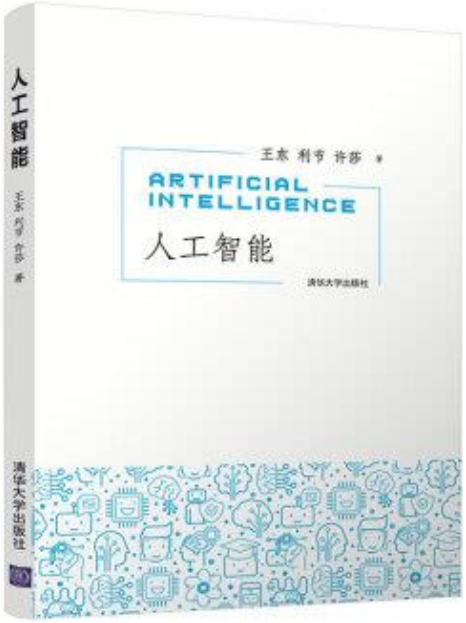
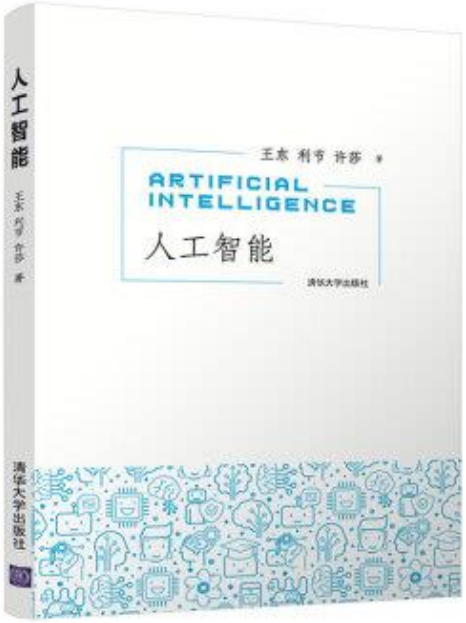


