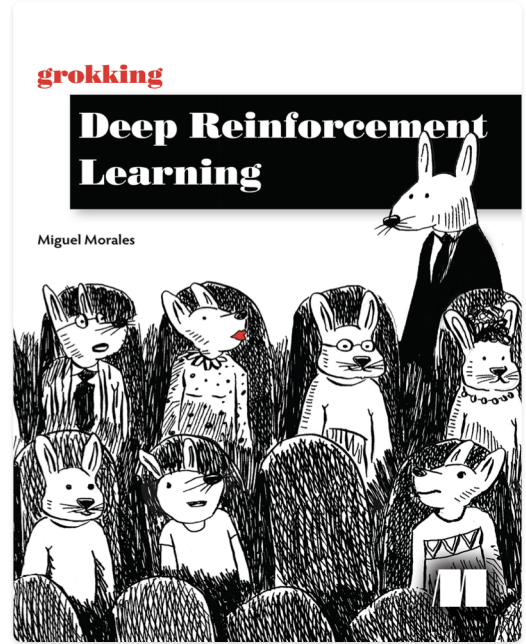
深度强化学习

bifnudozhao@tencent.com

概要

- 强化学习
- 一个场景(Grid World)
- 数学基础与相关概念
  - 组件
  - 概念
- 平衡即时与长期目标
- Q 学习
  - Q 学习网络
  - Q 学习流程
- Demo



Grokking Deep Reinforcement Learning

强化学习

强化学习是一种机器学习方法，或者说框架。

- 说到方法，那就意味着它是用来解决问题的。
- 说到框架，那就意味着它有一些组成部分，这些组成部分可以看作是组件，不同部分的组件，是可以进行替换的。

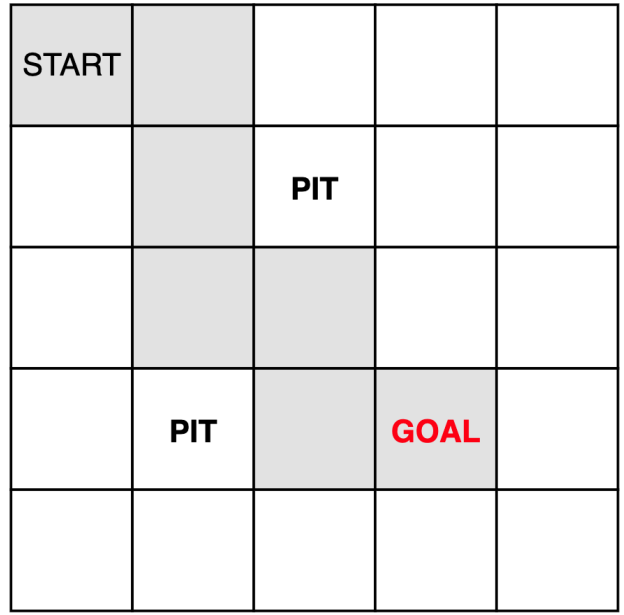
强化学习解决的问题

每种方法都有其适用的问题，强化学习善于解决一些能明确定义的任务，比如下棋，游戏。而强化学习的缺点在于，它需要很多训练样本，另外奖励函数不好定义，因为奖励是驱动整个学习循环的核心，直接关系到学习的质量与效果。

另外有一点值得注意，强化学习是学习解决问题的方法，而不是解决问题本身。比如在下棋这个场景中，强化学习是学习如何下棋；在玩游戏这个场景中，强化学习是学习如何通关（或者获取更多奖励）。更简单的寻路场景，强化学习是学习如何寻路，而不是求解某一个具体的迷宫。

一个场景(Grid World)

右图是一个格子世界的例子，其中灰色格子表明了这个地图的一个解法



说明

- 左上角为起点
- GOAL 为终点
- PIT 为深渊，掉进去就死了

目标

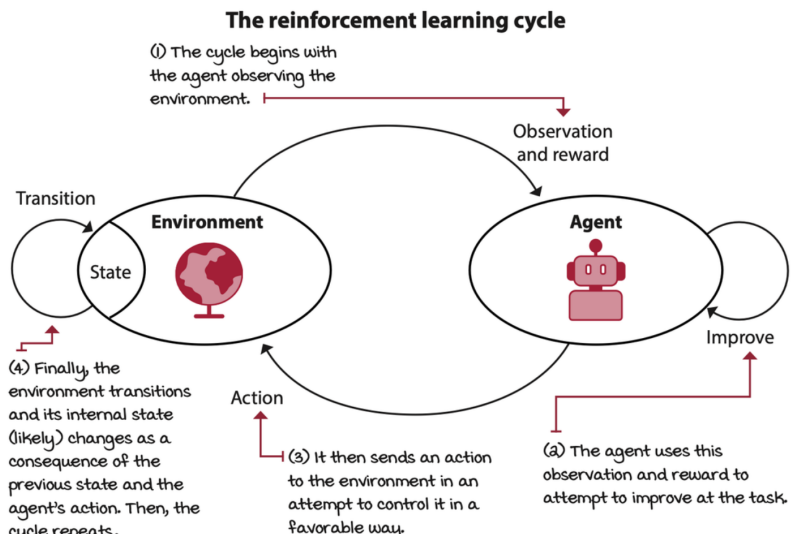
找出从起点到终点的路。其背后隐含的难点在于，需要学习理解地图不同类型格子所带来的影响，并且需要学会的是走随机地图，而不是右图展示这个固定的地图。

