教育部-华为智能基座课程

《人工智能基础与实践》

第9章：强化学习I

授课教师：丛润民

山东大学

控制科学与工程学院

章节目录 CONTENTS

章节目录 CONTENTS

强化学习基本概念

➢ 引言

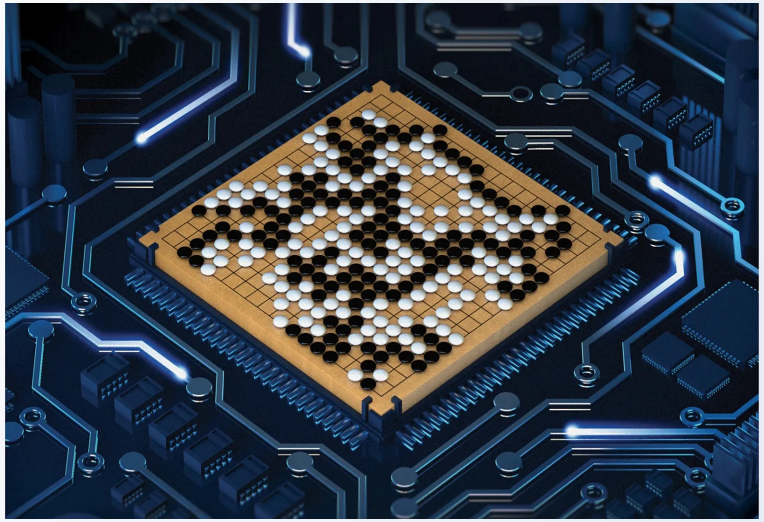
- 过去十年，人工智能领域取得了一系列突破，其中许多都与一个名为“**强化学习**”的技术紧密相关。比如著名的**AlphaGo**，它通过不断地自我对弈学习，掌握了人类围棋大师复杂技艺，并最终击败了世界冠军。又比如现在炙手可热的“**人形机器人**”，这些机器人能够完成流畅的后空翻和跑酷动作，其背后核心的平衡与动作规划算法，也深深依赖于强化学习。

- 强化学习，简而言之，它的目标是训练一个智能体，使其能够通过与环境的自主交互，学会独立完成某个特定任务。这里的“**智能体**”可以是一个程序、一个机器人，或任何一个决策实体；“**环境**”则是它所在的世界，比如围棋棋盘、游戏屏幕，或是物理空间。这个过程智能体自主尝试，我们根据其行为结果给予**反馈**，让它自己形成一套高效的决策策略。

强化学习基本概念

AlphaGo

- **训练对象**：下围棋的程序
- **环境**：虚拟围棋棋盘，遵循标准的围棋规则
- **任务**：学习如何选择每一步落子，最终为了赢棋，当落子优秀或赢棋时就会获得奖励。



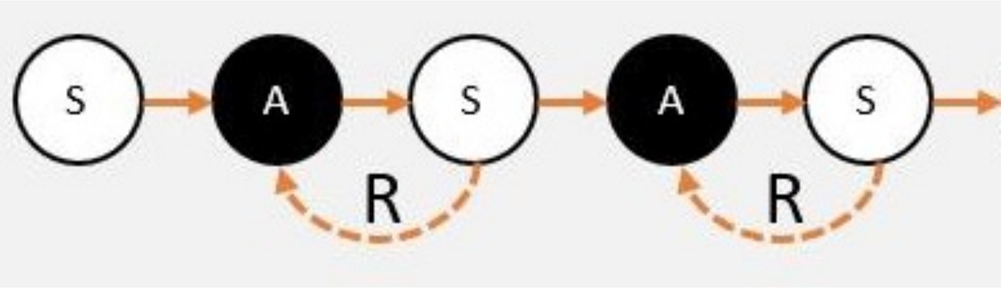
UniTree-H1

- **训练对象**：机器人运动控制算法
- **环境**：真实连续的物理世界，存在重力、摩擦等复杂物理条件。
- **任务**：协调身体各个关节和电机，完成各种指定动作，奖励在这里设计得会更加复杂。

