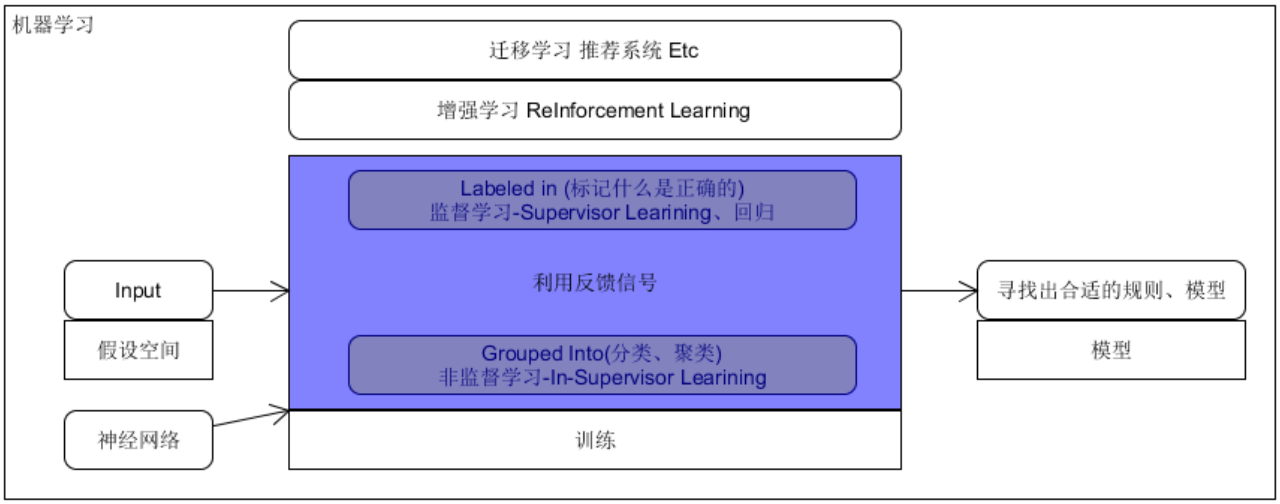
A·I 笔记

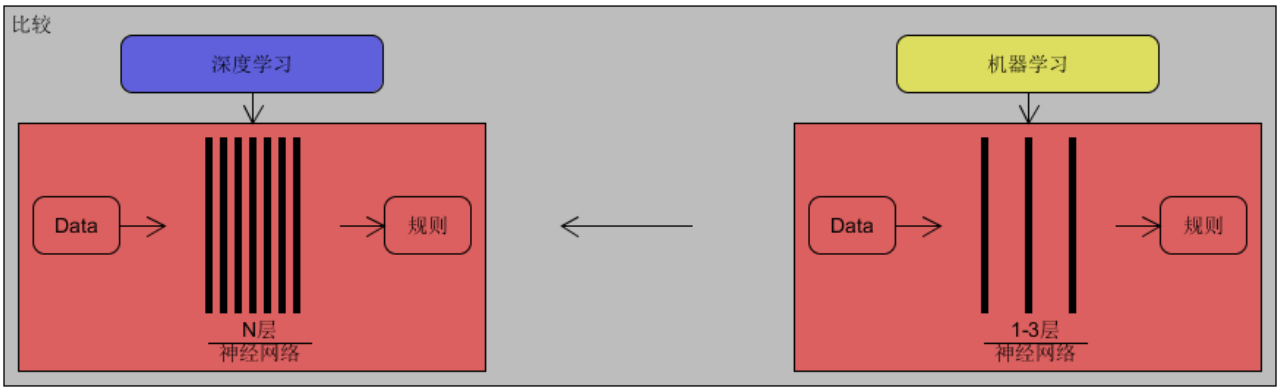
郑华

2019 年 11 月 30 日

第一章　基础概念

1.1　是什么





利用机器学习，人们输入的是数据和从这些数据中预期得到的答案，系统输出的是**规则**。这些规则随后可应用于新的数据，并使计算机自主生成答案。

机器学习的技术定义：在预先定义好的可能性空间中 (数据不同表示-预处理)，利用**反馈信号**的指引来寻找输入数据的有用表示。

机器学习从学习的种类一般分为 3 种：无监督学习、监督学习、强化学习。

监督学习：每一个样本都有明确的标签 (Right Answer), 最后总结出这些训练样本向量与标签的映射关系。

无监督学习：在没有标签的情况下尝试找出其内部蕴含关系的一种挖掘工作，常见的如分类、聚合。

强化学习：本质是解决 decision making 问题，即**自动进行决策**，并且可以做连续决策。

1.1.1 无监督学习

聚类 clustering

1.1.2 监督学习

分类 classifing

回归 regression 通过 70% 的数据训练出规则，通过 30% 剩下的数据进行回归测试拟合。

加入设计的线性关系类似于 $y = f(x) = wx + b$, 则训练函数类似于

$$Loss = \Sigma_{i=1}^{n} |wx_i + b - y_i|$$

1.1.3 迁移学习

专注于存储已有问题的解决模型，并**将其利用在其他不同但相关问题上**。比如说，用来辨识汽车的知识（或者是模型）也可以被用来提升识别卡车的能力。

1.1.4 强化学习

https://blog.csdn.net/j754379117/article/details/83037799

https://www.jianshu.com/p/5ceca53aff0b

强调如何基于环境而行动，以取得最大化的预期利益。其灵感来源于心理学中的行为主义理论，即有机体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为。

本质是解决 decision making 问题，即**自动进行决策**，并且可以做连续决策。

强化学习最早可以追溯到**巴甫洛夫的条件反射实验**，它从动物行为研究和优化控制两个领域独立发展，最终经 Bellman 之手将其抽象为**马尔可夫决策过程** (Markov Decision Process，MDP).

