



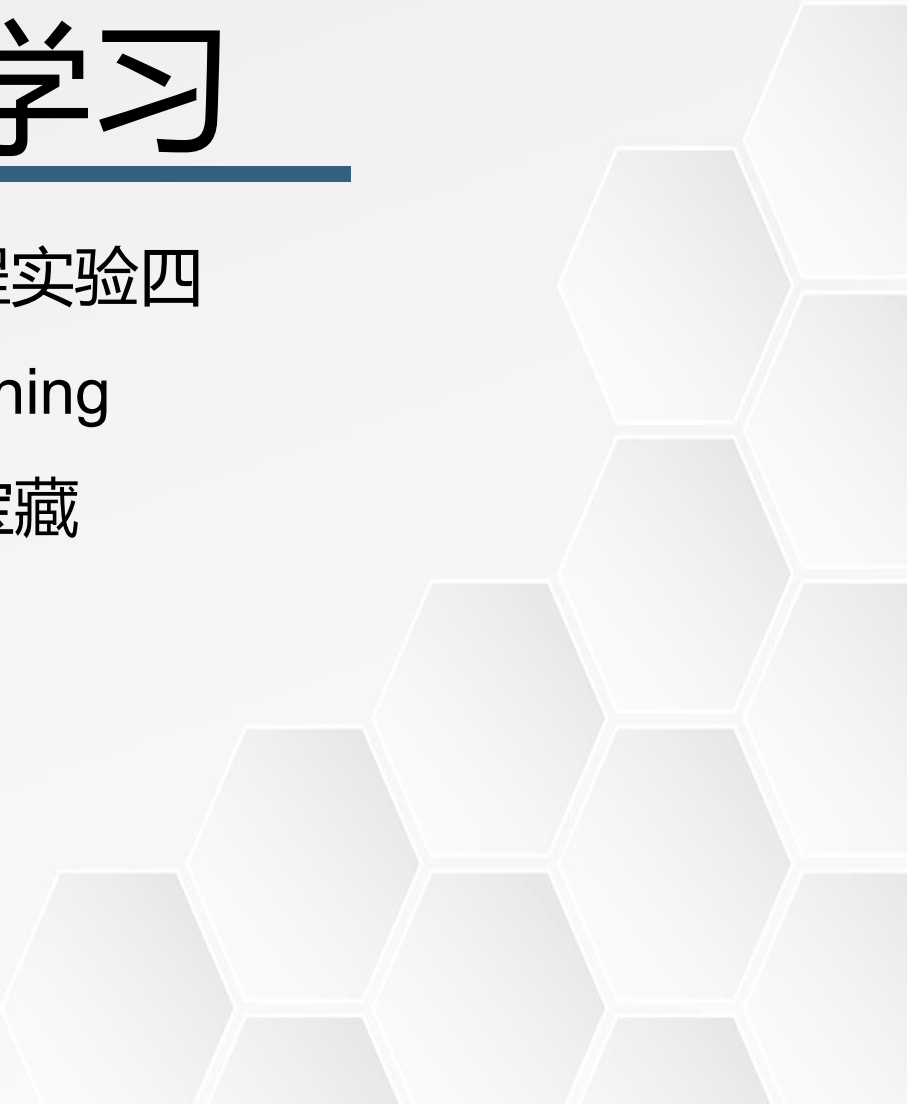
# 强化学习

---

Python编程实验四

Q-Learning

寻找宝藏





## ■ 实验理论知识

- 行为价值函数
- $\epsilon$  — 贪婪策略
- 现时策略蒙特卡罗控制
- 现时策略时序差分控制



## ■ 实验示例

### 寻找宝藏

智能体运用Q-Learning算法寻找宝藏。

O — — — — T

--o--T

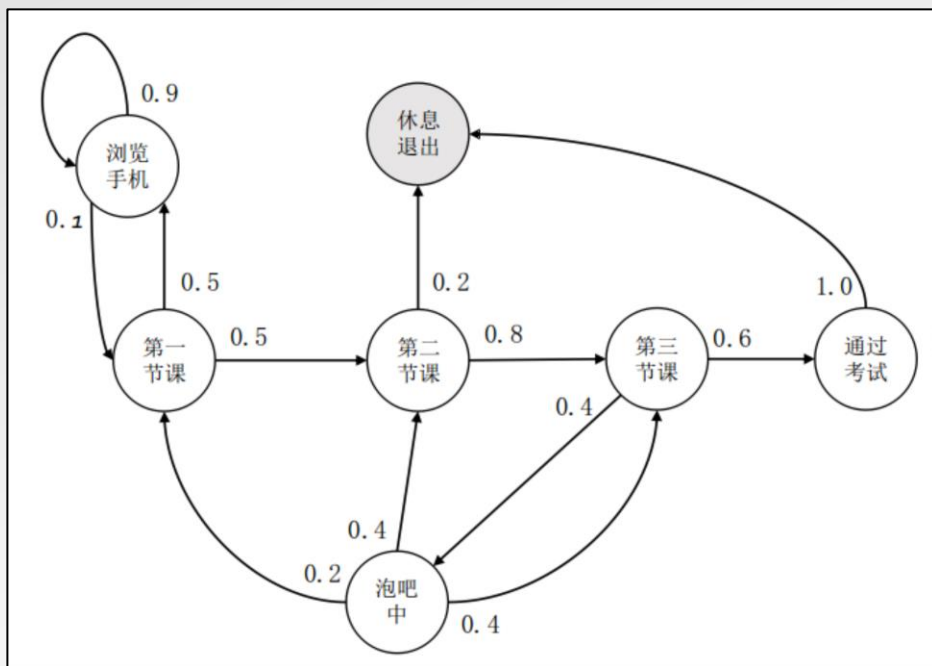
- ① O 表示智能体
- ② T 表示宝藏
- ③ — 表示路径



## ■ 行为价值函数

在**不基于模型**的控制时，我们将无法通过分析、比较基于状态的价值来改善贪婪策略，这是因为基于状态价值的贪婪策略的改善需要知晓状态间**转移概率**：

$$\pi'(s) = \underset{a \in A}{argmax} (R_s^a + P_{ss'}^a V(s'))$$



学生马尔可夫过程

基于**蒙特卡洛**的学习利用的是完整的状态序列，为了加快学习速度可以在只经历一个完整状态序列后就进行策略迭代；  
基于**时序差分**的学习，虽然学习速度可以更快，但要注意减少对事件估计的偏差。



## ■ $\epsilon$ — 贪婪策略

其基本思想就是保证能做到持续的探索，具体通过设置一个较小的  $\epsilon$  值，使用  $1 - \epsilon$  的概率贪婪地选择目前认为是**最大行为价值**的行为，而用  $\epsilon$  的概率随机的从**所有**  $m$  个可选行为中选择行为，即：

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{如果 } a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{其它情况} \end{cases}$$



## ■ 现时策略蒙特卡罗控制

现时策略蒙特卡罗控制通过  $\epsilon$ -贪婪策略采样一个或多个完整的状态序列后，平均得出某一状态行为对的价值，并持续进行策略的评估和改善。通常可以在仅得到一个完整状态序列后就进行一次策略迭代以加速迭代过程。

