initial_h

https://github.com/initial-h

MDP中值函数的求解

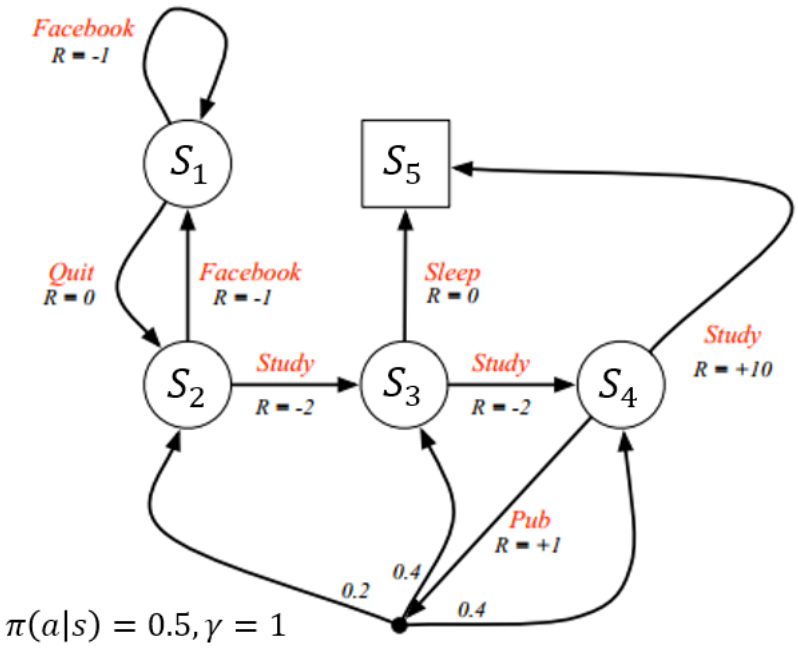
MDP概述

马尔科夫决策过程(Markov Decision Process)是强化学习(reinforcement learning)最基本的模型框架。它对序列化的决策过程做了很多限制。比如状态S_t和动作a_t只有有限个、(S_t, a_t)对应的回报R_t是给定的、状态转移只依赖于当前状态S_t而与之前的状态$S_{t-1}, S_{t-2}, \ldots$无关等等。

当给定一个MDP具体问题，常常需要计算在当前策略π之下，每个状态的值函数的值。而自己真正计算的时候，又不知怎么计算，这里给出一个具体的计算值函数的例子。分别用矩阵运算和方程组求解两种方式给出。

MDP问题设置

如图所示(来源:UCL Course on RL)



该例子可以理解为一个学生的生活状态，在不同的状态，他可以选择学习、睡觉、发文章等等动作，同时获得相应的回报R(reward)。

- 其中状态空间为$\{S_1, S_2, \ldots, S_5\}$,动作空间为{刷Facebook，睡觉，学习，发表文章，弃疗}。
- 每个状态有两个可选动作(S_5为终止状态)，每个动作被选择的概率都为0.5，折扣因子$\gamma = 1$。$(\pi(a|s) = 0.5, \gamma = 1)$

矩阵解法

据图可得如下转移概率矩阵P

$$P = \begin{array}{c|ccccc} & S_1 & S_2 & S_3 & S_4 & S_5 \\ S_1 & 0.5 & 0.5 & 0 & 0 & 0 \\ S_2 & 0.5 & 0 & 0.5 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 0.5 & 0.5 \\ S_4 & 0 & 0.5*0.2 & 0.5*0.4 & 0.5*0.4 & 0.5 \\ S_5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

化简可得

$$P = \begin{array}{c|ccccc} & S_1 & S_2 & S_3 & S_4 & S_5 \\ S_1 & 0.5 & 0.5 & 0 & 0 & 0 \\ S_2 & 0.5 & 0 & 0.5 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 0.5 & 0.5 \\ S_4 & 0 & 0.1 & 0.2 & 0.2 & 0.5 \\ S_5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

其中$P_{11} = 0.5$即表示从状态S_1转移到状态S_1的概率为0.5，其他同理。

每个状态转移的平均reward可如下计算，
例如$R_1 = P[Facebook|S_1] * R_{Facebook} + P[Quit|S_1] * R_{Quit} = 0.5 * (-1) + 0.5 * 0 = -0.5$

则有R的向量如下

$$R = \begin{array}{c} 0.5*(-1)+0.5*0 \\ 0.5*(-1)+0.5*(-2) \\ 0.5*0+0.5*(-2) \\ 0.5*10+0.5*1 \\ 0 \end{array} = \begin{array}{c} -0.5 \\ -1.5 \\ -1 \\ 5.5 \\ 0 \end{array}$$

那么已知转移概率矩阵P和回报向量R，利用贝尔曼方程，则可求得值函数的解v。贝尔曼方程可矩阵表示如下

$$v = R + \gamma P v$$

- 则有

$$\begin{aligned} v &= R + \gamma P v \\ (I - \gamma P)v &= R \\ v &= (I - \gamma P)^{-1} R \end{aligned}$$

其中$\gamma = 1$

- 剩下的就只是矩阵运算。我们运用python numpy计算如下

ments》论文解读(1)

3. AlphaZero并行五子

```
In [10]: P
Out[10]:
array([[0.5, 0.5, 0. , 0. , 0. ],
       [0.5, 0. , 0.5, 0. , 0. ],
       [0. , 0. , 0. , 0.5, 0.5],
       [0. , 0.1, 0.2, 0.2, 0.5],
       [0. , 0. , 0. , 0. , 0. ]])

In [11]: R
Out[11]:
array([[-0.5],
       [-1.5],
       [-1. ],
       [ 5.5],
       [ 0. ]])

In [12]: I
Out[12]:
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])

In [13]: np.dot(np.linalg.inv(I-P),R)
Out[13]:
array([[-2.30769231],
       [-1.30769231],
       [ 2.69230769],
       [ 7.38461538],
       [ 0.        ]])
```