强化学习基本概念

➢ 马尔可夫链

- 强化学习的模型可以抽象为一个马尔可夫链。其由三个重要的元素组成：**State状态**、**Action动作**、**Reward奖励**。



- **S（State）状态**，就是智能体观察到的当前环境的**部分**或者**全部特征**。

如无人驾驶汽车在十字路口，需要先“观察”这个路口的特征（如交通标志、交通信号灯、行人情况），再决定一下步的“动作”。

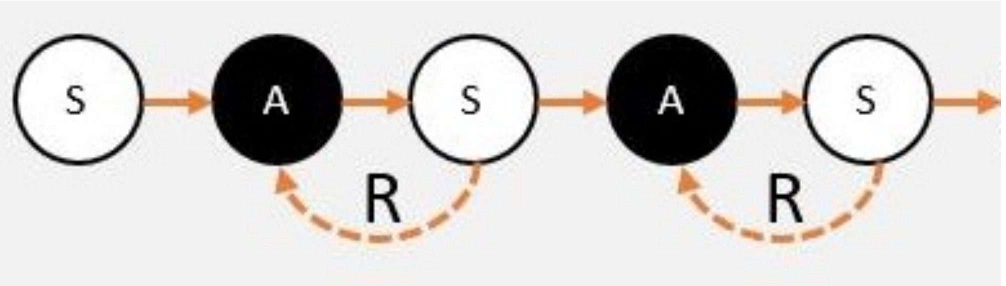
注意：环境的特征可能有许多，但只有智能体能够观察到的特征才算是状态。

因此通常用**Observation** 表示状态。

SHANDONG UNIVERSITY

➢ 马尔可夫链

- **A（action）动作**，就是智能体做出的行为。动作空间就是该智能体能够做出的动作数量。例如在围棋里，剩余可下的格数就是动作空间。

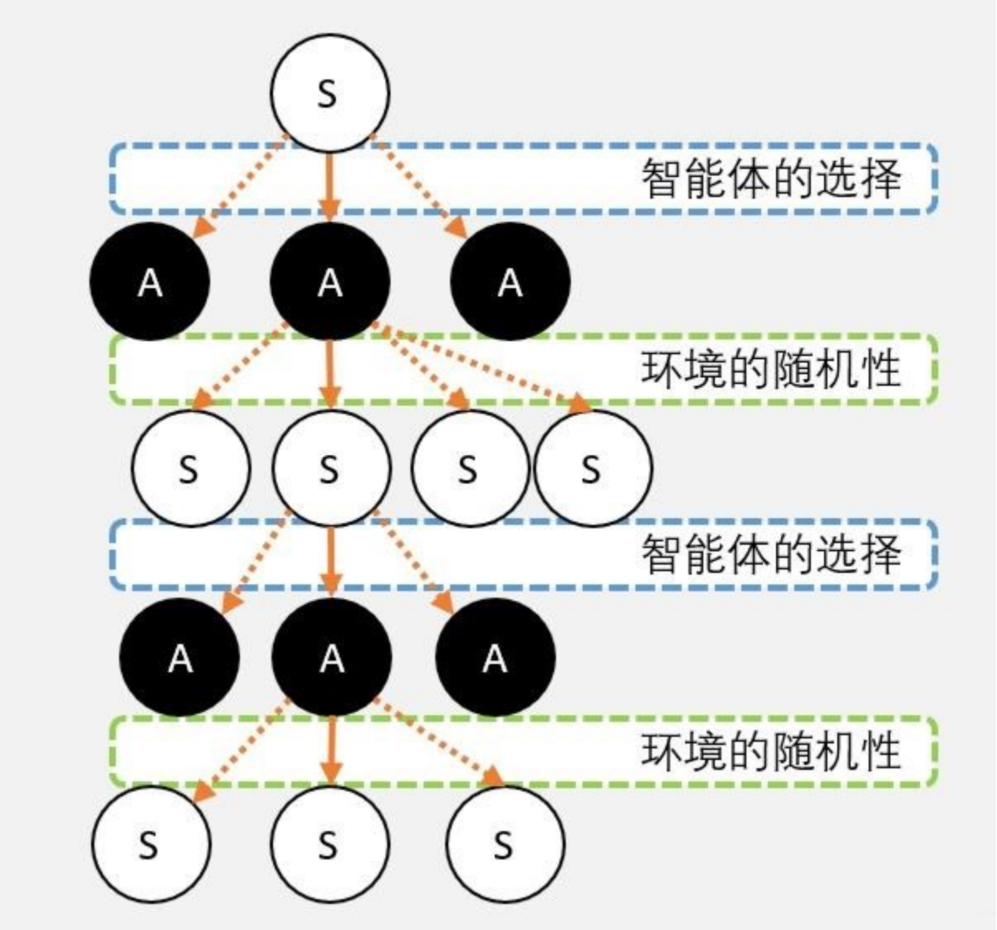


- **奖励的设定是主观的**，也就是说我们为了智能体更好地学习工作，自己定的。所以大家可以看到，很多时候我们会对奖励进行一定的修正，这也是加速智能体学习的方法之一。

