第十六章 强化学习RL(Reinforcement Learning)

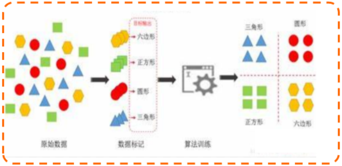
王星

中国人民大学统计学院

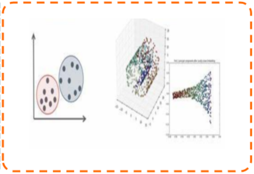
June 1, 2022

学习的分类

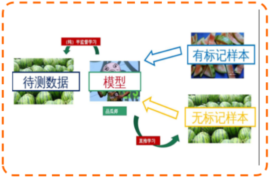
有指导学习



无指导学习



半指导学习

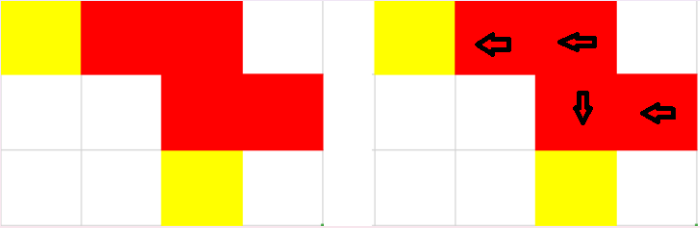


增强学习

大纲

- 强化学习简介
- K-摇臂赌博机（ϵ贪心+Softmax）
- 模型学习-马尔科夫决策过程
- 策略迭代
- 策略评估
- 时序差分
- Q值学习

场景：考虑一个格子游戏，每个格子等概率可以向4个方向移动，每次移动一步，收益为-1，移动到黄色出口结束游戏。若当前移动会导致出界，则移动后位置不变，要求最佳策略？



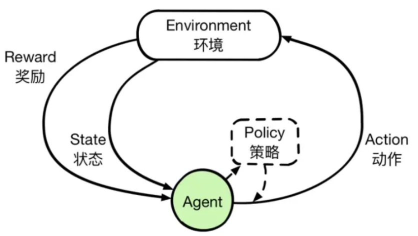
最终可以解出的策略如右图。

强化学习的定义2/4

- 在上述序列决策问题中，有一个智能体，智能体的行动和结果不确定，其行动具有随机性，有生存代价，也有停止条件。
- 智能体执行了某个动作后，环境将转换到一个新的状态，对于该新的状态环境会给出奖励信号（正奖励或者负奖励）。随后，智能体根据新的状态和环境反馈的奖励，按照一定的策略执行新的动作。智能体通过策略学习，可以知道自己在什么状态下，应该采取什么样的动作使得自身获得最大奖励.
- 定义：强化学习(Reinforcement Learning) 是机器学习中的一个领域，也称为增强学习，评价学习。
- 定义：强化学习是介于监督学习和非监督学习的另外一种学习方式。它的原理是从错误中学习，找到规律，不断逼近目标(Goal)
- 一个智能体(Agent)不断地在环境(Environment)的交互中学习，通过环境给予的反馈（奖励Reward）不断优化状态(State)- 动作(Action)完成特定目标（比如取得最大奖励目标）。

强化学习的基本元素3/4

- 特点1：强化学习关注智能体与环境之间的交互
- 特点2：反复实验(trial and error）和延迟奖励（delayed reward）是强化学习最重要的两个特征。
- 智能体:强化学习的本体，感知外界作为学习者或者决策者;
- 环境:强化学习智能体以外的一切，主要由状态集合组成;
- 状态:记作s_t，表示环境的数据，状态集则是环境中所有可能的状态;
- 动作:记作a_t，智能体可以做出的动作，动作集则是智能体可以做出的所有动作;