它主要包含四个元素，*agent*，环境状态，行动，奖励, 强化学习的目标就是**获得最多的累计奖励**。

让我们以小孩学习走路来做个形象的**例子**：

小孩想要走路，但在这之前，他需要先站起来，站起来之后还要保持平衡，接下来还要先迈出一条腿，是左腿还是右腿，迈出一步后还要迈出下一步。

小孩就是 **agent**，他试图通过采取**行动**（即行走）来适应**环境**（行走的表面），并且从一个**状态转变**到另一个状态（即他走的每一步），当他完成任务的子任务（即走了几步）时，孩子得到**奖励**（给巧克力吃），并且当他不能走路时，就不会给巧克力。

要素　几大元素分别是：

- **Agent** ，输入通常是状态 State，输出通常是策略 Policy
- **Action** ，就是从一点走到下一点 A -> B, C -> D, etc，动作空间。比如小人玩游戏，只有上下左右可移动，那 Actions 就是上、下、左、右。
- **States** ，就是节点 A, B, C, D, E, F，就是 Agent 的输入
- **Reward** ，就是边上的 cost，进入某个状态时，能带来正奖励或者负奖励。
- **Policy** ，就是完成任务的整条路径 A -> C -> F
- **Environment** ，接收 action，返回 state 和 reward。

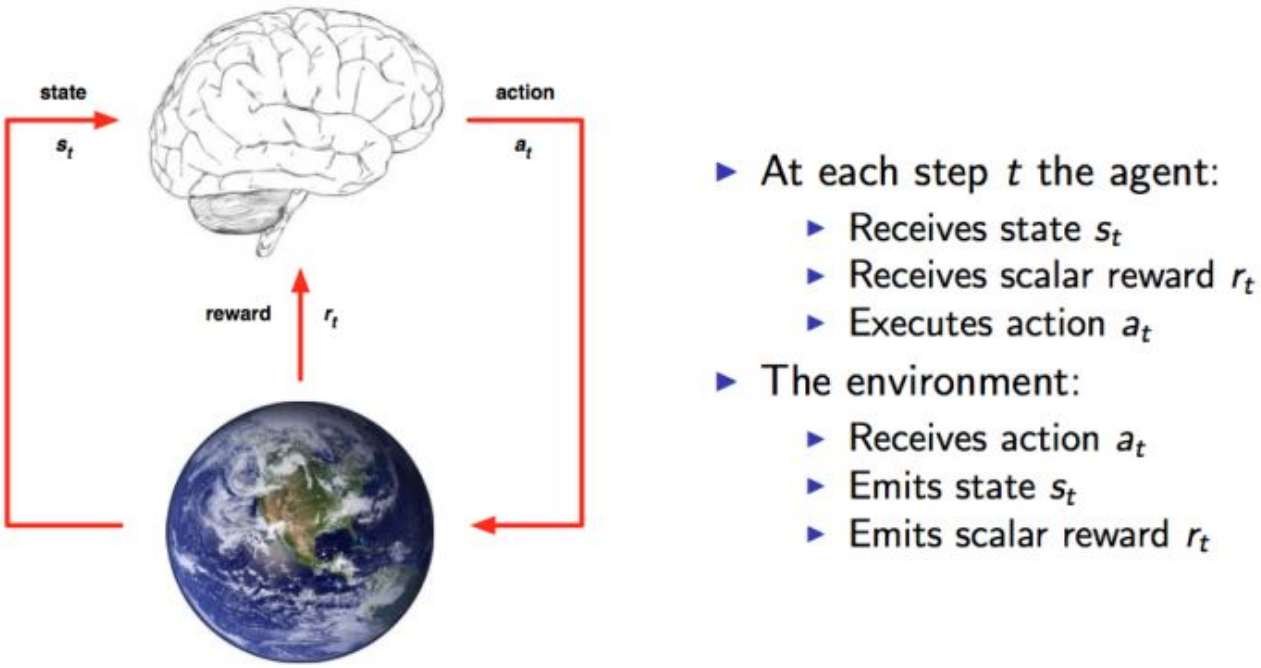


图 1.1: 强化学习示意

首先通过动作 a_t 与环境 env 进行交互，在动作 a_t 和环境 env 的作用下，Agent 会产生新的状态 s_t，同时环境会给出一个**立即回报** r_t。

如此循环下去，智能体与环境进行不断地交互从而产生很多数据。强化学习算法**利用产生的数据修改自身的动作策略**，再与环境交互，产生新的数据，并利用新的数据进一步改善自身的行为，经过数次迭代学习后，智能体能最终地学到完成相应任务的最优动作（最优策略）。

分类　从强化学习的几个元素的角度划分的话，方法主要有下面几类：

- **Policy based**, 关注点是找到最优策略。

- **Value based**, 关注点是找到最优奖励总和。
- **Action based**, 关注点是每一步的最优行动。

特点 强化学习所解决的问题的特点：

- 智能体和环境之间不断进行交互
- 搜索和试错
- 延迟奖励（当前所做的动作可能很多步之后才会产生相应的结果）

收敛条件或目标：

- 获取更多的累积奖励
- 获得更可靠的估计

与其他机器学习的区别

和监督式学习的区别 监督式学习就好比你在学习的时候，有一个导师在旁边指点，他知道怎么是对的怎么是错的.

