经典强化学习算法

1. 动态规划算法

对于算法设计与分析中，动态规划算法的思想是将大问题分解为小问题，解决问题，然后组合问题

算法设计与分析

动态规划算法类似与分治法，都是分解-解决-合并

不同点：前者子问题一层一层，为一个小问题分化为大问题

后者则是将大问题分为多个小问题，小问题之间的求解是独立的，指同一分支

1. 动态规划求解问题的步骤

根据问题性质： 1. 子问题最优解，问题最优解的重叠性

2. 递归地求解单个问题地最优解

3. 然后合成为大问题，再回到第二布，不断递归循环求解，直到至最大问题

1. 由于强化学习问题地马尔可夫性质

性质：下一个状态之只受上一个状态与动作影响

因此我们求出上一个状态最优解，从而可以求得当前状态最优解

新概念提出

**值函数(动作值函数与状态值函数)**

假设针对一个MDP序列，状态-动作-奖励 序列

st,at,rt+1,st+1,at+1,rt+2.....

1. 针对时间点t的pay off，一般称为累计折扣奖励，或者回报

Gt = rt+1 +γrt+2 +γ+ .....

1. 状态值函数

定义，对于当前状态开始的累计折扣奖励的期望，也就是某个状态的回报期望，用符号V表示

期望定义跟单个Gt 有区别，前者是所有可能的回报分布期望，后者则是单条MDP序列的状态回报

注意，策略是不变的，因为我们的状态值函数都是针对某个策略，某个状态下的回报期望

比如 与 尽管是同一个状态

但前者表示在k轮的策略下的状态st的值函数，后者则是在k+1轮，两轮的策略可能是不一样

公式表达：

其中st 可以是状态空间中的任何一个状态

, 其中 实际上就是

注意：此处的St+1是一个确定的状态，因为它取决于st时采取的动作at以及状态转移概率

并且 也是一个不缺的值，因为我们只能采取动作后才能得到对应的

那么递推公式表达就是贝尔曼方程

1. 根据状态值函数，同理推理动作值函数，一般用 表示

从当前状态开始(不包括当前状态)，采取动作at，所得到的后续积累折扣奖励的期望

动作值函数，关注的是在某个状态下，采取什么动作at能够使得此时的折扣最大化

公式表达： 同理可以推得对应的动作值函数的贝尔曼方程

此处的 可能情况是确定的，因为给定了状态st与动作at

经过推理，得到了动作值函数与状态值函数的联系方程

1. 状态值函数与动作值函数之间的关系

首先明确，任何动作at都处在策略Π对于状态s的决策动作中

因此，状态值函数即当前状态的所有动作值函数之和

同理，由于动作值函数下一个状态st+1在给定st与at情况下由，也就是状态转移矩阵决定

由于下一个状态的不确定性，引发出了动作值函数表示状态值函数

这里重点是下一个状态的值是由状态转移概率决定的

两个要点，一个策略，另外一个状态转移概率

后面我们设计算法，既可以通过状态值函数，也可以基于动作值函数

理解：如果基于状态值函数想要表示动作值函数，那么需要状态转移概率，然而我们可能不知道环境

如果基于动作值函数表示状态值函数，那么需要知道agent策略

1. Bellman 方程

将上面两个方程的推导进行结合

有

此时只有状态值函数，那么假设所有状态值函数是未知，我们可以通过解这个方程组来求解

状态值函数

也有

1. 策略评估 Policy Evaluation

假设给定你一个策略，那么如何去评估这个策略好还是不好

可以使用 状态值函数 或者 动作值函数 来评估

个人理解：如果所有状态的值迭代都不发生改变，那么就是最优策略

如果选定的动作为所有动作中的最大回报动作，也就是动作值最大，那就是最优策略

1. 基于状态值函数评估策略
2. 基于动作值函数策略评估

通过两个贝尔曼方程进行策略评估，目前还不是很了解评估过程

1. 策略改进

对当前的策略，基于两个评估方法，状态值函数与动作值函数，如何去优化策略，这一过程就是

策略改进，也称为策略求解

注意，这里我们只是对于当前值函数，来优化策略，但是并没有

而不是针对整个问题的优化或求解

跟机器学习，梯度下降法类似，这是一种贪心策略(为确定性策略)