基本元素序4/4

- 状态转移:记作$p(s'|s,a)$，智能体根据当前状态s做出动作a 后，下一时刻环境处于不同状态s'的概率;
- 即时奖励:智能体在t时刻执行一个动作后，在$t+1$时刻获得的正/负反馈信号r_{t+1}，奖励集则是智能体可以获得的所有反馈信息，该奖励与前一时刻的动作、动作和动作之后下一时刻的状态，$r : S \times A \times S' \rightarrow R$;
- 策略:强化学习是从环境状态到动作的映射学习，称该映射关系为策略。通俗的理解，即智能体如何选择动作的思考过程称为策略。
- 目标:智能体自动寻找在连续时间序列里的最优策略.最优策略通常指最大化长期累积奖励V 如下：

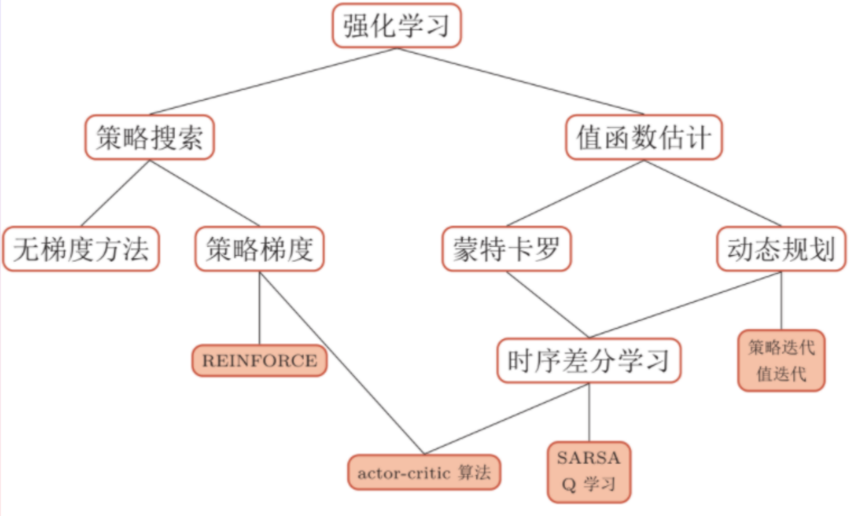
$$V = r_1 + \cdots + r_t$$

强化学习的分类1/2

强化学习可以分为两类：基于价值的学习；基于概率的学习

- 基于价值的学习：输出值的下一步要采取的各种动作的价值，根据价值最高的原则选择动作，经常应用于不连续的动作行为
  - 基于价值的方法（Value Based）：没有策略但是有价值函数:Q-leanring,Sarsa算法
  - 参与评价方法（Actor Critic）：既有策略也有价值函数
- 基于概率的学习：通过感官分析所处的环境，输出下一步要采取的各种动作的概率，根据概率选择行动，经常应用于连续的动作行为，常见的算法包括Policy Gradients算法等
  - 无模型的方法: 有策略和价值函数，没有模型，在无模型训练方法中，代理对环境一无所知，代理需要对环境的所有区域进行探索(explore)，意味着会花费大量时间去探索低奖励回报的区域，这样会浪费大量的时间去做无用功，但在大多数情况又不得不面对这样的情形。
  - 基于模型的方法：有策略和价值函数，也有模型；研发者知道环境中某些区域或者全部环境的情况下，研发者可将这些情况告知代理，就像一张地图，会告知代理哪些区域回报极低，避免探索低回报区域，加快训练速度，节约资源。

强化学习的分类



强化学习
策略搜索
值函数估计
无梯度方法
策略梯度
蒙特卡罗
动态规划
REINFORCE
时序差分学习
策略迭代
值迭代
actor-critic 算法
SARSA
Q 学习

- 与一般监督学习不同，强化学习任务的最终奖赏是在多步动作之后才能观察到，这里如果仅仅考虑最大化单步奖赏最佳策略依赖于两方面的信息：
  - 需知道每个动作带来的奖赏；
  - 获得奖赏最大的动作；如每个动作对应的奖赏是一个确定值，那么尝试一遍所有的动作就能确定奖赏最大的动作。然而现实中一个动作的奖赏值是来自于一个概率分布，仅通过一次尝试并不能确切地获得平均奖赏值，这就需要探索。
- 一方面，为了从环境中获取尽可能多的知识，需要让Agent进行探索，另一方面，为获得较大的奖励，还需要让agent 对已知的信息加以利用。
- 强化学习所面临的"探索-利用窘境" (Exploration-Exploitation dilemma)。显然，欲累积奖赏最大，则必须在探索与利用之间达成较好的折中。强化学习问题中存在的一个重要挑战即是如何权衡探索-利用之间的关系。这也是得到最优策略的算法纷纷努力的主要内容。