强化学习会**在没有任何标签的情况下**，通过先尝试做出一些行为得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为，就这样不断的调整，算法能够学习到在什么样的情况下选择什么样的行为可以得到最好的结果。

就好比你有一只还没有训练好的小狗，每当它把屋子弄乱后，就减少美味食物的数量（惩罚），每次表现不错时，就加倍美味食物的数量（奖励），那么小狗最终会学到一个知识，就是把客厅弄乱是不好的行为。

两种学习方式都会学习出输入到输出的一个映射，监督式学习出的是**之间的关系**，可以告诉算法什么样的输入对应着什么样的输出，强化学习出的是**给机器的反馈 reward function**，即用来判断这个行为是好是坏。

强化学习 的**结果反馈有延时**，有时候可能需要走了很多步以后才知道以前的某一步的选择是好还是坏，而监督学习 做了比较坏的选择会**立刻反馈给算法**。

强化学习 面对的**输入总是在变化**,每当算法做出一个行为,它影响下一次决策的输入,而监督学习 的输入是**独立同分布的**。

通过强化学习，一个 agent 可以**在探索和开发（exploration and exploitation）之间做权衡，并且选择一个最大的回报**。exploration 会尝试很多不同的事情，看它们是否比以前尝试过的更好。

exploitation 会尝试过去经验中最有效的行为。

一般的监督学习算法不考虑这种平衡，就只是是 exploitative。

和非监督式学习的区别 非监督式不是学习输入到输出的映射，而是**模式**。例如在向用户推荐新闻文章的任务中，非监督式会找到用户先前已经阅读过类似的文章并向他们推荐其一.

强化学习将通过向用户先推荐少量的新闻，并不断获得来自用户的**反馈**，最后构建用户可能会喜欢的文章的“知识图”。

DQN:Deep-Q-Network 深度强化学习全称是 Deep Reinforcement Learning（DRL），其所带来的推理能力是智能的一个关键特征衡量，真正的让机器有了自我学习、自我思考的能力。

深度强化学习 (Deep Reinforcement Learning，DRL) 本质上属于采用神经网络作为值函数估计器的一类方法，其主要优势在于它能够利用深度神经网络对状态特征进行自动抽取，避免了人工定义状态特征带来的不准确性，使得 *Agent* 能够在更原始的状态上进行学习。

1.2 视频游戏的 AI 史

`https://www.gameres.com/853687.html`

1.2.1 Fine State Machine

1.2.2 Monte Calor Search Tree

1.2.3 Behavioral Decision Trees

第二章　监督学习

第三章　非监督学习

第四章　迁移学习

第五章　强化学习

5.1　参考

知乎专栏：https://zhuanlan.zhihu.com/p/25498081

博客：https://www.cnblogs.com/jinxulin/p/3517377.html

Towards Data: https://towardsdatascience.com/understanding-markov-decision-processes-b5

通过例子了解强化学习：http://www.sohu.com/a/228536039_129720 +
https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learni

5.2　基本概念

5.3　MDP-马尔可夫决策

5.3.1　马尔科夫性

马尔科夫性是指系统的下一个状态 s_{t+1} 仅与当前状态 s_t 有关，而与以前的状态无关。

定义：状态 s_t 是马尔科夫的，当且仅当 $P[S_{t+1}|s_t] = P[s_{t+1}|s_1, ..., s_t]$。

定义中可以看到，当前状态 s_t 其实是蕴含了所有相关的历史信息 $s_1, ..., s_t$，一旦当前状态已知，历史信息将会被抛弃。

5.3.2　马尔科夫过程

马尔科夫性描述的是**每个状态的性质**，但真正有用的是如何描述**一个状态序列**。数学中用来描述**随机变量序列**的学科叫**随机过程**。所谓随机过程就是指随机变量序列。

若随机变量序列中的每个状态都是马尔科夫的则称此随机过程为马尔科夫随机过程。

马尔科夫过程的定义：马尔科夫过程是一个二元组 (S, P)，且满足：S 是有限状态集合，P 是状态转移概率。状态转移概率矩阵为：

$$P = \begin{bmatrix} \mathbf{P_{11}} & \cdots & \mathbf{P_{1n}} \\ \vdots & \vdots & \vdots \\ \mathbf{P_{n1}} & \cdots & \mathbf{P_{nn}} \end{bmatrix}$$