则可得

$$
\begin{aligned}
v_1 &\approx -2.3 \\
v_2 &\approx -1.3 \\
v_3 &\approx 2.7 \\
v_4 &\approx 7.4 \\
v_5 &= 0
\end{aligned}
$$

即为所求。

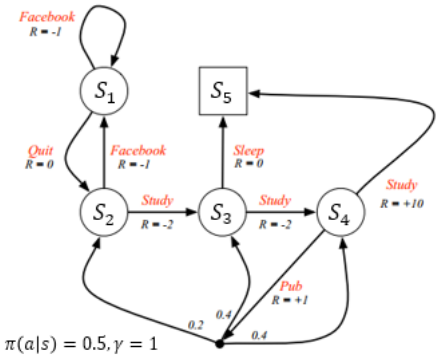
列方程组求解

矩阵运算表达和计算都很简便，但经常会忘记其缘由，为何可以这样求解。那么还可以回归最本质的解方程组的方式求解。即将所有的状态转移用方程的形式罗列出来，然后联立方程组求解即可。这种情况下，我们不必去记贝尔曼方程的矩阵表示形式，只需知道值函数的一步转移公式如下：

$$v_\pi(s_t) = \sum_{a \in A} \pi(a|s_t)(R^a_{s_t} + \gamma \sum_{s_{t+1} \in S} P^a_{s_t s_{t+1}} v_\pi(s_{t+1}))$$

其中，$\pi(a|s_t)$表示在状态s_t下选择动作a的概率，$R^a_{s_t}$表示在状态s_t下选择动作a得到的回报，$P^a_{s_t s_{t+1}}$表示在状态s_t下选择动作a之后转移到状态s_{t+1}的概率(当前问题是确定性的马尔科夫链，选择固定的动作后就一定会转移到确定的状态，即$P^a_{s_t s_{t+1}} = 1$)，$v_\pi(s_{t+1})$表示状态s_{t+1}的值函数。这个式子看起来很麻烦，其实它想表达的就是，前一个状态的值函数等于它能转移到的所有下一状态的值函数和所采取的动作的回报的概率加权之和。

我们用状态S_1举例。



从图中可以看出，在S_1状态下，有两个动作可以选择：刷Facebook或者弃疗。两个动作的选取率都为0.5。刷Facebook的回报是-1，同时又转移到状态S_1。弃疗的回报是0，同时转移到状态S_2。也就是说，在状态S_1的时候，要么刷Facebook并转移到状态S_1，要么弃疗并转移到状态S_2。用上面的数学表达式表示，即是

$$v_\pi(S_1) = 0.5(-1 + v_\pi(S_1)) + 0.5(0 + v_\pi(S_2))$$

对每个状态作同样的状态转移分析，则可求得状态值函数的方程组。这里将$v_\pi(S_i)$记为v_i,得到如下方程组：

$$\begin{aligned}
v_1 &= 0.5(-1 + v_1) + 0.5(0 + v_2) \\
v_2 &= 0.5(-1 + v_1) + 0.5(-2 + v_3) \\
v_3 &= 0.5(-2 + v_4) + 0.5(0 + v_5) \\
v_4 &= 0.5(10 + v_5) + 0.5(1 + 0.2v_2 + 0.4v_3 + 0.4v_4) \\
v_5 &= 0
\end{aligned}$$

化简有：

$$\begin{aligned}
v_1 &= -1 + v_2 \\
v_2 &= -1.5 + 0.5v_1 + 0.5v_3 \\
v_3 &= -1 + 0.5v_4 \\
8v_4 &= 55 + v_2 + 2v_3 \\
v_5 &= 0
\end{aligned}$$

求解方程组可得相同的结果：

$$\begin{aligned}
v_1 &\approx -2.3 \\
v_2 &\approx -1.3 \\
v_3 &\approx 2.7 \\
v_4 &\approx 7.4 \\
v_5 &= 0
\end{aligned}$$

注

需要注意，这里求解的MDP状态值函数，每个状态的回报(reward)是根据转移概率计算的平均回报，这称为贝尔曼期望方程(Bellman Expectation Equation)。而在强化学习中，通常是选取使得价值最大的动作执行，这种解法称为最优值函数(Optimal Value Function)，得到的解不再是期望值函数，而是理论最大值函数。此时相当于策略π已经改变，且非线性，线性方程组不再适用，需要通过数值计算的方式求出。通常的做法是设置状态值函数$v(s)$和状态-动作值函数$q(s,a)$迭代求解。最终得到

$$\begin{aligned}
v^*(s) &= \max_\pi v_\pi(s) \\
q^*(s,a) &= \max_\pi q_\pi(s,a)
\end{aligned}$$

其中通过$q^*(s,a)$选择动作即为最优策略π^*：

$$\pi^*(a|s) = \begin{cases} 1, if \quad a = \arg\max_{a \in A} q^*(s,a) \\ 0, otherwise \end{cases}$$

这里只给出迭代求解的方程如下(Bellman Optimality Equation)：

$$v^*(s) = \max_a (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s'))$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q^*(s',a')$$

随机设置初始值，通过迭代计算，即可收敛到最优解。

分类： RL algorithm

标签： RL， MDP， 贝尔曼方程

initial_h
关注 - 1
粉丝 - 6
+加关注

« 上一篇：1. Two Sum (Python)
» 下一篇：RuntimeWarning: invalid value encountered in true_divide

posted @ 2018-07-17 10:52 initial_h 阅读(1139) 评论(0) 编辑 收藏