强化学习（Reinforcement Learning）知识整理

我勒个矗 · 5 个月前

因为准备投入学习 CS294，具体见 知乎专栏，复习了下之前学习 Udacity 和 CS188 中有关强化学习部分的笔记和资料，再看了遍 David Silver 课程的 PPT，整理成了这篇文章。

马尔可夫决策过程（Markov Decision Processes,MDPs）

MDPs 简单说就是一个智能体（Agent）采取行动（Action）从而改变自己的状态（State）获得奖励（Reward）与环境（Environment）发生交互的循环过程。

MDP 的策略完全取决于当前状态（Only present matters），这也是它马尔可夫性质的体现。

其可以简单表示为： $M =< S, A, P_{s,a}, R >$

基本概念

1. $s \in S$: 有限状态 state 集合，s 表示某个特定状态
2. $a \in A$: 有限动作 action 集合，a 表示某个特定动作

3. Transition Model $T(S,a,S') \sim P_r(s'|s,a)$: Transition Model, 根据当前状态 s 和动作 a 预测下一个状态 s'，这里的 P_r 表示从 s 采取行动 a 转移到 s' 的概率
4. Reward $R(s,a) = E[R_{t+1}|s,a]$:表示 agent 采取某个动作后的即时奖励，它还有 R(s, a, s'), R(s) 等表现形式，采用不同的形式，其意义略有不同
5. Policy $\pi(s) \rightarrow a$: 根据当前 state 来产生 action，可表现为 $a = \pi(s)$ 或 $\pi(a|s) = P[a|s]$，后者表示某种状态下执行某个动作的概率

回报（Return）：

$U(s_0\, s_1\, s_2\, \cdots)$ 与 折扣率（discount）$\gamma \in [0,1]$: U 代表执行一组 action 后所有状态累计的 reward 之和，但由于直接的 reward 相加在无限时间序列中会导致无偏向，而且会产生状态的无限循环。因此在这个 Utility 函数里引入 γ 折扣率这一概念，令往后的状态所反馈回来的 reward 乘上这个 discount 系数，这样意味着当下的 reward 比未来反馈的 reward 更重要，这也比较符合直觉。定义

$$\begin{aligned} U(s_0\, s_1\, s_2\, \cdots) &= \sum_{t=0}^{\infty} \gamma^t R(s_t) \quad 0 \le \gamma < 1 \\ &\le \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} \end{aligned}$$

由于我们引入了 discount，可以看到我们把一个无限长度的问题转换成了一个拥有最大值上限的问题。

强化学习的目的是最大化长期未来奖励，即寻找最大的 U。（注：回报也作 G 表示）

基于回报（return），我们再引入两个函数

- 状态价值函数：$v(s) = E[U_t|S_t = s]$，意义为基于 t 时刻的状态 s 能获得的未来回报（return）的期望，加入动作选择策略后可表示为 $v_\pi(s) = E_\pi[U_t|S_t = s]$（$U_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$）
- 动作价值函数：$q_\pi = E_\pi[U_t|S_t = s,\ A_t = a]$，意义为基于 t 时刻的状态 s，选择一个 action 后能获得的未来回报（return）的期望

价值函数用来衡量某一状态或动作-状态的优劣，即对智能体来说是否值得选择某一状态或在某一状态下执行某一动作。

MDP 求解

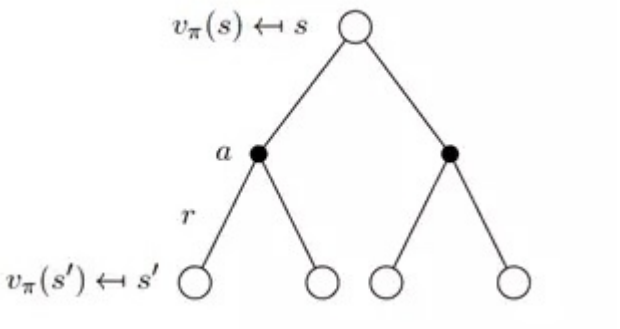
我们需要找到最优的策略使未来回报最大化，求解过程大致可分为两步,具体内容会在后面展开