强化学习框架

标准的强化学习过程如下。

1. 智能体（agent）向环境执行一个动作（action）。
2. 环境接收到动作之后做出响应，状态（state）发生了变化，并且告诉智能体一个奖励（reward）

整个学习循环的目标是尽可能最大化收益。



数学基础与相关概念 - 组件

智能体（Agent）

智能体主要是用来做决策的，这点和人工智能领域里的智能体基本一致。它主要工作流程是从环境中收集（感知）信息，对信息进行评价，然后做出决策。这个过程是可以不断进行改进的。

环境（Environment）

环境是智能体存在的空间，除了智能体之外的其他部分都能看成是环境。而强化学习中的环境，一般是用马尔科夫过程（Markov decision proces）进行建模的。每个环境都由一组变量进行定义。

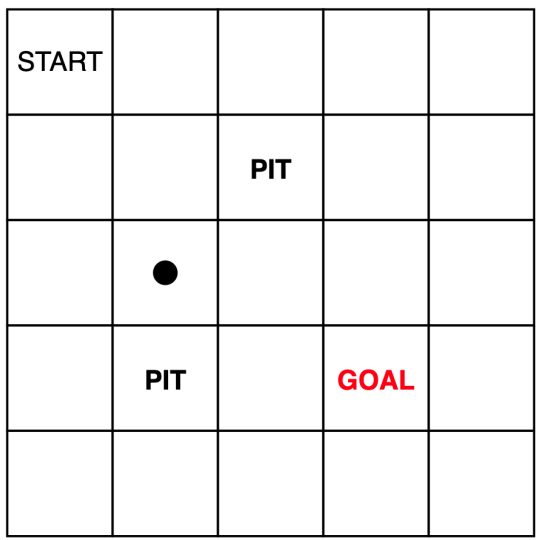
- **状态空间**：每一个时刻的环境状态称为一个**状态**，所有这些状态的集合称为**状态空间**。
- **观察空间**：环境中并不是所有状态都能被智能体感知，智能体能观察到的部分称为**观察空间**。
- **动作空间**：对于每一个状态，智能体都有一组可以执行的动作，所有状态的可执行动作集合，称为**动作空间**。
- **转换函数**：智能体通过动作与环境进行交互，环境在接受了一个动作之后，会从一个状态变换到另一个状态，这个转换方法称为**转换函数**。

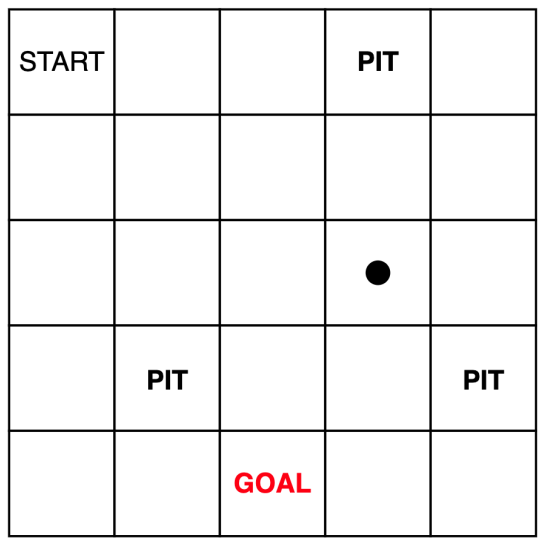
数学基础与相关概念 - 概念

状态

状态是描述环境在某一时刻的一组参数，是一个多维向量。在马尔科夫过程中，有一个基本假设是：**状态是无记忆的**。也就是当前状态与之前的状态是独立的，通过条件表达式可以表示为：

