强化学习

丁诚　2019.11.29

强化学习

- 强化学习
- 遗传算法
- Q-learning
- 期末Project

强化学习

- 本质是decision making问题，即自动进行决策，并且可以做连续决策
- 四个元素：agent、state、action、reward
- 策略（policy）：state到action的映射关系
- 评估方法（value function）：一个状态的价值就是从这个状态开始，期望在未来获得的奖赏，一种长期目标。
- 没有标签，反馈调节，不断调整策略，从而学习到：什么样的状态下选择什么样的动作可以获得最好的结果。

强化学习vs有监督学习

- 有监督学习是通过已有的数据和数据对应的正确标签，学习数据和标签对应逻辑。
- 若将状态看做数据属性，动作看作标签，监督学习和强化学习都是在试图寻找一个映射，从已知属性/状态推断出标签/动作，强化学习中的策略相当于有监督学习中的分类/回归器
- 强化学习刚开始并没有标签，它是在尝试动作后才能获得结果，通过反馈的结果信息不断调整之前的策略。
- 有监督学习的结果反馈是实时的，而强化学习的结果反馈有延时。
- 有监督学习的输入独立同分布，强化学习的输入总在变化，每个状态下的动作影响下一个状态。

强化学习vs无监督学习

- 无监督学习是通过未标签的数据，找到其中的隐藏模式
- 例如在向用户推荐新闻文章的任务中，无监督学习会找到用户先前阅读过的类似文章学习他们的偏向再进行推荐。而强化学习则会通过向用户推荐少量的新闻，得到用户的反馈，最后用于构建用户可能会喜欢的文章的“知识图”

遗传算法

- 染色体：问题的一个可行解。
- 基因：一个可行解一般由多个元素构成，每个元素称为染色体上的一个基因
- 利于问题$3x + 4y + 5z < 100$而言，$[1, 2, 3], [1, 3, 2], [2,3,1]$均为包含三个基因的染色体
- 适应度函数：在每次迭代中对所有生成染色体进行打分，评判这些染色体的适应度，决定淘汰或保留
- 交叉：两条染色体从某个位置切断，然后拼接，得到新的染色体
- 变异：在交叉得到的新染色上，随机选择若干个基因修改它们的值
- 复制：保留优良染色体，不作任何修改直接传给后代