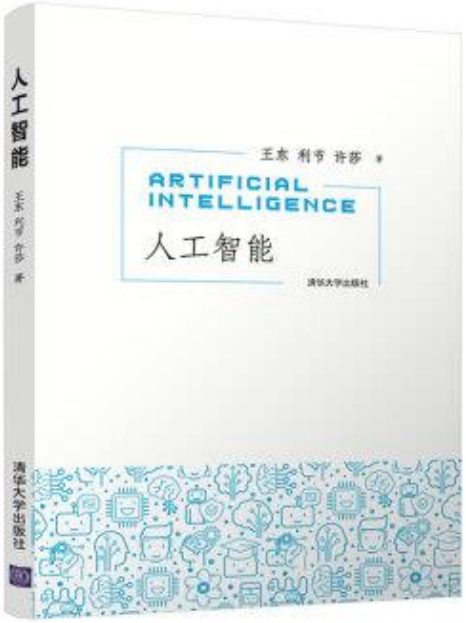
模仿你的行为

利节



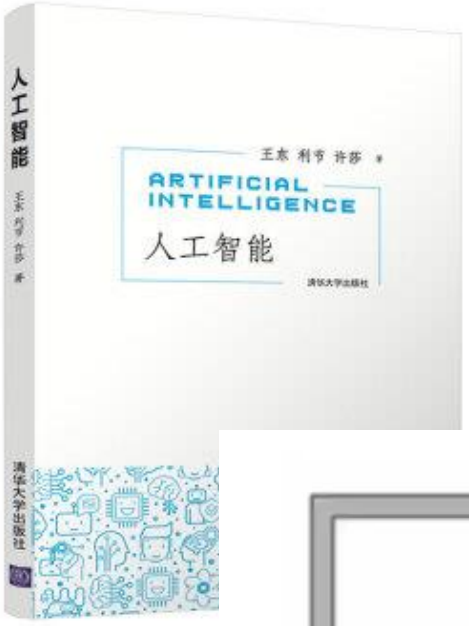
目录

- Q Learning
- Deep Q Learning

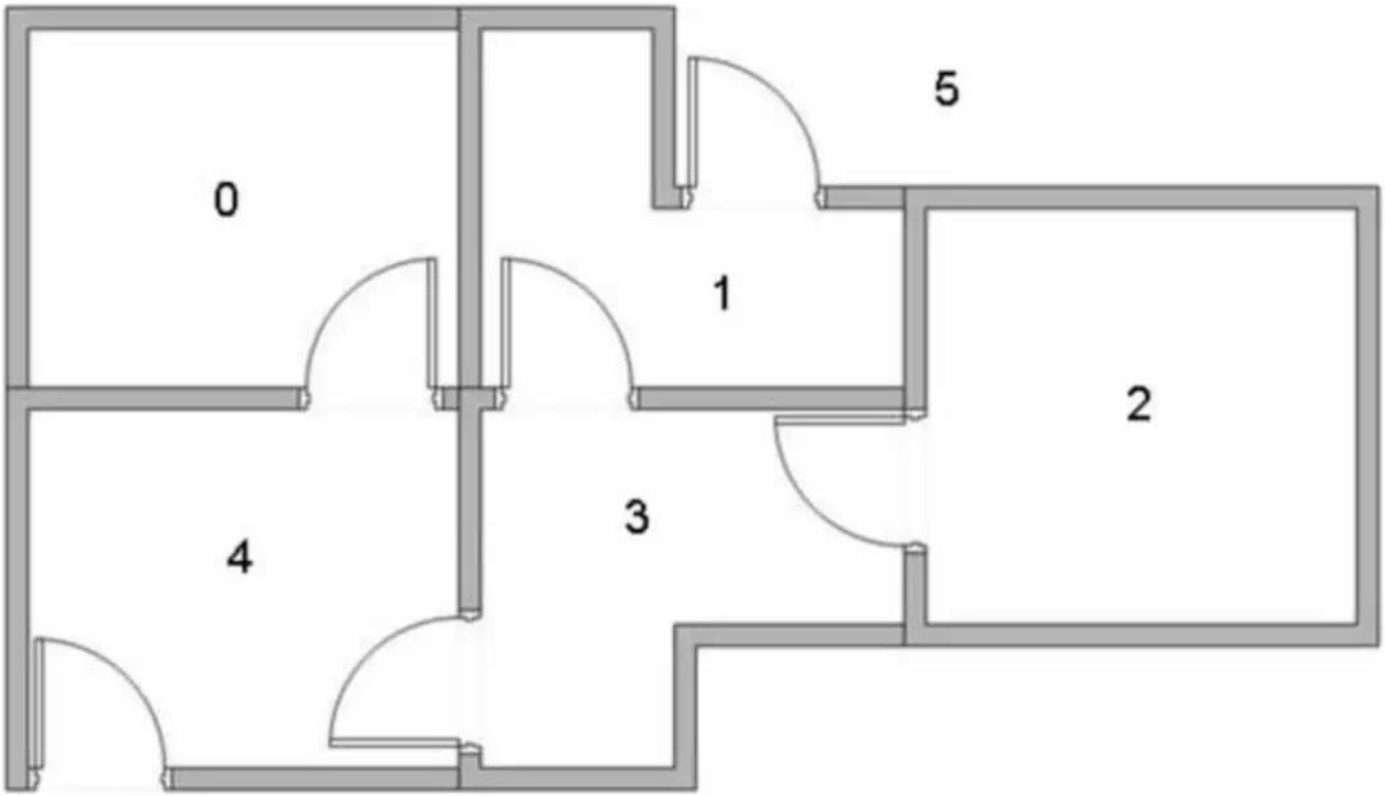


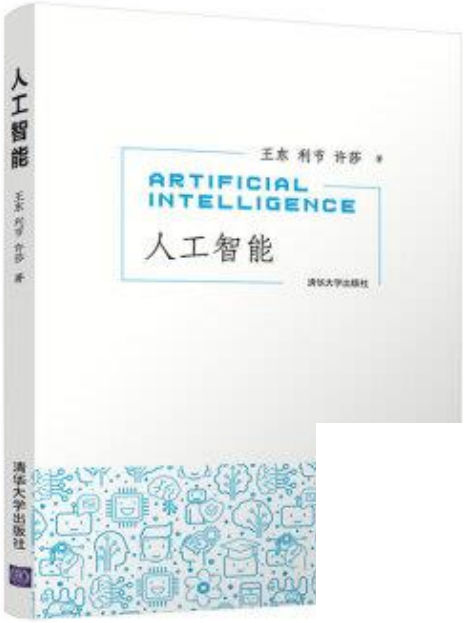
目录

- Q Learning
- Deep Q Learning

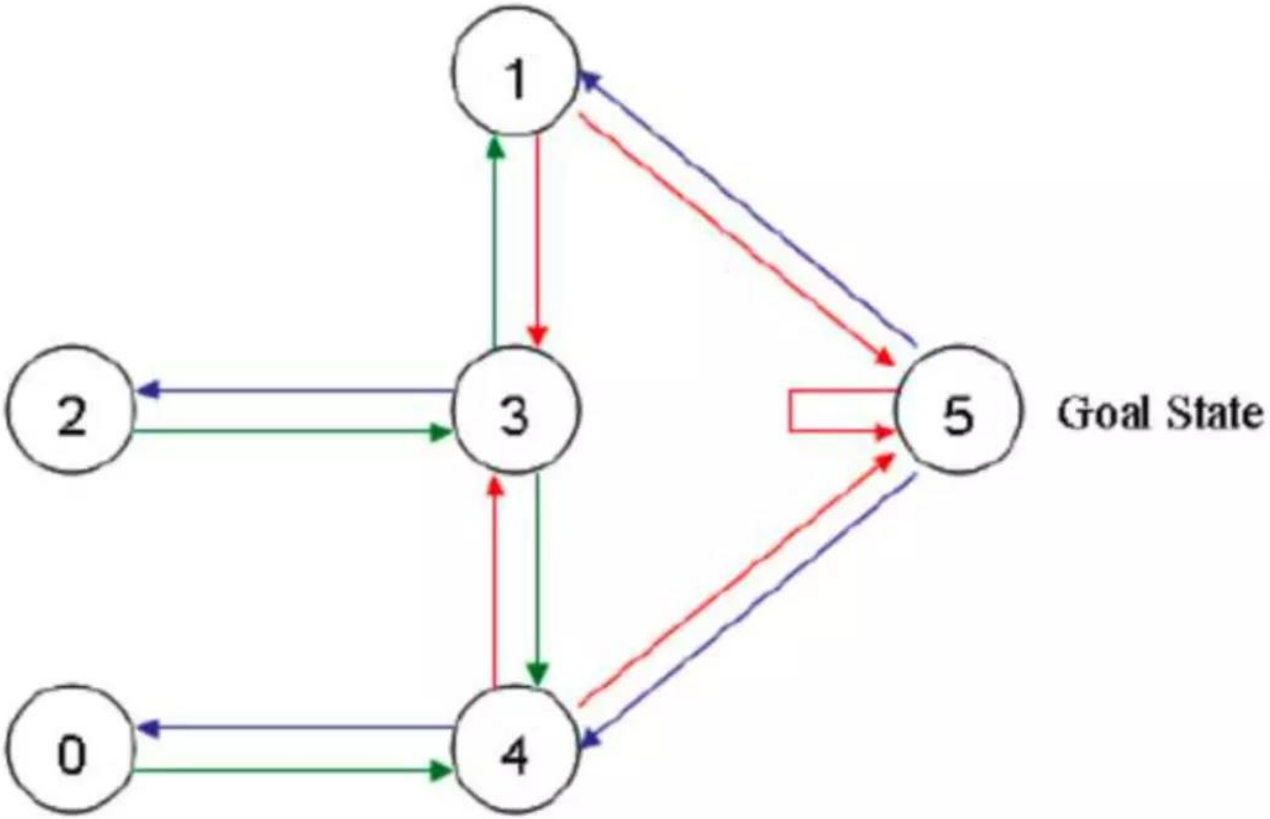

人工智能
王东 利节 许莎 著
ARTIFICIAL INTELLIGENCE
人工智能
清华大学出版社

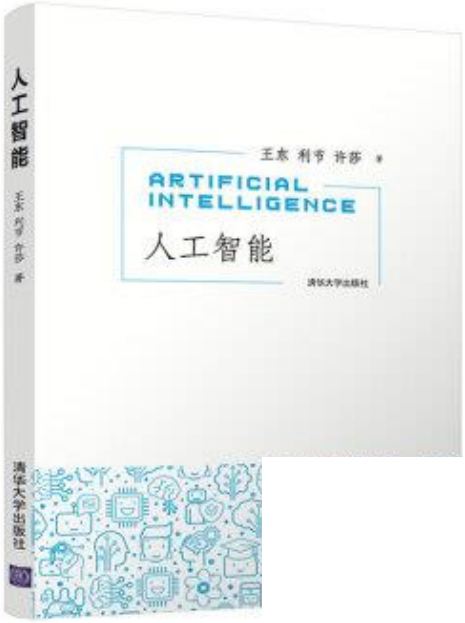
假设有这样的房间：