➢ 马尔可夫链

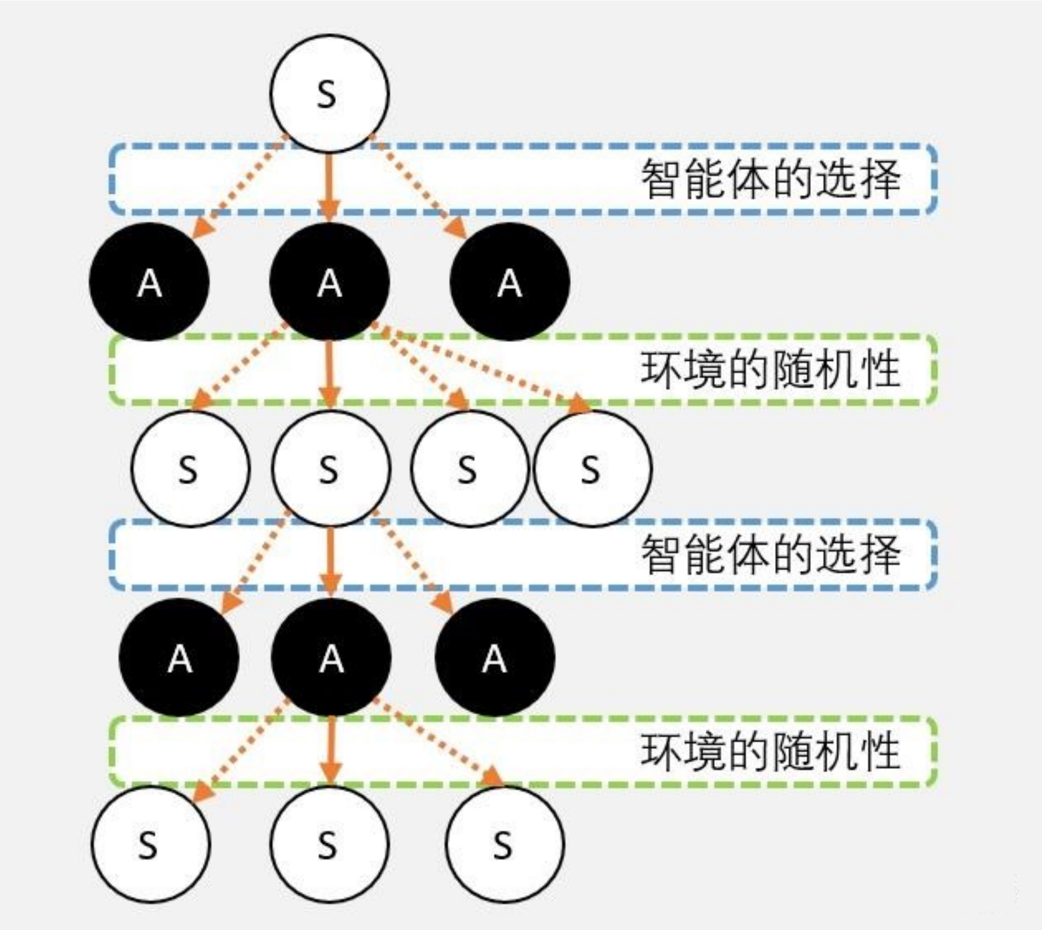
- 总结一下：
  - ✓ 智能体在环境中，观察到状态(S)；
  - ✓ 状态(S)输入到智能体，智能体经计算，选择动作(A);
  - ✓ 动作(A)使智能体进入另外一个状态(S)，并返回奖励(R)给智能体；
  - ✓ 重复以上步骤，一步一步创造马尔可夫链。
- 然而问题出现在：
  - ✓ 在某一状态下智能体有可能选择多种行为的其中一种；
  - ✓ 由于环境的随机性，哪怕是选择同一种行为也可能会引向不同的状态作为结果。

因此，马尔可夫链实际上更像是一个**树状结构**



➢ 马尔可夫链

- 进一步地，当智能体处在某一状态下，有可能会选择多种行为，这一选择也会影响到下一个状态。这种不同动作之间的选择，我们称为**智能体的策略**。策略我们一般用 π 表示。
- 具体地说，策略就是在各个状态下智能体选择各个行为的概率分布。
- 因此，强化学习的任务就是找到一个策略，能够获得最多的奖励。



章节目录 CONTENTS

贝尔曼公式

状态价值与动作价值

- 当智能体从一个状态S，选择动作A，会进入另外一个状态S'；同时，也会给智能体奖励R。 奖励既有正，也有负。正代表我们鼓励智能体在这个状态下继续这么做；负得话代表我们并不希望智能体这么做。 在强化学习中，我们会用奖励R作为智能体学习的引导，期望智能体获得尽可能多的奖励。
- 但更多的时候，让我们考虑这样一种可能：当智能体选择动作A1时虽然只得到了比动作A2更少的奖励，但是却让智能体更加接近最终的成功状态。换句话说，虽然牺牲了短期利益但是从长远来看却有了收获更多奖励的可能性。

因此，我们并不能单纯通过R来衡量一个状态或者动作的好坏

贝尔曼公式

状态价值与动作价值

- State Value状态价值，我们称为V值，用于评价一个状态S的好坏。

  V值代表了智能体在这个状态下，一直到最终状态的**奖励总和的期望**：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$