遗传算法

随机初始化一组问题的可行解作为初代染色体。

在每次迭代中，采用适应度函数分别计算每一条染色体的适应程度，并根据适应程度计算每一条染色体在下一次迭代中被选中的概率：

染色体i被选择的概率 = 染色体i的适应度 / 所有染色体的适应度之和

进化（迭代）过程：

- 通过“交叉”，生成N-M条染色体；
- 再对交叉后生成的N-M条染色体进行“变异”操作；
- 然后使用“复制”的方式生成M条染色体。

每次进化得到N条染色体，然后分别计算N条染色体的适应度和下次进化被选中的概率。

遗传算法

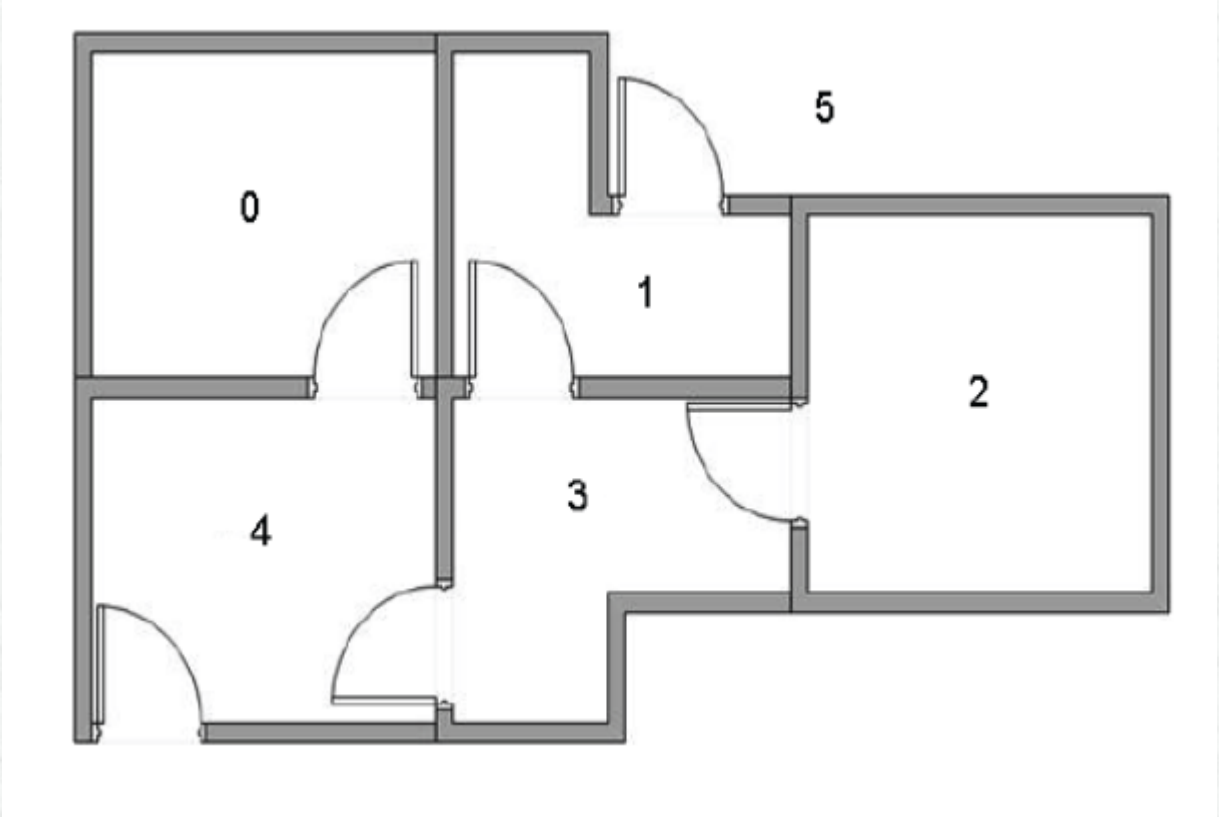
限定进化次数：

事先统计进化的次数或者人为限定次数

限定允许范围：

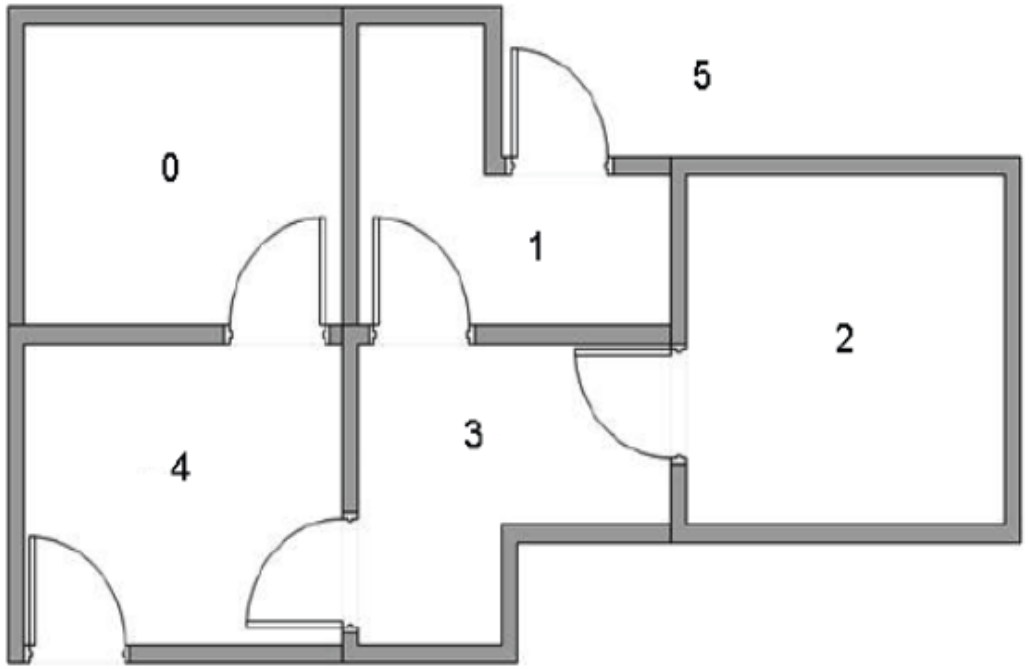
事先设定一个可以接收的结果范围，当算法进行若干次次进化后，一旦发现了当前的结果已经在误差范围之内了，那么就终止算法