1. 基于状态值函数的策略改进

对于策略Π(a|s) argmax a∈A

如果找到的这个a是使得状态值函数最大的发展方向，那么就将策略直接调整为a

其他情况下就干脆让其为0

实际上后者是

前者使用的状态值函数，因此说是基于状态值函数的改进

1. 基于动作值函数的策略改进

对于策略Π(a|s) argmax a∈A

对于所有动作，找到一个动作使得其值最大

实际动态规划，就是基于状态值函数

1. 首先给出一个随即策略

然后对随即策略进行状态评估

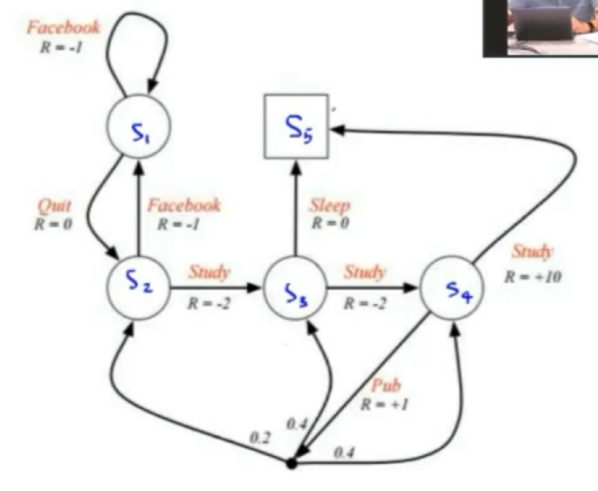
此处评估需要求解方程，那么求解过程必然很复杂

实际还可以动态求解

1. 在策略改进时，我们尝试根据已经求解的方程，最大化每一个状态价值

前提是我们确实知道了每一个状态的状态价值

针对实际问题进行策略评估与策略改进

1. 
2. 分析：

该问题的状态空间均是分散的，5个

动作空间也是分散的，5个

1. MDP 五元组

状态空间：s1,s2,s3,s4,s5

动作空间: study, pub, face, quit, sleep

折扣因子：1

转台转移函数，几乎都是确定的

但是对于pub这个动作，执行后得到的状态有三种可能，概率为0.4，0.4，0.2

奖励函数，同样也是固定的，标记在途中

策略Π(a|s) = 0.5

1. 我们初始化了一个均匀随机策略

在这里我们基于状态价值函数，进行策略评估，公式

针对每一个状态si 都能够列出对应式子，那么4个方程，4个未知数

因为最后一个状态s5为终止状态，我们无法再执行动作，没有方程，并且其v默认为0

列出方程后，能够求解出4个状态的值函数

1. 策略改进，迭代

根据新得到的状态价值函数来改进我们的策略

对于策略Π(a|s) argmax a∈A

动作能够使得得到的状态价值最大化

对于每个状态 s1，执行动作不同动作，其奖励 加上 这就因子乘以下一个状态的价值

找到使得s1状态价值最大argmax a的动作方向

贪心算法直接使得该动作对应的概率为1

最优值函数与最优策略之间的对应关系

强化学习的目标就是找到最优策略，符号标记为 Π\*

而最优策略一定对应着最优值函数

1. 最优值状态值函数，所有策略下产生的状态值函数最大的最大者

V\*(s) = max Π (s)

1. 最优动作值函数，所有策略下产生的动作值函数的最大者

Q\*(s,a) = max Π (s,a)