$$v_\pi(s) = \mathrm{E}\left[G_t \middle| S_t = s\right]$$

$\gamma \in [0,1]$ 是折扣率，能够对未来回报打一个折扣，可以控制算法在短期利益和长期利益中平衡

G_t 指智能体在策略π下得到的总回报

$v_\pi(s)$ 则是状态s在策略π下的V值，是状态s的函数

需要注意的是，在背景一定时，**V值只与策略有关**，也只有在给定策略时才能求得V值

贝尔曼公式

状态价值与动作价值

- Action Value动作价值，我们称为Q值，用于评价一个动作的好坏。与V值类似，Q值代表了智能体选择这个动作后，一直到最终状态奖励总和的期望：

$$q_{\pi}(s,a)=\mathrm{E}\left[G_t \middle| S_t=s, A_t=a\right]$$

与V值类似，$q_{\pi}(s,a)$是一个关于状态s和动作a的函数

Q值同样取决于策略π

Q值和V值意义相通：

1. 都是马尔可夫链上的节点；2. 价值评价的方式是一样的：从当前节点出发，一直走到最终节点，求所有的奖励的期望值。因此其实**Q值和V值是可以相互转换**的。

SHANDONG UNIVERSITY

状态价值与动作价值

- 转换形式：

一个状态的V值，就是这个状态下的所有动作的Q值，在策略π下的期望

一个动作的Q值，就是这个动作下得到的所有可能的V值的期望，再加上动作的奖励的期望

$$v_\pi(s)=\sum_a \pi(a\mid s)q_\pi(s,a)$$

$$q_\pi(s,a)=\sum_r p(r\mid s,a)r+\gamma\sum_{s'} p(s'\mid s,a)v_\pi(s')$$

其中$\pi(a\mid s),p(s'\mid s,a),p(r\mid s,a)$ 分别代表了策略π下状态s采取动作a的概率、状态s下采取动作a转移到状态s’ 的概率、状态s下采取动作a得到奖励r的概率。

贝尔曼公式

状态价值与动作价值

- 进一步的，根据上述的两个公式，我们发现两个相邻的V值也可以互相转化：

$$v_\pi(s)=\sum_{a\in A}\pi(a|s)\left(\sum_r p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_\pi(s')\right)$$

- 这就是著名的**贝尔曼公式**，它揭示了当我们给定一个策略π和环境模型（就是该公式中的两个概率分布）时，我们能够直接通过求解该公式得到每个状态的V值。对应的，V值得到后，Q值也能计算得来。
- 在求得了V值和Q值之后，我们可以认为，当策略π对应的V值在所有策略中最大时，该策略是最优的策略。强化学习的目标正是去求得这一**最优策略。**
- 然而具体我们应该如何求解贝尔曼公式，又如何利用求得的V、Q值来优化我们的策略呢？

贝尔曼公式

贝尔曼公式的求解

- 首先我们需要对刚刚的贝尔曼公式进一步展开：

$$
\begin{aligned}
v_\pi(s) &= \sum_a \pi(a|s)\left(\sum_r p(r|s,a)r + \gamma\sum_{s'} p(s'|s,a)v_\pi(s')\right) \\
&= \sum_a \pi(a|s)\sum_r p(r|s,a)r + \gamma\sum_a \pi(a|s)\sum_{s'} p(s'|s,a)v_\pi(s') \\
&= r_\pi(s) + \gamma\sum_{s'} p_\pi(s'|s)v_\pi(s')
\end{aligned}
$$

- 其中

$$r_\pi(s) = \sum_a \pi(a|s)\sum_r p(r|s,a)r$$

$$p_\pi(s'|s) = \sum_a \pi(a|s)p(s'|s,a)$$

贝尔曼公式

贝尔曼公式的求解

- 进一步我们将公式转化为矩阵形式：

$$v_\pi = r_\pi + \gamma P_\pi v_\pi$$

其中：

$$v_\pi = \left[v_\pi(s_1), \quad , v_\pi(s_n) \right]^T \in \quad {}^{n}$$

$$r_\pi = \left[r_\pi(s_1), \quad , r_\pi(s_n) \right]^T \in \quad {}^{n}$$

$$P_\pi \in \quad {}^{n\times n}, [P_\pi]_{ij} = p_\pi(s_j \mid s_i)$$

这个公式就是贝尔曼公式的矩阵形式，也是最常见最常用的形式。

我们可以很直接的通过求解这个矩阵方程得到V值。然而在实际的场景里，求一个如此庞大的矩阵的逆是非常困难的，因此我们会考虑其他方法。

贝尔曼公式的求解

使用一个迭代式的算法计算V值：

$$v_{k+1} = r_\pi + \gamma P_\pi v_k$$

通过迭代这个公式，得到一个序列 $\{v_0, v_1, v_2, \ldots\}$，可以证明：

$$v_k \to v_\pi = \left(I - \gamma P_\pi\right)^{-1} r_\pi, \quad k \to \infty$$

通过初始化一个v再通过上述公式不断迭代就可以无限逼近真实的V值。通常在迭代一定次数后就会停止，一般会设置一个阈值，当某次迭代后的差值小于该阈值时就会停止迭代。

贝尔曼最优公式

什么是最优策略？我们需要知道如何比较两个策略的好坏，这里直接给出定义：

当$v_{\pi_1}(s) \geq v_{\pi_2}(s), s \in \mathbf{S}$成立，我们认为策略$\pi_1$优于$\pi_2$。

最优策略可以定义为：

若对于所有策略π，有 $v_{\pi^*}(s) \geq v_{\pi}(s), s \in \mathbf{S}$，我们认为策略$\pi^*$是最优策略。

也就是说，求最优策略实际上转化为了求最大V值的问题。但是贝尔曼公式只能用来计算V值，却没告诉我们如何求最大V值，怎么办？

贝尔曼最优公式

这里直接给出贝尔曼最优公式:

$$v = \max_{\pi}\left(r_{\pi} + \gamma P_{\pi} v\right)$$

贝尔曼最优公式巧妙地描述了最优策略和最优状态值的关系。求解该公式实际上就是在求解最优状态值，同样也是在求解最优策略。这就解决了刚刚所提出的问题。

贝尔曼公式

值迭代算法

- 这个迭代的方法和求解贝尔曼公式的方法很类似。进一步观察这个式子，将它变换到代数形式，我们可以进一步发现公式右边实际上是：

$$
\begin{aligned}
v_{k+1}(s) &= \max_{\pi} \sum_{a} \pi(a|s) \left(\sum_{r} p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s') \right) \\
&= \max_{\pi} \sum_{a} \pi(a|s) q_k(s,a) \\
&= \max_{a} q_k(s,a)
\end{aligned}
$$

- 也就是说，在迭代时对V值的估算，需要通过根据估计的V值计算Q值得到。

山东大学 SHANDONG UNIVERSITY

值迭代算法

求解贝尔曼最优公式的流程：

1）对于任何给定的状态s，都有对应的估计值vk（s）；

2）对于状态s下可以选择的所有动作a，我们通过估计值vk（s）和策略πk下的动作选择概率（这里的动作选择一定是贪婪的）计算对应的qk（s，a）；

3）通过计算得到的qk（s，a），更新策略πk+1：策略只选择最大Q值的动作（贪婪策略）；

4）利用qk（s，a）更新vk+1（s）；

5）重复上述步骤直到求得最优策略，即策略不再更新。

这个算法就是**值迭代算法，**它通过不断维护一个状态价值函数vk（s）来求解贝尔曼最优公式。其中初始的v0（s）可以任意初始化，步骤1-3称为**policy update**，步骤4称为**value update**。

贝尔曼公式

山东大学 SHANDONG UNIVERSITY

值迭代算法

值迭代算法可以被这个式子给出：

$$v_{k+1} = \max_{\pi}\left(r_\pi + \gamma P_\pi v_k\right)$$

具体地，又可以分为两个步骤：

$$\pi_{k+1} = \arg\max_{\pi}\left(r_\pi + \gamma P_\pi v_k\right)$$

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

需要注意的是，这里的 v_k 实际上**并不是V值**，它并不能严格满足贝尔曼方程，他只是我们求解过程中不断维护的价值状态函数值。

SHANDONG UNIVERSITY

策略迭代算法

值迭代算法通过维护一个状态价值函数来寻找最优价值，通过直接求解贝尔曼最优公式的方法来求得最优策略。而事实上还存在另一种方法，我们可以通过维护一个策略，不断优化这个策略并最终得到最优策略，来间接的求解贝尔曼最优公式，这个算法就叫做**策略迭代算法。**

和值迭代算法一样，策略迭代算法也分为两步：

1）**Policy evaluation**：给定一个策略πk，利用贝尔曼公式求得对应的V值与Q值。

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$

2）**Policy improvement**：通过计算得到的Q值，我们更新策略得到策略πk+1。

$$\pi_{k+1} = \arg\max_{\pi}\left(r_\pi + \gamma P_\pi v_k\right)$$

策略迭代算法

- 策略迭代算法的思路非常直观，它是一种“逐步完善”的方法。

它维护一个明确的策略π，并通过两个步骤不断迭代。更具体地讲，在Policy evaluation时，我们根据目前的策略π来求得对应的V值与Q值；在Policy improvement时，我们则利用得到的Q值，来更新我们的策略。也就是贪婪法，让策略只选择Q值最大的动作。

值迭代和策略迭代的收敛性都是可以严谨证明的，通过不断地迭代，都可以得到最终的最优策略。然而这两个算法也都有着一个局限性。

贝尔曼公式

值迭代

- 隐式地只优化价值函数，通过求最大V值来得到最优策略；
- 由policy update和value update两个步骤组成；
- 收敛到最优值所需的迭代次数通常会更长，但单轮迭代需要的计算量通常更小；
- 可以看成是先全力画出一张准确的价值图，再按图索骥；
- 必须已知目前的环境模型（奖励和状态转移概率）才能使用。

策略迭代

- 显式地维护并交替优化策略和价值函数;
- 由policy evaluation和policy improvement两个步骤组成；
- 单轮迭代需要的计算量通常更大，但收敛到最优值所需的迭代次数通常会更短；
- 可以看成是在探索中不断地改进自己；
- 必须已知目前的环境模型（奖励和状态转移概率）才能使用。

贝尔曼公式

总结

- 值迭代和策略迭代非常类似，都由两个步骤往复组成。
- 这两个算法解决的问题都是**求解贝尔曼最优公式**，也就是说解决的是如何求解最优策略的问题。
- 这两个算法都基于一个关键假设：智能体已经完全知晓环境的模型，即知道状态转移概率 P 和奖励函数 R。
- 然而现实中大多数情况我们无法完全知晓环境的模型，尤其是状态转移概率，即无法知道采取行动后的全部情况。
- 为了解决这一问题，我们可以参考**蒙特卡洛方法（Monte-Carlo）**。