1. **预测：**给定策略，评估相应的状态价值函数和状态-动作价值函数
2. **行动：**根据价值函数得到当前状态对应的最优动作

Bellman 期望方程

Bellman 方程的分析

为了更加了解方程中期望的具体形式，可以见下图，第一层的空心圆代表当前状态（state），向下连接的实心圆代表当前状态可以执行两个动作，第三层代表执行完某个动作后可能到达的状态 s'。



根据上图得出状态价值函数公式：$v_\pi(s) = E[U_t|S_t = s] = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$

其中，$P_{ss\text{ß}}^a = P(S_{t+1} = s'|S_t = s, A_t = a)$

上式中策略 π 是指给定状态 s 的情况下，动作 a 的概率分布，即 $\pi(a|s) = P(a|s)$

我们将概率和转换为期望，上式等价于：

$$v_\pi(s) = E_\pi[R_s^a + \gamma v_\pi(S_{t+1}|S_t = s]$$

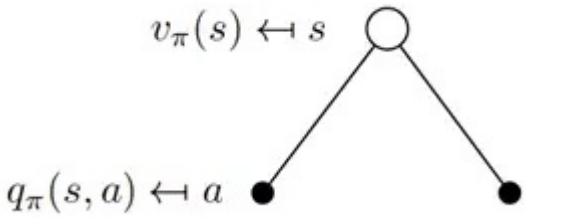
同理，我们可以得到动作价值函数的公式如下：

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

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

如上图，Bellman 方程也可以表达成矩阵形式：$v = R + \gamma P v$，可直接求出$v = (I - \gamma P)^{-1} R$；其复杂度为$O(n^3)$，一般可通过动态规划、蒙特卡洛估计与 Temporal-Difference learning 求解。

状态价值函数和动作价值函数的关系



$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a) = E[q_\pi(s,a)|S_t = s]$$
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

最优方程

最优价值函数（optimal state-value function）

$$v_*(s) = \max_\pi v_\pi(s)$$
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

其意义为所有策略下价值函数的最大值

Bellman最优方程

$$v_*(s) = \max_a q_*(s,a) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') = R_s^a + \gamma \sum_{s\beta \in S} P_{ss'}^a \max_{a\beta} q_*(s',a')$$

- v 描述了处于一个状态的长期最优化价值，即在这个状态下考虑到所有可能发生的后续动作，并且都挑选最优的动作来执行的情况下，这个状态的价值
- q 描述了处于一个状态并执行某个动作后所带来的长期最优价值，即在这个状态下执行某一特定动作后，考虑再之后所有可能处于的状态并且在这些状态下总是选取最优动作来执行所带来的长期价值

最优策略（Optimal Policy）

关于收敛性：（对策略定义一个偏序）

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

定理：

对于任意 MDP：

- 总是存在一个最优策略 π_* ，它比其它任何策略都要好，或者至少一样好
- 所有最优决策都达到最优值函数， $v_{\pi_*}(s) = v_*(s)$
- 所有最优决策都达到最优行动值函数， $q_{\pi_*}(s,a) = q_*(s,a)$

最优策略可从最优状态价值函数或者最优动作价值函数得出：

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0, & otherwise \end{cases}$$

求解 Bellman 最优方程

通过解 Bellman 最优性方程找一个最优策略需要以下条件:

- 动态模型已知
- 拥有足够的计算空间和时间
- 系统满足 Markov 特性

所以我们一般采用近似的办法，很多强化学习方法一般也是研究如何近似求解 Bellman 方程，一般有下面几种（后文会做具体讲解）：

- Value Iteration
- Policy Iteration
- Q-learning
- Sarsa

MDPs 还有下面几种扩展形式：

- Infinite and continuous MDPs
- Partially observable MDPs
- Undiscounted, average reward MDPs

动态规划求解 MDPs 的 Planning

动态规划是一种通过把复杂问题划分为子问题，并对自问题进行求解，最后把子问题的解结合起来解决原问题的方法。「动态」是指问题由一系列的状态组成，而且状态能一步步地改变，「规划」即优化每一个子问题。因为MDP 的 Markov 特性，即某一时刻的子问题仅仅取决于上一时刻的子问题的 action，并且 Bellman 方程可以递归地切分子问题，所以我们可以采用动态规划来求解 Bellman 方程。

