A Painless Q-learning Tutorial (一个Q-learning 算法的简明教程)

本文是对 http://mnemstudio.org/path-finding-q-learning-tutorial.htm 的翻译，共分

第一部分：中文翻译

§1.1 Step-By-Step Tutorial

本教程将通过一个简单但又综合全面的例子来介绍 Q-learning 算法. 该例子描述了一个利用无监督训练来学习未知环境的 agent.

假设一幢建筑里面有 5 个房间, 房间之间通过门相连 (如图 1 所示). 我们将这五个房间按照从 0 至 4 进行编号, 且建筑的外围可认为是一个大的房间, 编号为 5.

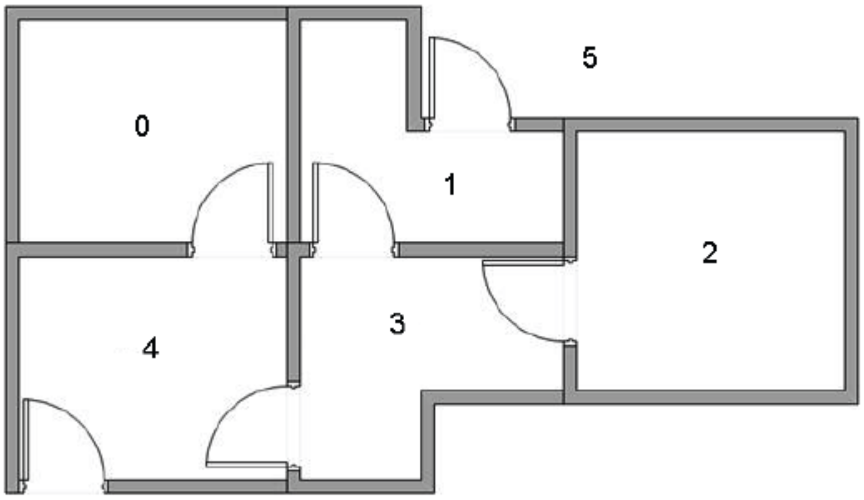


图 1 房间结构

上图的房间也可以通过一个图来表示, 房间作为图的节点, 两个房间若有门相连, 则相应节点间对应一条边, 如图 2 所示.

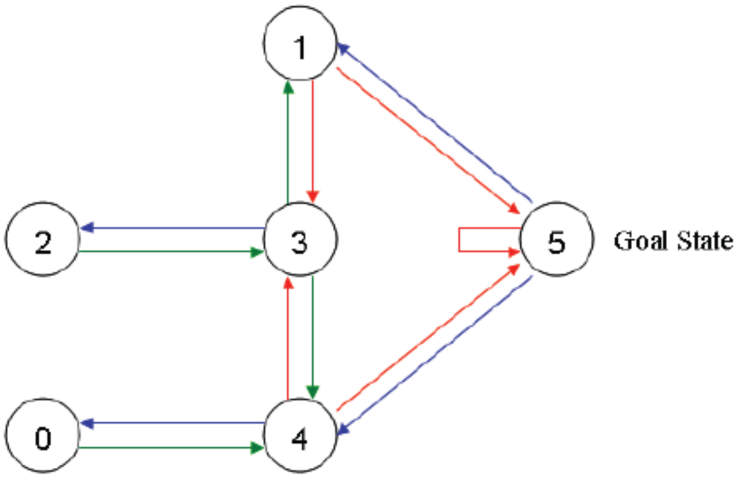
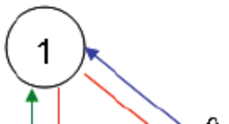


图 2 房间结构对应的图

对于这个例子, 我们首先将 agent 置于建筑中的任意一个房间, 然后从那个房间开始, 让其走到建筑外, 那是我们的目标房间 (即编号为 5 的房间). 为了将编号为 5 的房间设置为目标, 我们为每一扇门 (即相应的边) 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. 因为每一扇门都有两个方向 (如由 0 号房间可以去 4 号房间, 而由 4 号房间也可以返回 0 号房间), 因此每一个房间上指定两个箭头 (一个指进一个指出), 且每个箭头上带有一个 reward 值 (如图 3 所示).