- $\epsilon-$贪心算法基于一个概率对探索和利用进行折中：每次尝试时，以ϵ概率进行探索，即以均匀概率随机选取一个摇臂；以$1-\epsilon$的概率进行利用，即选择当前平均奖赏最高的摇臂(若有多个，则随机选取一个)。

$$\pi^{\epsilon}(s)=\begin{cases}\pi(s), & \text{按概率}1-\epsilon,\\ \text{随机选择}\mathcal{A}\text{中的动作}, & \text{按概率}\epsilon.\end{cases}$$

ϵ贪心

- 若摇臂奖赏不确定较大，则需要探索的时间比较长，需要较大的ϵ值；若摇臂奖赏确定性较集中，则需要探索时间较短，需要较小的ϵ值。$\epsilon = \frac{1}{\sqrt{T}}$

输入: 摇臂数 K;
奖赏函数 R;
尝试次数 T;
探索概率 ϵ.

过程:

$r = 0$;
$\forall i = 1, 2, \ldots K: \; Q(i) = 0, \; \mathrm{count}(i) = 0$;
for $t = 1, 2, \ldots, T$ **do**
  if $\mathrm{rand}() < \epsilon$ **then**
    $k =$ 从 $1, 2, \ldots, K$ 中以均匀分布随机选取
  else
    $k = \arg\max_i Q(i)$
  end if
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times \mathrm{count}(k) + v}{\mathrm{count}(k) + 1}$;
  $\mathrm{count}(k) = \mathrm{count}(k) + 1$;
end for

输出: 累积奖赏 r

Softmax3/4

- 若各摇臂的平均奖赏相当，选取各摇臂的概率也一样；若某些摇臂的平均奖赏明显高于其他摇臂，则被选中的概率也明显更高。
- Sotfmax 算法中摇臂概率的分配是基于Boltzmann 分布:
- 其中，$Q(i)$记录当前摇臂的平均奖赏；$\tau > 0$称为"温度"，τ越小平均奖赏高的摇臂被选取的概率越高；τ越趋于0 时，Softmax 将趋于"仅利用"；τ趋于无穷大时，Softmax 则趋于"仅探索"。

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}}$$

输入: 摇臂数 K;
奖赏函数 R;
尝试次数 T;
温度参数 τ.

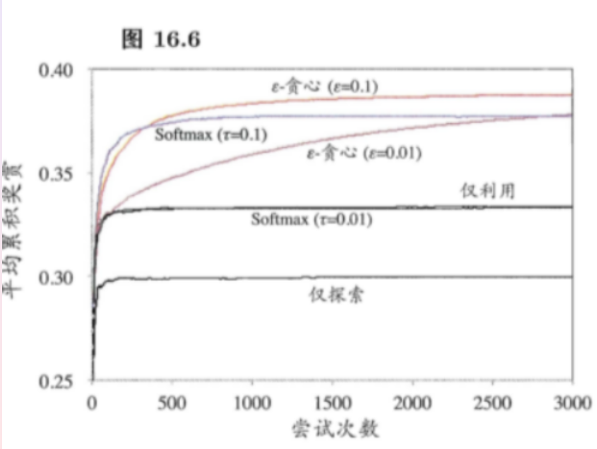
过程:

$r = 0$;
$\forall i = 1, 2, \ldots K: \ Q(i) = 0, \ \mathrm{count}(i) = 0$;
for $t = 1, 2, \ldots, T$ **do**
  $k =$ 从$1, 2, \ldots, K$ 中根据式(16.4)随机选取
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times \mathrm{count}(k) + v}{\mathrm{count}(k) + 1}$;
  $\mathrm{count}(k) = \mathrm{count}(k) + 1$;
end for