章节目录 CONTENTS

SHANDONG UNIVERSITY

➢ 蒙特卡洛方法（Monte-Carlo）

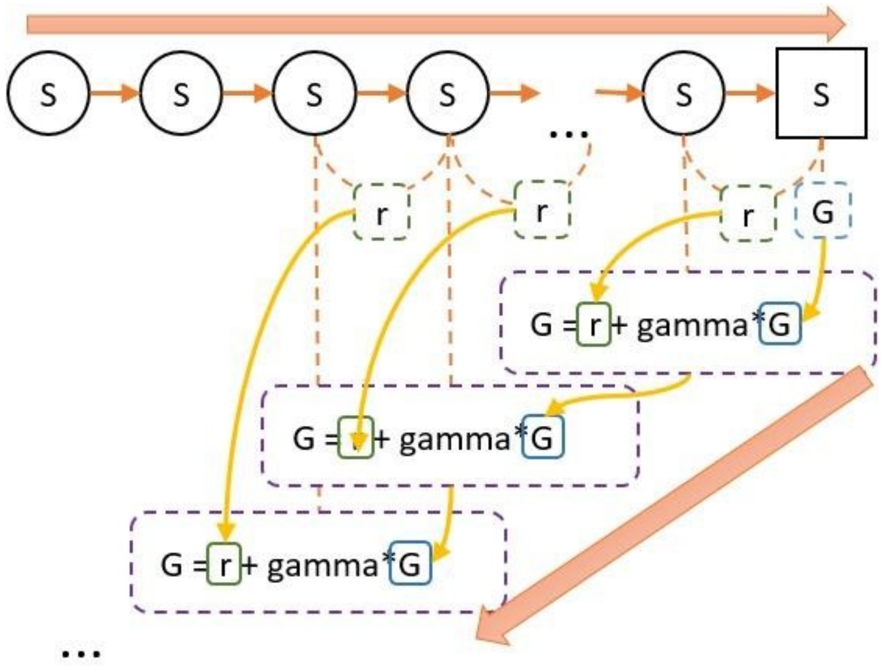
蒙特卡洛方法首先要解决的问题是，在我们没有环境模型时如何计算Q值。这一过程也被称为从**Model-based**方法到**Model-free**方法。其核心思路概括：**用频率代替概率，平均代替期望。**

➢ 具体流程如下：

1. 我们把智能体放到环境的任意状态；
2. 从这个状态开始按照策略进行选择动作，并进入新的状态。
3. 重复步骤2，直到最终状态；
4. 我们从最终状态开始向前回溯：计算每个动作执行后的G值。
5. 重复1-4多次，然后平均每个状态的G值，这就是我们需要求的Q值。

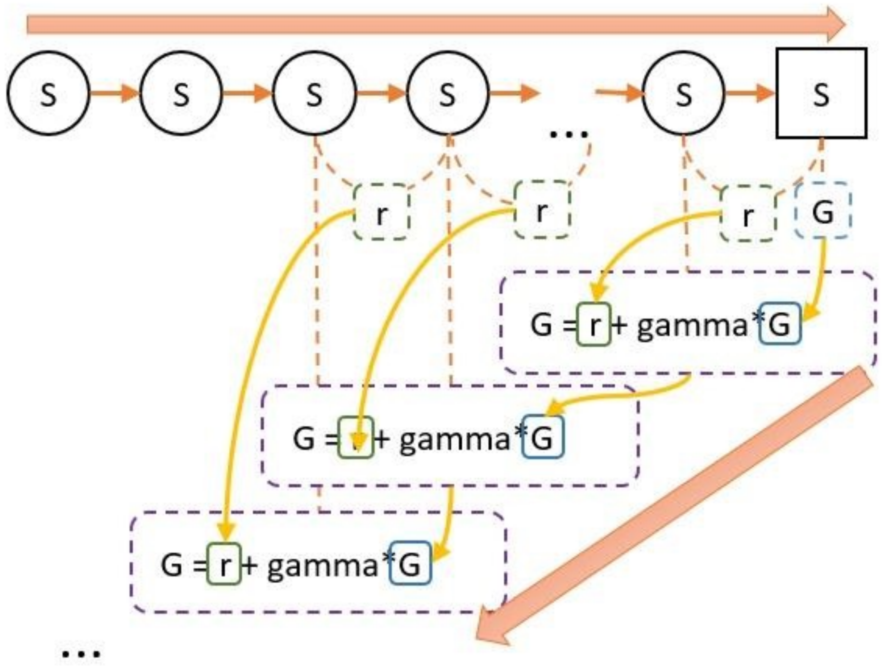
➢ 蒙特卡洛方法（Monte-Carlo）

- 我们首先根据给定的策略π探索环境，得到一段马尔可夫链；然后再向前回溯计算每一次状态转移后的G值并记录。通过多次探索环境，得到多个马尔可夫链后，通过平均G值来估算Q值。这一过程就是**policy evaluation。**