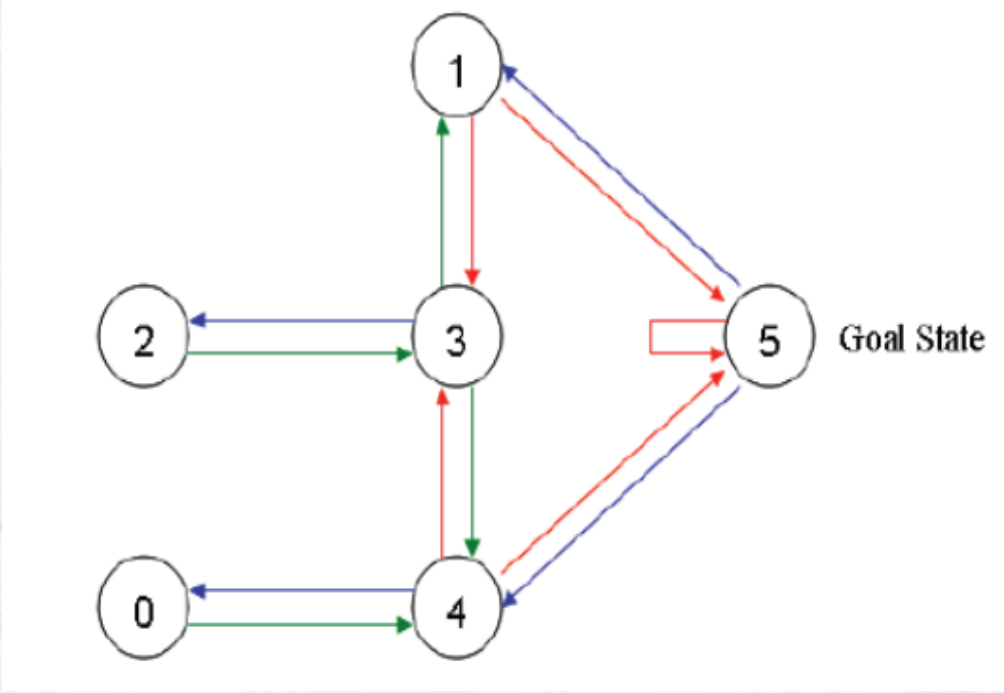
Q-learning



假设屋子有5个房间，编号为0-4，房间之间通过门相连，屋子外可视为一个大方向，编号为5

Q-learning

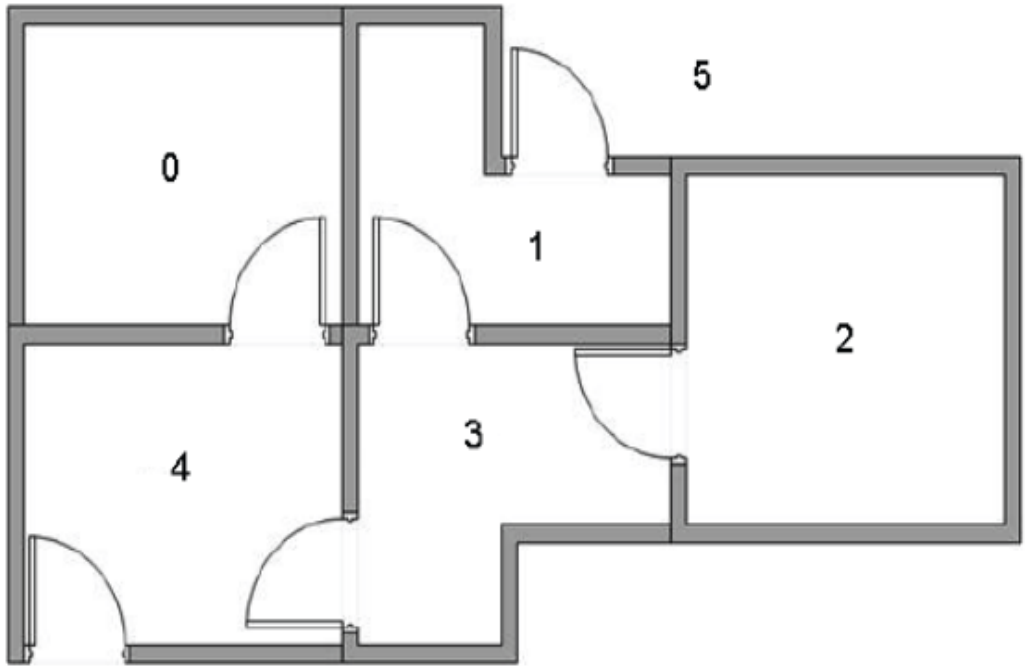