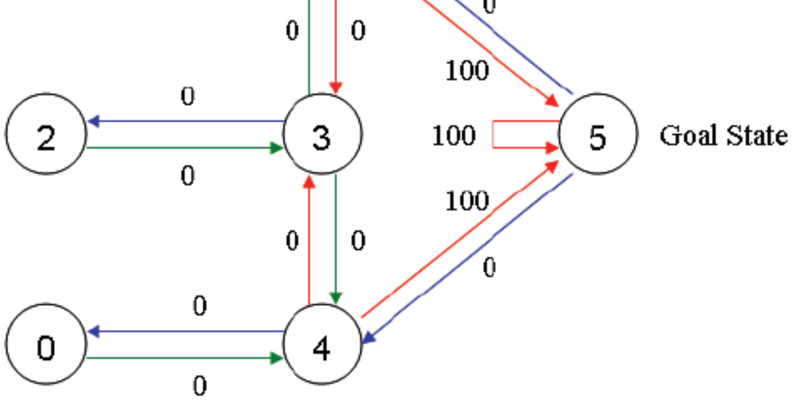


图 3

注意, 编号为 5 的房间有一个指向自己的箭头, 其 reward 值为 100, 其他直接指向目标房间的边的 reward 值也为 100. Q-learning 的目标是达到 reward 值最大的 state, 因此, 当 agent 到达目标房间后将永远停留在那里. 这种目标也称为 **"吸收目标"**.

想象一下, 我们的 agent 是一个可以通过经验进行学习的 "哑巴虚拟机器人", 它可以从一个房间走到另一个房间, 但是, 它不知道周边的环境, 也不知道怎样走到建筑的外面去.

下面我们想对 agent 从建筑里的任意房间的简单撤离进行建模. 假定现在 agent 位于 2 号房间, 我们希望 agent 通过学习到达 5 号房间.

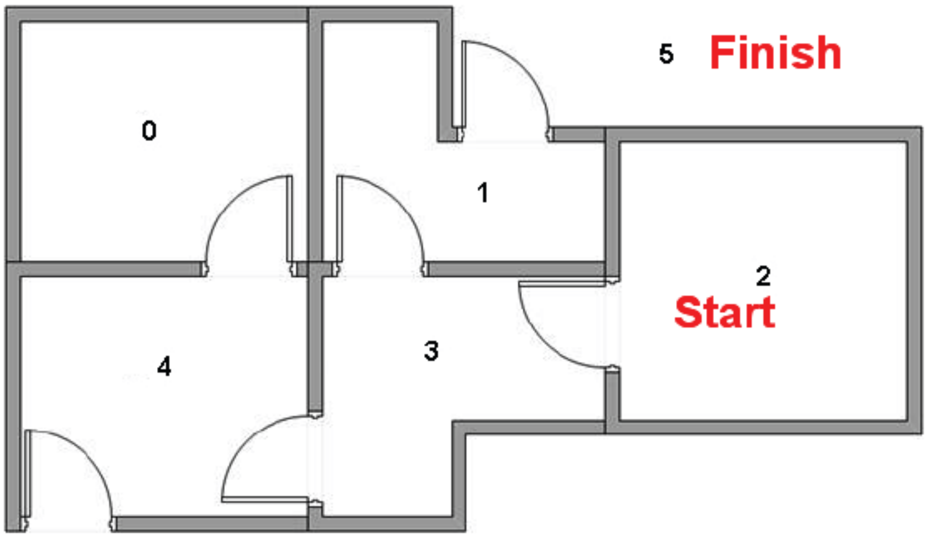
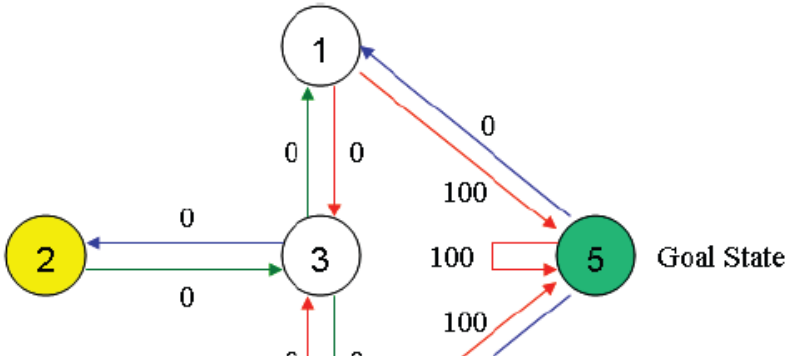


图 4

Q-Learning 算法中有两个重要术语: **"状态** (state)" 和 **"行为** (action)".

我们将每一房间 (包括 5 号房间) 称为一个 **"状态"**, 将 agent 从一个房间走到另外一个房间称为一个 **"行为"**. 在图 2 中, 一个 "状态" 对应一个节点, 而一种 "行为" 对应一个箭头.