MDP 的问题主要分两类

- Prediction 问题
  - 输入：MDP $< S, A, P, R, \gamma >$ 和策略（policy）π
  - 输出：状态价值函数 v_π
- Control 问题
  - 输入：MDP $< S, A, P, R, \gamma >$
  - 输出：最优状态价值函数 v_* 和最优策略 π

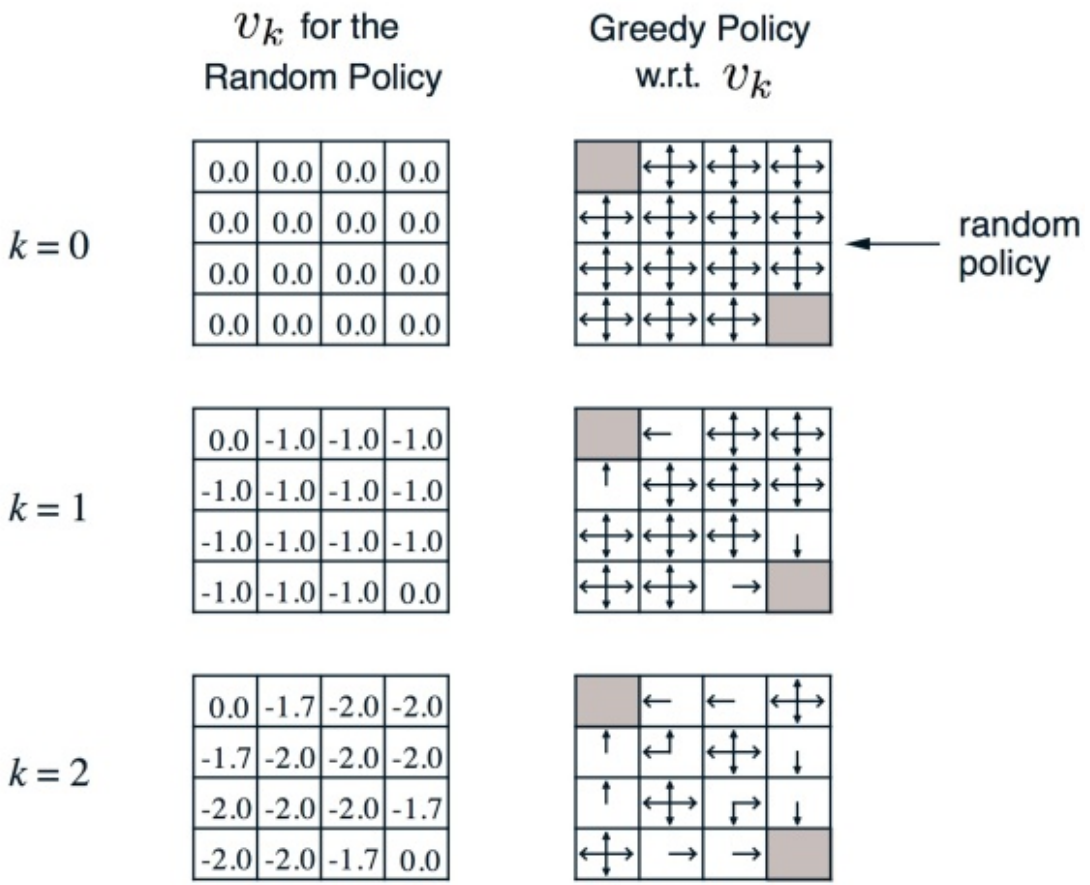
解决也是分两种，见下文

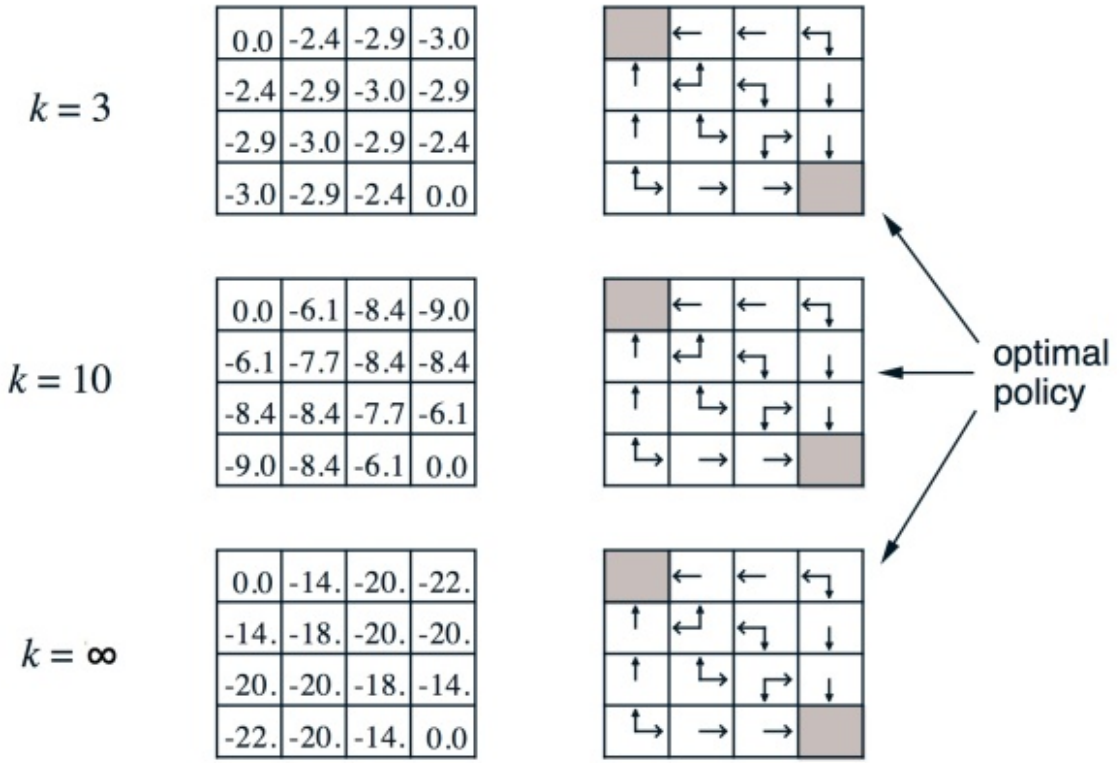
Policy Iteration

步骤：

- Iterative Policy Evaluation:
  - 基于当前的 Policy 计算出每个状态的 Value function
  - 同步更新：每次迭代更新所有的状态的 v

    $$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$
  - 矩阵形式： $\mathbf{v^{k+1}} = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v^k}$


v_k for the Random Policy
Greedy Policy w.r.t. v_k
$k = 0$
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
random policy
$k = 1$
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
$k = 2$
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0



- 左边是第 k 次迭代每个 state 上状态价值函数的值，右边是通过贪心（greedy）算法找到策略
- 计算实例：
  - k=2, -1.7 $\approx$ -1.75 = 0.25*(-1+0) + 0.25*(-1-1) + 0.25*(-1-1) + 0.25*(-1-1)
  - k=3, -2.9 $\approx$ -2.925 = -0.25*(-1-2) + 0.25*(-1-2) + 0.25*(-1-2) + 0.25*(-1-1.7)

- Policy Improvement

- 基于当前的状态价值函数（value function），用贪心算法找到最优策略 $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$
- v_π 会一直迭代到收敛，具体证明如图:

  - This improves the value from any state s over one step,

    $$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

  - It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

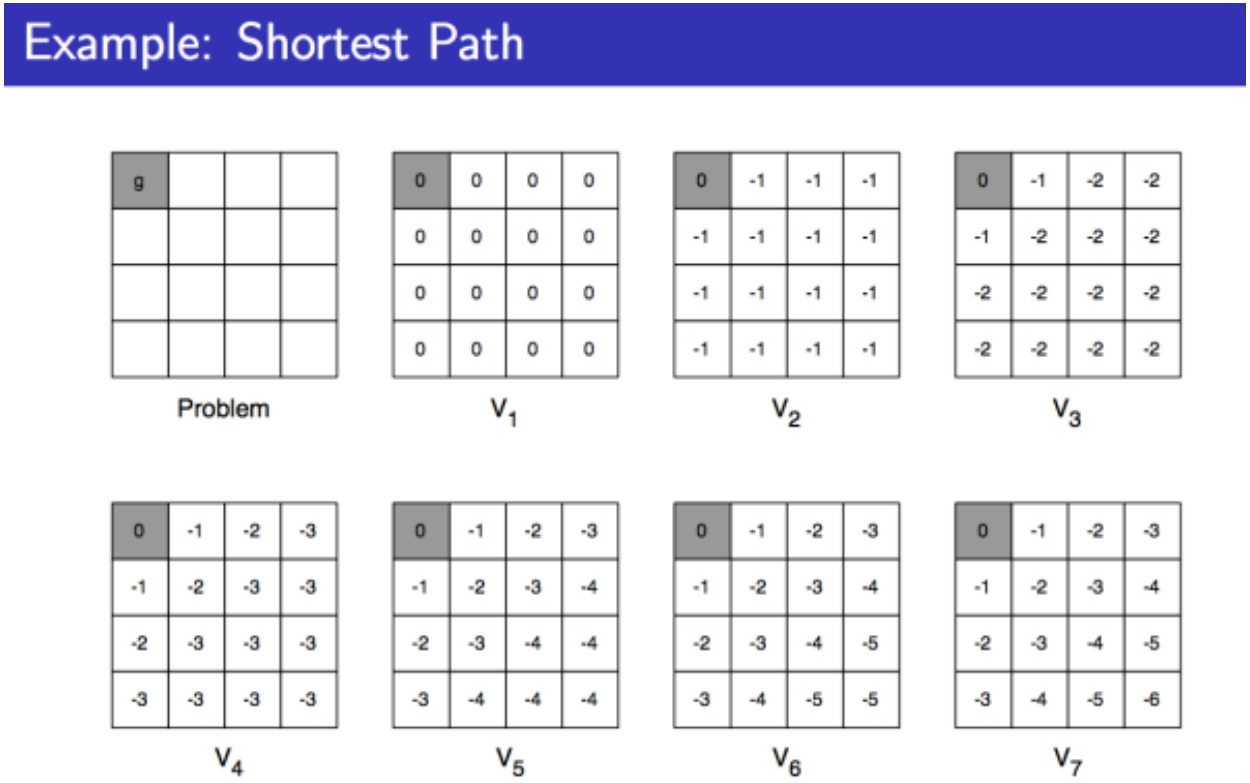
    $$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

- 扩展
  - 事实上在大多数情况下 Policy evaluation 不必要非常逼近最优值，这时我们通常引入 $\epsilon - convergence$ 函数来控制迭代停止
  - 很多情况下价值函数还未完全收敛，Policy 就已经最优，所以在每次迭代之后都可以更新策略（Policy），当策略无变化时停止迭代

Value Iteration

- 最优化原理：当且仅当状态 s 达到任意能到达的状态 s‘ 时，价值函数 v 能在当前策略（policy）下达到最优，即 $v_\pi(s') = v_*(s')$，与此同时，状态 s 也能基于当前策略达到最优，即 $v_\pi(s) = v_*(s)$

- 状态转移公式为：$v_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$
- 矩阵形式为：$\mathbf{v_{k+1}} = \max_{a \in A} \mathbf{R_s^a} + \gamma \mathbf{P^a v_k})$
- 下面是一个实例，求每个格子到终点的最短距离，走一步的 reward 是 -1:



同步动态规划算法小结

1. 迭代策略评估（Iterative Policy Evaluation）解决的是 Prediction 问题，使用了贝尔曼期望方程（Bellman Expectation Equation）

2. 策略迭代（Policy Iteration）解决的是 Control 问题，实质是在迭代策略评估之后加一个选择 Policy 的过程，使用的是贝尔曼期望方程和贪心算法
3. 价值迭代（Value Iteration） 解决的是 Control 问题，它并没有直接计算策略（Policy），而是在得到最优的基于策略的价值函数之后推导出最优的 Policy，使用的是贝尔曼最优化方程（Bellman Optimality Equation）

还有关于 Model-Free, Value Function Approximation, Policy Gradient 等等一堆内容明天再弄。。。。

参考资料：

优达学城（Udacity）纳米学位增强学习部分　Reinforcement Learning By David Silver
UC Berkeley CS188 Intro to AI -- Course Materials

强化学习 (Reinforcement Learning)

人工智能 机器学习

115 收藏 分享 举报

文章被以下专栏收录

矗不知其几千万落

就是记下 AI/ML 学习过程里杂七杂八的东西咯 ~

进入专栏

22 条评论

写下你的评论...