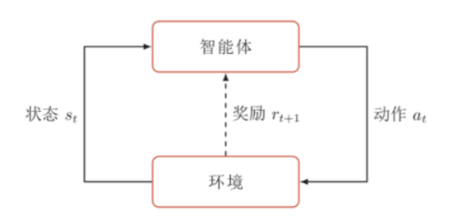
即根据Softmax函数

输出: 累积奖赏 r

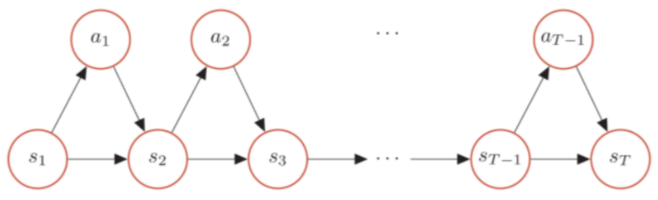
- 离散状态空间，离散动作空间上多步强化学习任务可以将每个步骤看成是一个$K-$摇臂赌博机，这样做事比较简单的一种方式，它没有考虑马尔科夫决策过程的结构



马尔科夫决策过程（Markov Decision Process，MDP）适用于一种环境完全可观测的强化学习环境，即马尔科夫性质，结果部分随机、部分在决策者的控制下的决策过程建模的数学框架。在智能体在与环境进行交互的过程中，根据经验调整策略最优化决策序列的过程。考虑多步学习四元组$E = \{S, A, P, R\}$均为已知.



$$p(s_{t+1}|s_t, a_t, \cdots, s_0, a_0) = p(s_{t+1}|s_t, a_t);$$



- 马尔可夫决策过程是一条“轨迹”(trajectory)
  $\tau = s_0 \to a_0 \to s_1 \to r_1 \to a_1 \to \cdots, a_{t-1} \to s_t \to r_t, \cdots$
- 策略$\pi(a|s)$

$$p(\tau) = p(s_0, a_0, s_1, a_1, \cdots)$$

$$= p(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

有模型的学习:策略评估

在模型已知的前提下，可以对任意策略的进行评估。一般常使用以下两种值函数来评估某个策略的优劣：

- 状态值函数（V）：从状态s出发，使用策略π所带来的累积奖赏；
- 状态动作值函数（Q）:从状态s 出发，执行动作a后再使用π策略所带来的累积奖赏。

根据累积奖赏的定义，可以引入T步累积奖赏与γ折扣累积奖赏：

- 期望累积奖赏函数
  - T步累积奖赏: $V_T(s) = E[\frac{1}{T}\sum_{t=1}^{T} r_t | s_0 = x_0]$;
  - γ折累积奖赏:$V_\gamma(s) = E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = x_0]$
- 状态值函数：
  - T步累积奖赏: $Q_T(s) = E[\frac{1}{T}\sum_{t=1}^{T} r_t | s_0 = x_0 a_0 = a]$;
  - γ折累积奖赏:$Q_\gamma(s) = E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = x_0, a_0 = a]$

有模型学习的策略改进

对某个策略的积累奖赏进行评估后发现非最优，改进最优状态动作值函数：

由于最优值函数的累积奖赏值已达最大，因此可对前面的 Bellman 等式(16.7)和(16.8)做一个改动，即将对动作的求和改为取最优：

$$\begin{cases} V_T^*(x) = \max\limits_{a\in A} \sum\limits_{x'\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \frac{T-1}{T} V_{T-1}^*(x')\right); \\ V_\gamma^*(x) = \max\limits_{a\in A} \sum\limits_{x'\in X} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma V_\gamma^*(x')\right). \end{cases} \tag{16.13}$$

换言之，

$$V^*(x) = \max_{a\in A} Q^{\pi^*}(x,a). \tag{16.14}$$

代入式(16.10)可得最优状态-动作值函数

$$\begin{cases} Q_T^*(x,a) = \sum\limits_{x'\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \frac{T-1}{T} \max\limits_{a'\in A} Q_{T-1}^*(x',a')\right); \\ Q_\gamma^*(x,a) = \sum\limits_{x'\in X} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma \max\limits_{a'\in A} Q_\gamma^*(x',a')\right). \end{cases} \tag{16.15}$$