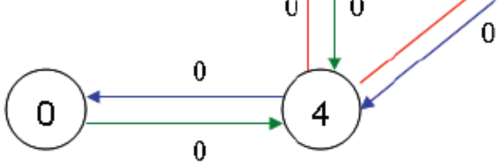


图 5

假设 agent 当前处于状态 2. 从状态 2, 它可以转至状态 3 (因为状态 2 到状态 3 有边相连). 但从状态 2 不能转至状态 1 (因为状态 2 到状态 1 没边相连). 类似地, 我们还有

- 从状态 3, 它可以转至状态 1 和 4, 也可以转回至状态 2.
- 从状态 4, 它可以转至状态 0, 5 和 3.
- 从状态 1, 它可以转至状态 5 和 3.
- 从状态 0, 它只能转至状态 4.

我们可以以**状态**为行, **行为**为列, 构建一个如图 6 所示的关于 reward 值的矩阵 R, 其中的 -1 表示空值 (相应节点之间没有边相连).

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

图 6 reward 值矩阵

类似地, 我们也可以构建一个矩阵 Q, 它用来表示 agent 已经从经验中学到的知识. 矩阵 Q 与 R 是同阶的, 其行表示**状态**, 列表示**行为**.

由于刚开始时 agent 对外界环境一无所知, 因此矩阵 Q 应初始化为零矩阵. 为简单起见, 在本例中我们假设状态的数目是已知的 (等于 6). 对于状态数目未知的情形, 我们可以让 Q 从一个元素出发, 每次发现一个新的状态时就可以在 Q 中增加相应的行列.

Q-learning 算法的**转移规则**比较简单, 如下式所示:

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

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 $0\le\gamma<1$ 的常数.

在没有老师的情况下, 我们的 agent 将通过经验进行学习 (也称为**无监督学习**). 它不

断从一个状态转至另一状态进行探索, 直到到达目标. 我们将 agent 的每一次探索称为一个 **episode**. 在每一个 episode 中, agent 从任意初始状态到达目标状态. 当 agent 达到目标状态后, 一个 episode 即结束, 接着进入另一个 episode.

下面给出整个 Q-learning 算法的计算步骤.

算法 1.1 *(Q-learning 算法)*

```
Step 1 给定参数 $\gamma$ 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 $s := \widetilde{s}$.
```

Agent 利用上述算法从经验中进行学习. 每一个 episode 相当于一个 training session. 在一个 training session 中, agent 探索外界环境, 并接收外界环境的 reward, 直到达到目标状态. 训练的目的是要强化 agent 的 "大脑" (用 Q 表示). 训练得越多, 则 Q 被优化得更好. 当矩阵 Q 被训练强化后, agent 便很容易找到达到目标状态的最快路径了.

公式 (1.1) 中的 γ 满足 $0 \leq \gamma < 1$. γ 趋向于 0 表示 agent 主要考虑 immediate reward, 而 γ 趋向于 1 表示 agent 将同时考虑 future rewards.

利用训练好的矩阵 Q, 我们可以很容易地找出一条从任意状态 s_0 出发达到目标状态的行为路径, 具体步骤如下:

1. 令当前状态 $s := s_0$.

2. 确定 a, 它满足 $Q(s,a) = \max\limits_{\widetilde{a}}\{Q(s,\widetilde{a})\}$.

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.

§1.2 Q-Learning Example By Hand

为进一步理解上一节中介绍的 Q-learning 算法是如何工作的, 下面我们一步一步地迭

为进一步理解上一节中介绍的 Q-learning 算法是如何工作的, 下面我们一步一步地迭代几个 episode.

首先取学习参数 $\gamma = 0.8$, 初始状态为房间 1, 并将 Q 初始化为一个零矩阵.

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

图 7 将 Q 初始化为一个零矩阵

观察矩阵 R 的第二行 (对应房间 1 或状态 1), 它包含两个非负值, 即当前状态 1 的下一步行为有两种可能: 转至状态 3 或转至状态 5. 随机地, 我们选取转至状态 5.

$$\begin{array}{cc} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \end{array}$$

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

图 8

想象一下, 当我们的 agent 位于状态 5 以后, 会发生什么事情呢? 观察矩阵 R 的第 6 行 (对应状态 5), 它对应三个可能的行为: 转至状态 1, 4 或 5. 根据公式 (1.1), 我们有

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

现在状态 5 变成了当前状态. 因为状态 5 即为目标状态, 故一次 episode 便完成了, 至此, agent 的 “大脑” 中的 Q 矩阵刷新为

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

图 9 一次 episode 后的 Q 矩阵

接下来, 进行下一次 episode 的迭代, 首先随机地选取一个初始状态, 这次我们选取状态 3 作为初始状态.