使用  $\epsilon$ -贪婪策略进行现时蒙特卡罗控制仍然只能得到基于该策略的近似行为价值函数，这是因为该策略一直在进行探索，没有一个终止条件。因此我们必须关注以下两个方面：一方面我们不想丢掉任何更好信息和状态，另一方面随着我们策略的改善我们最终希望能终止于某一个最优策略。为此引入了另一个理论概念：GLIE (greedy in the Limit with Infinite Exploration)。它包含两层意思，一是所有的状态行为对会被无限次探索：

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

二是另外随着采样趋向无穷多，策略收敛至一个贪婪策略：

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = 1 \left( a = \underset{a' \in A}{\operatorname{argmax}} Q_k(s, a') \right)$$

存在如下的定理：GLIE 蒙特卡洛控制能收敛至最优的状态行为价值函数：

$$Q(s, a) \rightarrow q^*(s, a)$$



## ■ 现时策略蒙特卡罗控制

如果在使用  $\epsilon$ -贪婪策略时，能令  $\epsilon$  随采样次数的无限增加而趋向于 0 就符合 GLIE。这样基于 GLIE 的蒙特卡洛控制流程如下：

1. 基于给定策略  $\pi$ ，采样第  $k$  个完整的状态序列： $\{S_1, A_1, R_2, \dots, S_T\}$
2. 对于该状态序列里出现的每一状态行为对  $(S_t, A_t)$ ，更新其计数  $N$  和行为价值函数  $Q$ ：

$$\begin{aligned} N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t)) \end{aligned}$$

3. 基于新的行为价值函数  $Q$  以如下方式改善策略：

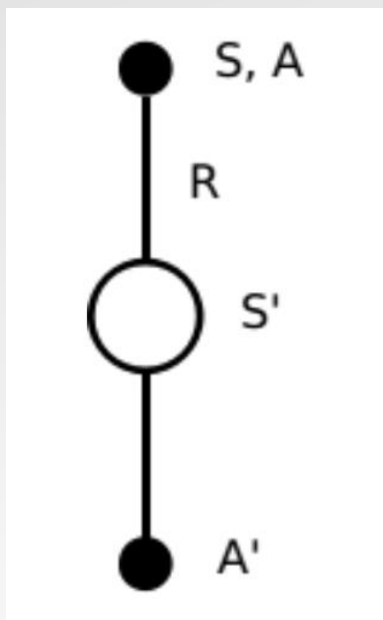
$$\begin{aligned} \epsilon &\leftarrow 1/k \\ \pi &\leftarrow \epsilon - greedy(Q) \end{aligned}$$

