

# INTRODUCTION

## Characteristics of Reinforcement Learning

- No supervisor
- Delayed Feedback, not instantaneous
- Time matters (non iid data, sequential)
- Agent's actions affect the subsequent data it receives

## CONTENT

### PART I Theory

- Introduction
- MDP 马尔科夫决策过程
- 动态规划
- 不基于模型的预测
- 不基于模型的控制

### PART II Practice

- 价值函数的近似表示
- 策略梯度方法
- 整合学习与规划
- 探索和利

## Reward Reinforcement Learning is based on Reward Hypothesis

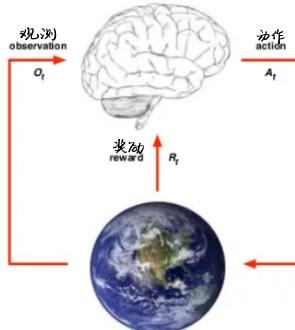
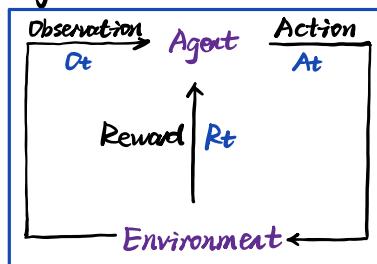
- A reward  $R_t$  is a scalar feedback signal 标量反馈信号
- Indicates how well agent is doing at step  $t$
- The agent's job is to maximize cumulative reward 最大化累计奖励

All goals can be described by the maximisation of expected cumulative reward.

## Sequential Decision Making 序列决策

- Goal: Select actions to maximize total future reward 最大化未来总体奖励
- Actions may have long term consequences 长期序列
- Reward may be delayed 延迟奖励
- It may be better to sacrifice immediate reward to gain more long-term reward

## Agent & Environment



在  $t$  时刻, Agent

- 观察环境  $O_t$
- 做出行为  $A_t$
- 从环境中获得奖励  $R_{t+1}$

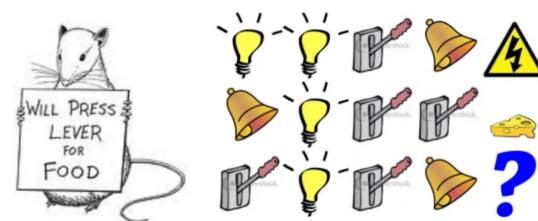
在  $t$  时刻, Environment

- 接收 Agent 的动作  $A_t$
- 更新环境信息, 使 Agent 得到下一个观测  $O_{t+1}$
- 给个体奖励信号  $R_{t+1}$

## History & State

- History 观测 (Observation)、行为 (Action)、奖励 (Reward) 的序列
- State 既有决定将来的已有信息, 关于 History 的一个函数  $S_t = f(H_t)$

有如下三个针对老鼠的事件序列, 其中前两个最后的事件分别是老鼠遭电击和获得一块奶酪, 现在请分析比较这三个事件序列的特点, 分析第第三个事件序列中, 老鼠是获得电击还是奶酪?



假如个体状态 = 序列中的后三个事件 (不包括电击、获得奶酪, 下同), 事件序列3的结果会是什么? (答案是: 电击)

假如个体状态 = 亮灯、响铃和拉电闸各自事件发生的次数, 那么事件序列3的结果又是什么? (奶酪)

假如个体状态 = 完整的事件序列, 那结果又是什么? (未知)

## 环境状态 环境的外部呈现

- 包含环境用来决定下一个 Observation / Reward 的所有数据
- 对 Agent 不完全可见 Agent 不知道 Environment 状态的所有细节