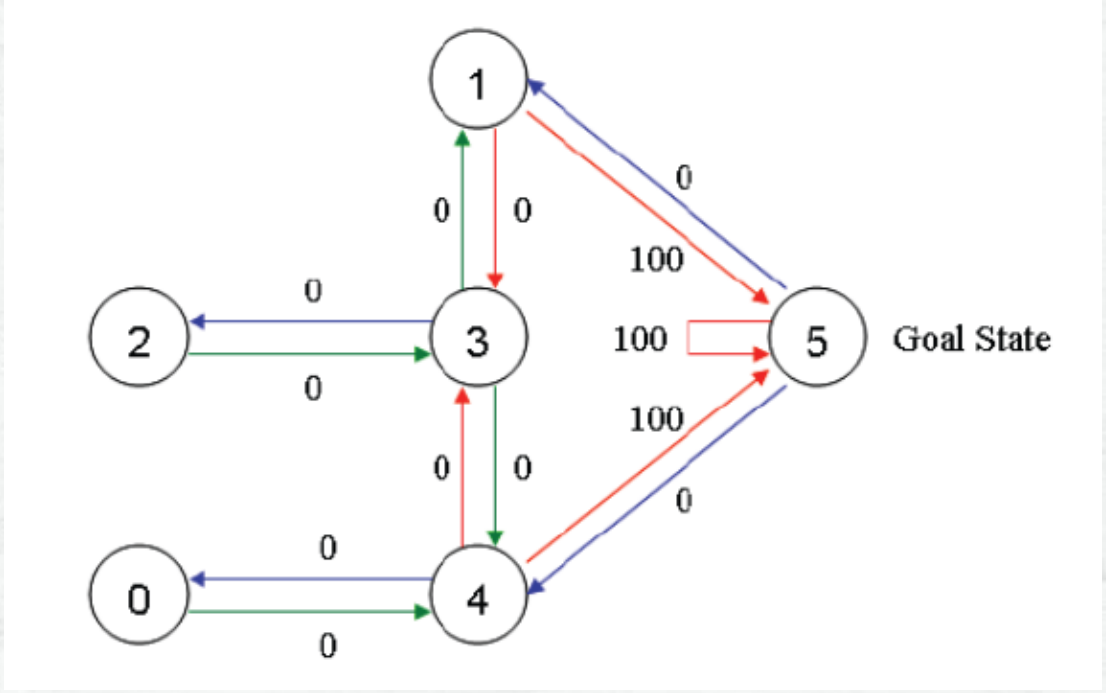


房间作为图的节点，两房间之间若有门相连则对应节点间的一条边，
每个门都有两个方向。

State：房间（节点）　　action：从一个房间走到另一个房间（箭头）

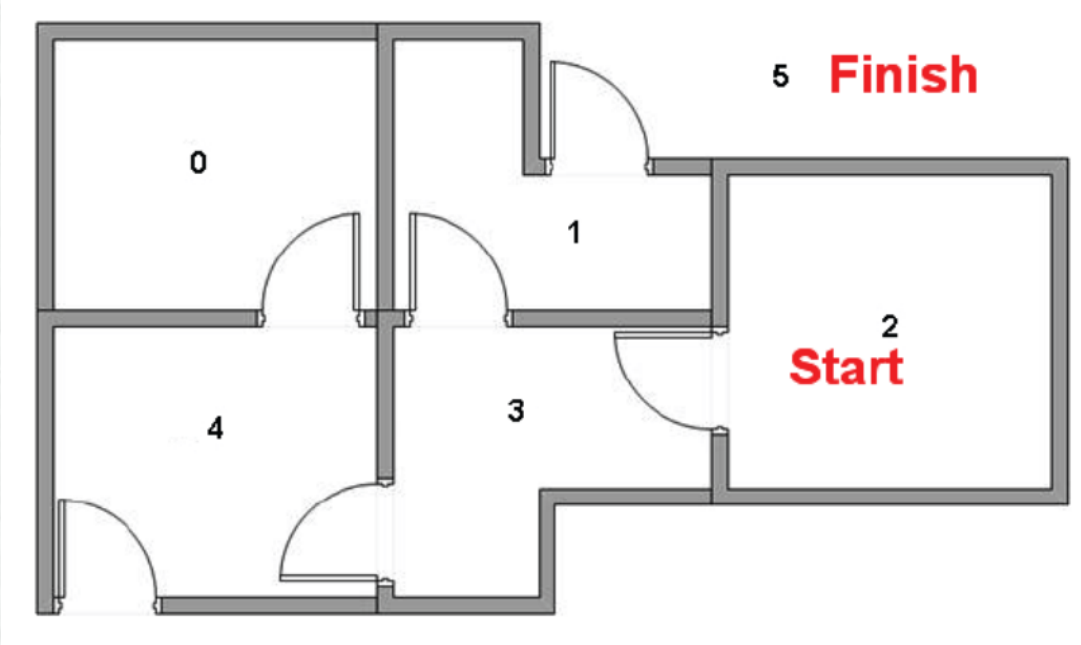
Q-learning