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \dots)$$

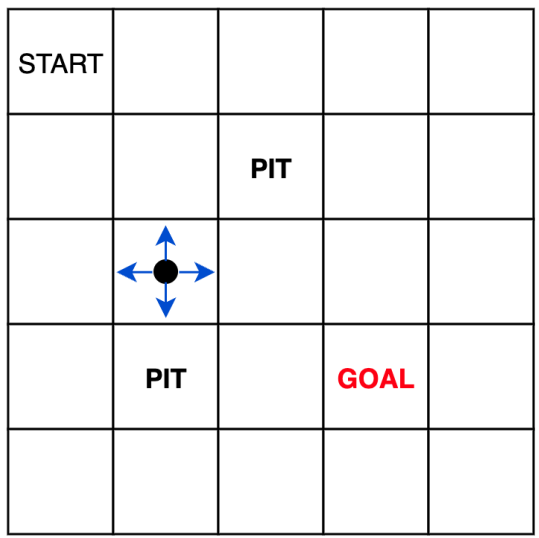


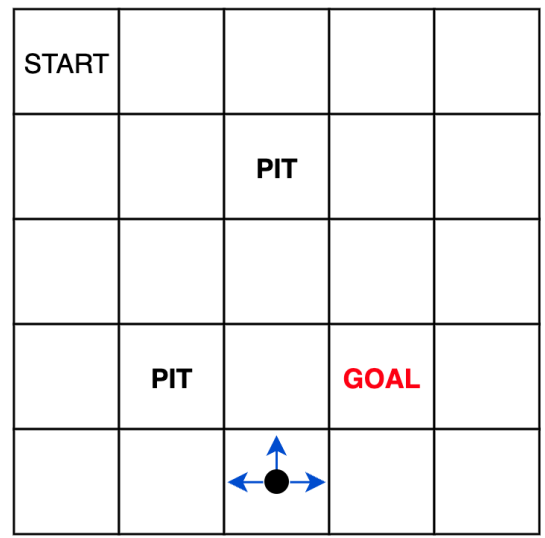


数学基础与相关概念 - 概念

动作

动作是智能体与环境交互的方式，马尔科夫过程可以对于任何给定的状态，返回一系列可以执行的动作，记为$A(s)$。



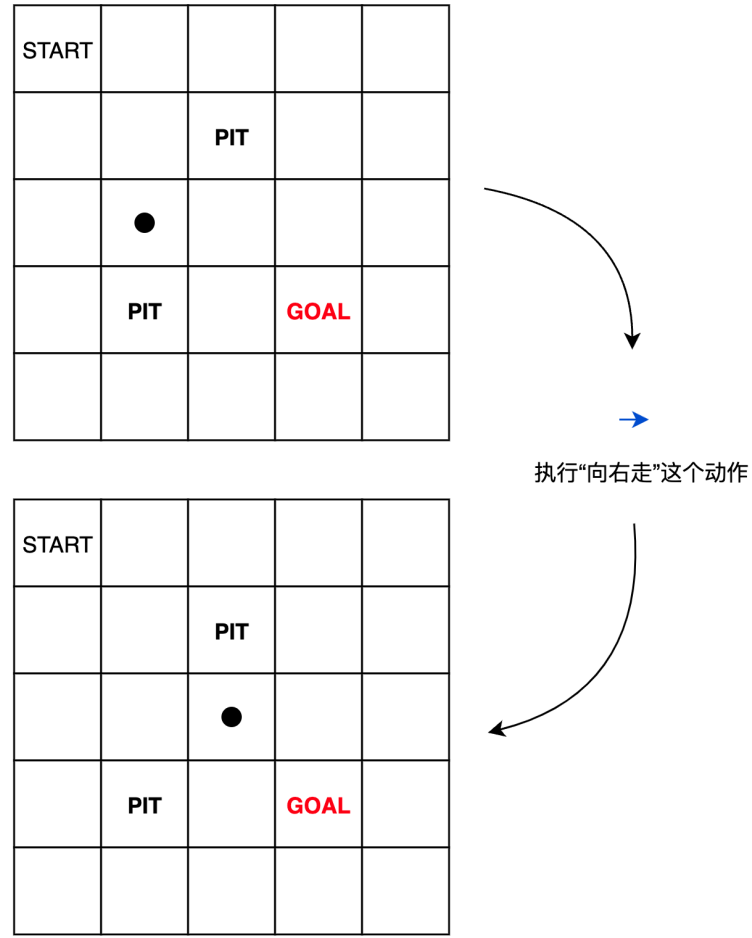


数学基础与相关概念 - 概念

转换函数

状态转换函数标记为 $T(s, a, s')$，这个函数返回一个概率，表明在状态 s 下，执行动作 a，环境转化为状态 s' 的概率。这个转换函数还可以有其他的形式，它可以看成是一个带有状态转移信息的速查表，入参 s, a, s' 都能看成是查询条件。当入参是 s, a, s' 的时候，直接查到的是一个具体的概率；当入参是 s, a 的时候，得到一个概率分布函数，表明执行动作 a 后，环境变换为不同状态的概率。此时的 T 定义如下。

$$p(s'|s,a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a)$$
$$\sum_{s' \in S} p(s'|s,a) = 1, \forall s \in S, \forall a \in A(s)$$

