1. 动态规划算法，agent并未与环境进行互动

而是知晓了个人策略，状态转移概率，以及奖励函数

根据前三者进行策略迭代或者值迭代，拟合值函数，然后更新策略

从始至终，agent未曾与环境互动过

1. 蒙塔卡罗算法，由于不知道环境模型，无法进行值迭代，策略迭代中的策略评估也不好计算

在策略确定后，agent会与环境多次互动，然后拟合获得评估策略的动作值函数

根据动作值函数，来确定新的策略

还会使用到 ε在互动中影响agent的最优动作选择

那么动态规划的优点是，算法效率高，不需要等待互动；缺点要知晓环境

蒙特卡洛算法优点是，无需对环境建模；缺点，需要多次对环境进行完整采样，花费时间多

那么时序差分算法 Temporal-difference TD

利用智能体在环境中时间步之间的时序差，学习有时间间隔产生的差分数据求解强化学习任务。

**时许差分算法准确，高效，是目前强化学习的主流方法**

时序差分法原理

1. 对于每个状态动作对(st,at)，在mento carlo algorithm 蒙特卡洛算法中，运用增量法来确定其动作值期望

公式

其中k表示第k次增量估计，理解公式，比平均多出来的部分分给所有人，也包括自己

先切掉不平均的，然后剩下的平均分给大家，包括自己

推导 其中 = 第k次的 rt + γrt+1 + .... + γ^(T-1) rT

= Rk(st,at) + γ

此时第k次的 ，可以尝试使用第k-1次的估计值来代替

这里需要理解其中st+1,at+1，状态动作对是确定的，值该条G链后面的st+1，at+1

因为我们仅仅是针对该条链后面的状态动作对进行评估

因此状态动作对， 只对应 st，at后面的一个动作对

此时将Gt 带入到前者公式中，**不管是第k次还是第几次都是固定的**

这样我们不进行第k次采样，也能根据第 k - 1 次采样

得到对应的估计值

*叫做TD 目标值，近似真实值，近似于*

干脆用 α来代替 1/k ，并且去掉k的下表，因为都是在第k-1次进行的计算，只使用了单挑链

观察公式会发现，更新单个动作序列对的动作值期望，只需要知道

自身原本到的期望，以及该状态动作对的奖励，还有后一个动作对的期望

在机器学习中 α 理解为学习率

上面考虑的为单步预测，一步预测，也就是TD(0)

后面介绍的 Sarsa算法 与 Q-learning算法，都是单步预测 TD(0)