在实际应用中， $\epsilon$  的取值可不局限于取  $1/k$ ，只要符合 GLIE 特性的设计均可以收敛至最优策略(价值)。



## ■ 现时策略时序差分控制

针对一个状态  $S$ ，个体通过行为策略产生一个行为  $A$ ，执行该行为进而产生一个状态行为对  $(S, A)$ ，环境收到个体的行为后会告诉个体即时奖励  $R$  以及后续进入的状态  $S'$ ；个体在状态  $S'$  时遵循当前的行为策略产生一个新行为  $A'$ ，个体此时并不执行该行为，而是通过行为价值函数得到后一个状态行为对  $(S', A')$  的价值，利用这个新的价值和即时奖励  $R$  来更新前一个状态行为对  $(S, A)$  的价值。



$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

Sarsa 算法示意图





## ■ 现时策略时序差分控制

---

### 算法 1: Sarsa 算法

---

输入:  $episodes, \alpha, \gamma$

输出:  $Q$

initialize: set  $Q(s, a)$  arbitrarily, for each  $s$  in  $\mathbb{S}$  and  $a$  in  $\mathbb{A}(s)$ ; set  $Q(\text{terminal state}, \cdot) = 0$

repeat for each episode in episodes

    initialize:  $S \leftarrow$  first state of episode

$A = \text{policy}(Q, S)$  (e.g.  $\epsilon$ -greedy policy)

    repeat for each step of episode

$R, S' = \text{perform\_action}(S, A)$

$A' = \text{policy}(Q, S')$  (e.g.  $\epsilon$ -greedy policy)

$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$

$S \leftarrow S'; A \leftarrow A';$

    until  $S$  is terminal state;

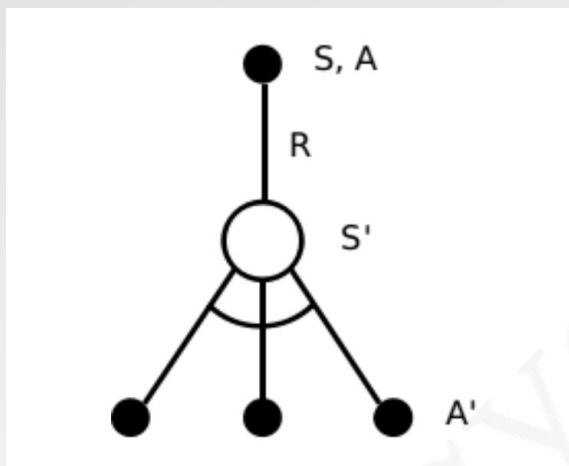
until all episodes are visited;

---



## ■ 借鉴策略Q学习算法

**现时策略**学习的特点就是产生实际行为的策略与更新价值（评价）所使用的策略是同一个策略，而**借鉴策略**学习（off-policy learning）中产生指导**自身行为的策略**  $\mu(a|s)$  与**评价策略**  $\pi(a|s)$  是不同的策略，具体地说，个体通过策略  $\mu(a|s)$  生成行为与环境发生实际交互，但是在更新这个状态行为对的价值时使用的是目标策略  $\pi(a|s)$ 。



Q 学习算法示意图

借鉴学习 TD 学习任务就是使用 TD 方法在目标策略  $\pi(a|s)$  的基础上更新行为价值，进而优化行为策略：

$$V(S_t) \leftarrow V(S_t) + \alpha \left( \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

借鉴策略 TD 学习中一个典型的行为策略  $\mu$  是基于行为价值函数  $Q(s, a)$   **$\epsilon$ -贪婪策略**，借鉴策略  $\pi$  则是基于  $Q(s, a)$  的完全**贪婪策略**，这种学习方法称为 Q 学习（Q learning）。



## ■ 现时策略时序差分控制

---

### 算法 3: Q 学习算法

---

输入:  $episodes, \alpha, \gamma$

输出:  $Q$

initialize: set  $Q(s, a)$  arbitrarily, for each  $s$  in  $\mathbb{S}$  and  $a$  in  $\mathbb{A}(s)$ ; set  $Q(\text{terminal state}, \cdot) = 0$

repeat for each episode in episodes

    initialize:  $S \leftarrow$  first state of episode

    repeat for each step of episode

$A = \text{policy}(Q, S)$  (e.g.  $\epsilon$ -greedy policy)

$R, S' = \text{perform\_action}(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$

$S \leftarrow S'$

    until  $S$  is terminal state;

until all episodes are visited;

---



## ■ Python代码实现

■ 更改维度

■ 增加障碍物

O — H — — T

O — — — — —  
— — — — — T

O — — — — —  
— — H — — T