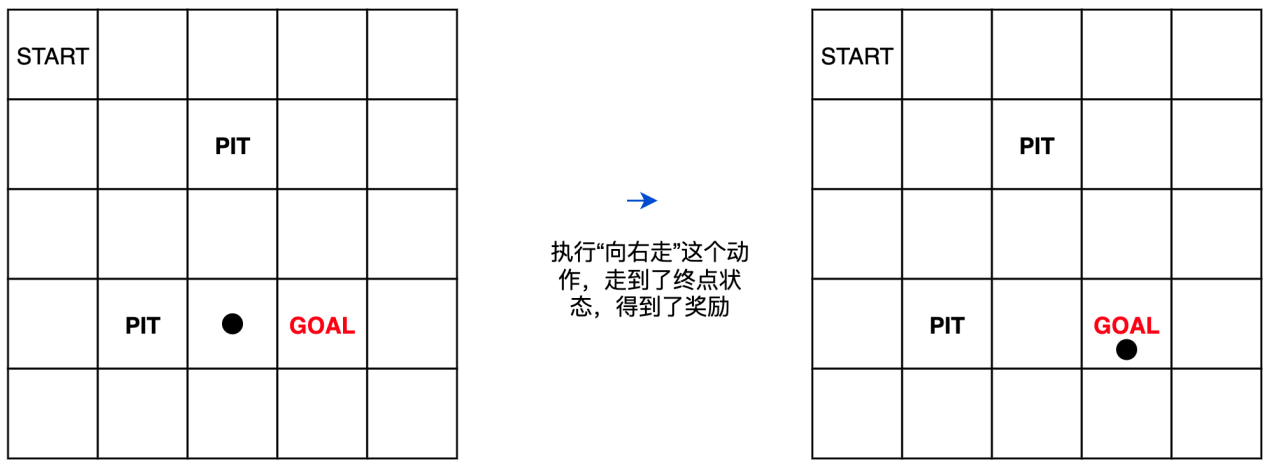


数学基础与相关概念 - 概念

奖励信号

奖励信号是动作执行之后，环境进行了状态转换，此时会伴随发出一个奖励信号。奖励函数 R 可以看成是将一组转移变量映射为一个标量。

$$r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$$
$$R_t \in R \subset \mathbb{R}$$



数学基础与相关概念 - 概念

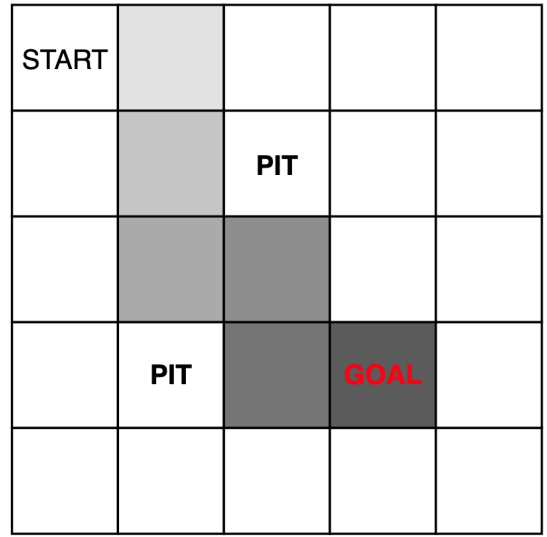
折损收益

更通用的角度来看，在一次流程（episode）里，在每个时刻（step）都可能得到一个奖励，这些奖励会随着时间的递增而折损。假设整个流程持续 T 步，当前在 t 时刻，此时的总收益可以表示为

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

考虑折损的话，实际上是为每个时间步的收益乘以一个逐渐递减的折损因子（discount factor）

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned}$$



平衡即时与长期目标

从上面的概念来看，环境是不可控的部分，需要设计的是智能体。从最终目标来看，智能体的目标是最大化整体收益。但是在一个流程里，每一步都可能有不同收益，有时候一连串的小收益可能会最终得到一个很大的收益，而有时候则相反，一个很大的即时收益之后引来一连串的损失，这些对智能体来说都是不可控并且是未知的，智能体需要在不停的学习中平衡即时与长期的目标。

智能体的目标

智能体需要收益最大化，它实际上需要产出的是一个策略（**policy**）。因为智能体在一个不确定性的环境中，它需要能处理每一种可能出现的状态，以及在这种状态下执行最优的动作。而策略正是以状态为输入，给出执行的动作的概率分布。

下面我们先来看一些衡量方法，智能体需要进行各种衡量，才能不断学习，做出更好的决策，所以在每一个时刻内，智能体其实都能通过现有信息多角度地衡量当前的状态，衡量动作的好坏，甚至衡量这个动作能带来什么好处。