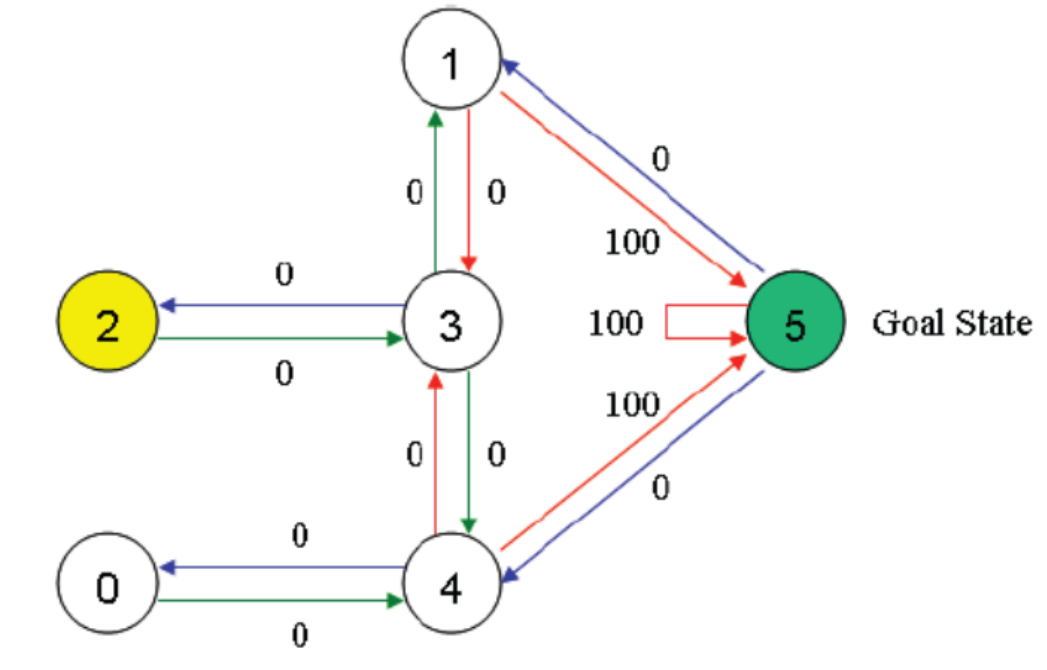




将agent置于建筑中的任意一个房间，目标是走到屋子外，即房间5。
每一条边关联一个reward值，直接连接到目标房间的门的reward值为100，其他门的reward值为0。
房间5有一个指向自己的箭头，reward值也为100。Q-learning的目标是达到value值最大的state。