示例 一个学生的 7 种**状态**娱乐，课程 1，课程 2，课程 3，考过，睡觉，论文，每种状态之间有一定的**转换概率**。具体如下图所示。

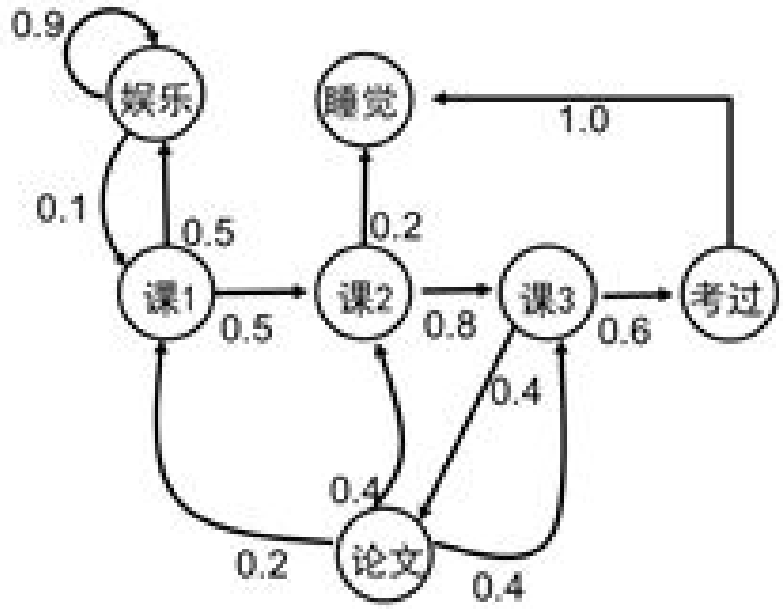


图 5.1: 马尔可夫决策示例

以上状态序列称为马尔科夫链。当给定状态转移概率时，从某个状态出发存在多条**马尔科夫链**。对于游戏或者机器人，马尔科夫过程不足以描述其特点，因为不管是游戏还是机器人，他们都是通过动作与环境进行交互，并从环境中获得奖励，**而马尔科夫过程中不存在动作和奖励**。**将动作（策略）和回报考虑在内的马尔科夫过程称为马尔科夫决策过程**。

5.3.3 马尔科夫决策过程

基本组成 基本组成：五元组 $M = (S, A, P, \gamma, R)$.

- **S**: 表示状态集 (states)，有 $s \in S$，s_i 表示第 i 步的状态。
- **A**: 表示一组动作 (actions)，有 $a \in A$，a_i 表示第 i 步的动作, 由状态与策略函数 π 决定。
- **P**: 表示状态转移概率。s 表示的是在当前 $s \in S$ 状态下，经过 $a \in A$ 作用后，会转移到的其他状态的**概率分布情况**。比如，在状态 s 下执行动作 a，转移到 s' 的概率可以表示为 $p(s'|s, a)$。
- **R**: $S \times A -> \mathbb{R}$，R 是回报函数 (reward function)。有些回报函数状态 S 的函数，可以简化为 $R : S -> \mathbb{R}$。如果一组 (s, a) 转移到了下个状态 s'，那么回报函数可记为 $r(s'|s, a)$。如果 (s, a) 对应的下个状态 s' 是唯一的，那么回报函数也可以记为 $r(s, a)$。
- γ : 折现因子：对未来的价值抱有的希望参数，取值在 $[0, 1]$

状态转移策略 $\pi(s)$ 马尔科夫决策过程的状态转移概率是包含动作的:

$$P^a_{ss1} = P[S_{t+1} = s1|S_t = s, A_t = a]$$

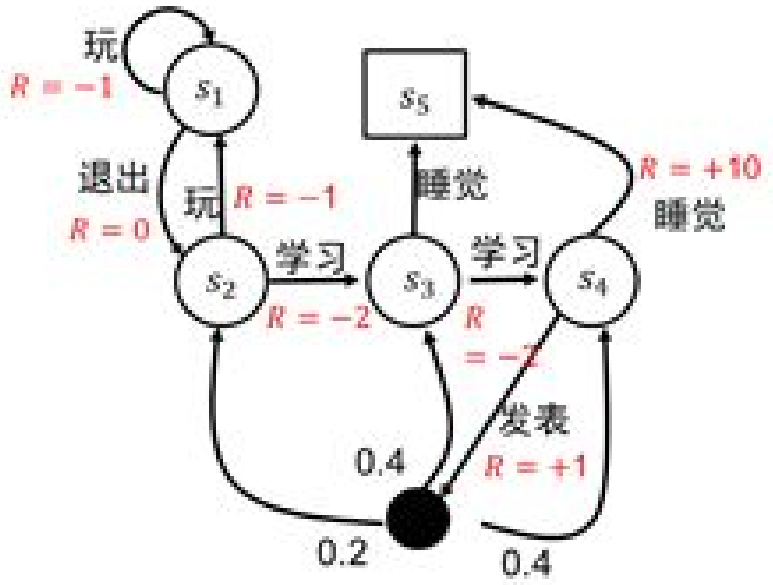


图 5.2: 马尔可夫决策过程示例

学生有五个状态，状态集为 $S = s_1, s_2, s_3, s_4, s_5$，动作集为 $A =$ 玩 退出 学习 发论文 睡觉，在上图中**立即回报** R 用红色标记。

强化学习的**目标**是给定一个马尔科夫决策过程，**寻找最优策略**。所谓策略是指状态到动作的映射，策略常用符号 π 表示，它是指给定状态 s 时，动作集上的一个分布，即

$$\pi(a|s) = p[A_t = a_t|S_t = s] \tag{5.1}$$

策略的定义是用条件概率分布给出的。策略 π 在每个状态 s 指定一个动作概率。如果给出的策略 π 是确定性的，那么策略 π 在每个状态 s 指定一个确定的动作。

例如其中一个学生的策略为 $\pi_1(\text{玩}|s_1) = 0.8$，是指该学生在状态 s_1 时玩的概率为 0.8，不玩的概率是 0.2，显然这个学生更喜欢玩。