平衡即时与长期目标

状态价值函数

状态价值函数其实是衡量当前状态的价值，比如在进行一个游戏，如果快要赢了，那么当前状态就是一个“比较好”的状态，具有比较高的价值，反之，如果将近输了，那么当前状态就不是一个好状态。

但我们不能凭空评价一个状态，评价必须是建立在“遵循某策略下，当前状态的价值”，如俗话所说“一手好牌也能打的稀烂”，策略才是影响状态价值的关键，所以我们必须是基于某一策略下来讨论一个状态的价值的。

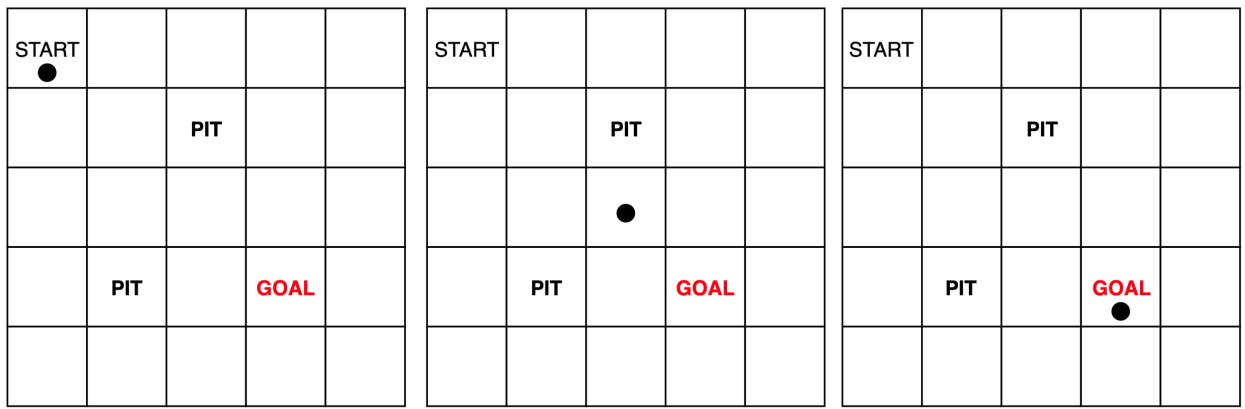
那么一个状态的价值如何衡量呢，在遵循某种策略下，从当前状态开始，得到的**总收益期望**，可以作为这个状态的价值。下面是**状态价值函数**的定义

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t=s] && (1)\\
&= \mathbb{E}_\pi[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|S_t=s] && (2)\\
&= \mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1}|S_t=s] && (3)\\
&= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')], \forall s\in S && (4)
\end{aligned}
$$

平衡即时与长期目标

下标 π 表明我们遵循 π 这个策略，输入为当前状态 s。下面是对上面定义的一个细致解读。

1. 状态价值函数接受当前状态为输入，返回在当前状态下 $S_t = s$，总收益 G_t 的期望值。
2. 将总收益 G_t 进行展开，这里考虑了奖励的折损（折损因子是 γ）。
3. 将总收益的展开式用递归的方式改写（$G_t = R_{t+1} + \gamma G_{t+1}$）。
4. 总收益可以看成：先计算执行某一个动作的期望收益，然后再将所有动作的期望收益相加。



状态 s_1：$v(s_1) = 0$　　状态 s_2：$v(s_2) = 7$　　状态 s_3：$v(s_3) = 10$

平衡即时与长期目标

动作价值函数

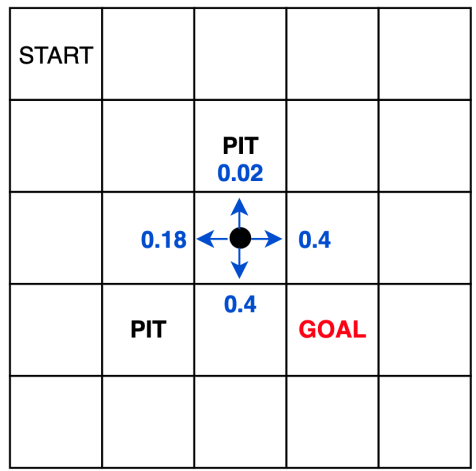
有时候另一个非常有用的评价是动作评价，当执行某个动作的时候，这个动作的价值如何呢？如果我们有一种方式能评价不同的动作，那么我们就可以通过一些策略来选择动作，比如“选择评价最高的动作”。**动作价值函数**，有时候也被称为 **Q 函数**，接受一个当前状态以及可执行的动作，然后返回在这个状态以及动作下，总收益的期望。其定义如下所示