➢ 蒙特卡洛方法（Monte-Carlo）

- 在得到估算的Q值后，下一步就需要优化我们的策略。最简单的方法就是贪婪法，和策略迭代一样，只选择最大Q值的动作更新策略。
- 但是受限于数据的不确定性，更常见的做法是对于Q值最大的动作设置最高的选择概率，同时又保留一小部分概率分配给剩余动作以保证探索性。这个方法叫做**ε-greedy**，因为其通过ε这一参数来平衡探索性。这一步也就是**policy improvement**。



SHANDONG UNIVERSITY

➢ 蒙特卡洛方法（Monte-Carlo）

总结:

MC方法的核心是通过多次采样计算频率来代替概率。这一思路指出了在强化学习中，当我们**没有模型时就需要利用数据**。与此同时，由于是通过采样数据来进行学习，不同于有已知模型的情况下，我们必须要保证智能体训练中的探索性，因此我们通常会使用**ε-greedy**来更新策略。

需要指出，MC方法有着一个重大的缺点：**效率低下!**

- 每一次采样，都需要先从头走到尾，再进行回溯更新。如果最终状态很难达到，那小猴子可能每一次都要转很久很久才能更新一次G值。同时每一次采样的数据只会被利用一次，而这在数据宝贵的强化学习里是非常让人难以接受的。
- 为了解决这一问题，我们接下来要介绍**时序差分算法（TD算法）**

SHANDONG UNIVERSITY

➢ 时序差分算法

- TD-Learning即时序差分算法，通常指一类广泛的强化学习算法，有时候也特指一种专门用于估算V值的方法。我们这里先介绍后者，为大家引入时序差分的核心思想，再以此推广，介绍最著名的一个时序差分算法——**Q-learning**算法。
- TD算法改进自MC算法，为了解决其必须完整采样后才能训练的弊端，我们自然而然会想到，能不能在到达最终状态之前就对估计的V值/Q值进行更新呢？答案是可以的。

1. 和MC不同，TD算法只需要走N步。就可以开始回溯更新。
2. 需要先走N步，每经过一个状态，把奖励记录下来。然后开始回溯。
3. 我们就假设N步之后，就到达了最终状态了。 如果这个“最终状态”我们之前没有走过，所以这个状态是空白的，我们就当这个状态值为0；假设“最终状态”上我们已经走过了，这个状态的V值就是当前值。然后我们开始回溯。

➢ 时序差分算法

我们可以把TD看成是这样一种情况：

- 我们从A状态，经过1步，到B状态。我们什么都不管就当B状态是最终状态了。但B状态本身就带有一定的价值，也就是V值。其意义就是从B状态到最终状态的总价值期望。
- 我们假设B状态的V值是对的，那么，通过回溯计算，我们就能利用B状态的V值更新A状态的估计值了。

这里关键点在于，为什么B状态的V值是对的？或者说为什么B状态的估计会比A状态要更准确？

- 答案在于**奖励**，从A状态到B状态，B状态的估计包含了一个客观的奖励信息，因为其中多了真实值r，所以我们不断用更可靠的估计取代现有估计，就能慢慢接近真实的估计值。

➢ 时序差分算法

- 举个例子：

想象你在山脚下（状态A），要目测到山顶（目标）的距离。你眯着眼猜测：“我估计从这到山顶还有10公里远。”这就是你对状态A的价值估计 **V(A) = 10**。

现在，你实际向山顶方向走了一段路，到达了一个中途点（状态B）。你再次目测，并基于新的、更近的视角猜测：“从B点到山顶，我估计还有7公里远。”这就是 **V(B) = 7**。

那么，你实际从A走到B的这一段路，经地图确认，刚好是 2公里。这就是你获得的沿途奖励

关键问题来了：现在有两个对“A点到山顶总距离”的估计：

1）旧的纯猜测：10 公里（一开始在A点的目测）。

2）基于一步实测的新估计：实际走的2公里 + 从B点目测的7公里 = 9 公里。

哪一个更可信？ 毫无疑问是 9公里。

SHANDONG UNIVERSITY

➢ 时序差分算法

下面我们给出时序差分算法的公式：

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t)\Big[\overbrace{v_t(s_t) - [\underbrace{r_{t+1} + \gamma v_t(s_{t+1})}_{\text{TD target } \bar{v}_t}]}^{\text{TD error } \delta_t}\Big]$$

Here,

$$\bar{v}_t \doteq r_{t+1} + \gamma v(s_{t+1})$$

is called the TD target.

$$\delta_t \doteq v(s_t) - [r_{t+1} + \gamma v(s_{t+1})] = v(s_t) - \bar{v}_t$$

is called the TD error.