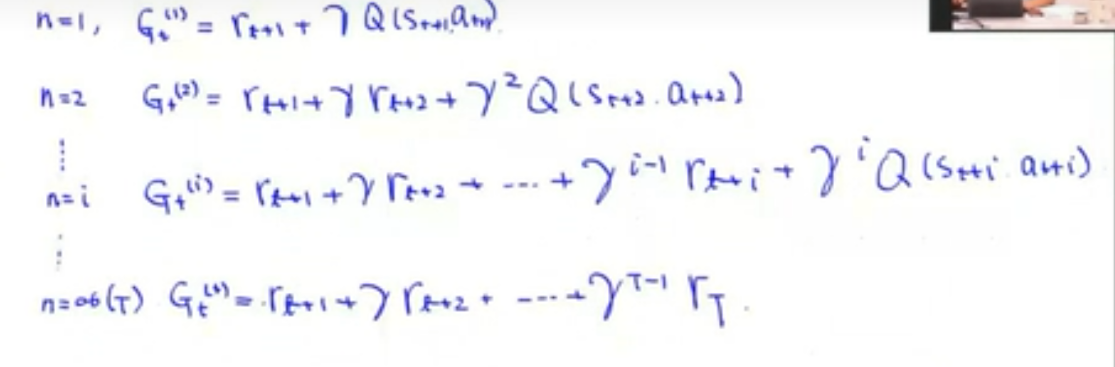
个人疑惑：教材中讲明，只考虑向前一步的差分，个人感觉是向后一步的差分，有待理解

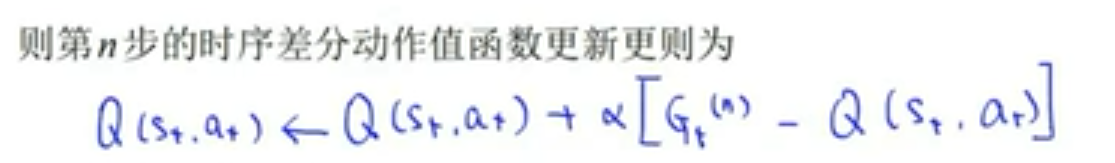
如果向前移动多步，再更新动作值函数，则称之为n步预测

当n接近正无穷，实际上 TD(λ)就是蒙特卡洛算法

个人理解还是向后预测，下图中的＝应该换位≈，除了最后一个，就是真实的Gt了

这个G是第k次相对于k-1的序列





时序差分算法的特点：

1. 免模型的强化学习算法
2. 在线学习Online-learning: 每一个时间步都可以进行更新，收敛更快

公式理解，与蒙特卡洛算法看似从k+1轮与k轮区别上去理解

实际上是，针对单条链的重复利用

因为agent在游玩的时候，并不需要知道完整的后续链，就能够更新，比如

第7次采样，st，at，rt，st+1, at+1, rt+2, 后面状态动作还没有开始，也就是第七次没完成

但是没完成也可以更新，因为奖励， 都是已知的

而TD(1), 进行两步预测就需要知道

，st，at，rt，st+1, at+1, rt+2，st+2, at+2, rt+3

对于蒙特卡罗法，需要知道整条链子才能更新

所以这个时间步指的是agent每一次看到状态，做出动作，获得奖励的时间步

个人深层理解，向后(向前)预测的项数越多，但是越靠后项数越多，γ叠加次数越多

那么后面项占比就越小，可以理解为 1/x 函数图像

个人理解，根据函数图像，我们向后预测3步左右，TD(2)，应该是比较精确的

经过资料查阅，TD(0)确实是one step，单步，教程上的原文也是 look head 向前看

尽管个人理解是向后看，就理解为只看前面几步

TD（0）向前看一步，就是后续链中的只看前一布，所以look ahead

1. 必须要终止状态 实际我们更新的时候并不需要等待agent抵达终止状态，只要做出了动作就可以更新

时序差分算法实例

假设有一个强化学习问题，有两个状态 A，B；模型位置，不涉及策略和行为。

只知道状态转换 和 即使奖励，以下序列中有8个完整的状态奖励对

A,0 B,0 B,1 B,1 B,1 B,1 B,1 B,0

假定衰减因子为y = 1

蒙特卡洛法预测：

公式： V(A) <- V(A) + 1/k(G(A)-V(A))

其中1/k 理解为 系数即可

假定初始V(A) = 0

V(A) = 0 + α(5-0)

时序差分法预测(一阶)：

公式 V(A) <- V(A) + 1/k(0+γV(B)-V(A)) 实际蒙特卡洛法也需要考虑gama

针对第一个状态的评估

这么算会发现V(A) = 6/8, 因为V(A)自身默认为0，并且此处的k=1

实际上k应该取较小数字 α

时序差分算法，策略迭代

1. Sarsa 算法 TD(0)

输入：针对环境有 E(环境信息)，S(状态)，A(动作)