1. 基于最优状态值函数 定义 最优策略

Π(a|s) argmax a∈A

以上的策略以及状态值函数均有最优解

1. 基于最优动作值函数定义的最优策略

Π(a|s) argmax a∈A

两种定义实际上是一样的，因为 =

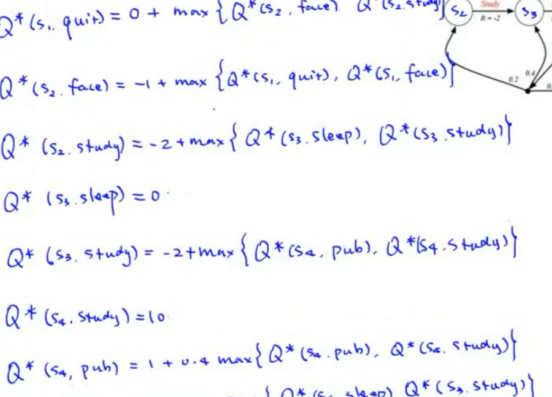
以上方式为通过迭代值函数来求解最优策略，因此一般称为值迭代

后面还会讲另一种，通过迭代策略来寻找最优策略，叫做策略迭代

1. 最优状态值函数与最优动作值函数之间的关系

我们会发现最优值函数的求解

以上的V 与 Q 均为最优值函数

 列出如图所示的方程组，由于有max，不是线性方程组，无法直接求解 我们只能使用迭代的方法就行求解，这个后面会讲，暂时不看

这是第二种求解方式，也就是列出关于最优值函数的方程，此时并未给出初始策略

然后使用迭代的方式一步一步，知道求解出最优值函数

最后根据最优值函数来确定 最优策略

第一种方式，给定初始策略，然后策略评估

策略迭代，更新策略，再次评估，依次循环，得到最优策略

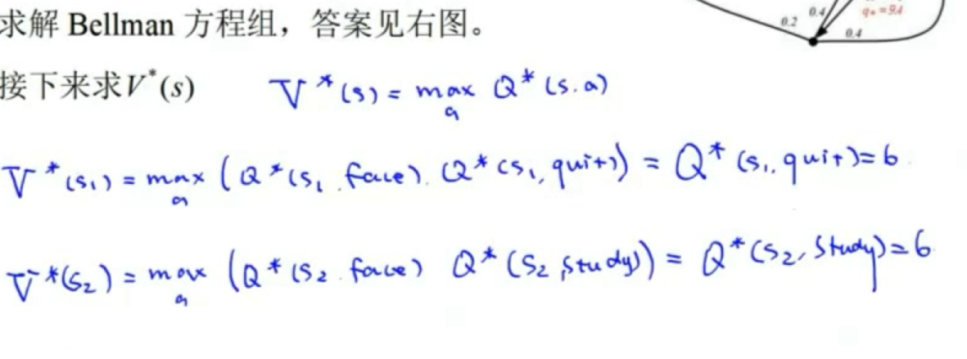
两种都是值迭代的求解方式

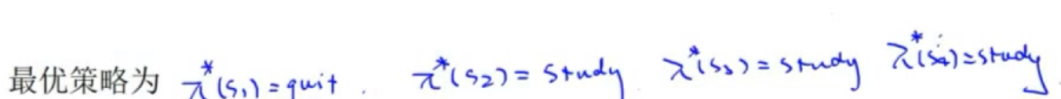
实际 一般情况下，根据最优值函数列方程，然后一步一步求解使用的比较少

因为这个要求状态空间与动作空间都是离散的，空间维度比较小(维度大了，计算复杂度很高)