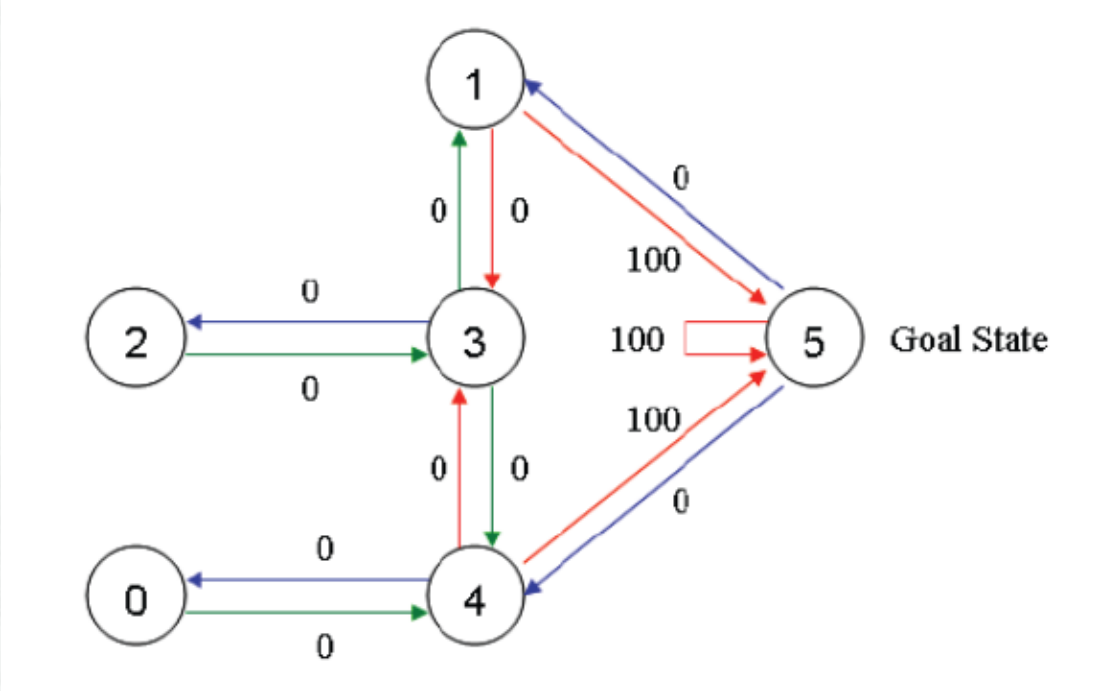
Q-learning





假设agent从状态2开始，我们希望通过学习到达状态5.

Q-learning



Action

State	0	1	2	3	4	5
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

$R=$

以state为行，action为列构建reward值矩阵R，其中-1表示空值，即节点间没有边相连。

Q-learning

类似地，构建一个与R同阶的矩阵Q，表示agent已经从经验中学到的知识。
由于agent刚开始对外界环境一无所知，Q初始化为零矩阵。
本例中的状态数目是已知的（等于6），对于状态数目未知的情形，可以让Q从一个元素出发,每发现一个新的状态就增加相应的行列。

Q-learning算法的状态转移规则：

$$Q(s,a) = R(s,a) + \gamma \cdot \max_{\widetilde{a}}\{Q(\widetilde{s},\widetilde{a})\}, \tag{1.1}$$

其中 s, a 表示当前的状态和行为，$\widetilde{s}, \widetilde{a}$ 表示 s 的下一个状态及行为，学习参数 γ 为满足 $0 \leq \gamma < 1$ 的常数.

Q-learning

Q-learning算法：

Step 1 给定参数 γ 和 *reward* 矩阵 R.

Step 2 令 $Q := 0$.

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 $s := \widetilde{s}$.

Agent利用该算法从经验中学习,每一个episode相当于一个training epoch，agent不断探索外界环境，并接收外界环境的reward，直至达到目标状态。训练得越多，Q被优化得更好，agent就能根据训练后的Q更容易地找到到达目标状态的最快路径。

Q-learning

得到充分训练的Q之后：

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.