初始状态s0，折扣系数γ，更新步长α

随机初始化一个动作值函数 Q（s，a），一般通过字典，而不是矩阵

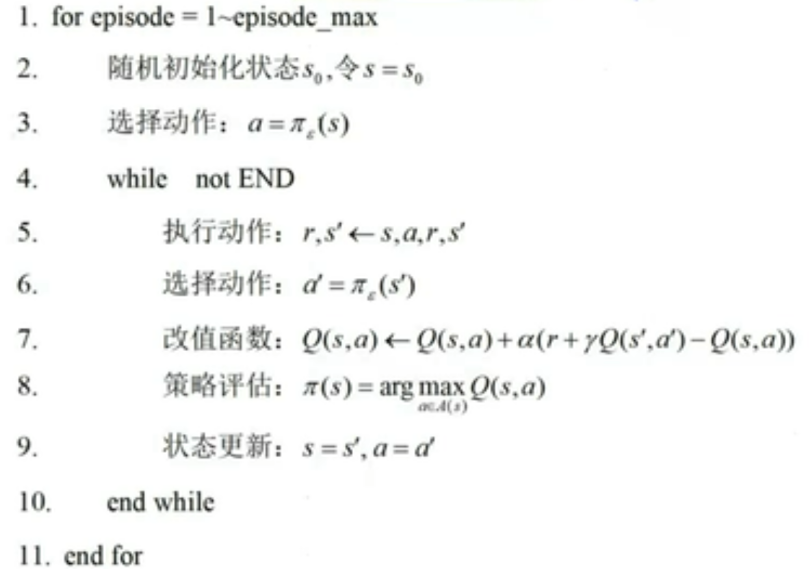
该字典的key为状态，可以理解的是，字典之间下表是没有顺序的，然而矩阵是有下表的