0
1
2
3
4
5

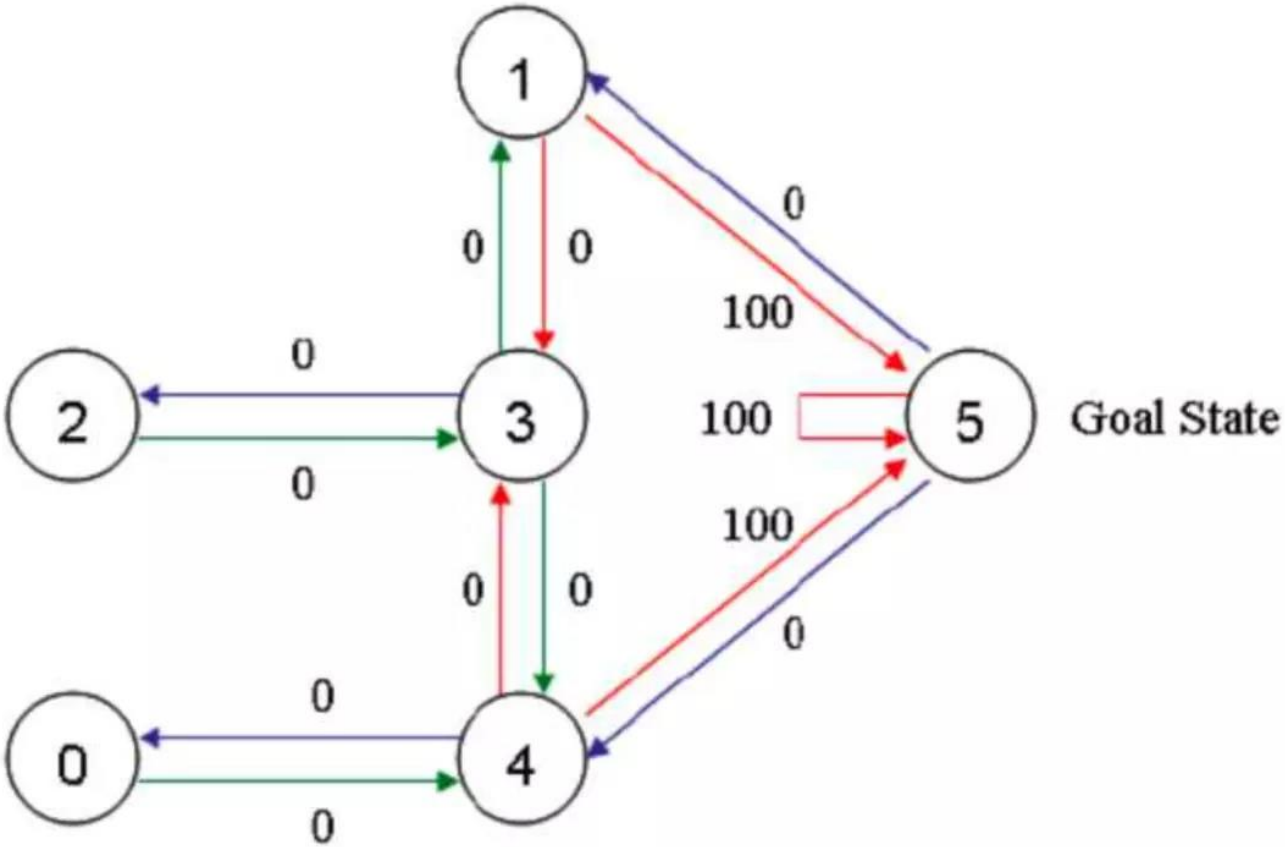


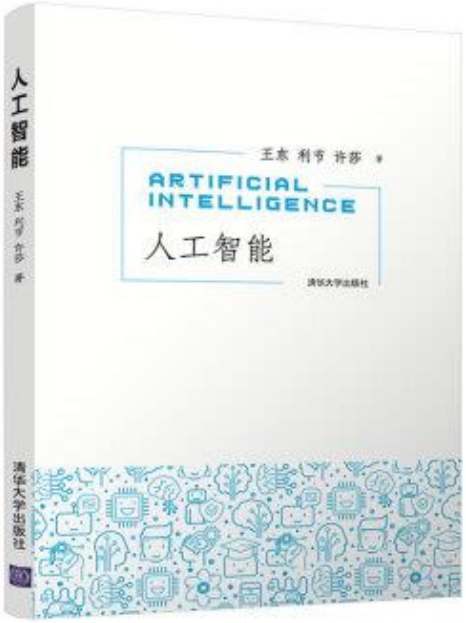
如果将房间表示成点，然后用房间之间的连通关系表示成线，如下图所示：





首先将agent（机器人）处于任何一个位置，让他自己走动，直到走到5房间，表示成功。为了能够走出去，我们将每个节点之间设置一定的权重，能够直接到达5的边设置为100，其他不能的设置为0，这样网络的图为：

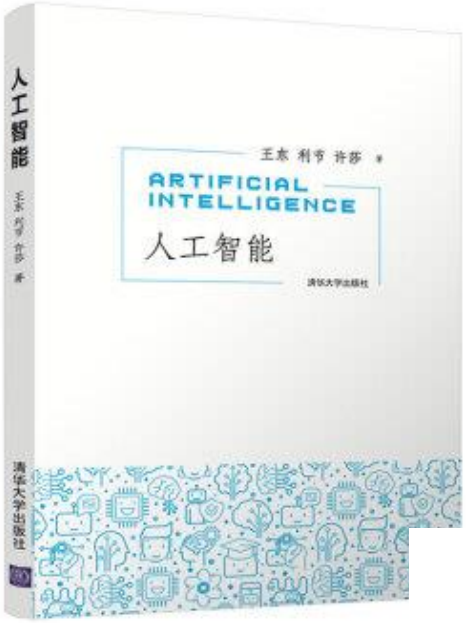




Q Learning中，最重要的就是“状态”和“动作”，状态表示处于图中的哪个节点，比如2节点，3节点等等，而动作则表示从一个节点到另一个节点的操作。

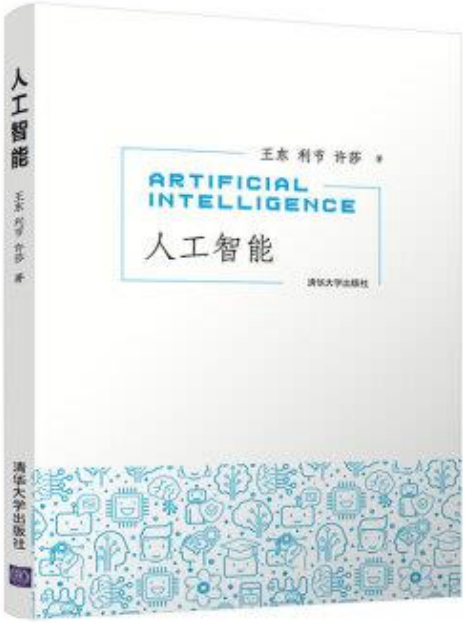
首先我们生成一个奖励矩阵矩阵，矩阵中，-1表示不可以通过，0表示可以通过，100表示直接到达终点：

$$
R = \begin{array}{c|cccccc}
 & & & \text{Action} & & & \\
\text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & -1 & -1 & -1 & -1 & 0 & -1 \\
1 & -1 & -1 & -1 & 0 & -1 & 100 \\
2 & -1 & -1 & -1 & 0 & -1 & -1 \\
3 & -1 & 0 & 0 & -1 & 0 & -1 \\
4 & 0 & -1 & -1 & 0 & -1 & 100 \\
5 & -1 & 0 & -1 & -1 & 0 & 100
\end{array}
$$