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t=s,A_t=a] && (5)\\
&= \mathbb{E}_\pi[R_t+\gamma G_{t+1}|S_t=s,A_t=a] && (6)\\
&= \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')], \forall s\in S, \forall a\in A && (7)
\end{aligned}
$$

平衡即时与长期目标

动作价值函数(解释)

1. 动作价值函数接受当前状态为输入，返回在当前状态下 $S_t = s$ 执行动作 $A_t = a$，总收益 G_t 的期望值。
2. 将总收益 G_t 进行展开，这里考虑了奖励的折损（折损因子是 γ）。
3. 和状态价值函数所不同的是，动作价值函数是计算某个具体动作的收益均值，所以不需要再对动作进行求和。



平衡即时与长期目标

动作优势函数

策略返回的是关于动作的概率密度函数，此时我们可以基于这个概率分布进行动作的选择，概率大的动作被选中的概率会相应更大。**动作优势函数（action-advantage function）**从名字可以看出，它是用来衡量，当我选择了动作 a 之后，带来的优势是多少。其定义如下

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

动作优势函数是动作价值函数与状态价值函数只差，这个其实比较容易理解，将动作带来的价值，减去当前状态的价值，就是选择这个动作所带来的优势了。如果为 0，那意味着这个动作毫无优势，如果是负数，那意味着执行这个动作会让我们的收益减少。

平衡即时与长期目标

上面所提到的衡量函数，理论上都存在最优。一个最优的策略是指，在每一种状态下，都能做出收益最多的动作选择。

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S \tag{8}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in S, \forall a \in A(s) \tag{9}$$

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{10}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{11}$$

1. 一个最优策略，实际上是对于状态集中的任何状态，都能获得最大状态价值的策略。
2. 而一个最优的 Q 函数，对于所有状态下的所有动作而言，都选取该情况下令 Q 值达到最大化的选择。
3. 最优策略的递归定义。
4. 最优 Q 函数的递归定义。

Q 学习

Q 学习的更新函数如下所示。

$$\overbrace{Q(S_t, A_t)}^{\text{更新后的 Q 值}} = \overbrace{Q(S_t, A_t)}^{\text{当前的 Q 值}} + \alpha[\overbrace{\underbrace{R_{t+1}}_{\text{奖励}} + \gamma \underbrace{\max Q(S_{t+1}, a)}_{\text{所有动作里最大的 Q 值}}}^{\text{估算目标 target}} - Q(S_t, A_t)]$$

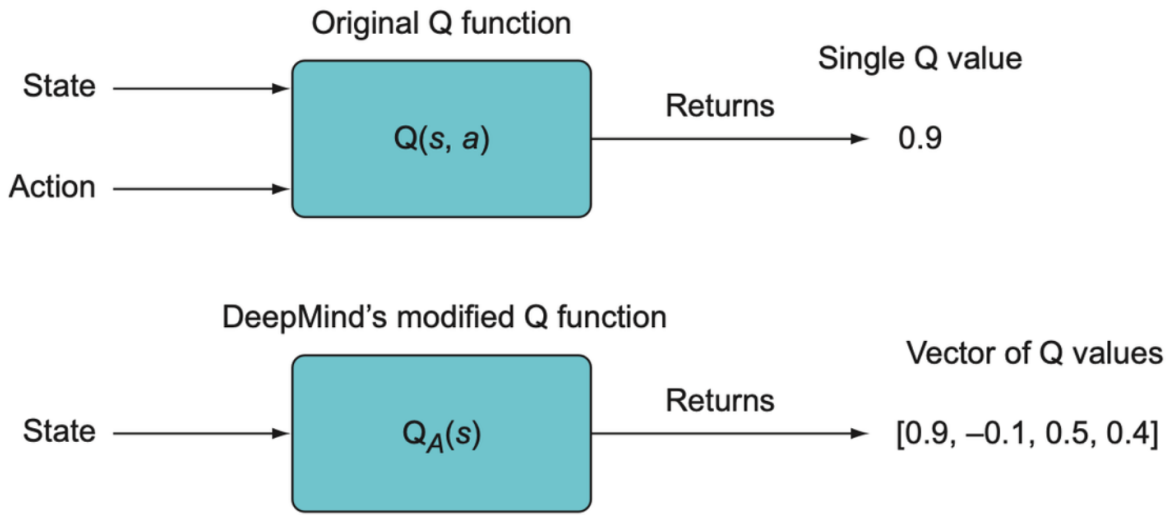
其中 α 是学习率，γ 是折损因子。

Q 学习这种学习目标与策略分离的思路，可以通过一个比喻来理解。假设你现在面临一个难题，此时你根据自己的经验来产生一个自认为最优的动作，并执行这个动作，得到了收益 R_{t+1}，到达一个状态 S_{t+1}，但是对于下一个状态的 Q 值，你不会自己估计，而是别人告诉你，别人经过一番探究，下一个状态的 Q 的最优值能达到某个值 Q^*_{next}。此时你用别人得到的这个“经验”，加上自己获得的收益 R_{t+1} 来估计当前的 Q 值。