实际两种都可以，但是字典更便于理解，并且如果需要展示转换成dateframe

然后据此 进行策略评估得到 Π（a|s）= argmax a Q（s，a） 贪心策略评估

初始化策略与对应的动作值函数的方法

接着正式开始算法



先对初始状态进行动作选择与执行，得到s0 a0，动作选择为 ε贪心策略

其中，while循环让，agent执行动作a0，得到reward r0，

再次选择动作，一旦动作被执行，就可以进行单个动作对评估，此时不需要等待reward出来

这里的进行了单个策略评估后就直接尝试策略改进，接着循环

一般终止条件是人为定义终止步数，或者抵达终止状态

其中episode 为agent 一次完整的迭代

**说明：**

Sarsa 算法是同策略算法，即选择动作(生成轨迹)所用的策略与策略改进的动作是同一个动作

而Q-learning 策略选择与评估所使用的动作可能不是一个动作

Sarsa 需要维持一个Q(s,a)矩阵

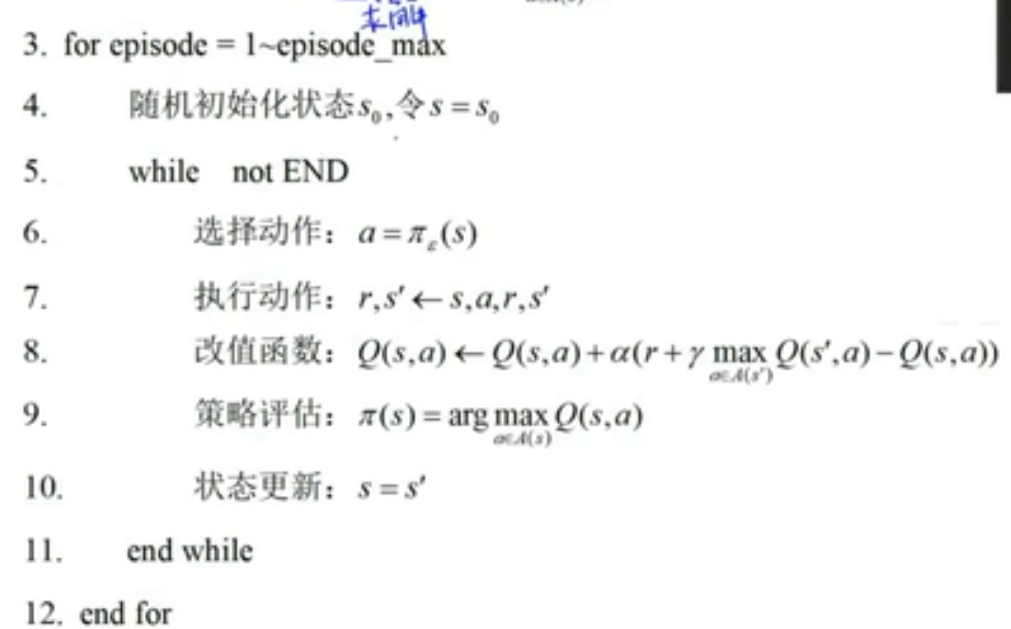
1. learning 算法

输入：针对环境有 E(环境信息)，S(状态)，A(动作)

初始状态s0，折扣系数γ，更新步长α

初始化策略迭代的动作值函数Q(s, a)

策略评估得到 Π（a|s）= argmax a Q(s，a) 贪心策略评估



Q-learning 算法初始化状态后并没有，先执行动作

待到 while 循环内部执行动作

同样根据ε贪心策略选择动作，然后执行动作

但是策略评估时，使用的并非 Q(s’, a),

也就是更新的估计值并非后一个动作序列对，而是状态对应的所有可能动作的最大值

理论上这种策略评估方式，似乎不符合时序差分法的原理推导

也就是只观察了状态，并没有观察伴随的动作

从理解上，这是一种更加贪心的策略评估，不仅只向前看一布

看的时候还只考虑状态，动作自己选，使得Q(s,a) 评估更大

**说明：**

1. Q-learning选择的动作并没有用在策略评估中，异策略算法
2. 也需要维持一个动作值函数矩阵

时序差分算法的改进

1. 期望Sarse算法

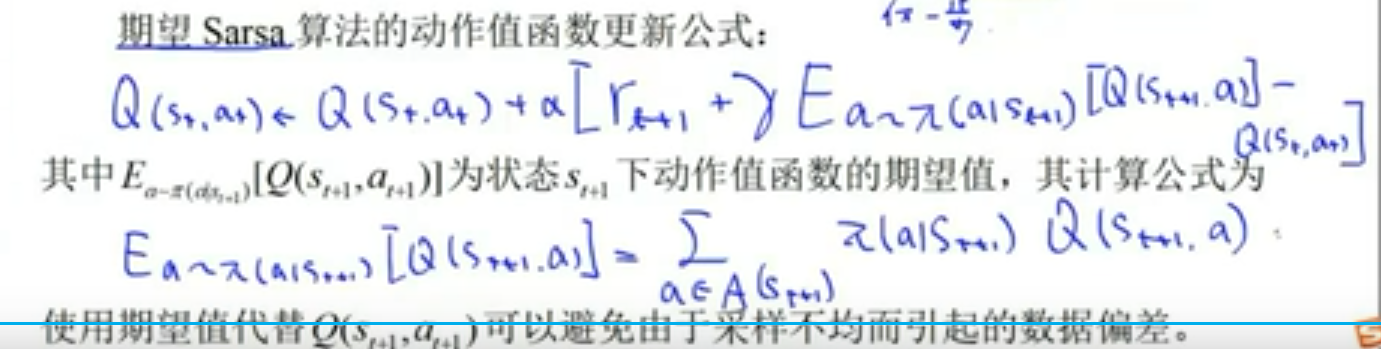
由于采样过程，序列更新采样的过程

面对每一个状态，总是贪婪算法选择最优的动作

这样会导致采样不均匀而引起数据偏差，尽管有ε加持

原本的动作值函数更新公式：，下面的at+1，总是根据st+1最优情况选择的

可以尝试改进，将换成所有at+1动作空间的期望Q值



可以避免由于采样不均匀而引起的数据偏差，相当于用的是V(s)