David 9

我想这篇文章是很好的补充: http://nooverfit.com/wp/15-%E5%A2%9E%E5%BC%BA%E5%AD%A6%E4%B9%A0101-%E9%97%AA%E7%94%B5%E5%85%A5%E9%97%A8-reinforcement-learning/

5 个月前

张洛阳 回复 **我勒个矗**（作者） 查看对话

你好， 我阅读完代码的程序，发现你的解释有一些疑问具体的请参考https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter04/GridWorld.py， 我的理解应该是0.25*(-1+0)+0.25*(-1-1)+0.25*(-1-1)+0.25*(-1-1)=-1.75， 这样子才对。。

4 个月前

以上为精选评论

武小喵

赞一个

5 个月前

梁小风

你是研究生吗？

5 个月前

我勒个矗（作者） 回复 **梁小风** 查看对话

本科生

5 个月前

jinchi chen

你好，请问下面这个是怎么算的？k=2, -1.7 = -1.0 + 2 x (1/3) x (-1)
我的理解，根据reinforcement learning an introduction 书中(4.5)公式（抱歉没把公式搬上

来），其中的pi（a|s）=1/4, for any a={up,down,right, left}, and R = -1, for s = {1,2,...14}, 有

V_2 (1) = 1/4 * 1* (0 + v_1(0)) (a = left)

+ 1/4 * 0* (-1 + v_1(1)) （a = up)

+ 1/4 * 1* (-1 + v_1(2)) （a = right)

+ 1/4 * 1* (-1 + v_1(5)) （a = down)

= 1/4*(0) + 0 + 1/4 * (-2) + 1/4*(-2) = -1

是我理解错了？请指正，谢谢

5 个月前

banana

强化学习在信息安全方面的应用多吗？我目前只看到在人工智能方面的实现。

5 个月前

潘剑民

会这个有啥用，能讲讲吗？

5 个月前

我勒个矗（作者） 回复 **潘剑民** 查看对话

强化学习用处太大了…机器人控制、无人驾驶，在医疗、股票预测方面也有应用…Alpha Go 原理之一就是 RL

5 个月前

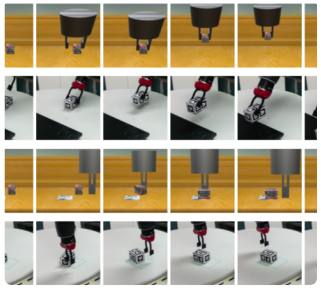
潘剑民 回复 **我勒个矗**（作者） 查看对话

机器人空间、咋控制，能简单说一下原理。我感觉学、机器学习、深度学习，方向就错了。都在用谷歌别的学习库。

5 个月前

1 2 3 下一页

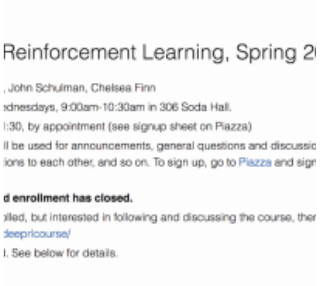
推荐阅读



模仿学习（Imitation Learning）完全介绍（一）

在传统的强化学习任务中，通常通过计算累积奖赏来学习最优策略（policy），这种方式简单直接… 查看全文 ›

我勒个矗 · 5 个月前 · 发表于 矗不知其几千万落



【Berkeley CS 294：深度增强学习，2017年春季学期】学习资源（附字幕）

3 月 9 日更新：课程 2-22 至 3-8 的视频共 5 课已上传至网盘。2 月 20 日更新：2-15 这一课… 查看全文 ›

我勒个矗 · 5 个月前 · 发表于 矗不知其几千万落

欠条可能无效的情形

在欠别人钱的时候，一般都会留存一张用来证明欠钱的欠条。欠条的真实性以及欠条的格式等方面往往可能因为某些原因导致无效，那么对于被欠钱的人来说无疑吃了一个闷亏。下面马律师教教大家认... 查看全文 >

杨锡锋律师 · 4 天前 · 编辑精选

第12期：《趁洪灾哄抢商铺财物，洪水冲不走罪责！》

陕西省榆林市本周起受暴雨侵袭，加上大理河上游有水库发生溃坝，下辖的绥德县城被洪水围困。... 查看全文 >

邓学平律师 · 3 天前 · 编辑精选