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

标签： Q-learning state action agent reward

2013-07-18 00:08 29590人阅读 评论(9)

分类： 强化学习

本文是对 http://mnemstudio.org/path-finding-q-learning-tutorial.htm 的翻译，共分两部分，第一部分为中文翻译，第二部分为英文原文。翻译时为方便读者理解，有些地方采用了意译的方式，此外，原文中有几处笔误，在翻译时作了更正。这篇教程通俗易懂，是一份很不错的学习理解 Q-learning 算法工作原理的材料。

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.

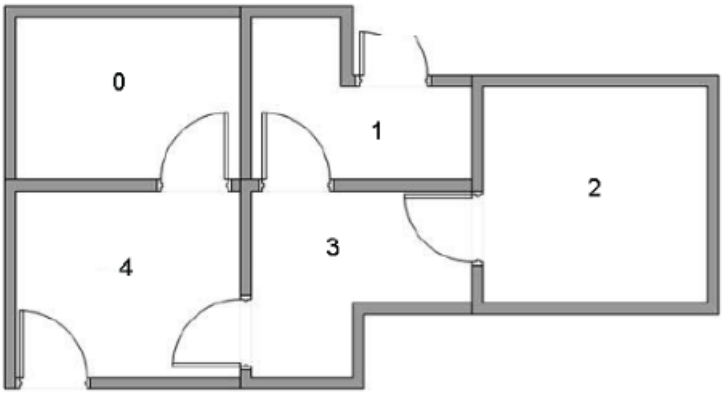


图 1 房间结构

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.

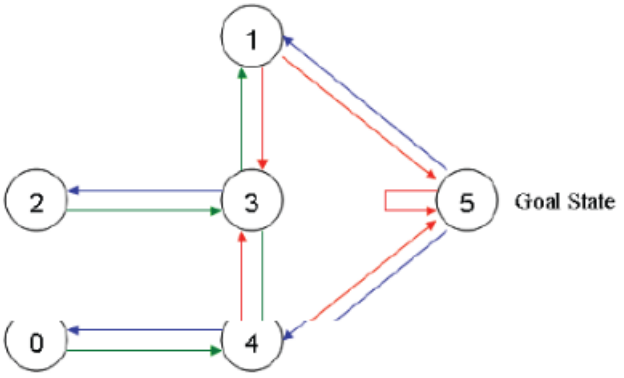


图 2 房间结构对应的图

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, 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 rewa 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 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).

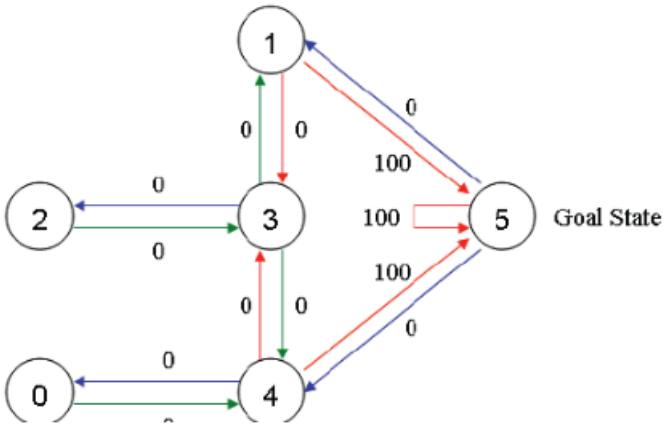


图 3

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, 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.