这里并不是算平均值，而是算期望，动作概率大的更有可能被选取

1. Double Q learning 算法改进

Q learning 算法中 策略评估 选定的都是最大值来评估值函数

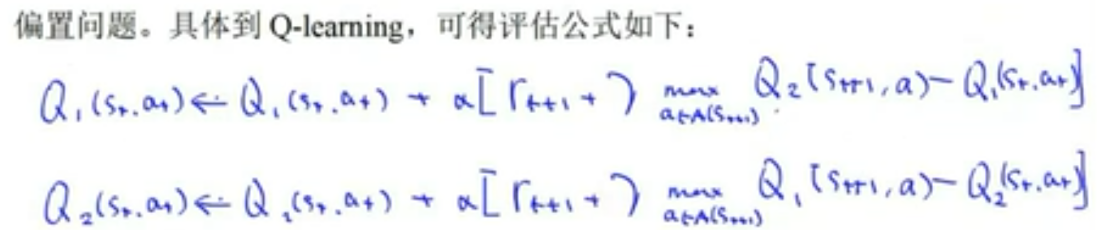
带来了一个正向偏置的问题，因为最大值函数对应的动作这么评估导致TD误差总是会比

真实值更大一点，至多相等

因而采用Double Q learning 来避免正向偏置。Double 的核心思想是同时维持两个动作值函数矩阵，

评估其中一个动作值函数时，利用另一个动作值函数的最大值来计算TD误差，然后交替的评估

实验证明可以有效地避免正向偏置问题



下一页中详细给出Double Q learning 算法过程

Double Q learning 算法流程

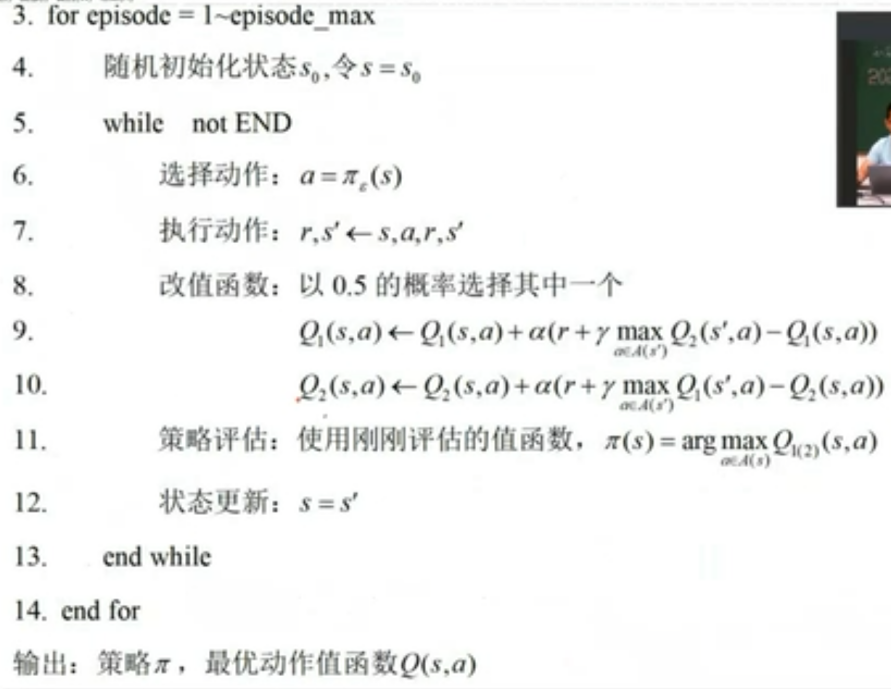
输入：针对环境有 E(环境信息)，S(状态)，A(动作)

初始状态s0，折扣系数γ，更新步长α

过程：

1. 随机初始化两个动作值函数 Q1(s,a) 和 Q2(s,a)
2. 执行第一次策略评估： Π(s) = arg max Q1(s,a)

第一次策略评估是为了得到该动作值函数对应的策略



选定动作，跟执行都一样

但是策略评估时，会有0.5的概率选择其中一个进行评估

且评估时使用的TD目标却是另外一个不相同的Q值

策略改进，则是使用上一步策略评估时的更新的 Q值矩阵