同时，我们创建一个Q表，表示学习到的经验，与R表同阶，初始化为0矩阵，表示从一个state到另一个state能获得的总的奖励的折现值。

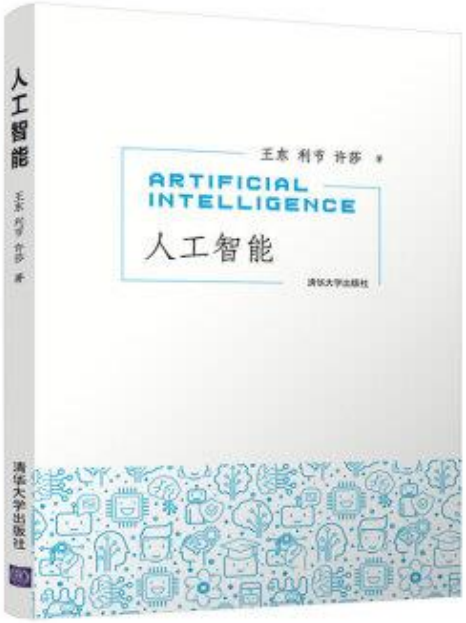
$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$



Q表中的值根据如下的公式来进行更新：

$$Q(s,a) = R(s,a) + \gamma \cdot \max_{\widetilde{a}}\{Q(\widetilde{s},\widetilde{a})\}$$

在上面的公式中，S表示当前的状态，a表示当前的动作，s表示下一个状态，a表示下一个动作，λ为贪婪因子，0<λ<1,一般设置为0.8。Q表示的是，在状态s下采取动作a能够获得的期望最大收益，R是立即获得的收益，而未来一期的收益则取决于下一阶段的动作。



所以，Q-learning的学习步骤可以归结为如下：

算法 1.1 *(Q-learning 算法)*

Step 1 给定参数 γ 和 *reward* 矩阵 R.

Step 2 令 $Q := \mathbf{0}$.

Step 3 *For each episode:*

3.1 随机选择一个初始的状态 s.

3.2 若未达到目标状态，则执行以下几步

(1) 在当前状态 s 的所有可能行为中选取一个行为 a.

(2) 利用选定的行为 a，得到下一个状态 $\widetilde{s}$.

(3) 按照 *(1.1)* 计算 $Q(s,a)$.

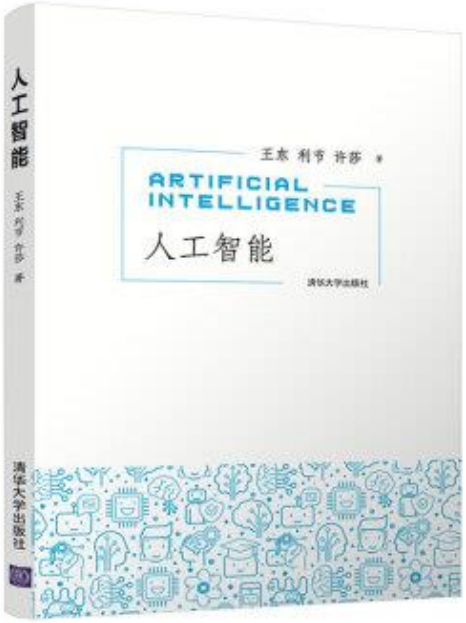
(4) 令 $s := \widetilde{s}$.

在迭代到收敛之后，我们就可以根据Q-learning来选择我们的路径走出房间。

看一个实际的例子，首先设定λ=0.8，奖励矩阵R和Q矩阵分别初始化为：

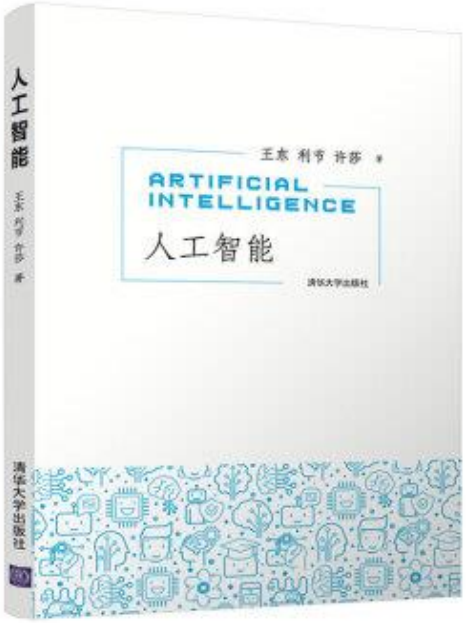
$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



随机选择一个状态，比如1，查看状态1所对应的R表，也就是1可以到达3或5，随机地，我们选择5，根据转移方程：

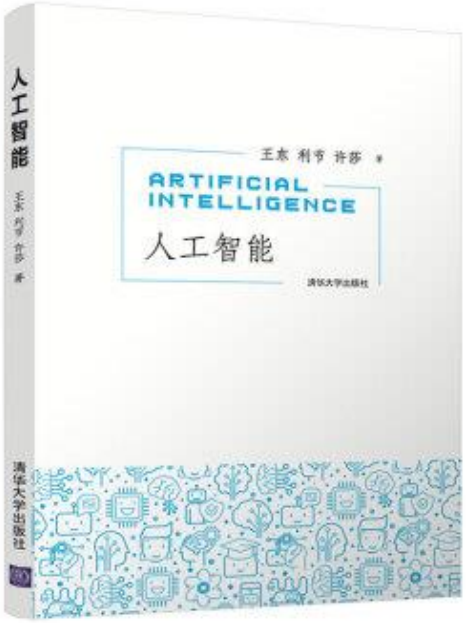
$$
\begin{aligned}
Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\
&= 100 + 0.8 * \max\{0,0,0\} \\
&= 100.
\end{aligned}
$$