上述关于最优值函数的等式，称为最优 Bellman 等式

有模型学习的策略迭代：T步奖赏累积策略迭代

- 策略迭代：不断迭代进行策略评估和改进知道策略收敛不在改变位置

图 16.8

输入： MDP 四元组 $E = \langle X, A, P, R \rangle$;
　　累积奖赏参数 T.

过程：

$\forall x \in X:\ V(x) = 0,\ \pi(x,a) = \frac{1}{|A(x)|}$;
loop
　for $t = 1, 2, \ldots$ **do**
　　$\forall x \in X: V'(x) = \sum_{a \in A} \pi(x,a) \sum_{x' \in X} P^a_{x \to x'} \left(\frac{1}{t} R^a_{x \to x'} + \frac{t-1}{t} V(x')\right)$;
　　if $t = T + 1$ **then**
　　　break
　　else
　　　$V = V'$
　　end if
　end for
　$\forall x \in X: \pi'(x) = \arg\max_{a \in A} Q(x,a)$;
　if $\forall x: \pi'(x) = \pi(x)$ **then**
　　break
　else
　　$\pi = \pi'$
　end if
end loop

输出： 最优策略 π

有模型学习的值迭代：T步奖赏累积奖赏值迭代

- 值迭代：减少耗时

图 16.9

输入： MDP 四元组 $E = \langle X, A, P, R \rangle$;
累积奖赏参数 T;
收敛阈值 θ .

过程:

```
∀x ∈ X : V(x) = 0;
for t = 1, 2, ... do
    ∀x ∈ X : V'(x) = max_{a∈A} Σ_{x'∈X} P^a_{x→x'} ( (1/t) R^a_{x→x'} + ((t−1)/t) V(x') );
    if max_{x∈X} |V(x) − V'(x)| < θ then
        break
    else
        V = V'
    end if
end for
```

$\forall x \in X : V(x) = 0$;
for $t = 1, 2, \ldots$ **do**
  $\forall x \in X : V'(x) = \max_{a \in A} \sum_{x' \in X} P^a_{x \to x'} \left(\frac{1}{t} R^a_{x \to x'} + \frac{t-1}{t} V(x') \right)$;
  if $\max_{x \in X} |V(x) - V'(x)| < \theta$ **then**
    break
  else
    $V = V'$
  end if
end for

输出： 策略 $\pi(x) = \arg\max_{a \in A} Q(x, a)$

免模型的学习-时序差分学习

时序差分(Temporal Difference，简称TD) 学习结合了动态规划与蒙特卡罗方法的思想，能做到更高效的免模型学习。对于状态-动作(x,a)不妨假设基于t个采样已估计出值函数$Q_t^\pi(x,a)=\frac{1}{t}\sum_{i=1}^{t} r_i$,在得到第$t+1$个采样$r_{t+1}$时，有

$$Q_{t+1}^\pi(x,a)=Q_t^\pi(x,a)+\frac{1}{t+1}\left(r_{t+1}-Q_t^\pi(x,a)\right).$$

SARSA算法(State Action Reward State Action,SARSA)

输入: 环境 E;
动作空间 A;
起始状态 x_0;
奖赏折扣 γ;
更新步长 α.

过程:

$Q(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;
$x=x_0,\ a=\pi(x)$;
for $t=1,2,\ldots$ **do**
  $r,x'=$ 在 E 中执行动作 a 产生的奖赏与转移的状态;
  $a'=\pi^\epsilon(x')$;
  $Q(x,a)=Q(x,a)+\alpha\left(r+\gamma Q(x',a')-Q(x,a)\right)$;
  $\pi(x)=\arg\max_{a''} Q(x,a'')$;
  $x=x', a=a'$
end for

输出: 策略 π

图 16.12 Sarsa 算法

无模型的学习-Q-学习

输入: 环境 E;
动作空间 A;
起始状态 x_0;
奖赏折扣 γ;
更新步长 α.