## Agent 状态 个体的内部呈现

- 包含个体可使用的、决定未来动作的所有信息
- 是 RL 算法可以利用的信息
- 可以是 History 的一个函数  $S_t^a = f(H_t)$

## 信息状态

- 包含历史上所有有用的信息, 即 Markov 状态
- Markov Property

### Definition

A state  $S_t$  is **Markov** if and only if

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

如果信息状态可知, 则所有历史信息都可以丢弃

只需  $t$  时刻的信息状态

e.g. History  $H_t$ , Environment 状态

## 完全可观测的环境 Fully observable Environments

个体能够直接观测到环境状态 个体对环境的观测 = 个体状态 = 环境状态

→ Markov Decision Process, MDP

## 部分可观测的环境 Partially Observable Environments

个体状态 + 环境状态 部分可观测 MDP, Agent 需要构建自己的状态呈现形式, e.g. 记住完整的历史  $S_t^0 = H_t$

### • Beliefs of Environment

Agent 利用已有经验(数据), 用 Agent 已知状态的概率分布作为当前时刻的 Agent 状态的呈现

$$S_t^a = (\mathbb{P}[S_t^a = s^1], \dots, \mathbb{P}[S_t^a = s^n])$$

### • Recurrent Neural Network

不需要知道概率, 只根据当前 Agent 状态与当前时刻 Agent 的 Observation, 送入 RNN 得到当前 Agent 状态呈现

$$S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$$

## RL Agent 主要组成 Policy、Value Function、Model

### • 策略 Policy 决定 Agent 动作的机制

• 从 State 到 Action 的映射

• 可以是确定性, 也可以是不确定性

### • 价值函数 Value Function 对未来 Reward 的预测, 用来评价当前 State

• Agent 用一个 Value 评估 State 可能获得的最终 Reward, 继而指导选择不同的 Action, 制定不同的 Policy

• Value Function 是基于某一个特定的 Policy 的, 不同 Policy 下同一 State 的 Value 不同

$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

### • 模型 Model Agent 对 Environment 的建模

• 体现 Agent 是如何思考 Environment 运行机制的

• Agent 希望模型能模拟 Environment 与 Agent 的交互机制

• 解决两个问题

状态转移概率 预测下一个可能 State 发生的概率

$$P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

预测可能获得的即时 Reward

$$R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

• Model 不是构建 Agent 需要的 Agent 不依赖 Model

Dynamics of Environment

• Model 仅针对 Agent 而言, Environment 实际运行机制不称为 Model, 而称为 环境动力学

## RL Agent 分类

### • 基于价值函数 Value Based

• 有对 State 的 Value Function, 但没有直接的 Policy Function

• Policy Function 从 Value Function 得到

### • 基于策略 Policy Based

• Action 由 Policy Function 产生

• Agent 不维护一个对各 State Value 估计函数

### • 演员-评判家形式 Actor-Critic

既有 Value Function, 也有 Policy Function

## 学习和规划 Learning & Planning

### 常用解决思路

先学习 Environment to work (了解 Environment 工作的方式), 学习到一个 Model, 利用这个 Model 进行规划

### • 学习

• 初始时环境未知, Agent 不知道 Environment to work

• Agent 通过与 Environment 交互, 逐渐改善 Action 的 Policy

### • 规划

• Environment to work 对于 Agent 是已知或近似已知的

• Agent 不与 Environment 发生实际交互, 而是利用构建的 Model 进行计算, 以此为基础改善行为 Policy

## 探索和利用 Exploration & Exploitation Agent 进行决策时需要平衡的两个方面

Agent 需要从与环境的交互中发现好的 Policy, 但又不能在试错的过程中丢失太多 reward

## 预测和控制 Prediction & Control

需要先解决 Prediction 问题，在此基础上解决关于 Control 的问题

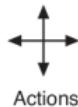
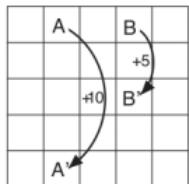
### • 预测 Prediction 给定 Policy, 评估未来

→ 求解在给定 Policy 下的 Value Function 的过程

How will the agent do if it follows a specific policy?

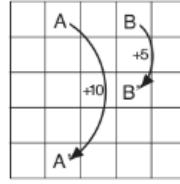
### • 控制 Control

找到一个好的 Policy 最大化未来的奖励



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

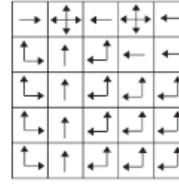
(b)



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b)  $v_*$



c)  $\pi_*$

## MARKOV DECISION PROCESS

MDP 描述完全可观测的 Environment → 观测到的 state 完整决定了决策需要的特征

### 马尔科夫过程 Markov Process

#### • 马尔科夫性 Markov Property

→ 某一 State 的信息包含了所有相关的 History, 只要当前 State 已知, 所有历史信息都不再需要  
当前 state 即可决定未来

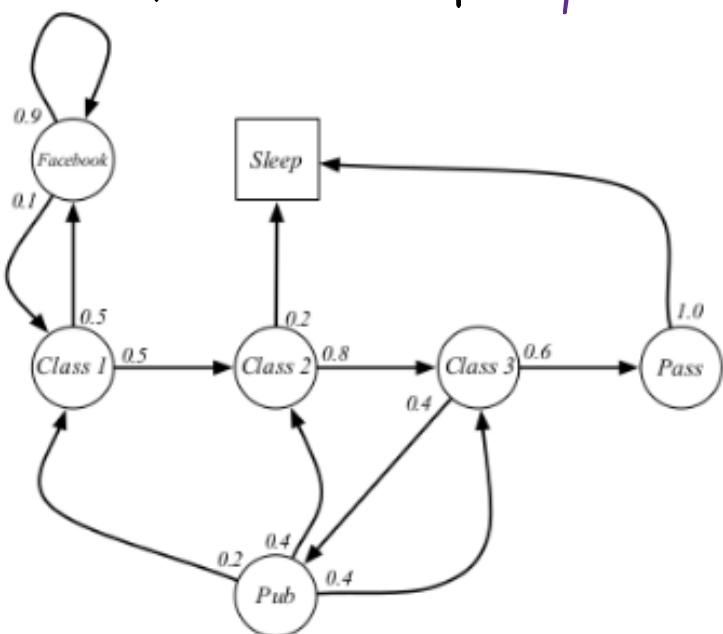
$P_{ss'} = P[S_{t+1} = s' | S_t = s]$  可以用状态转移概率公式描述马尔科夫性

$$P = \text{from} \begin{bmatrix} P_{11} & \dots & P_{1n} \\ \vdots & & \vdots \\ P_{n1} & \dots & P_{nn} \end{bmatrix}^{\text{to}}$$

