策略（Actor）参数：

使用策略梯度更新策略参数：

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A_t$$

- α 是学习率。

5. 更新价值函数（**Critic**）参数：

使用TD误差更新价值函数参数：

$$\delta_t = r_t + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t)$$

通过梯度下降法最小化均方误差：

$$\phi \leftarrow \phi - \beta \nabla_{\phi} (\delta_t)^2$$

- β 是学习率。
- 缺点：
 - 价值函数的估计可能会引入偏差。
 - 需要同时训练两个函数（策略和价值函数），增加了模型复杂性。

信赖域（Trust Region）优化方法是一种在每次更新参数时将其限制在一个相对可信赖的区域内的策略，从而保证优化过程中的稳定性和可靠性。在强化学习中的策略梯度优化中，信赖域优化方法可以防止策略更新过大而导致策略性能不稳定。

Trust Region

信赖域优化方法的核心思想是在每次策略更新时，将更新步长限制在一个信赖域内，从而控制更新的幅度。这样可以避免因更新步长过大而导致的策略崩溃或不稳定。

Trust Region Policy Optimization (TRPO)

- TRPO是一种信赖域方法，它通过约束策略更新步长来保证策略优化的稳定性。TRPO通过解决以下优化问题来实现策略更新：

$$\max_{\theta} \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q^{\pi_{\theta_{\text{old}}}}(s, a) \right]$$

subject to

$$\mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}} [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s) \parallel \pi_{\theta}(\cdot|s)]] \leq \delta$$

- 其中：
 - $\pi_{\theta_{\text{old}}}$ 是旧策略。
 - π_{θ} 是新策略。
 - $Q^{\pi_{\theta_{\text{old}}}}(s, a)$ 是旧策略的动作价值函数。
 - δ 是预设的KL散度限制，控制策略更新步长。

算法步骤

1. 受策略 $\pi(\cdot | s; \theta_{\text{old}})$ 控制，智能体玩游戏并观察到一个轨迹：
 - 智能体根据当前的策略 $\pi(\cdot | s; \theta_{\text{old}})$ 进行游戏，生成一个包含状态、动作和奖励的序列。
2. 对于 $i = 1, \dots, n$ ，计算折扣回报：
 - 对于每个时间步 i ，计算从 i 时刻开始的折扣累积回报 u_i 。
3. 近似：
 - 计算近似的目标函数 $\tilde{L}(\theta | \theta_{\text{old}})$ ，这是使用重要性采样来估计新策略与旧策略之间的回报。
4. 最大化：
 - 找到使得近似目标函数 $\tilde{L}(\theta | \theta_{\text{old}})$ 最大化的策略参数 θ_{new} ，同时约束新旧策略参数之间的距离在 Δ 以内，以确保策略更新的步长在信赖域内。

Proximal Policy Optimization (PPO)

- PPO是一种改进的信赖域方法，它简化了TRPO的实现，同时保留了稳定性。PPO通过剪辑的方式来限制策略更新

PPO的核心思想

PPO 的核心思想是通过限制每次策略更新的幅度来保证策略的稳定性，具体实现时采用了“剪辑”机制。PPO 主要有两种形式：剪辑版本（Clipped Surrogate Objective）和自适应KL惩罚版本（Adaptive KL Penalty）。其中，剪辑版本更为常用。

剪辑版本的PPO

剪辑版本的PPO 通过以下的目标函数来限制策略的更新：

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

其中：

- $r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ 是当前策略与旧策略的概率比。
- \hat{A}_t 是优势函数的估计值，用于衡量在特定状态下采取特定动作的相对好坏。
- ϵ 是一个很小的常数，通常在 0.1 到 0.3 之间。

目标函数的第一项 $r_t(\theta) \hat{A}_t$ 代表了普通的策略梯度目标，而第二项 $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t$ 则通过剪辑概率比来限制策略的更新幅度，确保更新不会偏离过远。

自适应KL惩罚版本的PPO

自适应KL惩罚版本的PPO 则通过以下的目标函数来限制策略的更新：

$$L^{\text{KL}}(\theta) = \mathbb{E}_t \left[r_t(\theta) \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t) || \pi_{\theta}(\cdot|s_t)] \right]$$

其中， β 是一个权重系数，用于控制KL散度项的影响。该方法通过在目标函数中添加一个KL散度项来限制新旧策略之间的变化幅度。

实现步骤

1. 采样: 使用当前策略 (π_{θ}) 采样一批轨迹数据。
2. 计算优势函数: 使用GAE（Generalized Advantage Estimation）等方法计算优势函数 (\hat{A}_t)。
3. 更新策略: 使用上述的目标函数（剪辑版本或自适应KL惩罚版本）优化策略参数 (θ)。