于是，Q表为：

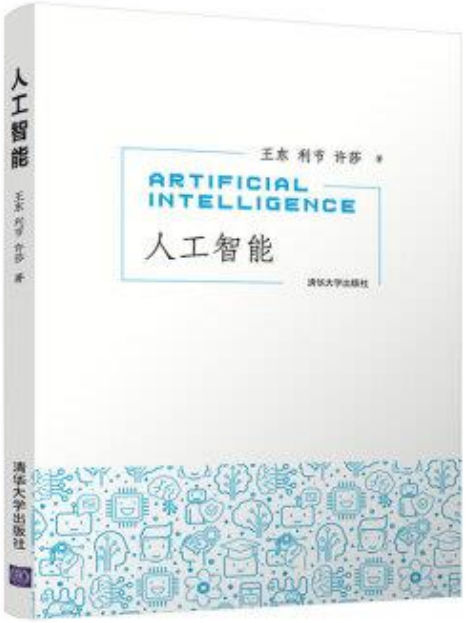
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0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

这样，到达目标，一次尝试结束。



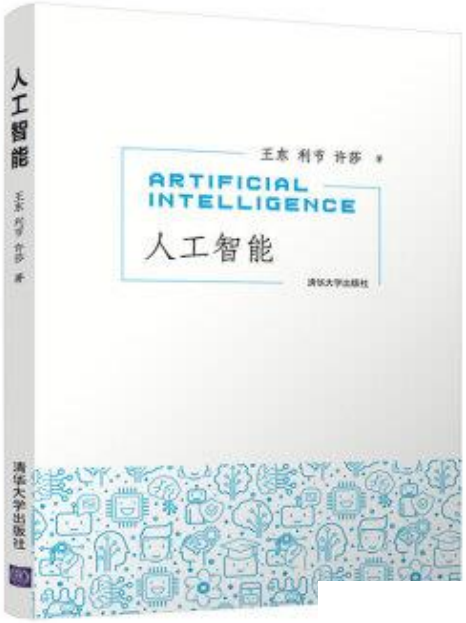
接下来再选择一个随机状态，比如3，3对应的下一个状态有（1，2，4都是状态3对应的非负状态），随机地，我们选择1，这样根据算法更新：

$$
\begin{aligned}
Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\
&= 0 + 0.8 * \max\{0, 100\} \\
&= 80.
\end{aligned}
$$



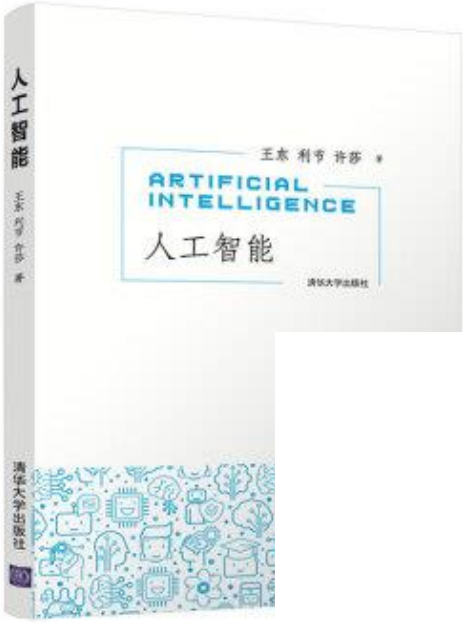
这样，Q表为：

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 80 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