另外一个学生的策略为 $\pi_2(\text{玩}|s_1) = 0.3$，是指该学生在状态 [公式] 时玩的概率是 0.3，显然这个学生不爱玩。依此类推，每学生都有自己的策略。

累积回报 $G_t(s)$　**强化学习是找到最优的策略，这里的最优是指得到的总回报最大。**

当给定一个策略 π 时，就可以计算**累积回报**了。首先定义累积回报：

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{5.2}$$

在策略 π 下，可以计算累积回报 G_1，此时 G_1 有多个可能值。**由于策略 π 是随机的，因此累积回报也是随机的。**为了评价状态 s_1 的价值，我们需要定义一个确定量来描述状态 s_1 的价值，很自然的想法是利用累积回报来衡量状态 s_1 的价值。然而，累积回报 G_1 是个随机变量，不是一个确定值，因此无法进行描述。**但其期望是个确定值，可以作为状态值函数 的定义。**

状态值函数 $V_t(s)$　累积回报 G 是个随机变量，不是一个确定值，因此**无法评价**状态 s 的价值。但其期望是个确定值 (**状态值** v_π)，可以作为评价依据。

当 Agent 采用策略 π 时，累积回报服从一个分布，累积回报在状态 s 处的期望值定义为**状态-值函数**：

$$\begin{aligned} v_\pi(s) &= E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s\right] \\ &= R_s + \gamma\sum_{s'\in S} P_{ss'}v(s') \end{aligned} \tag{5.3}$$

--> 状态值函数是与策略 π 相对应的，这是因为策略 π 决定了累积回报 G 的状态分布.

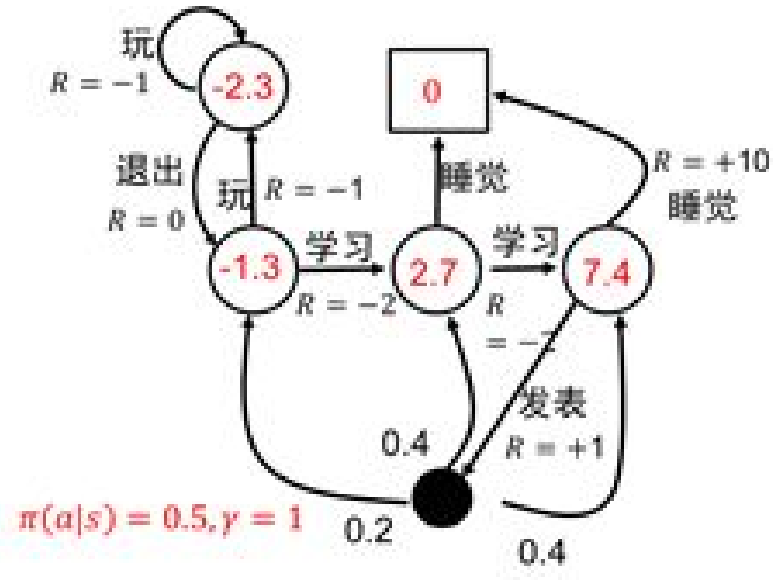


图 5.3: 状态值函数示例

图中白色圆圈中的数值为该状态下的值函数。即：$v_\pi(s_1) = -2.3, v_\pi(s_2) = -1.3, v_\pi(s_3) = 2.7, v_\pi(s_4) = 7.4, v_\pi(s_5) = 0$

状态-行为值函数 $q_\pi(s,a)$　相应地，状态-行为值函数为：

$$\begin{aligned} q_\pi(s,a) &= E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s, A_t=a\right] \\ &= R_s^a + \gamma\sum_{s'\in S} P_{ss'}^a \sum_{a'\in A}\pi(a'|s')q_\pi(s',a') \end{aligned} \tag{5.4}$$

推导见后文。

状态值函数与状态-行为值函数的贝尔曼方程　由状态值函数的定义式可以得到：

$$
\begin{aligned}
v(s) &= E[G_t|S_t = s] \\
&= E[R_{t+1} + \gamma R_{t+2} + \cdots |S_t = s] \\
&= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)|S_t = s] \\
&= E[R_{t+1} + \gamma G(t+1)|S_t = s] \\
&= E[R_{t+1} + \gamma E_{s_{t+1},\ldots}(G(S_{t+1}))] \\
&= E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \\
&= \text{立即回报} + \text{未来回报}
\end{aligned} \tag{5.5}
$$