4. 迭代: 重复上述步骤，直到策略收敛或达到预设的训练步数。

规划与学习

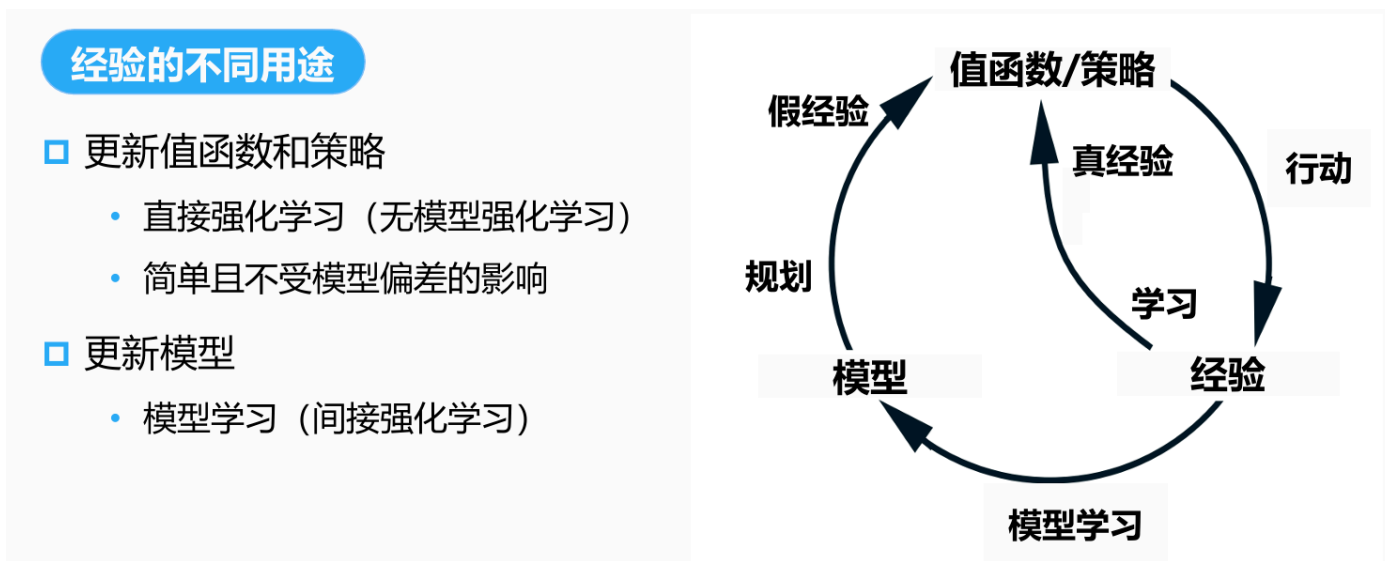
结合学习模型（Model-Based Learning）

结合学习模型方法既可以借助已知的环境模型，也可以通过与环境的交互来学习模型。

Dyna架构：

- 核心思想：结合模型学习和模型无关学习。既使用实际交互数据更新策略，也使用环境模型进行模拟以加速学习。
- 工作流程：
 1. 与环境交互，收集数据 $((s, a, r, s'))$ 。
 2. 更新环境模型 $(P(s'|s,a))$ 和 $(R(s,a))$ 。
 3. 基于实际数据进行Q学习更新。
 4. 使用模型生成虚拟样本，进行模拟更新。
- 优点：能够更高效地利用数据，提高学习速度。

Dyna-Q算法详细介绍



Dyna-Q是结合模型学习和直接强化学习的一种典型方法。以下是Dyna-Q算法的具体步骤：

1. 初始化：

- 初始化Q值函数 ($Q(s, a)$)。
- 初始化模型 ($P(s'|s, a)$) 和 ($R(s, a)$)。

2. 循环直到收敛:

- 实际交互:

1. 在状态 s 下选择动作 a 。
2. 执行动作 a , 观察奖励 r 和下一个状态 s' 。
3. 使用实际交互数据更新Q值:

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

4. 更新模型 $P(s'|s, a)$ 和 $R(s, a)$ 。

- 模拟更新:

1. 从模型中随机抽取一组虚拟样本 (s, a, r, s') 。
2. 使用虚拟样本更新Q值:

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

3. 策略更新:

- 基于最新的Q值函数选择最优策略

决策时规划 (Planning at Decision Time)

在每次决策时动态进行规划。它不同于预先计算所有策略的规划方法，而是在需要做出决策时才进行规划。

决策时规划的关键方法

1. 蒙特卡罗树搜索 (Monte Carlo Tree Search, MCTS) :

- 核心思想: 通过模拟未来的可能路径, 评估当前决策的价值, 并不断调整搜索树中的策略以找到最优路径。

。步骤：

1. 选择（**Selection**）：从根节点开始，选择一个子节点，直到到达一个叶节点。选择策略通常是基于某种探索与利用权衡（如UCB1算法）。
2. 扩展（**Expansion**）：如果叶节点不是终止节点，扩展该节点，添加一个或多个子节点。
3. 模拟（**Simulation**）：从新扩展的节点开始，模拟游戏到结束，得到一个结果。
4. 回溯（**Backpropagation**）：将模拟结果回溯到所有经过的节点，更新这些节点的统计信息（如胜率）。

2. Model Predictive Control (MPC):

- 。核心思想：在每个决策时刻，通过模型预测未来若干步的结果，并选择最优动作。
- 。步骤：
 1. 模型预测：使用当前的环境模型，预测未来若干步的状态转移和奖励。
 2. 优化：通过滚动优化找到使得累积奖励最大的动作序列。
 3. 执行：执行第一个动作，然后重复上述过程。

3. Rollout算法:

- 。核心思想：在每个决策时刻，从当前状态开始模拟多条可能的轨迹，评估每条轨迹的累积奖励，选择最优的动作。
- 。步骤：
 1. 模拟轨迹：从当前状态出发，模拟多个动作序列，得到多条轨迹。
 2. 评估轨迹：计算每条轨迹的累积奖励。
 3. 选择动作：选择具有最高累积奖励的轨迹的初始动作。

决策时规划的优点和缺点

优点:

- 实时优化：能够根据当前的最新信息进行决策优化，适应动态环境变化。
- 灵活性：可以在每个决策点考虑多种可能的未来路径，从而做出更加灵活和适应性的决策。
- 有效性：在一些复杂和高维度的环境中，决策时规划往往能够找到更优的策略。

缺点:

- **计算成本高：**每次决策都需要进行大量的模拟和计算，可能导致计算开销较大，尤其是在实时应用中。
- **依赖模型准确性：**MPC等方法依赖于环境模型的准确性，如果模型不准确，可能导致次优甚至错误的决策。
- **复杂性：**实现和调优这些算法可能较为复杂，需要精细的参数调整和大量的计算资源。