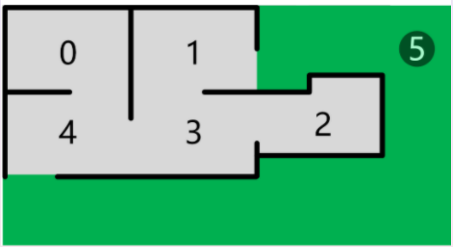
过程:

```
Q(x,a) = 0, π(x,a) = 1/|A(x)|;
x = x_0;
for t = 1,2,... do
    r, x' = 在 E 中执行动作 π^ε(x) 产生的奖赏与转移的状态;
    a' = π(x');
    Q(x,a) = Q(x,a) + α(r + γQ(x',a') − Q(x,a));
    π(x) = arg max_{a''} Q(x,a'');
    x = x', a = a'
end for
```

输出: 策略 π

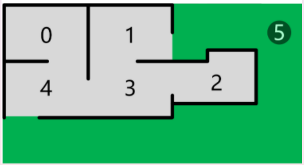
图 16.13 Q-学习算法

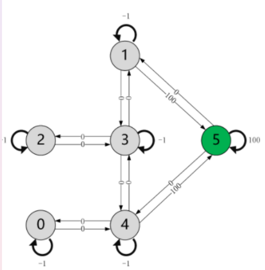
- 问题:大楼有一组房间，每个房间编号为$0\sim4$，有的房间之间可互通，有的不能互通，5号区域为楼外。如下图所示：



- 目标:将Agent放在任何房间，它自己能够从目标房间5号走出大楼。

- 构造连通图，并约定：能够直接到达5号区域的边，其奖励值为100（包括5号自己到达5号）；房间之间彼此连通的边，其奖励值为0；房间之间彼此阻隔的边，其奖励值为-1；房间自己到达自己的边，其奖励值为-1。根据这样的规则，构造如下连通图（对于互相阻隔的房间，它们之间的边被省略了）：





- 定义state和action。约定将代理当前所在的区域编号作为state，将代理前往哪一个区域的编号作为action。

Q-学习3/8

建立两张表：根据连通图中的奖励值，构造奖励矩阵R，如左图所示；再创建一张Q-table，对其进行初始化，将内部所有元素置为0，如右图所示：

$$
\begin{array}{cc}
 & \textit{action} \\
\textit{state} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
R= \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & \textit{action} \\
\textit{state} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
Q= \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

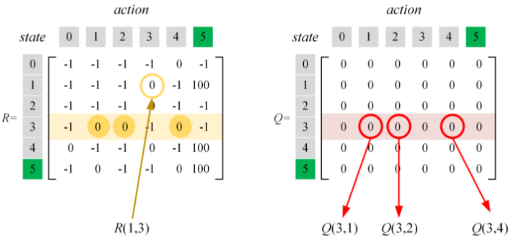
后续对Q值进行更新(固定$\gamma = 0.8$)：

$$Q(s_t, a_t) \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a)$$

假定代理当前位于1号房间，查询奖励矩阵R，它可到达区域3和区域5，随机选择一个动作“前往区域3”，通过查询Q-table来更新Q值,具体更新的是元素$Q(1,3)$，此时，需要查表获得两类数据：

Q-学习4/8

- 奖励（Reward）。指“从房间1（当前state）前往房间3（采取的action）所获得的即时奖励”，因此需要查询的数据为R(1,3)。
- 未来的价值（Value）。指“进入房间3后（next state）所能获得的最大即时奖励”，需要查询的数据为：$Q(3,1), Q(3,2)$和$Q(3,4)$。