需要注意的是对哪些变量求期望。

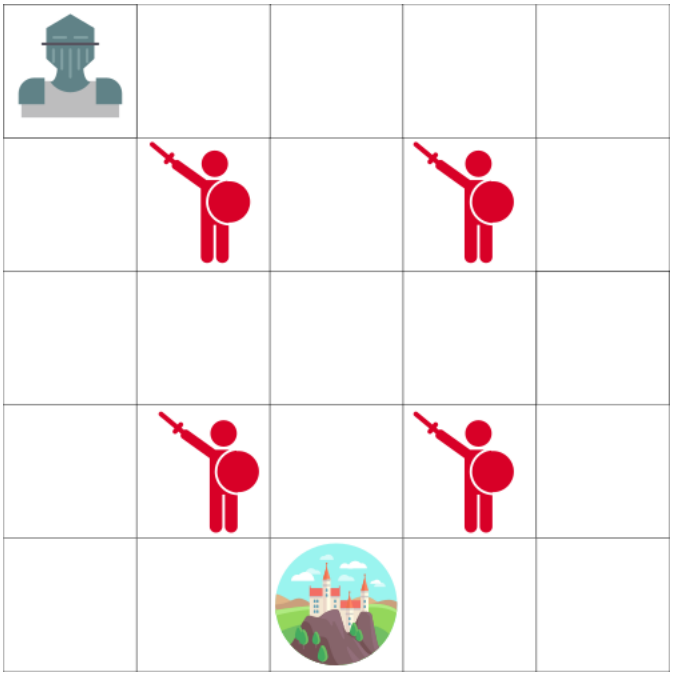
同样我们可以得到状态-动作值函数的贝尔曼方程：

$$
q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{5.6}
$$

5.3.4 Q-learning 从案例到原理

http://www.sohu.com/a/228536039_129720

Environment: Game *the Knight and the Princess*

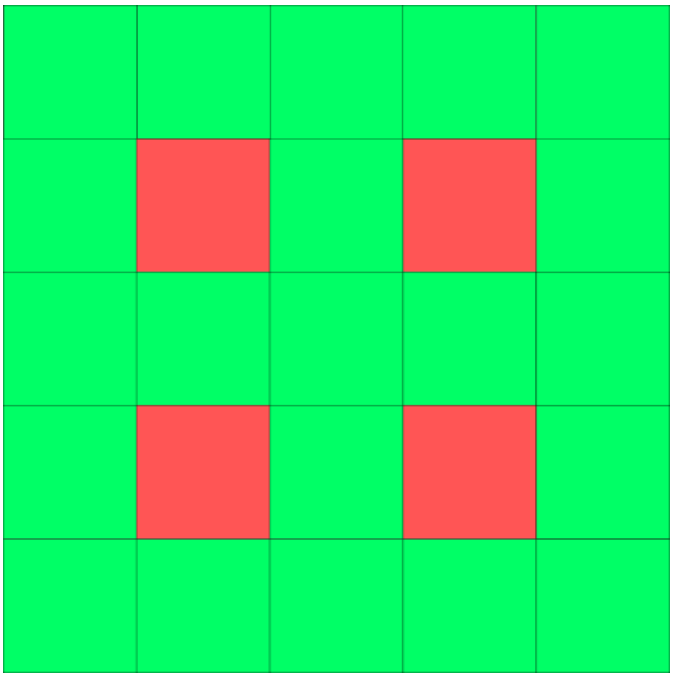


你每次可以移动一个方块的距离。敌人是不能移动的，但是如果你和敌人落在了同一个方块中，你就会死。你的目标是以尽可能快的路线走到城堡去。这可以使用一个「按步积分」系统来评估。

- 你在每一步都会失去 1 分, rewardx = -1;
- 如果碰到了一个敌人，你会失去 100 分，并且训练 episode 结束。rewardEnemy = -100;
- 如果进入到城堡中，你就获胜了，获得 100 分。rewardPrincess = 100;

Q-table:MDP

Policy π_1 第一个策略。假设智能体试图走遍每一个方块，并且将其着色。绿色代表「**安全**」，红色代表「**不安全**」。



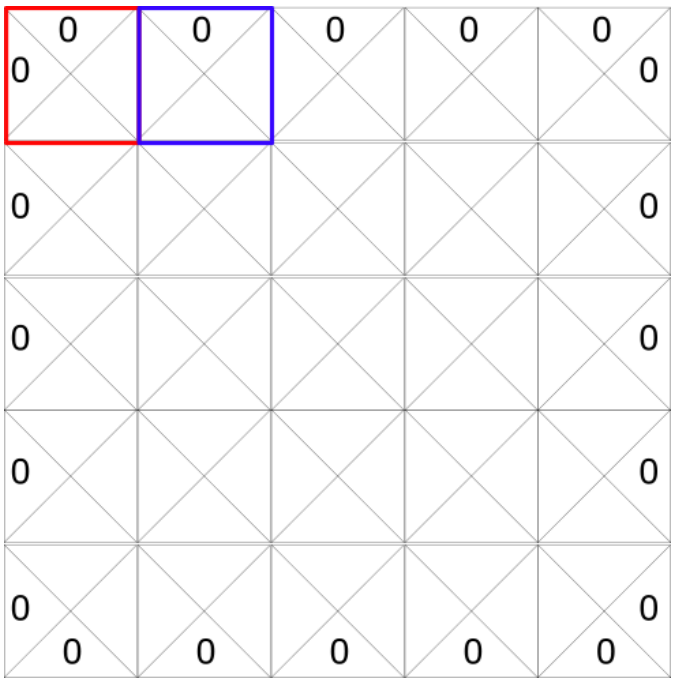
同样的地图，但是被着色了，用于显示哪些方块是可以被安全访问的。接着，我们告诉智能体只能选择绿色的方块。

但问题是，这种策略并不是十分有用。当绿色的方块彼此相邻时，我们不知道选择哪个方块是最好的。所以，Agent 可能会在寻找城堡的过程中陷入无限的循环。

Policy π_2 第二种策略：创建一个表格。通过它，我们可以为每一个**状态（state）**上进行的每一个**动作（action）**计算出最大的未来**奖励（reward）的期望**。