状态转移矩阵定义了所有状态的转移概率  
n 为 state 数量, 矩阵中每一行元素之和为 1

#### • 马尔科夫过程 Markov Process $\leftrightarrow$ 马尔科夫链 (Markov Chain)

无记忆的随机过程, 可以用 tuple  $\langle s, p \rangle$  表示 S 是有限数量的 state 集, P 是状态转移矩阵



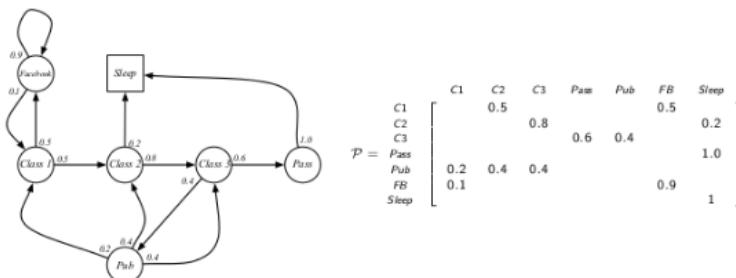
- 圆圈表示学生所处的状态
- 方格 Sleep 是一个终止状态, 或者可以描述成自循环的状态, 也就是 Sleep 状态的下一个状态 100% 的几率还是自己
- 箭头表示状态之间的转移, 箭头上的数字表示当前转移的概率

**举例说明:** 当学生处在第一节课 (Class1) 时, 有 50% 的几率会参加第 2 节课 (Class2); 同时在也有 50% 的几率不在认真听课, 进入到浏览 facebook 这个状态中。在浏览 facebook 这个状态时, 有 90% 的几率在下一时刻继续浏览, 也有 10% 的几率返回到课堂内容上来。当学生进入到第二节课 (Class2) 时, 会有 80% 的几率继续参加第三节课 (Class3), 也有 20% 的几率觉得课程较难而退出 (Sleep)。当学生处于第三节课这个状态时, 他有 60% 的几率通过考试, 继而 100% 的退出该课程, 也有 40% 的可能性需要到去图书馆之类寻找参考文献, 此后根据其对课堂内容的理解程度, 又分别有 20%、40%、40% 的几率返回值第一、二、三节课重新继续学习。一个可能的学生马尔科夫链从状态 Class1 开始, 最终结束于 Sleep, 其间的过程根据状态转化图可以有很多种可能性, 这些都称为 Sample Episodes。

课堂内容的理解程度, 又分别有 20%、40%、40% 的几率返回值第一、二、三节课重新继续学习。一个可能的学生马尔科夫链从状态 Class1 开始, 最终结束于 Sleep, 其间的过程根据状态转化图可以有很多种可能性, 这些都称为 Sample Episodes。

以下四个Episodes都是可能的：

- C1 - C2 - C3 - Pass - Sleep
- C1 - FB - FB - C1 - C2 - Sleep
- C1 - C2 - C3 - Pub - C2 - C3 - Pass - Sleep
- C1 - FB - FB - C1 - C2 - C3 - Pub - C1 - FB - FB - FB - C1 - C2 - C3 - Pub - C2 - Sleep



### • 马尔科夫奖励过程 Markov Reward Process

在马尔科夫的基础上增加奖励 R 和衰减系数  $\gamma < S, P, R, \gamma >$

$$R_s = E[R_{t+1} | S_t = s] \quad \bullet R \text{ 是奖励函数}$$

$S$  状态下的 Reward 是  $t$  时刻处在状态  $S$  下在下一刻  $(t+1)$  能获得的 Reward 期望  
在表述上可以把奖励描述为“当进入某个状态会获得相应奖励”

衰减系数 Discount Factor  $\gamma \in [0, 1]$

为什么引入衰减系数？

- 数学表达的方便，避免陷入无限循环
- 远期利益具有一定的不确定性
- 符合人类对于眼前利益的追求
- 符合金融学上获得的利益能够产生新的利益因而更有价值

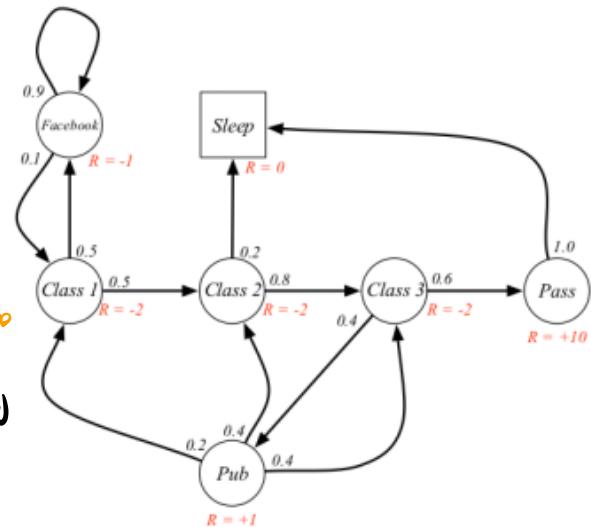
### • 收获 Return $G_t$

在一个MRP上从  $t$  时刻开始往后所有的 Reward 的有衰减的总和

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

• 衰减系数  $\gamma$  体现了未来的 Reward 在当前时刻的价值比例

- 在  $k+1$  时刻获得的奖励  $R$  在  $t$  时刻体现出的价值是  $\gamma^k R$
- $\gamma \rightarrow 0$ , 则趋向短视评估;  $\gamma \rightarrow 1$ , 则偏向考虑远期利益