➢ 这里的 $\alpha_t(s_t)$ 是学习率，但不同于深度学习里GD使用的那个学习率，这里的学习率控制的是新探索得到的经验，也就是TD-error，对数值更新的贡献。

➢ 时序差分算法

总结：
这一算法最大的好处在于我们并不需要一直探索到最终状态，我们可以先用后面的估算，调整当前状态。直接解决了MC算法的问题。

➢ 需要指出的是这一算法不涉及策略优化，实际上解决的只是如何求Q值和V值的问题。那么我们如何利用这一思路解决策略优化的问题呢？

下面我们会介绍今天的最后一个算法，**Q-learning**算法。

SHANDONG UNIVERSITY

➢ 时序差分算法

Q-learning

回忆MC算法是如何优化策略的？分为两步：1）policy evaluation，2）policy improvement。

关键在于我们的策略优化是基于Q值的！

因此我们自然而然的开始考虑如何利用时序差分的思路来估算Q值，如果能够直接估算Q值，便可以利用Q值来进行策略优化了。

最简单的办法就是：用下一个动作的Q值代替V值。这样我们直接得到：

$$q_{t+1}(s_t,a_t)=q_t(s_t,a_t)-\alpha_t(s_t,a_t)\Big[q_t(s_t,a_t)-[r_{t+1}+\gamma q_t(s_{t+1},a_{t+1})]\Big]$$

这个公式就是Sarsa，其实很简单，就是一个Q值版本的TD算法。利用这个公式完成Q值估计后，就可以按照MC算法中说明过的步骤进行迭代了。

➢ 时序差分算法

Q-learning

- 那么什么是Q-learning呢？我们也直接给出公式：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)]\right]$$

- 与Sarsa不同的是，它的TD-target从简单的下一个动作的Q值加奖励，变成了当前状态下可能的最大Q值的加上奖励。
- 道理其实也很简单：因为我们需要寻着的是能获得最多奖励的动作，Q值就代表我们能够获得今后奖励的期望值。所以我们也只会选择Q值最大的路径用于更新估计值。
- 进一步，这还有一个更加巧妙的好处：它实际上促成了负责探索的策略和被更新策略的分离，这种变化就是从**On-policy**到**Off-policy**的变化。

SHANDONG UNIVERSITY

➢ 时序差分算法

Q-learning

- 在说明Q-learning是如何做到Off-policy之前，我们先考虑其进行policy evaluation的流程：

  1. 在状态 S，根据**当前策略（如ε-greedy）**实际选择动作 A。

  2. 执行 A，获得奖励 R，进入新状态 S′ 。

  3. 在状态 S′ ，**它并不根据当前策略选择动作**，而是直接“放眼望去”，选择能带来最大Q值的那 个动作，即 max Q(S', a)。注意，这个“最大Q值动作”它不一定真的会执行。

  4. 用这个理论上的最优动作对应的Q值，来更新 (S, A) 的Q值。

- 请注意步骤3，这里的 max Q(S′ , a) 完全无视了策略实际会做什么动作。它直接使用这个理想化的、最优的价值来更新 Q(S, A)。**因此这里所有的Q值实际上对应的是一个贪婪策略**，这和步骤1中进行动作选择的**策略（可能是ε-greedy ）**是不同的。

➢ 时序差分算法

Q-learning

- 在说明Q-learning是如何做到Off-policy之前，我们先考虑其进行policy evaluation的流程：

Q值对应的贪婪策略和进行动作选择的策略是两个策略！！

和步骤1中进行动作选择的**策略（可能是ε-greedy）**是不同的。

SHANDONG UNIVERSITY

➢ 时序差分算法

Q-learning

现在我们清楚了，在Q-learning算法里实际上存在了两个策略：

1）**行为策略**：在环境中负责进行探索（选择Action）的策略

2）**目标策略**：最终我们需要的策略，因此是一个贪婪策略，同时也是实际上我们更新的目标

所以这种Off-policy比之前MC，Sarsa这些On-policy算法究竟有什么好处?

- Off-policy的本质就是：我们用一个探索性的行为策略去收集经验数据，却用这些数据来评估和改进另一个贪婪的目标策略。这种分离带来了巨大优势：Q-learning可以用任何方式探索（行为策略），哪怕是很随机、很冒险的方式去收集数据，但它学到的始终是那个隐藏在数据背后的、不包含探索噪声的最优策略（目标策略）。
- Q-learning是后续一系列目前主流的算法的基石，大量著名的算法都改进自它，如DQN等。

课程总结

1.首先我们介绍了**马尔可夫链**，它是整个强化学习的基石。**状态、动作、奖励**组成了一个最基本的架构。

2.紧接着我们介绍了**V值和Q值**，他们可以用来评估状态和动作。并且进一步介绍了**贝尔曼方程**和**贝尔曼最优方程**，通过贝尔曼方程我们可以求解V与Q值，并进一步通过**值迭代**和**策略迭代**来求得最优策略。

3.然后我们介绍了**MC算法**，它解决了当**环境模型未知**的情况下我们如何计算Q值V值并优化策略

4.然而MC算法的效率太低，为了提高数据利用率，**TD算法**通过不断用更可靠的估计取代现有估计解决了如何在探索时同步进行更新，**大大提高了计算效率**。

5.为了进一步把TD思想推广到策略更新和Q值计算上，我们尝试直接把TD算法里的V值改为Q值，自然而然得到了**Sarsa算法**。进一步的，**Q-learning**又将TD-error里的Q值替换为最大的Q值，通过这一trick，巧妙地分离出了行为策略和目标策略，实现了强大的**Off-policy Learning**方法，大大提高了算法的效果。