Q 学习

Q 学习网络

从上面的式子来看，我们需要学习的是在给定状态以及给定动作下的 Q 值，但是这样比较低效，因为在某个状态下有多个动作。DeepMind 提出的一种改进办法是直接学习某个状态下所有动作的 Q 值。这样后面再通过 ε-greedy 之类的方法对动作进行选择。



Q 学习

Q 学习流程

需要注意的一点是，当我们计算下一个状态的时候，是需要在 `no_grad` 的环境下进行的。对于 pytorch 这种运行时构建动态图的框架来说这点非常重要。我们只希望使用当前状态来学习，而不是需要下一个状态来学习。

由于 `q_network` 是一次过预测给定状态下的所有 Q 值，我们要计算的误差是动作 Q 值之间的误差，真实执行的动作是 `action`，所以需要将 `target` 设置为实际选择的那个动作的 Q 值。

```
for epoch in range(epochs):
    while (not done):
        # 使用 Q 网络预测当前状态下的所有动作 Q 值
        q_values = q_network(state)

        # 用 epsilon-greedy 来选择动作，然后执行动作
        action = epsilon_greedy(actions, p=q_values)
        next_state, reward, done = env.step(action)

        with no_grad():
            # 使用 Q 网络预测下一个状态下的所有动作 Q 值
            new_q_values = q_network(next_state)

        # 选择动作 Q 值最大的动作
        max_q_value = max(new_q_values)

        # 用新的动作计算 target
        y = reward + gamma * max_q_value * (not done)
        target = q_values[action]

        # 计算损失函数以及反向传播
        loss = loss_function(y, target)
        loss.backward()
        optimizer.step()

        state = next_state
```