价值函数 Value Function 给出了某 state 或 Action 的长期价值

一个MRP中某State的 Value Function 为从该State开始的 MP 的 Return 的期望

$$v(s) = E[G_t | S_t = s]$$

• 价值可以仅描述 state, 也可以描述某 state 下的 Action, 特殊情况下还可以仅描述 Action

• 约定: State Value Function 或 Value Function 描述针对 state 的价值

Action Value Function 描述某一 State 下执行某一 Action 的价值  
(State-Action Value Function 的简写)

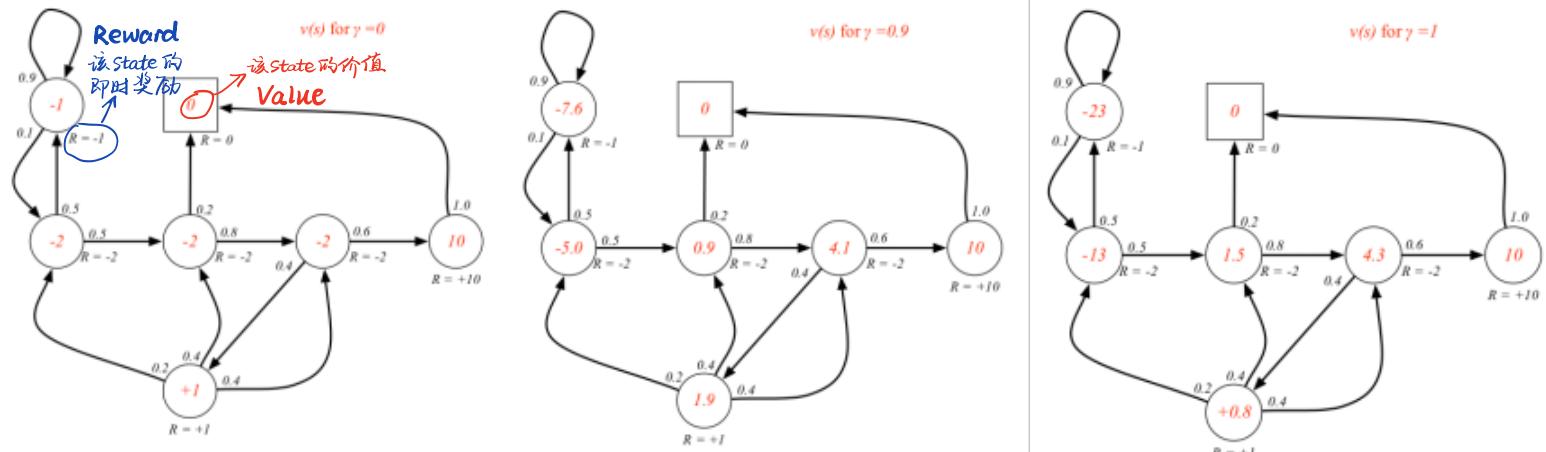
- 表中第二行对应各状态的即时奖励值
- 数字为状态转移概率, 表示从所在行状态转移到所在列状态的概率

States	C1	C2	C3	Pass	Pub	FB	Sleep
Rewards	-2	-2	-2	10	1	-1	0
C1	0.5				0.5		
C2			0.8				0.2
C3				0.6	0.4		
Pass							1
Pub	0.2	0.4	0.4				
FB	0.1				0.9		
Sleep							1

C1 C2 C3 Pass Sleep	$G_1 = -2 + (-2) * 1/2 + (-2) * 1/4 + 10 * 1/8 + 0 * 1/16 = -2.25$
C1 FB FB C1 C2 Sleep	$G_1 = -2 + (-1) * 1/2 + (-1) * 1/4 + (-2) * 1/8 + (-2) * 1/16 + 0 * 1/32 = -3.125$
C1 C2 C3 Pub C2 C3 Pass Sleep	$G_1 = -2 + (-2) * 1/2 + (-2) * 1/4 + (1) * 1/8 + (-2) * 1/16 + \dots = -3.41$
C1 FB FB C1 C2 C3 Pub C1 FB FB C1 C2 C3 Pub C2 Sleep	$G_1 = -2 + (-1) * 1/2 + (-1) * 1/4 + (-2) * 1/8 + (-2) * 1/16 + (-2) * 1/32 + \dots = -3.20$

以上4个马尔科夫链, 当  $\gamma = 1/2$  时, 在  $t=1$  时刻状态  $S_1$  的 Return 分别表中所示。 $(S_1 = C_1)$

→ Return 是针对 MP 中某一个 State 而言的  
当  $\gamma = 0$ , MRP 中各 State 的 Return 与该 State 的价值相同; 当  $\gamma \neq 0$ , 各 State 的价值需计算得到  
→ State 的 Value 确实十分关键  
RL 许多问题都可以归结为求 State 的 Value  
(如何求各 State 的 Value, 即寻找 Value Function  
十分重要)