观察矩阵 R 的第四行 (对应状态 3), 它对应三个可能的行为: 转至状态 1, 2 或 4. 随机地, 我们选取转至状态 1. 因此观察矩阵 R 的第二行 (对应状态 1), 它对应两个可能的行为: 转至状态 3 或 5. 根据公式 (1.1), 我们有

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

注意上式中的 $Q(1,5)$ 用到了图 9 中的刷新值. 此时, 矩阵 Q 变为

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

图 10

现在状态 1 变成了当前状态. 因为状态 1 还不是目标状态, 因此我们需要继续往前探索. 状态 1 对应三个可能的行为: 转至状态 3 或 5. 不妨假定我们幸运地选择了状态 5.

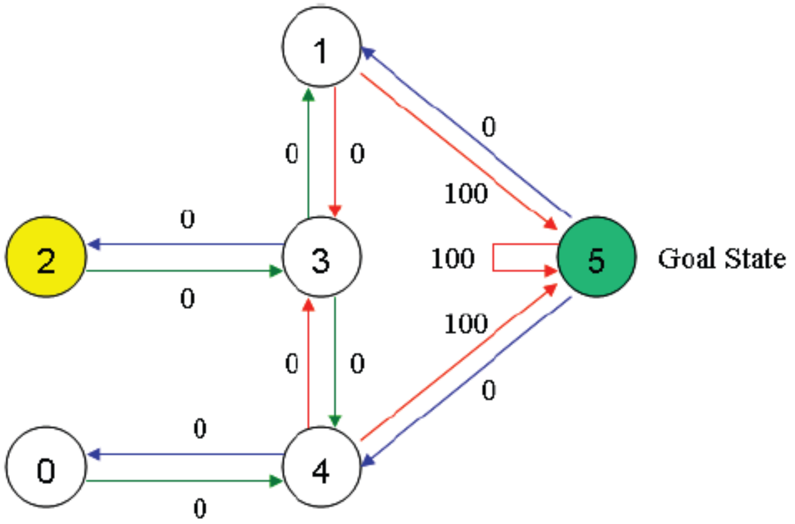


图 11

此时, 同前面的分析一样, 状态 5 有三个可能的行为: 转至状态 1, 4 或 5. 根据公式 (1.1), 我们有

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

$$= \quad 100.$$

注意, 经过上一步刷新, 矩阵 Q 并没有发生变化.

因为状态 5 即为目标状态, 故这一次 episode 便完成了, 至此, agent 的 “大脑” 中的 Q 矩阵刷新为

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

图 12

若我们继续执行更多的 episode, 矩阵 Q 将最终收敛成

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

图 13

对其进行规范化, 每个非零元素都除以矩阵 Q 的最大元素 (这里为 500), 可得 (这里省略了百分号)

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

图 14 规范化后的矩阵 Q

一旦矩阵 Q 足够接近于收敛状态, 我们的 agent 便学习到了转移至目标状态的最佳路径. 只需按照上一节结尾时介绍的步骤, 即可找到最优的路径 (如图 15 所示).

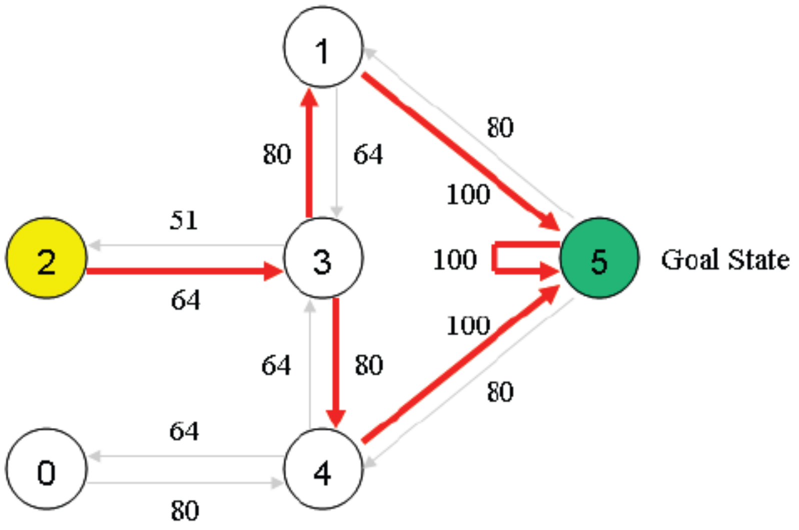


图 15

例如, 从 2 为初始状态, 利用 Q, 可得

- 从状态 2, 最大 Q 元素值指向状态 3;
- 从状态 3, 最大 Q 元素值指向状态 1 或 4 (这里假设我们随机地选择了 1);
- 从状态 1, 最大 Q 元素值指向状态 5,

因此最佳路径的序列为 2-3-1-5.

第二部分：英文原文

§2.1 Step-By-Step Tutorial

This tutorial introduces the concept of **Q-learning** through a simple but comprehensive numerical example. The example describes an **agent** which uses **unsupervised training** to learn about an unknown environment. You might also find it helpful to compare this example with the accompanying source code examples.