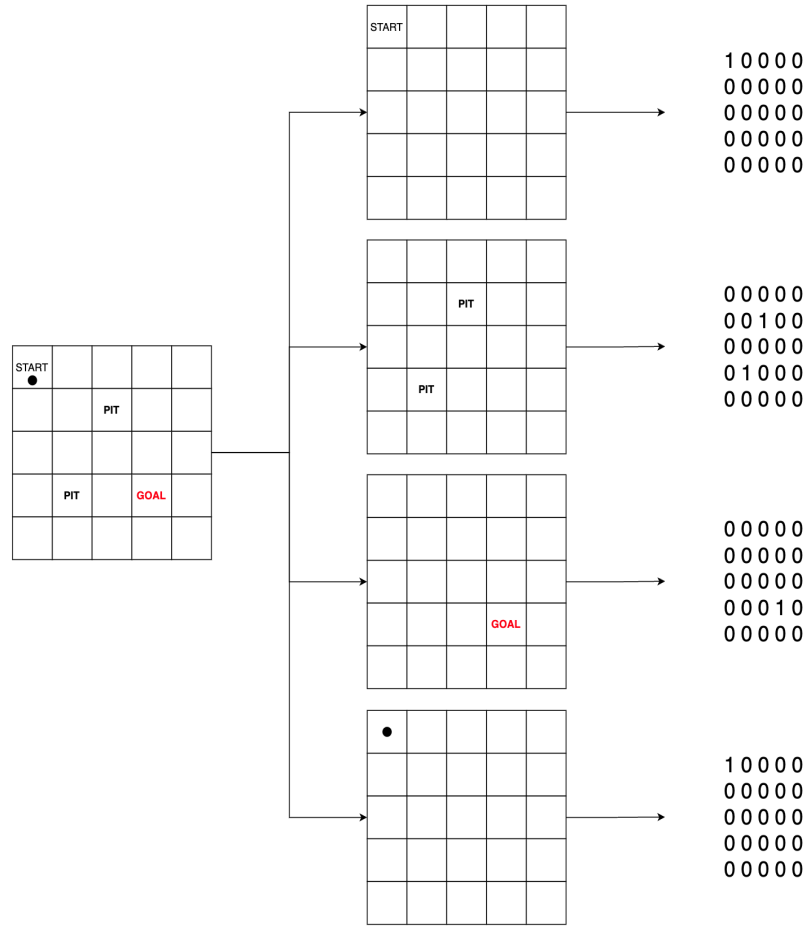
Q 学习

状态编码

由于深度神经网络只能接受数字作为输入，所以需要将 GridWorld 转化为深度神经网络可以处理的状态。

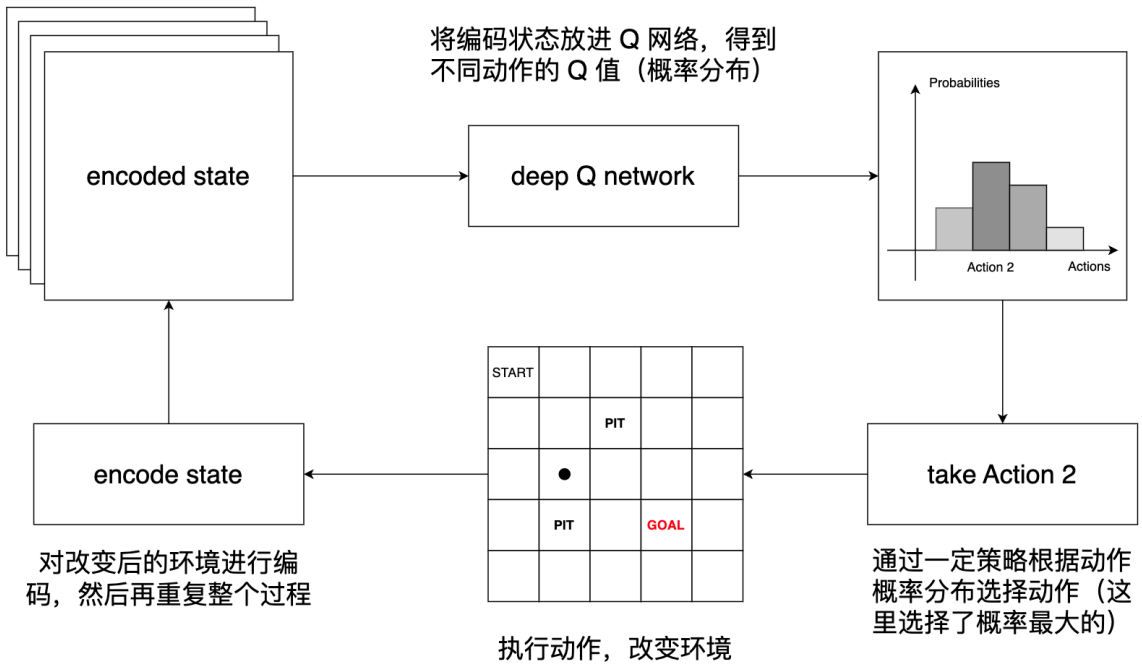
- 将 GridWorld 信息分层拆分
- 方便深度学习感知到地图中的不同成分
- 在每个信息层，将有信息和无信息用 0/1 标记
- 最后再将多层信息合成一个多维数组

这种方法在强化学习游戏的时候比较普适，AlphaGo 也是采取这种方式对围棋棋盘进行编码。



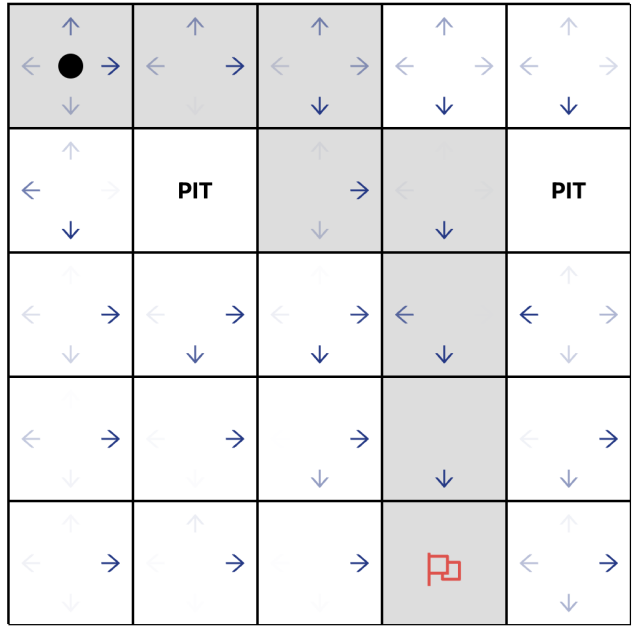
Q 学习

学习流程


encoded state
deep Q network
将编码状态放进 Q 网络，得到
不同动作的 Q 值（概率分布）
Probabilities
Action 2
Actions
take Action 2
通过一定策略根据动作
概率分布选择动作（这
里选择了概率最大的）
START
PIT
PIT
GOAL
执行动作，改变环境
encode state
对改变后的环境进行编
码，然后再重复整个过程

Demo

右图是一个经过学习的 Agent 的策略函数展示



说明

- 每个格子的四向箭头表明四个方向的动作
- 箭头的深浅表明执行该动作的概率大小
- 越深色的概率越大

一些结论

- 学习了不走向边界的方向
- 学习了尽量避开 PIT
- 学习了尽可能朝着终点方向走

Q & A