Q-learning

取γ=0.8，初始状态为房间1，Q初始化为0矩阵

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

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

若agent随机地转移到状态5，更新Q矩阵，得到一次episode后的Q矩阵。

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

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

Q-learning

第二次episode：随机选择一个初始状态，此处选状态3。让agent执行走到状态1的action，更新Q：

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

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

现在状态1变成了当前状态,因为状态1还不是目标状态，仍需继续探索，让随机选择可能的action。假定agent选择了走到状态5的action,更新Q：

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

此处更新并没有引起矩阵Q的变化，Q保持不变。

Q-learning

执行更多的episode：矩阵Q最终收敛为：

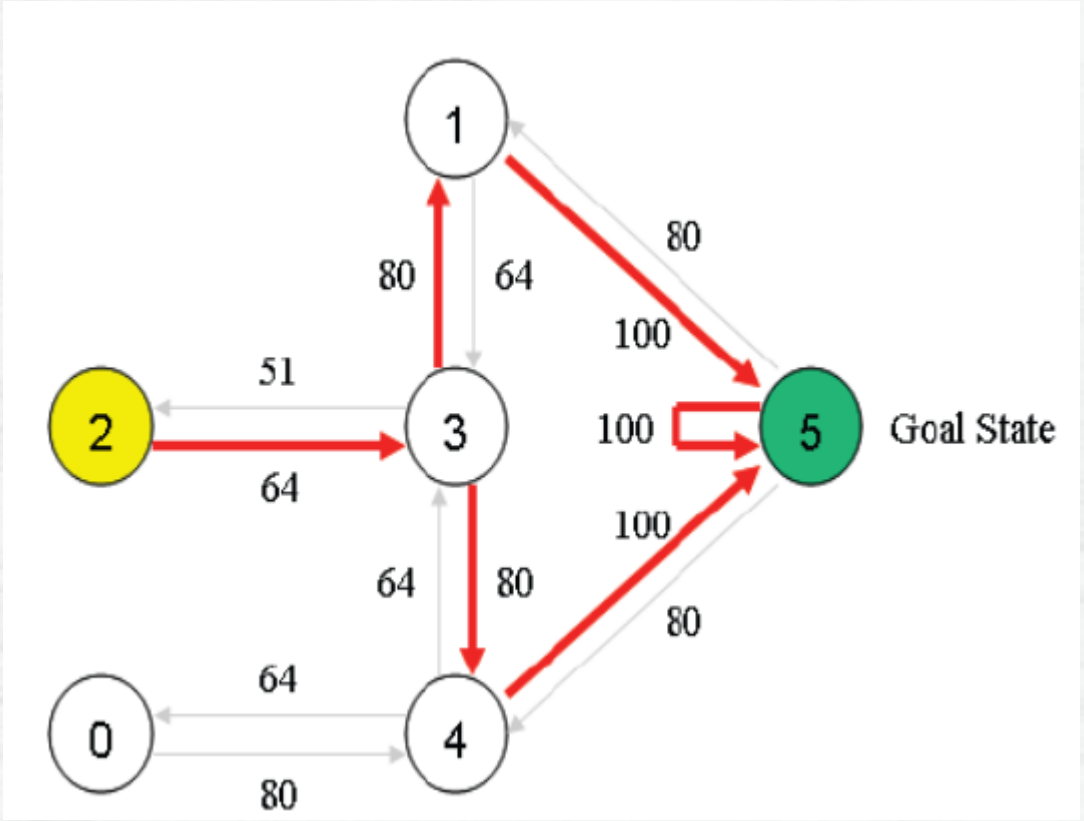
$$Q=\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

归一化，每个元素都除以Q的最大元素：

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

Q-learning

当矩阵Q接近收敛状态，agent便学习到了转移至目标状态的最佳路径：



Q-learning算法总结

$Q(s,a)$ 表示在状态s下执行动作a能够获得的value

每一个时刻的 $Q(s,a)$和当前得到的reward以及下一时刻的 $Q(s,a)$有关

初始化$Q(s,a), \forall s \in S, a \in A(s)$,任意的数值，并且$Q(terminal-state,\cdot)=0$

重复（对每一节episode）：

初始化 状态S

重复（对episode中的每一步）：

使用某一个policy比如（$\epsilon-greedy$)根据状态S选取一个动作执行

执行完动作后，观察reward和新的状态S'