Suppose we have 5 rooms in a building connected by doors as shown in the figure below. We'll number each room 0 through 4. The outside of the building can be thought of as one big room (5). Notice that doors 1 and 4 lead into the building from room 5 (outside).

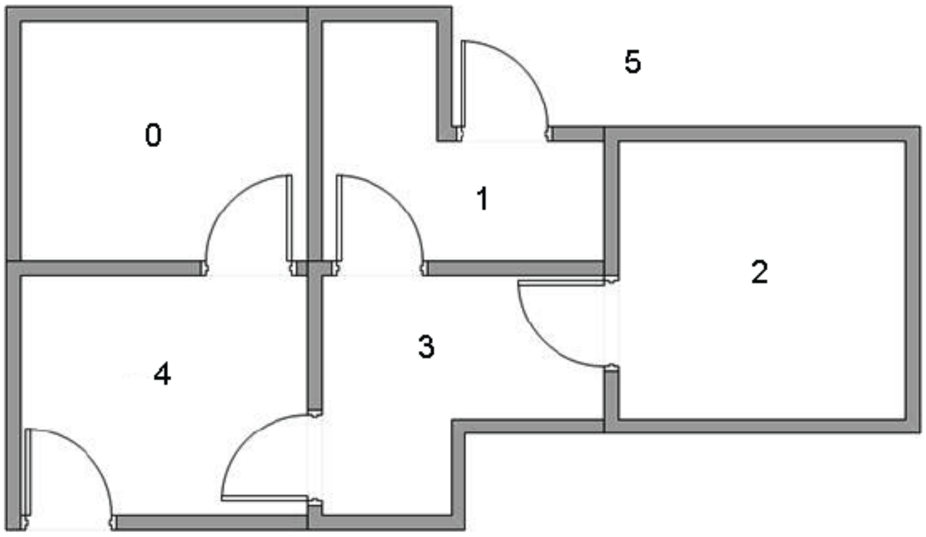


图 16

We can represent the rooms on a graph, each room as a node, and each door as a link.

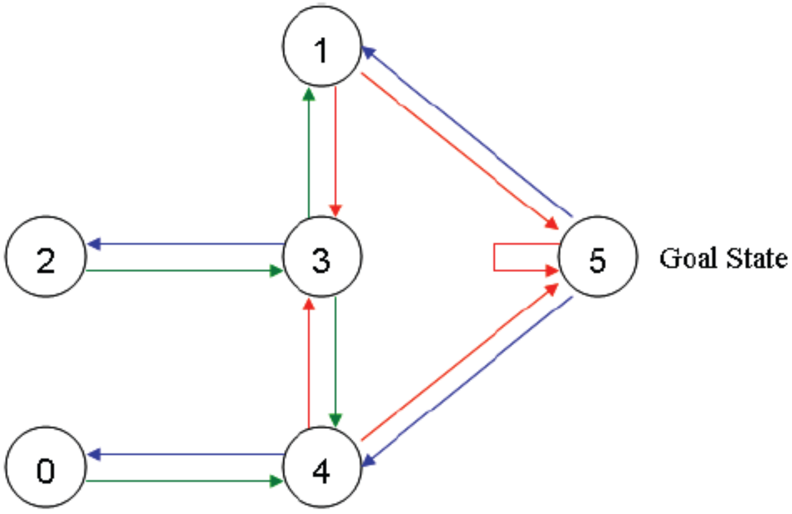


图 17

For this example, we'd like to put an agent in any room, and from that room, go outside the building (this will be our target room). In other words, the goal room is number 5. To set this room as a goal, we'll associate a **reward value** to each door (i.e. link between nodes). The doors that lead immediately to the goal have an instant reward of 100. Other doors not directly connected to the target room have zero reward. Because doors are two-way (0 leads to 4, and 4 leads back to 0), two arrows are assigned to each room. Each arrow contains an instant reward value, as shown below:

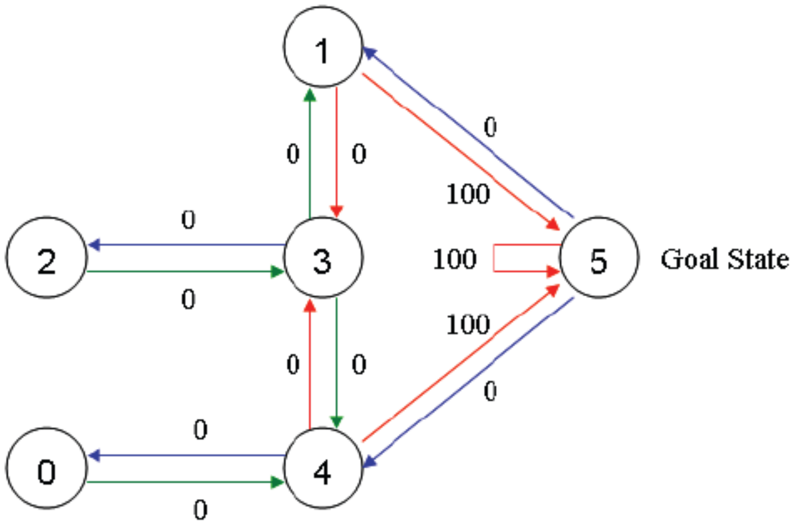


图 18

Of course, Room 5 loops back to itself with a reward of 100, and all other direct connections to the goal room carry a reward of 100. In Q-learning, the goal is to reach the state with the highest reward, so that if the agent arrives at the goal, it will remain there forever. This type of goal is called an ”**absorbing goal**”.

Imagine our agent as a dumb virtual robot that can learn through experience. The agent can pass from one room to another but has no knowledge of the environment, and doesn’t know which sequence of doors lead to the outside.

Suppose we want to model some kind of simple evacuation of an agent from any room in the building. Now suppose we have an agent in Room 2 and we want the agent to learn to reach outside the house (5).

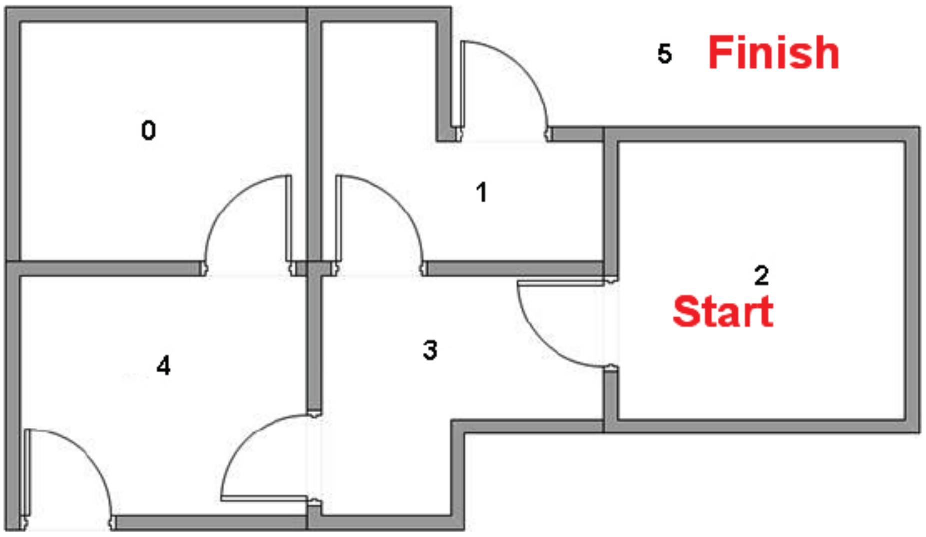


图 19

The terminology in Q-Learning includes the terms ”**state**” and ”**action**”.

We’ll call each room, including outside, a ”state”, and the agent’s movement from one

We'll call each room, including outside, a "state", and the agent's movement from one room to another will be an "action". In our diagram, a "state" is depicted as a node, while "action" is represented by the arrows.

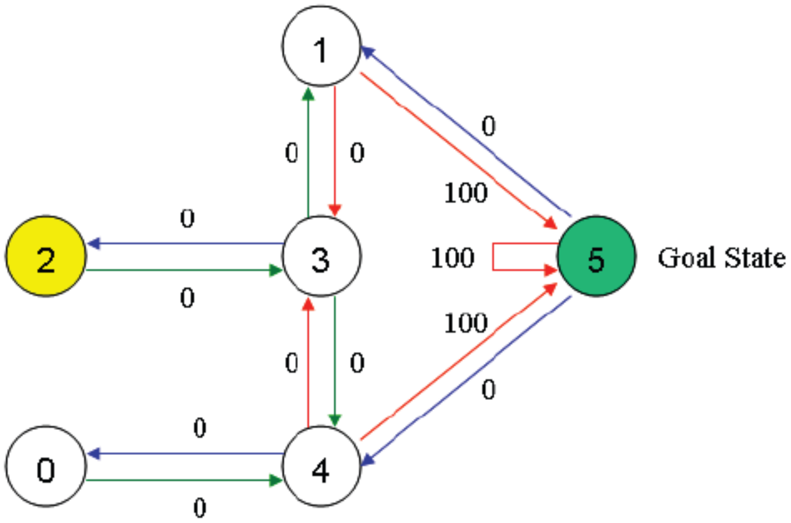


图 20

Suppose the agent is in state 2. From state 2, it can go to state 3 because state 2 is connected to 3. From state 2, however, the agent cannot directly go to state 1 because there is no direct door connecting room 1 and 2 (thus, no arrows). From state 3, it can go either to state 1 or 4 or back to 2 (look at all the arrows about state 3). If the agent is in state 4, then the three possible actions are to go to state 0, 5 or 3. If the agent is in state 1, it can go either to state 5 or 3. From state 0, it can only go back to state 4.