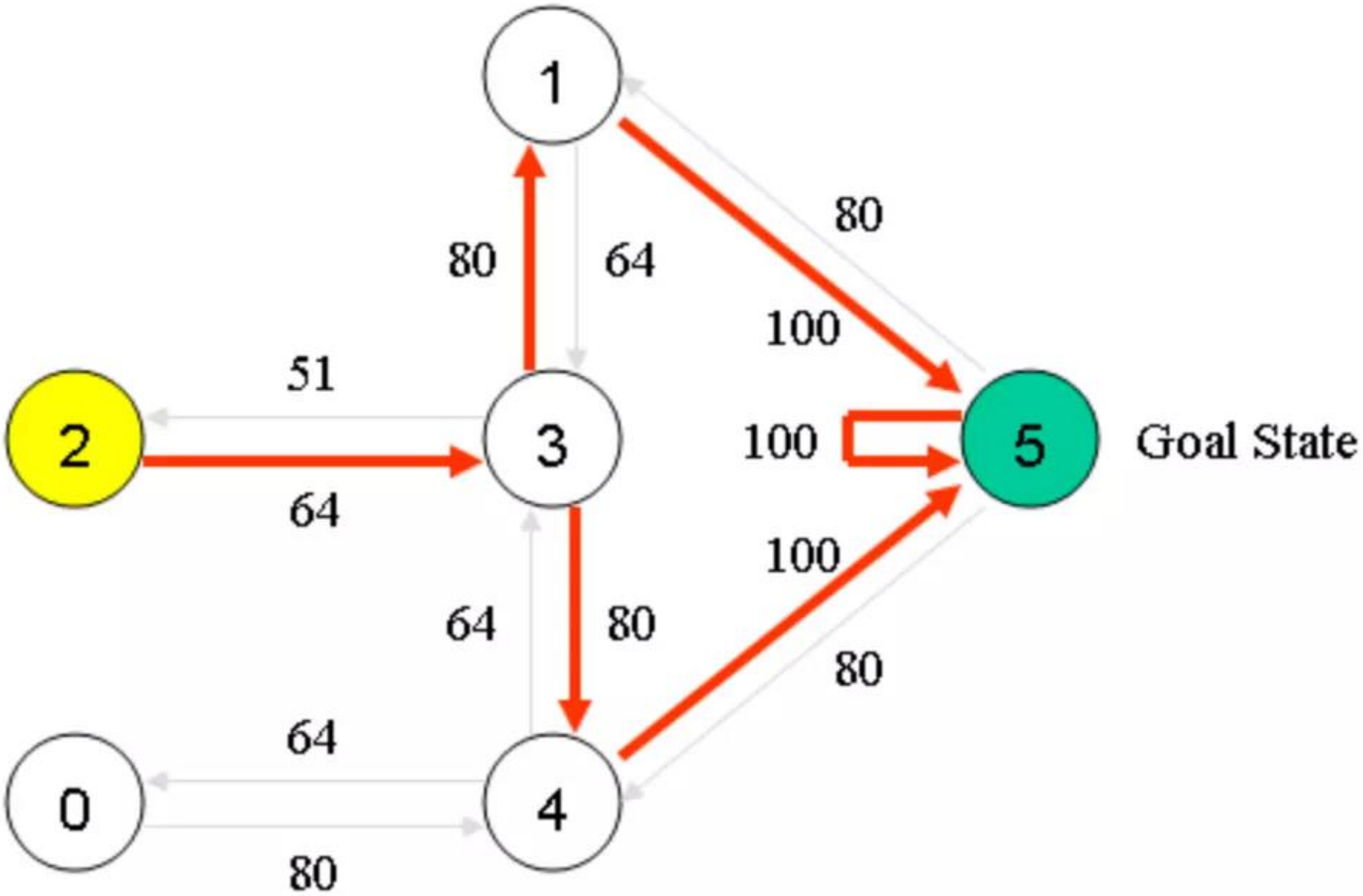


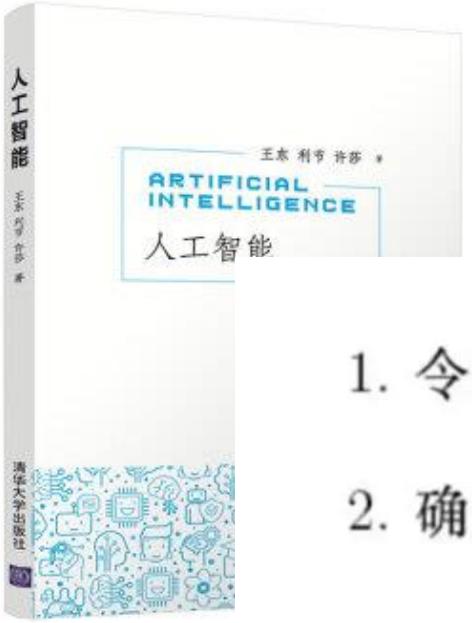
经过不停的迭代，最终我们的Q表为：

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 80 & 0 \\
0 & 0 & 0 & 64 & 0 & 100 \\
0 & 0 & 0 & 64 & 0 & 0 \\
0 & 80 & 51 & 0 & 80 & 0 \\
64 & 0 & 0 & 64 & 0 & 100 \\
0 & 80 & 0 & 0 & 80 & 100
\end{bmatrix}
\end{array}
$$