## 价值函数的推导 Value Function

### • Bellman 方程 - MRP

$$\begin{aligned}
 v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \rightarrow \text{Return 的期望} \\
 &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] = \text{Return 期望的期望}
 \end{aligned}$$

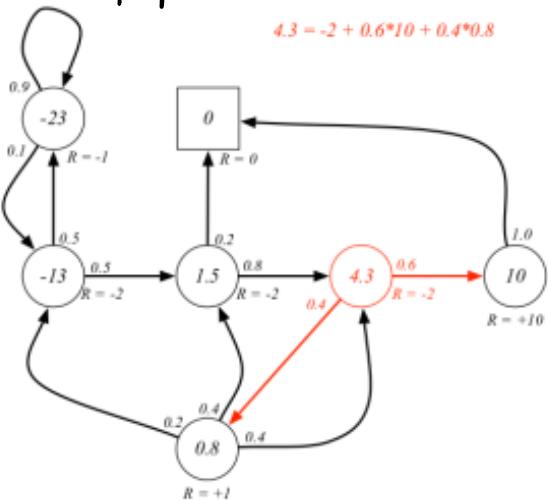
针对 MRP 的 Bellman 方程：

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$

- 该 State 的即时 Reward 期望 = 即时 Reward (即时 Reward 与下一个状态无关)
- 下一时刻 State 的 Value 期望 (根据下一时刻 State 的概率分布得到期望)

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

### • 方程的解释



### • Bellman 方程的矩阵形式和求解

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$(I - \gamma \mathcal{P}) v = \mathcal{R}$$

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

• 计算复杂度  $O(n^3)$   $n$  是状态数量

• 小规模 MRP - 直接求解

大规模 MRP - 迭代法

动态规划 Dynamic Programming  
蒙特卡洛评估 Monte-Carlo Evaluation  
时序差分学习 Temporal-Difference

$\gamma=1$  时各状态的 Value (V 必须确定才能计算)

State C3 的 Value 通过 Value 以及状态转移概率计算

$$4.3 = -2 + 1.0 * (0.6 * 10 + 0.4 * 0.8)$$

### 马尔科夫决策过程 Markov Decision Process

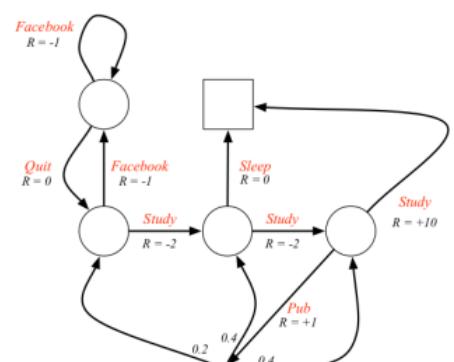
• 相对于马尔科夫奖励过程 (MRP), MDP 多了行为集合  $A < S, A, P, R, \gamma >$

• MDP 的  $P, R$  都与具体的行为  $a$  对应, 而 MRP 只对应状态  $S$

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

示例：图中红色的文字表示的是采取的行为，而不是先前的状态名。对比之前的 MRP 例题可以发现，即时奖励与行为对应了，同一个状态下采取不同的行为得到的即时奖励是不一样的。由于引入了 Action，容易与状态名混淆，因此此图没有给出各状态的名称；此图还把 Pass 和 Sleep 状态合成为一个终止状态；另外当选择“去查阅文献”这个动作时，主动进入了一个临时状态（图中用黑色小实点表示），随后被动的被环境按照其动力学分配到另外三个状态，也就是说此时 Agent 没有选择权决定去哪一个状态。



## • 策略 Policy

- 策略  $\pi$  是概率的集合或分布  $\pi(a|s)$  为对过程中某一 State  $s$  采取可能的 Action  $a$  的概率

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

- Policy 完整定义了 Agent 的行为方式 即 Agent 在各个 state 下各种可能的行为方式以及其概率的大小
- Policy 仅和当前 State 有关, 与 History 无关
- 某一确定的 Policy 是静态的, 与时间无关
- Agent 可以随时间更新 Policy

给定 MDP:  $M = \langle S, A, P, R, \gamma \rangle$  加一个策略  $\pi$ , 则 State 序列  $S_1, S_2, \dots$  是一个 MP  $\langle S, P^{\pi} \rangle$

State 和 Reward 序列  $S_1, R_2, S_2, R_3, S_3, \dots$  是一个 MRP  $\langle S, P^{\pi}, R^{\pi}, \gamma \rangle$ , 并且满足方程

$$P_{s,s'}^{\pi} = \sum_{a \in A} \pi(a|s) P_{ss'}^a \quad \text{执行 Policy } \pi \text{ 时, State } s \text{ 转移至 } s' \text{ 的概率等于一系列概率的和}$$