We can put the state diagram and the instant reward values into the following reward table, "matrix R".

$$
\begin{array}{cc}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}
\end{array}
$$

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

图 21

The -1's in the table represent null values (i.e.; where there isn't a link between nodes). For example, State 0 cannot go to State 1.

Now we'll add a similar matrix, "Q", to the brain of our agent, representing the memory of what the agent has learned through experience. The rows of matrix Q represent the current state of the agent, and the columns represent the possible actions leading to the next state (the links between the nodes).

The agent starts out knowing nothing, the matrix Q is initialized to zero. In this example, for the simplicity of explanation, we assume the number of states is known (to be six). If we didn't know how many states were involved, the matrix Q could start out with only one element. It is a simple task to add more columns and rows in matrix Q if a new state is found.

The transition rule of Q learning is a very simple formula:

Q(state, action) = R(state, action) + Gamma·Max[Q(next state, all actions)]

According to this formula, a value assigned to a specific element of matrix Q, is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state.

Our virtual agent will learn through experience, without a teacher (this is called unsupervised learning). The agent will explore from state to state until it reaches the goal. We'll call each exploration an **episode**. Each episode consists of the agent moving from the initial state to the goal state. Each time the agent arrives at the goal state, the program goes to the next episode.

The Q-Learning algorithm goes as follows:

1. Set the gamma parameter, and environment rewards in matrix R.

2. Initialize matrix Q to zero.

3. For each episode:

    (a) Select a random initial state.

    (b) Do While the goal state hasn't been reached.

        i. Select one among all possible actions for the current state.

ii. Using this possible action, consider going to the next state.

iii. Get maximum Q value for this next state based on all possible actions.

iv. Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

v. Set the next state as the current state.

The algorithm above is used by the agent to learn from experience. Each episode is equivalent to one **training session**. In each training session, the agent explores the environment (represented by matrix R), receives the reward (if any) until it reaches the goal state. The purpose of the training is to enhance the 'brain' of our agent, represented by matrix Q. More training results in a more optimized matrix Q. In this case, if the matrix Q has been enhanced, instead of exploring around, and going back and forth to the same rooms, the agent will find the fastest route to the goal state.

The Gamma parameter has a range of 0 to 1 ($0 \leq Gamma < 1$). If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

To use the matrix Q, the agent simply traces the sequence of states, from the initial state to goal state. The algorithm finds the actions with the highest reward values recorded in matrix Q for current state:

Algorithm to utilize the Q matrix:

1. Set current state = initial state.

2. From current state, find the action with the highest Q value.

3. Set current state = next state.

4. Repeat Steps 2 and 3 until current state = goal state.

The algorithm above will return the sequence of states from the initial state to the goal state.

§2.2 Q-Learning Example By Hand

To understand how the Q-learning algorithm works, we'll go through a few episodes step by step. The rest of the steps are illustrated in the source code examples.

We'll start by setting the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Initialize matrix Q as a zero matrix:

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

图 22

Look at the second row (state 1) of matrix R. There are two possible actions for the current state 1: go to state 3, or go to state 5. By **random selection**, we select to go to 5 as our action.

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

图 23

Now let's imagine what would happen if our agent were in state 5. Look at the sixth row of the reward matrix R (i.e. state 5). It has 3 possible actions: go to state 1, 4 or 5.

Q(state, action) = R(state, action) + Gamma·Max[Q(next state, all actions)]

Q(1,5) = R(1,5) + 0.8*Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100

Since matrix Q is still initialized to zero, Q(5,1), Q(5,4), Q(5,5), are all zero. The result of this computation for Q(1,5) is 100 because of the instant reward from R(5,1).

The next state, 5, now becomes the current state. Because 5 is the **goal state**, we've finished one episode. Our agent's brain now contains an updated matrix Q as:

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

图 24

For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state.

Look at the fourth row of matrix R; it has 3 possible actions: go to state 1, 2 or 4. By **random selection**, we select to go to state 1 as our action.

Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5. Then, we compute the Q value:

Q(state, action) = R(state, action) + Gamma·Max[Q(next state, all actions)]

Q(3,1) = R(3,1) + 0.8*Max[Q(1,3), Q(1,5)] = 0 + 0.8*Max(0, 100) = 80

We use the updated matrix Q from the last episode. Q(1,3) = 0 and Q(1,5) = 100. The result of the computation is Q(3,1) = 80 because the reward is zero. The matrix Q becomes:

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

图 25

The next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

So, starting the new loop with the current state 1, there are two possible actions: go to state 3, or go to state 5. By lucky draw, our action selected is 5.

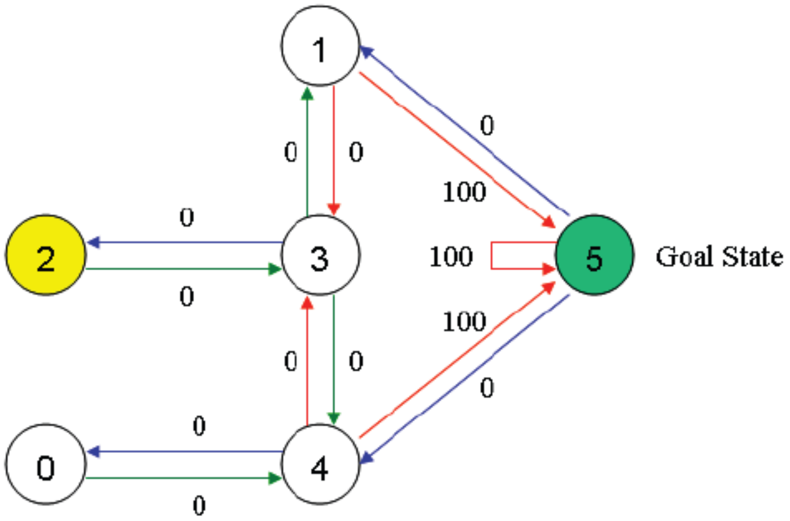


图 26

Now, imaging we're in state 5, there are three possible actions: go to state 1, 4 or 5. We compute the Q value using the maximum value of these possible actions.

Q(state, action) = R(state, action) + Gamma·Max[Q(next state, all actions)]

Q(1,5) = R(1,5) + 0.8*Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8*0 = 100

The updated entries of matrix Q, Q(5,1), Q(5,4), Q(5,5), are all zero. The result of this computation for Q(1,5) is 100 because of the instant reward from R(5,1). This result does not change the Q matrix.

Because 5 is the goal state, we finish this episode. Our agent's brain now contain updated matrix Q as:

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

图 27

If our agent learns more through further episodes, it will finally reach convergence values in matrix Q like:

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

图 28

This matrix Q, can then be normalized (i.e.; converted to percentage) by dividing all non-zero entries by the highest number (500 in this case):

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

图 29

Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state. Tracing the best sequences of states is as simple as following the links with the highest values at each state.

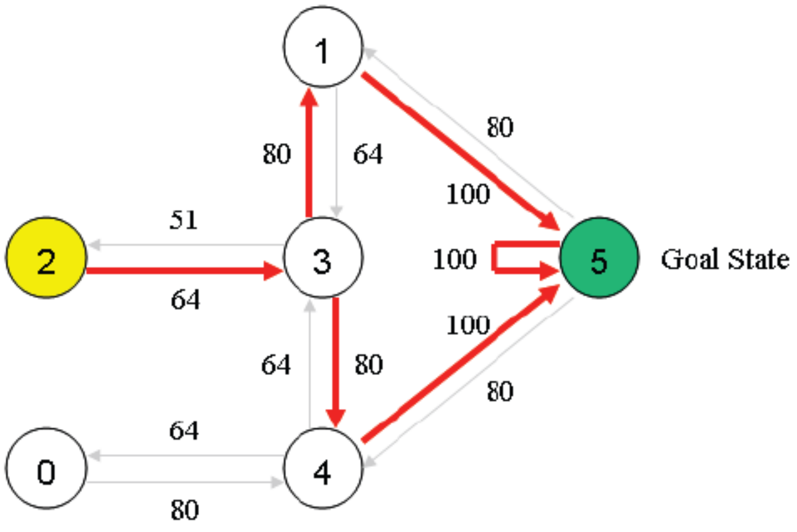


图 30

For example, from initial State 2, the agent can use the matrix Q as a guide:

From State 2 the maximum Q values suggests the action to go to state 3.

From State 3 the maximum Q values suggest two alternatives: go to state 1 or 4. Suppose we arbitrarily choose to go to 1.

From State 1 the maximum Q values suggests the action to go to state 5.

Thus the sequence is 2 - 3 - 1 - 5.

作者: peghoty

出处: http://blog.csdn.net/peghoty/article/details/9361915

欢迎转载/分享, 但请务必声明文章出处.