- Q值的更新算式

$$Q(1,3) \leftarrow R(1,3) + 0.8 \cdot \max(Q(3,1), Q(3,2), Q(3,4))$$

$$\leftarrow 0 + 0.8 \cdot (0,0,0) = 0$$

Q-学习5/8

- 选择另一个动作“前往区域5”，通过查询Q-table更新Q值,具体更新的是元素$Q(1,5)$，此时，需要查表获得两类数据：$R(1,5)Q(5,1),Q(5,4),Q(5,5)$

$$Q(1,5) \leftarrow R(1,5) + 0.8 \cdot \max(Q(5,1), Q(5,4), Q(5,5))$$

$$\leftarrow 100 + 0.8 \cdot (0,0,0) = 100$$

$$\begin{array}{c} \\ \\ \\ Q= \end{array}
\begin{array}{c} \textit{state} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \textit{action} \quad \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{100} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{c} \\ \\ \\ Q= \end{array}
\begin{array}{c} \textit{state} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \textit{action} \quad \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51.2 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-学习6/8

- 接下来按照上述思路不断进行迭代，最终得到Q-table,在得到了Q-table后，可以根据该矩阵令代理独自行走寻找路径了。基本想法是查找当前位置具有最大Q值的action，然后执行，如此反复，直到到达目的地为止。
- 初始化*s*，如果不是出口，循环选择Q最大的方向前进，更新s直到结束。
- 具体代码如下：

```python
import numpy as np
import random

# Build Q-table
q = np.zeros((6, 6))
q = np.matrix(q)

# Build R-table
r = np.array([[-1, -1, -1, -1, 0, -1], [-1, -1, -1, 0, -1, 100], [-1, -1, -1, 0, -1, -1], [-1, 0, 0, -1, 0, -1],
              [0, -1, -1, 0, -1, 100], [-1, 0, -1, -1, 0, 100]])
r = np.matrix(r)

# Discount factor
gamma = 0.8

# Training
for i in range(1000):
    # For each training episode, randomly select a state
    state = random.randint(0, 5)
    while state != 5:
        # Select the action of a non-negative value in the R-table
        r_pos_action = []
        for action in range(6):
            if r[state, action] >= 0:
                r_pos_action.append(action)
        next_state = r_pos_action[random.randint(0, len(r_pos_action) - 1)]
        q[state, next_state] = r[state, next_state] + gamma * q[next_state].max()
        state = next_state

print('Finally, the content of Q-table is:\n')
print(q)
```

Q-学习7/8代码实现

```python
# verification
for i in range(10):
    print("\n{}-th verification:".format(i + 1))
    state = random.randint(0, 5)
    print('The robot is at {}'.format(state))
    count = 0
    while state != 5:
        if count > 20:
            print('fail')
            break
        # Choose the largest q_max
        q_max = q[state].max()

        q_max_action = []
        for action in range(6):
            if q[state, action] == q_max:
                q_max_action.append(action)

        next_state = q_max_action[random.randint(0, len(q_max_action) - 1)]
        print("the robot goes to " + str(next_state) + '.')
        state = next_state
        count += 1
```

运行过程中，会输出最终Q-table中的数据，如下图所示；部分验证阶段的输出，对照房间平面图可以发现，代理成功地通过了测试。

```
Finally, the content of Q-table is:

[[   0.     0.     0.     0.    80.     0. ]
 [   0.     0.     0.    64.     0.   100. ]
 [   0.     0.     0.    64.     0.     0. ]
 [   0.    80.    51.2    0.    80.     0. ]
 [  64.     0.     0.    64.     0.   100. ]
 [   0.     0.     0.     0.     0.     0. ]]
```

```
1-th verification:
The robot is at 2
the robot goes to 3.
the robot goes to 1.
the robot goes to 5.

2-th verification:
The robot is at 1
the robot goes to 5.

3-th verification:
The robot is at 1
the robot goes to 5.

4-th verification:
The robot is at 3
the robot goes to 1.
the robot goes to 5.

5-th verification:
The robot is at 3
the robot goes to 1.
the robot goes to 5.

6-th verification:
The robot is at 2
the robot goes to 3.
the robot goes to 4.
the robot goes to 5.

7-th verification:
The robot is at 3
the robot goes to 4.
the robot goes to 5.

8-th verification:
The robot is at 1
the robot goes to 5.
```