$\pi(a|s) \leftarrow$  在执行当前 Policy 时, 执行某一 Action 的概率与  $P_{ss'}^a \leftarrow$  该 Action 能使状态从  $s$  转移至  $s'$  的乘积

$$R_s^{\pi} = \sum_{a \in A} \pi(a|s) R_s^a \quad \text{当前 State } s \text{ 下执行某指定 Policy 得到的即时 Reward 是该 Policy 下所有可能 Action 得到的 Reward 与该 Action 发生的概率的乘积的和}$$

$$R_s^a \quad \pi(a|s)$$

Policy 在 MDP 中的作用: • Agent 可以在某个状态时做出选择  $\rightarrow$  有形成各种 MP 的可能

• 基于 Policy 产生的每一个 MP 是一个 MRP

各过程间的差别是不同的选择产生了不同的后续状态以及对应不同的 Reward

## • 基于 Policy $\pi$ 的 Value Function

$v_{\pi}(s)$  是 MDP 下基于 Policy  $\pi$  的 State Value Function

从状态  $s$  开始, 遵循当前 Policy 时, 获得的 Return 的期望

$\Leftrightarrow$  在执行当前 Policy 时, 衡量 Agent 处在状态  $s$  时的 Value

Policy 是静态的、关于整体的, 不随 State 改变

在某个 State 下, 依据 Policy 可能产生的具体行为可能会变化  
有一定概率

Policy 用于描述不同 State 下执行不同 Action 的概率

行为价值函数 Action Value Function  $q_{\pi}(s, a)$

$\rightarrow$  在执行 Policy  $\pi$  时, 对当前 State  $s$  执行具体 Action  $a$  所能得到的 Return 的期望

$\rightarrow$  遵循当前 Policy  $\pi$  时, 衡量对当前 State 执行 Action  $a$  的 Value

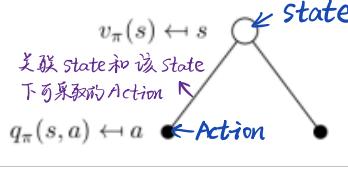
$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

一般与某特定的 State 对应 更精细的表达: State-Action Value Function

## • Bellman 期望方程 Bellman Expectation Equation

MDP 下的 State Value Function 和 Action Value Function 可以用下一时刻的 State Value Function 表达

$$v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s] \quad q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$



$v_{\pi}(s) \leftarrow s$  遵循 Policy  $\pi$  时, State  $s$  的 Value 体现为该 State 下遵循 Policy 采取所有可能的 Action 的 Value 按行为发生概率的乘积求和

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s')$$

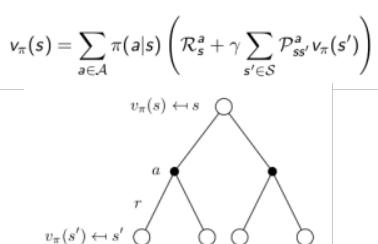
$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) q_{\pi}(s, a)$

$\rightarrow$  Action Value Function 可以表示为 State Value Function 形式

$\rightarrow$  某状态下采取一个 Action 的 Value 可分为两部分:

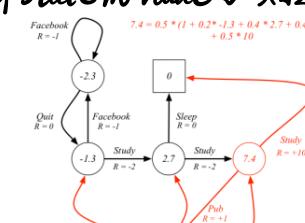
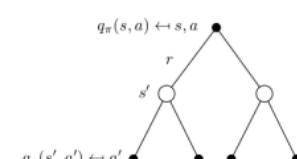
• 离开这个 State 的 Value

• 所有进入新 State 的 Value 与其转移概率乘积的和



$$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right)$$

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} \sum_{a' \in A} \pi(a'|s') P_{ss'}^a q_{\pi}(s', a')$$



Student MDP  
 $\rightarrow$  随机 Policy  
 所有可能的 Action 有相同的概率被选择执行

## Bellman 期望方程矩阵形式

$$v_{\pi} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} v_{\pi} \rightarrow v_{\pi} = (I - \gamma \mathcal{P}^{\pi})^{-1} \mathcal{R}^{\pi}$$

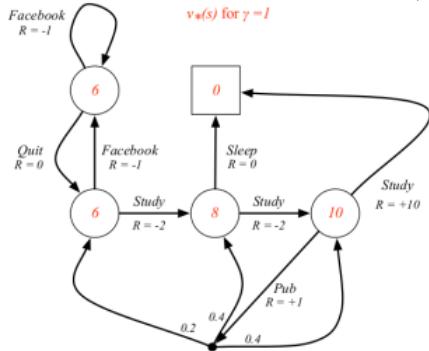
- 最优 Value Function 明确了 MDP 的最优可能表现, 知道最优 Value Function 也就知道了每个 State 的最优价值  
→ 最优状态 Value Function  $v_*(s)$

$$v_* = \max_{\pi} v_{\pi}(s) \text{ 从所有 Policy 产生的 State Value Function, 选择使 State } s \text{ Value 最大的函数}$$

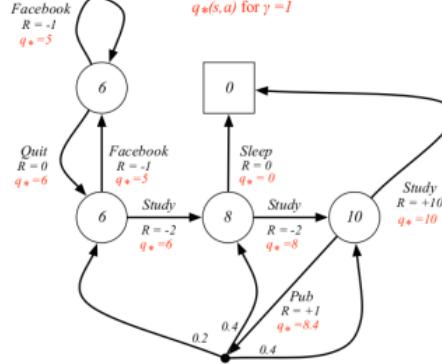
- 最优 Action Value Function

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a) \text{ 从所有 Policy 产生的 Action Value Function, 选择使 } (s, a) \text{ Value 最大的函数}$$