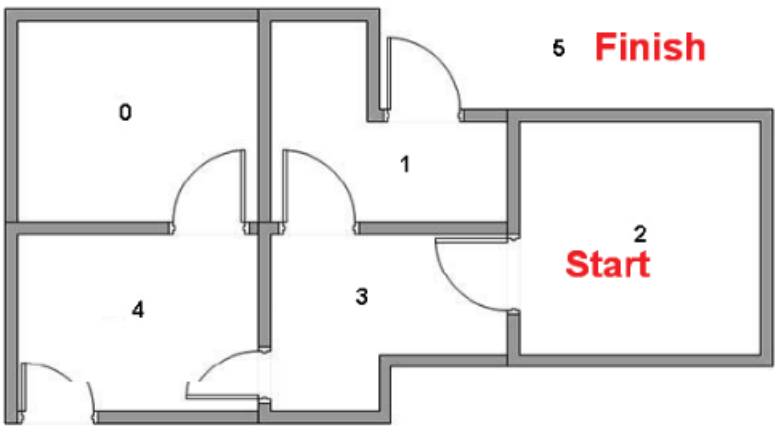


图 4

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

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

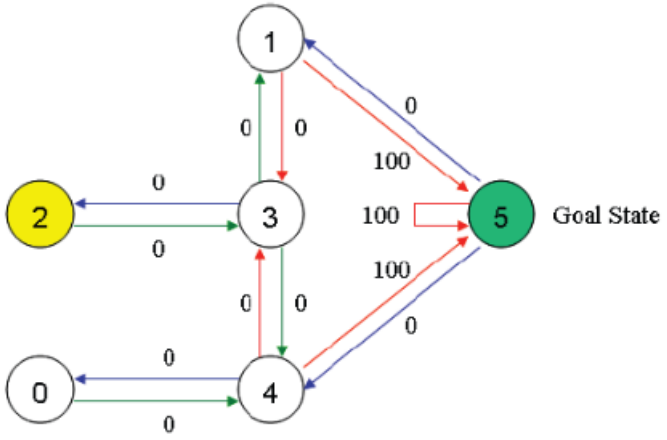


图 5

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, 其的 -1 表示空值 (相应节点之间没有边相连).

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

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. 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:

$$Q(s,a) = R(s,a) + \gamma \cdot \max_{\widetilde{a}}\{Q(\widetilde{s},\widetilde{a})\}, \tag{1}$$

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 $0 \leq \gamma < 1$ 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 **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 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 $s := \widetilde{s}$.

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 sessio 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 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 rewa 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 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 $Q(s,a) = \max_{\widetilde{a}}\{Q(s,\widetilde{a})\}$.
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.

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

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 (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, 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

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

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搜索. 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.

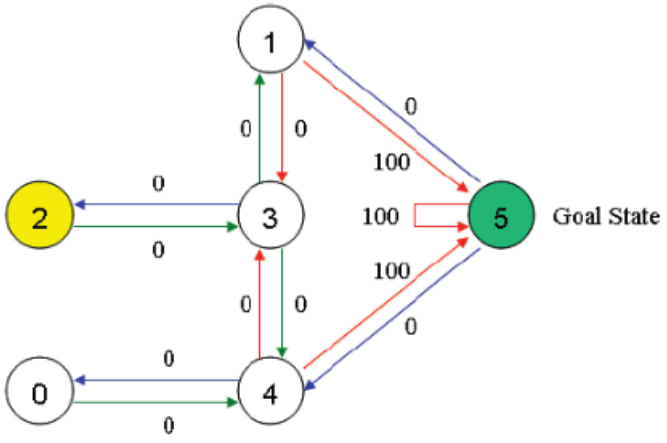


图 11

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

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

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

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

图 12

若我们继续执行更多的 episode, 矩阵 Q 将最终收敛成

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

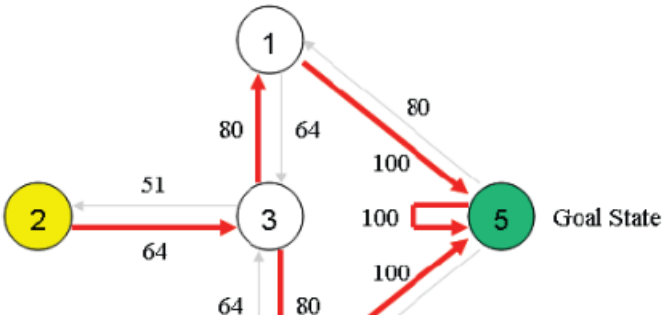
图 13

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 (这里略了百分号)

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

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).



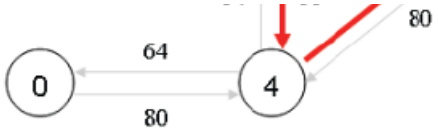


图 15

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.