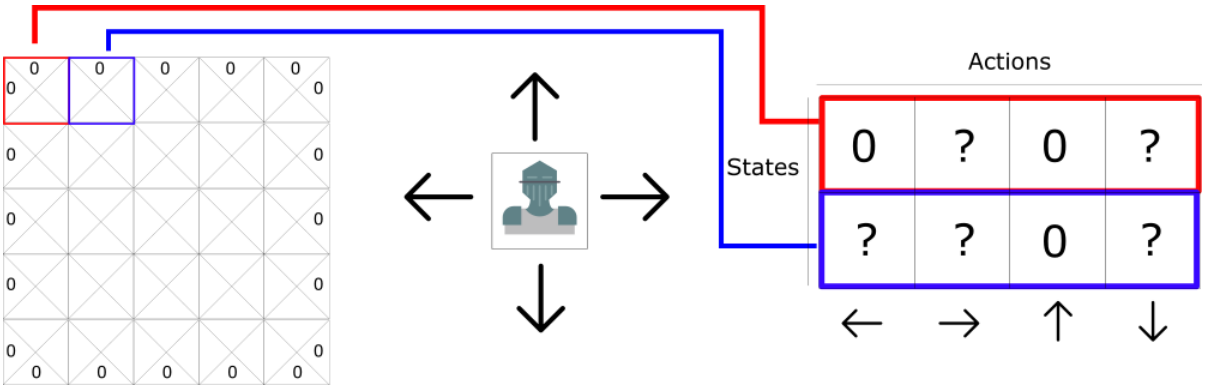
得益于这个表格，我们可以知道为每一个状态采取的最佳动作。

每个状态（方块）允许四种可能的操作：左移、右移、上移、下移。



「*0*」代表不可能的移动（如果你在左上角，你不可能向左移动或者向上移动！）

在计算过程中，我们可以将这个网格转换成一个表。**这种表格被称为 Q-table（「Q」代表动作的「未来价值」）**。每一列将代表四个操作（左、右、上、下），行代表状态。每个单元格的值代表给定状态和相应动作的最大未来奖励期望。



将这个 Q-table 想象成一个「备忘纸条」游戏。得益于此，我们通过寻找每一行中最高的分数，可以知道对于每一个状态（Q-table 中的每一行）来说，可采取的最佳动作是什么。这样就解决了这个城堡问题！但是，如何计算 Q-table 中每个元素的值呢？也就是 MDP 中提到的 BellManEquation。

BellMan-Equation 为了求出 Q-table 中的每个值，**将使用 Q-learning 算法**求解 BellMan Equation.

Q-learning 算法：学习动作值函数 *动作-值函数*（或称「*Q* 函数」）有两个输入：「**状态**」和「**动作**」。它将返回**在该状态下执行该动作的未来奖励期望**。

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t, a_t]$$

Q value for that state given that action

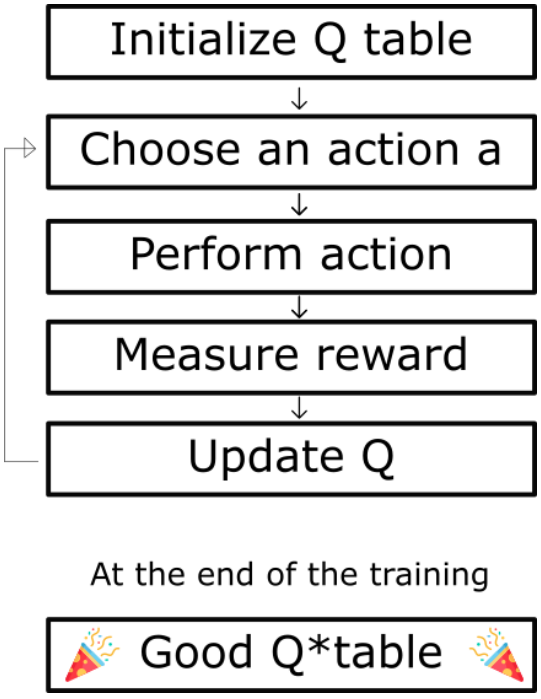
Expected discounted cumulative reward ...

given that state and that action

可以把 Q 函数视为一个在 Q-table 上滚动的读取器，用于寻找与当前状态关联的行以及与动作关联的列。**它会从相匹配的单元格中返回 Q 值。这就是未来奖励的期望。**

在**探索环境（environment）之前**，Q-table 会给出相同的任意的设定值（大多数情况下是 0）。**随着对环境的持续探索**，这个 Q-table 会通过迭代地**使用 Bellman 方程（动态规划方程）更新 Q(s,a) 来给出越来越好的近似**。具体求解参考动态规范方法一节。