学生 MDP 问题的最优状态价值



学生 MDP 问题的最优行为价值示例



## 最优策略

对于任何 State  $s$ , 遵循 Policy  $\pi$  的 Value 不小于遵循 Policy  $\pi'$  下的 Value, 则 Policy  $\pi$  优于 Policy  $\pi'$

$$\pi \geq \pi' \text{ if } v_{\pi}(s) \geq v_{\pi'}(s), \forall s$$

Theorem 定理

对于 MDP • 存在一个最优 Policy, 比任何其他 Policy 更好或至少相等

- 所有最优 Policy 有相同的最优 Value Function
- 所有最优 Policy 具有相同的 Action Value Function

## 寻找最优 Policy 通过最大化 Action Value Function 来找到最优 Policy

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

对于任何 MDP 问题, 总存在一个确定性最优 Policy  
如果知道了最优 Action Value Function, 则表明找到了最优策略  
(行为价值函数)

## Bellman 最优方程 Bellman Optimality Equation

针对  $v_*$ , 一个 State 的最优 Value

为从该 State 出发采取的所有 Action

产生的 Action Value 中最大的 Action Value

针对  $q_*$ , 在某个状态  $s$  下, 采取某

个 Action 的最优 Value 由 2 部分组成:

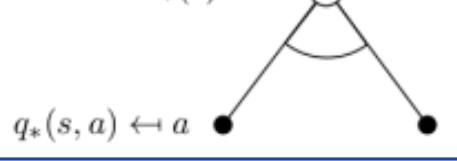
- 离开 State  $s$  的即刻 Reward

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- 所有能到达的状态  $s'$  的最优 State Value 按出现的概率求和

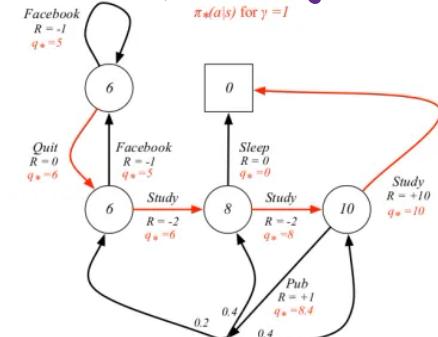
$$v_*(s) = \max_a q_*(s, a)$$

$$v_*(s) \leftrightarrow s$$

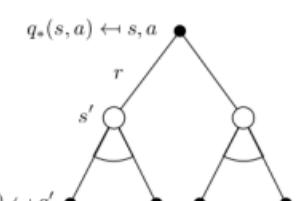
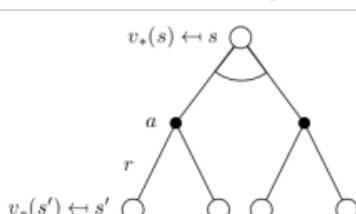


$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

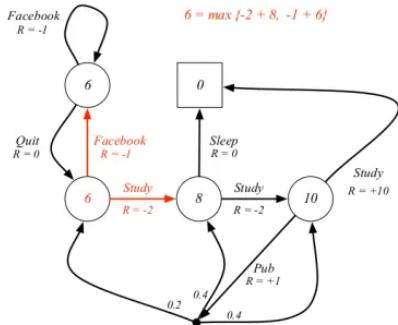
学生 MDP 最优 Policy 示例



$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$



## • Bellman 最优方程学生MDP示例



→ 求解 Bellman 最优方程

Bellman 最优方程是非线性的，没有固定的解决方案  
通过迭代方法解决

- 价值迭代
- 策略迭代
- Q-Learning
- Sarsa

# PLANNING BY DYNAMIC PROGRAMMING

动态规划：进行RL的“规划”，即在已知模型上判断一个 Policy 和 Value Function，并在此基础上寻找最优的 Policy 和 Value Function，或直接寻找最优 Policy 和 Value Function

## 动态规划算法

- 将复杂问题分解为子问题，通过求解子问题进而得到整个问题的解
- 子问题的结果通常需要存储起来用于解决后续复杂问题
- 问题具有特性
  - 一个复杂问题的最优解由若干个小问题的构成
  - 子问题在复杂问题内重复出现

MDP具有上述两个属性：Bellman方程把问题递归为求解子问题，价值函数就相当于存储了一些子问题的解，可以复用。因此可以使用动态规划来求解MDP。

预测：给定 MDP  $\langle S, A, P, R, \gamma \rangle$  和 Policy  $\pi$  或给定 MRP  $\langle S, P^\pi, R^\pi, \gamma \rangle$ ，预测基于当前 Policy  $\pi$  的 Value Function  $V^\pi$

控制：给定 MDP  $\langle S, A, P, R, \gamma \rangle$ ，确定最优 Value Function  $V^*$  和最优 Policy  $\pi^*$

## 迭代策略评估 Iterative Policy Evaluation

### • 理论

Question：评估一个给定的 Policy  $\pi$ ，解决“预测”问题

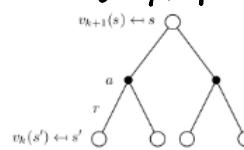
Solution：反向迭代应用 Bellman 期望方程

方法：同步反向迭代 每次迭代中，对第  $k+1$  次迭代，所有 State  $s$  的 Value 用  $V_k(s')$  计算并更新该状态第  $k+1$  次迭代中使用的价值  $V_k(s)$ ，其中  $s'$  是  $s$  的后继状态。→ 反复迭代后最终收敛至  $V^\pi$