我们不妨将Q表中的数转移到我们一开始的示意图中：

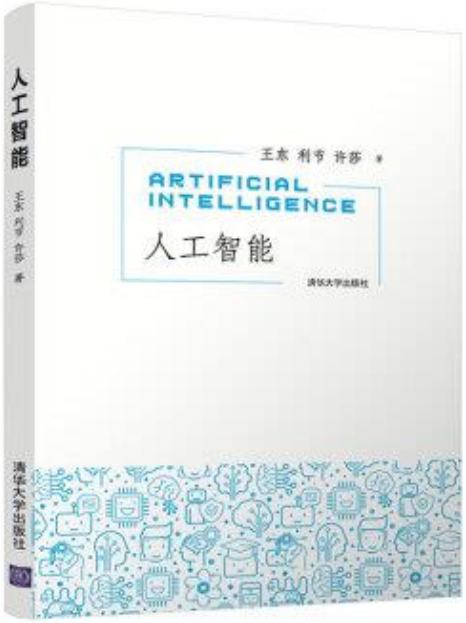




在得到Q表之后，我们可以根据如下的算法来选择我们的路径：

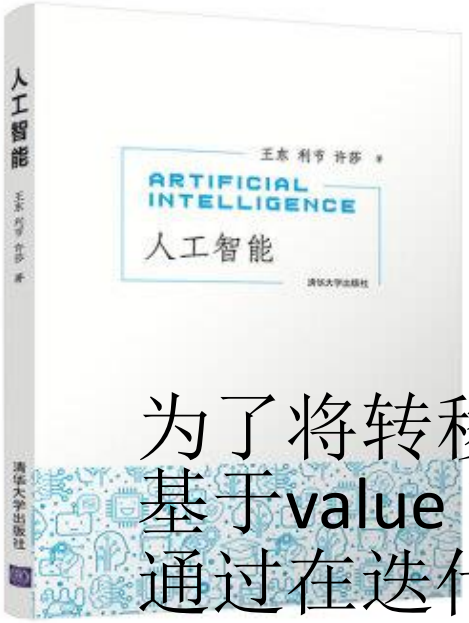
1. 令当前状态 $s := s_0$.
2. 确定 a, 它满足 $Q(s,a) = \max_{\widetilde{a}}\{Q(s,\widetilde{a})\}$.
3. 令当前状态 $s := \widetilde{s}$ ($\widetilde{s}$ 表示 a 对应的下一个状态).
4. 重复执行步 2 和步 3 直到 s 成为目标状态.

举例来说，假设我们的初始状态为2，那么根据Q表，我们选择2-3的动作，然后到达状态3之后，我们可以选择1，2，4。但是根据Q表，我们到1可以达到最大的价值，所以选择动作3-1，随后在状态1，我们按价值最大的选择选择动作1-5，所以路径为2-3-1-5.



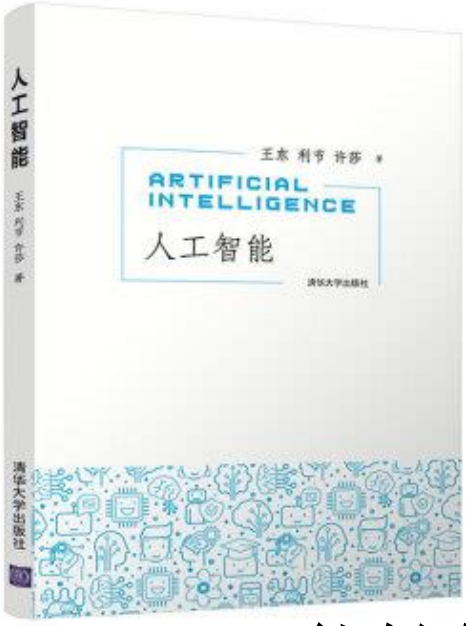
目录

- Q learning
- Deep Q learning

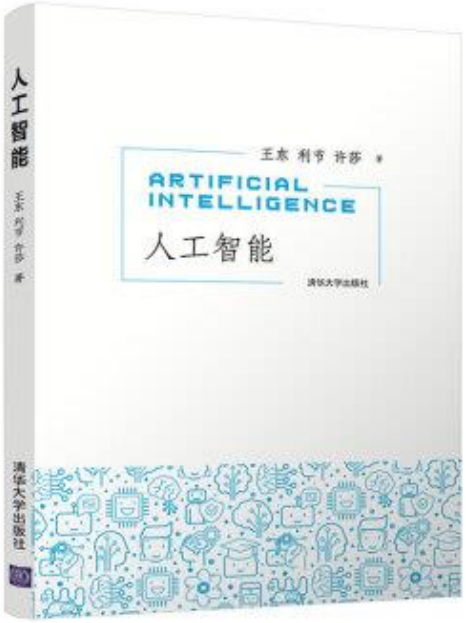


为了将转移概率以逼近实际情况的方式计算出来，基于value iteration的Q-Learning算法应运而生，它通过在迭代过程中不断更新Q-table的方式来近似转移概率矩阵P。

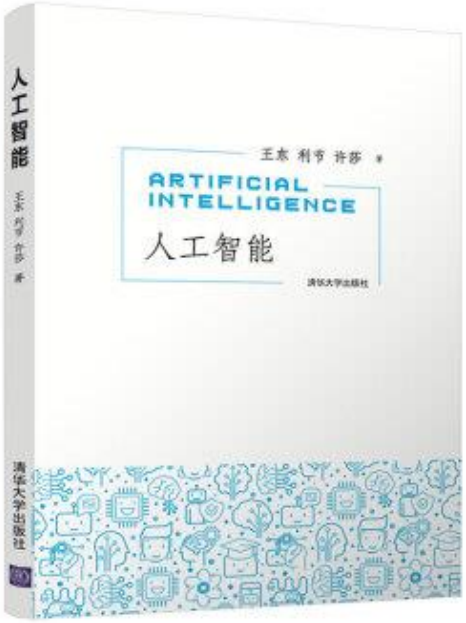
DQN (Deep Q-Learning Network)，当转移矩阵P (Q-table) 过大时计算困难甚至无法计算，而DQN其利用Deep网络结构拟合Q-table，使得Q-Learning框架具备了解决状态无限（动作仍旧有限）的强化学习问题。



DQN 能够解决状态无限，动作有限的问题；具体来说就是将当前状态作为输入，输出的是各个动作的 Q 值。以 Flappy Bird 这个游戏为例，输入的状态近乎是无限的（当前 bird 的位置和周围的水管的分布位置等），但是输出的动作只有两个(飞或者不飞)。实际上，已经有人通过 DQN 来玩这个游戏了。



在 DQN 中的核心问题在于如何训练整个神经网络，其实训练算法跟 Q-Learning 的训练算法非常相似，需要利用 Q 估计和 Q 现实的差值，然后进行反向传播。



The end !