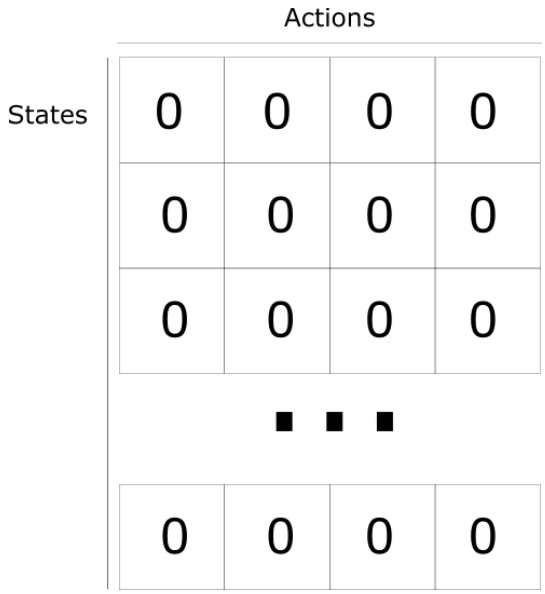
Q-learning 流程 一般计算机模拟流程大概如下：



1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

- **Step 1: Initialize Q-values**

  初始化 Q 值。我们构造了一个 m 列（m = 动作数），n 行（n = 状态数）的 Q-table，并将其中的值**初始化为 0**。



- **Step 2: For life (or until learning is stopped)**

  在整个生命周期中（或者直到训练被中止前），步骤 3 到步骤 5 会一直被重复，直到达到了最大的训练次数（由用户指定）或者手动中止训练。

- **Step 3: Choose an action**

  选取一个动作。在基于当前的 Q 值估计得出的状态 s 下选择一个动作 a。

  但是……如果每个 Q 值都等于零，我们一开始该选择什么动作呢？请转到 ϵ **epsilon 探索/利用比率**小节。

- **Step 4-5: Evaluate!**

  评价！采用动作 a 并且观察输出的状态 s' 和奖励 r。现在我们更新函数 Q（s，a）。

  我们采用在步骤 3 中选择的动作 a，然后执行这个动作会返回一个新的状态 s' 和奖励 r。接着我们使用 Bellman 方程去更新 Q（s，a）：

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

Maximum expected future reward **given the new s' and all possible actions at that new state**

```
New Q value = Current Q value + lr * [Reward + discount_rate * (highest Q
    value between possible actions from the new state s'  ) — Current Q
    value]
```

Dynamic Solve

α **学习率**　可以将学习率看作是网络**有多快地抛弃旧值、生成新值的度量**。如果学习率是 1，新的估计值会成为新的 Q 值，并完全抛弃旧值。

ϵ **epsilon 探索/利用比率**　在刚开始初始化后，每个 Q 值都等于零，一开始该选择什么动作呢？在这里，就可以看到**探索/利用（exploration/exploitation）**的权衡有多重要了。

思路就是，在一开始，将使用 **epsilon 贪婪策略**：

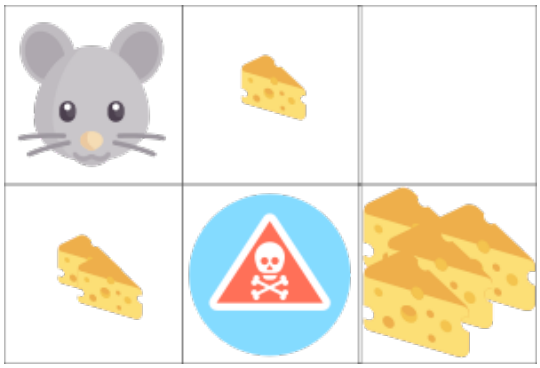
1. 指定一个**探索速率「epsilon」**，一开始将它设定为 1。这个就是我们将随机采用的步长。在一开始，这个速率应该处于最大值，因为我们不知道 Q-table 中任何的值。这意味着，我们需要通过随机选择动作进行大量的探索。

2. 生成一个随机数。如果**这个数大于 epsilon，那么我们将会进行「利用」**（这意味着我们在每一步利用已经知道的信息选择动作）。否则，我们将继续进行探索。

3. 在刚开始训练 Q 函数时，我们必须有一个大的 epsilon。随着智能体对估算出的 Q 值更有把握，我们将逐渐减小 epsilon。

5.3.5 Q-learning 案例示例

Environment　老鼠奶酪游戏。



- 一块奶酪 = +1

- 两块奶酪 = +2
- 一大堆奶酪 = +10（训练结束）
- 吃到了鼠药 = -10（训练结束）

Step1：初始化 Q-table 初始化

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Step2、3：选择一个动作 从起始点，你可以在向右走和向下走其中选择一个。由于有一个**大的 epsilon 速率**（因为我们至今对于环境一无所知），我们随机地选择一个。例如向右走。

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

我们随机移动（例如向右走）

我们发现了一块奶酪（+1），现在我们可以更新开始时的 Q 值并且向右走，通过 Bellman 方程实现。

Step4、5：更新 Q 函数 更新 Q 函数

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

Maximum expected future reward **given the new s' and all possible actions at that new state**

$$NewQ(start,right) = Q(start,right) + \alpha[\Delta Q(start,right)]$$

$$\Delta Q(start,right) = R(start,right) + \gamma\ max\ Q'(1cheese,a')) - Q(start,right)$$

$$\Delta Q(start,right) = 1{+}0.9{*}max(Q'(1cheese,left), Q'(1cheese,right), Q'(1cheese,down)) - Q(start,right)$$

$$\Delta Q(start,right) = 1 + 0.9 * 0 - 0 = 1$$

$$NewQ(start,right) = 0 + 0.1 * 1 = 0.1$$

- 首先，我们计算 Q 值的改变量 $\Delta Q(start, right)$。
- 接着我们将初始的 Q 值与 $\Delta Q(start, right)$ 和学习率 α 的积相加。

	←	→	↑	↓
Start	0	0.1	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

刚刚更新了第一个 Q 值。现在我们要做的就是**一次又一次地做这个工作直到学习结束**。

5.4 Q-Learning

5.5 Saras

5.6 Deep-Q-Network

第六章　案例分析

第七章　应用