异步反向迭代 在第  $k$  次迭代使用当次迭代的状态价值来更新状态价值

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$$

一次迭代内，状态  $s$  的 Value 为前一次迭代该状态的即时 Reward 与所有状态的下一个可能状态  $s'$  的 Value 与其概率乘积和



$$v^{k+1} = R^\pi + \gamma P^\pi v^k$$

### • 改善策略

→ 在一个给定的 Policy 下迭代更新 Value Function  
 $v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$

→ 在当前 Policy 的基础上，贪婪选取行为，使后继状态 Value 增加最多

$$\pi' = \text{greedy}(v_\pi)$$

## 策略迭代 Policy Iteration

在当前 Policy 上迭代计算 Value ( $V$ )，根据  $V$  值贪婪更新 Policy。反复多次  
最终得到最优 Policy  $\pi^*$  和最优状态价值函数  $V^*$

贪婪 指仅采取那些使得状态 Value 最大的行为

### • 改善策略

1. 考虑一个确定的策略： $a = \pi(s)$

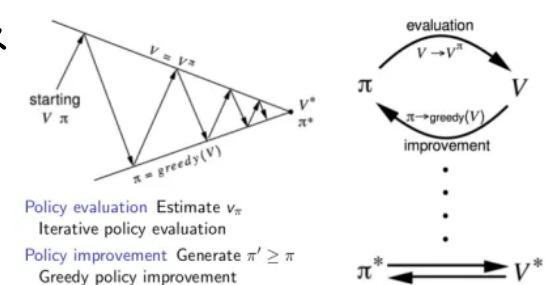
2. 通过贪婪计算优化策略： $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$

3. 这会用一步迭代改善状态  $s$  的  $q$  值，即在当前策略下，状态  $s$  在动作  $\pi'(s)$  下得到的  $q$  值等于当前策略下

状态  $s$  所有可能动作得到的  $q$  值中的最大值。这个值一般不小于使用当前策略得到的行为所得到的  $q$  值，因而也就是该状态的状态价值。

4. 如果  $q$  值不再改善，则在某一状态下，遵循当前策略采取的行为得到的  $q$  值将会是最优策略下所能得到的最大  $q$  值，上述表示就满足了 Bellman 最优方程，说明当前策略下的状态价值就是最优状态价值。

5. 因而此时的策略就是最优策略。



## • 修饰过的策略迭代 Modified Policy Iteration

有时无需持续迭代至最优 Value Function, 可设置条件提前终止迭代

- 设定  $\epsilon$
- 比较两次迭代的 Value Function 平方差
- 直接设置迭代次数
- 每迭代一次更新一次 Policy

## 价值迭代 Value Iteration

### • 优化原则 Principle of Optimality

- 一个最优 Policy 可分为 两个部分
- 从状态  $s$  到下一个状态  $s'$  采取了最优行为  $A^*$
  - 在状态  $s'$  时遵循一个最优 Policy

Theorem 一个 Policy 能够使状态  $s$  获得最优 Value, 当且仅当:

对于从状态  $s$  可以到达的任何状态  $s'$ , 该 Policy 能够使状态  $s'$  的价值是最优 Value

### • 确定性价值迭代 Deterministic Value Iteration

在前一个定理的基础上, 如果我们清楚地知道我们期望的最终 (goal) 状态的位置以及反推需要明确的状态间关系, 那么可以认为是一个确定性的价值迭代。此时, 我们可以把问题分解成一些列的子问题, 从最终目标状态开始分析, 逐渐往回推, 直至推至所有状态。

Question: 寻找最优 Policy  $\pi$

Solution: 从初始 state Value 开始同步迭代计算, 最终收敛, 整个过程没有遵循任何 Policy

Caution: • Value Iteration 中算法不会给出明确的 Policy

迭代中得到的 Value Function 不对应任何 Policy

• 价值迭代没有 Policy 参与, 但仍需 state 间的转移概率 (需要知道模型)

预测问题: 在给定策略下迭代计算价值函数。

控制问题: 策略迭代寻找最优策略问题则先在给定或随机策略下计算状态价值函数, 根据状态函数贪婪更新策略, 多次反复找到最优策略; 单纯使用价值迭代, 全程没有策略参与也可以获得最优策略, 但需要知道状态转移矩阵, 即状态  $s$  在行为  $a$  后到达的所有后续状态及概率。

## 动态规划的扩展

### • 异步动态规划 Asynchronous Dynamic Programming

→ 原位动态规划 In-place Dynamic Programming

→ 重要状态优先更新 Prioritised Sweeping

→ 实时动态规划 Real-time Dynamic Programming

### • 采样更新 Sample Backups

### • 近似动态规划 Approximate Dynamic Programming

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function  $v_\pi(s)$  or  $v_*(s)$
- Complexity  $O(mn^2)$  per iteration, for  $m$  actions and  $n$  states
- Could also apply to action-value function  $q_\pi(s, a)$  or  $q_*(s, a)$
- Complexity  $O(m^2n^2)$  per iteration