我们在求解除了最优动作值函数，自然就能求解最有状态值函数

理论上求解最优状态值函数的过程 就 已在求解 最优动作值函数了



那么最优策略即为 

第二种值迭代方法一般是对简单问题适用

大部分时候还是，基于值函数的策略评估，策略迭代的方式求解问题

实际第一种求解方式就是策略迭代，只不过是基于值函数的策略迭代

因为在求解过程中，我们并没有迭代值函数，只是针对每个迭代的策略

都求解出对应的值函数

1. 因此大多数强化学习问题，一般都采用策略迭代

初始化策略，策略评估(很多情况下，对于初始策略随机，并且值函数也是随机为0的)，策略迭代，2 3 循环

因为似乎评估初始策略这个过程比较复杂，因为要列方程求解

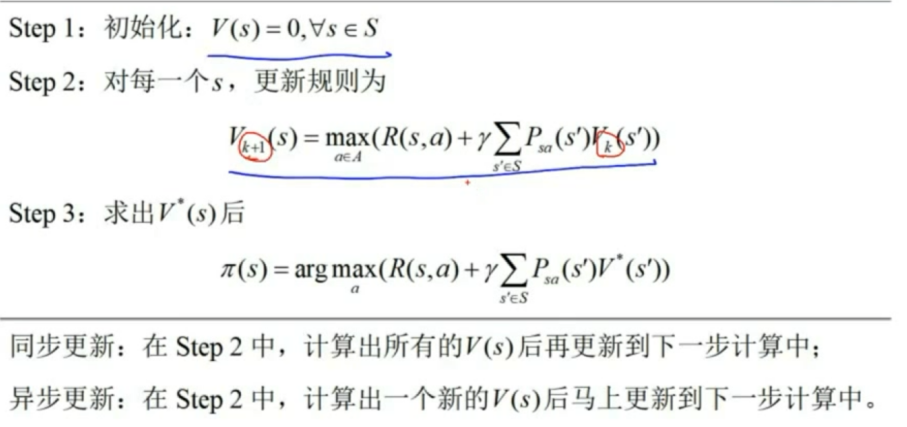
目前个人观点，初始策略的初始值函数求解可以随机初始化

1. 对于价值迭代，此时并没有定义策略，动作选择为max

列出价值迭代方程，然后一步一步求解，直到求解出最优值函数

根据最优值函数来确定最优策略

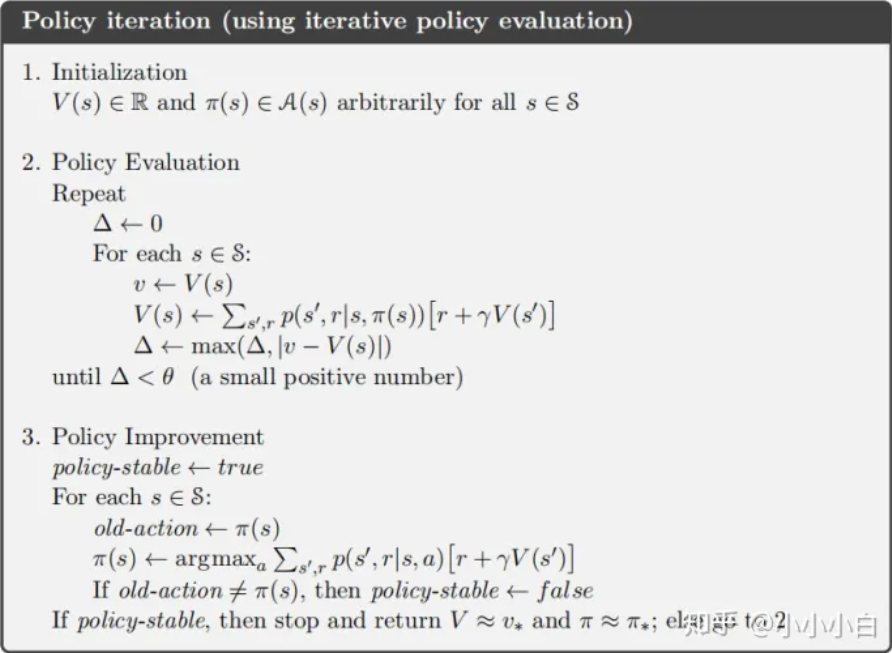
一般情况下



值函数迭代中，分为同步更新与异步更新

实际情况往往使用异步更新，能够更快得得到最优解

价值迭代



策略迭代

前者

1. 初始化策略与其对应的值函数
2. 策略评估，评估该策略的好坏，实际更新的是值函数



1. 根据更新的值函数，选择使自己奖励最大的动作，在某个状态下

此时就已经更新了策略

然后返回第二步，进行策略评估，再次更新，知道策略不再改变

思考，其实初始化策略，伴随的值函数并非对应策略的值函数

根据策略评估，我们获得更有可能的对应策略的好坏

实际上策略评估是一个循环过程，我们要保证评估的状态值函数

足够接近真实的该策略下的状态值函数

因此有设置一个参数θ保证评估的是最优的