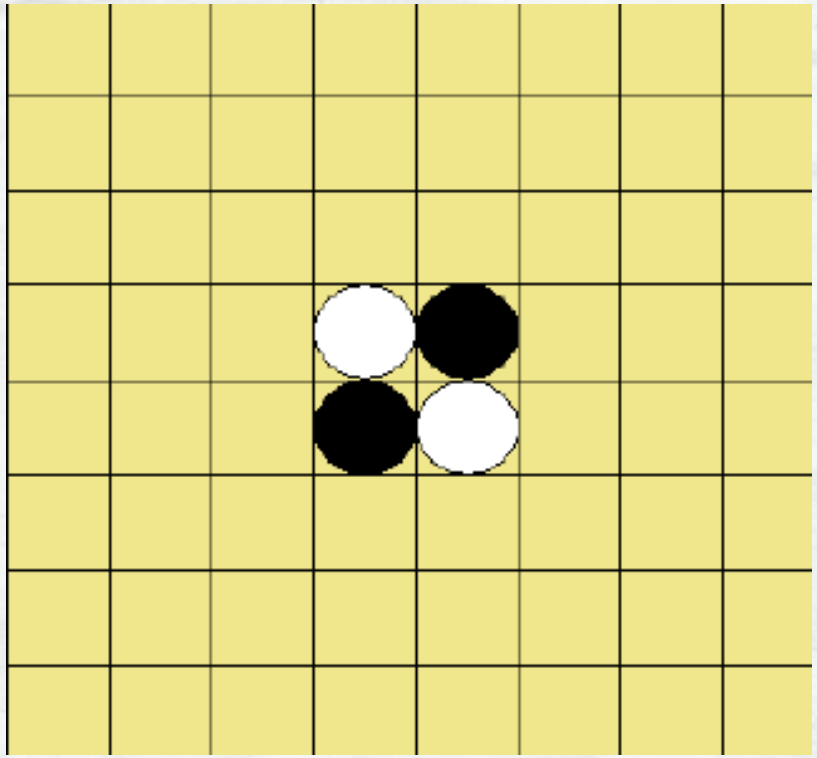
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t)+\alpha(R_{t+1}+\lambda \max_a Q(S_{t+1},a)-Q(S_t,A_t))$$

$$S \leftarrow S'$$

循环直到S终止

Project

- 实现8X8黑白棋的人机对战，要求：
- 横排、竖排、对角线均可翻转
- 搜索算法不限，如minimax，mcts等
- 搜索深度和评价函数不限
- 结合机器学习算法，如遗传算法、DQN、Policy Network等



初始状态如图所示，要求有电脑先手和电脑后手两种模式

组队

- 一组人数**1~2人**，队伍名字，共同评分，推荐组队完成
- **12月7日0点前提前组队名单**
- https://docs.qq.com/form/fill/DRklhZ01ZVVVBT29k?_w_tencentdocx_form=1

评分标准

- Rank 20%
- 报告 80%

Rank

- 内容：安排小组之间进行博弈
- 具体的时间和形式之后公布

报告要求

- 实验原理
- 实现过程
- 实验结果分析
- 鼓励创新（不同算法之间的对比）

最终提交

- 报告提交DDL：**（2019年12月26日晚上11点）23:00:00**
- 提交内容：
  - 实验报告，每个人一份，命名为：组长学号_学号_姓名拼音_report.pdf，如“17350000_17350001_report.pdf”
  - 源码zip，包含代码文件和**readme**文件，命名为：组长学号_学号_code.zip，如“17350000_17350001_code.zip”，**readme** 文件阐述各文件用途，工程如何运行。

相关算法

MSTC : https://www.cntofu.com/book/85/dl/reinforcement/monte-carlo.md

Policy Network : https://www.cntofu.com/book/85/dl/reinforcement/policy-gridient.md

Paper: Policy Gradient Methods for Reinforcement Learning with Function Approximation

TRPO : https://blog.csdn.net/meizhulei/article/details/85138538

Paper: Trust Region Policy Optimization