为何要用值函数近似法

对于三种经典强化学习算法，动态规划算法，蒙特卡洛，时序差分算法(这是Q learning，没有Deep)

实际上针对到的都是

求解离散状态空间的强化学习问题，毕竟值函数都是给出矩阵形式，包括状态转移空间

然而现实生活，或者大多数强化学习问题，其状态与动作都是连续的

尽管可以对连续值进行离散化，但会出现维度灾难

矩阵的值函数表达形式是一个离散函数。对于连续空间问题：

实际可以尝试用神经网络来表达一个值函数Q(s,a)

输入：状态(动作)，输出：对应的值函数

不仅如此，对于策略的表达，因为策略为Π(s|a)其输入为s

一般这个神经网络表示为 Q(s, a; w) 其中w为矩阵的可学习函数，其中；表明为并非输入变量

其中 Q(s,a;w)-> R 其中 s，a 为子变量，w为函数的参数，其中R表示输出值为实数

表明对动作值函数的评估

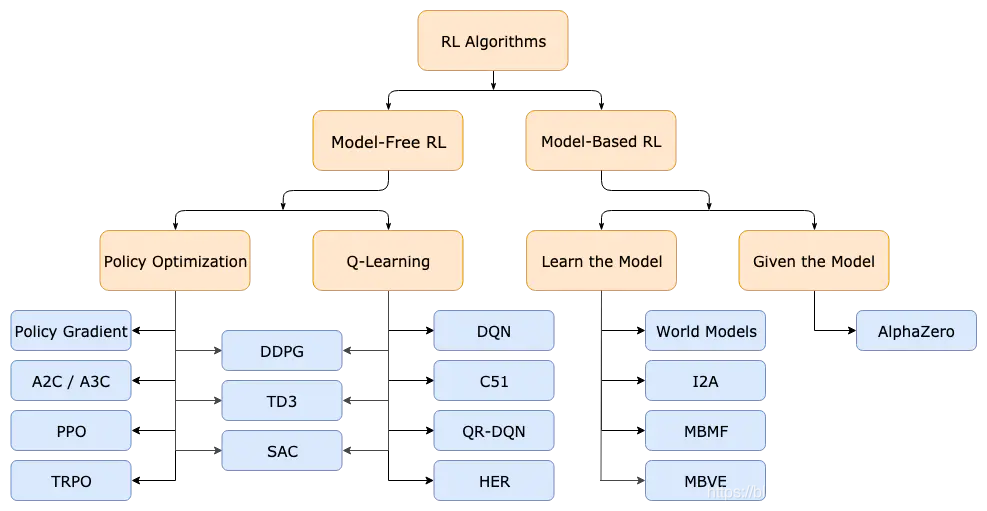
举出非常简单的例子，线性值函数 Q = w1s²+ w2a²，根据线性回归理解，w为可学习参数

实际上是w1，w2，决定了这个函数的映射方式，s，a则是固定的输入集合中的某个对象

目前我们先考虑，只用深度神经网络模拟值函数，策略用矩阵模拟 这种方法为值函数近似法

第二种，值函数用矩阵模拟，策略用深度神经网络模拟 这种方法叫做策略梯度法

第三种，神经网络 模拟值函数与策略 这种叫做Actor-critic框架 AC framework



可以注意到 AlphaZero 算法是基于模型的，是由于围棋这种环境变化是可以直接给出模型

并且状态转移函数是固定的，没有第二种例外可能

1. 线性值函数近似原理

并不是特别关注用矩阵来表示值函数，仅仅是尝试用线性函数来模拟值函数矩阵

1 假设 s 是连续空间中的一个状态空间，而动作a是有限的且离散的

将其中的a作出适当改造为 one-hot向量 [0,0,0,0,0,0,1,0,0,...,0]

将两者自变量放在一起为 (s,a) (s1,s2,s3,0,0,0,0,0,0,1,0,0,...,0)

如果s细化的化，可能一个维度表示，也可能多个维度才能表示一个s

比如 单个状态 s 表示为 s1，s2，s3......sn, 理解为神经网络，三通道输入s1，s2，s3

2 在Sarsa 与 Q-learning 算法中，使用的是动作值函数，此处用线性函数来近似动作值函数

该动作值函数的参数定义为θ, 将输入向量称之为 特征向量

3 我们希望模拟的线性值函数，最终的结果能够与真实的值函数近似

实际上我们并不用去担心由于输入向量之间，尤其是状态的轻微差距，导致的值不同，实际是类似

对于图像识别，并不能对所有图像进行模拟，但是仍然能够准确分类，其输入特征相似即可

同样对于有限的数据来模拟 线性值函数 与 神经网络值函数，依旧是成功的，有效的

4 Q(s,a;θ) = θ(s,a)

学到的动作值函数与实际的动作值函数较为接近，这个目标就是减少误差，均方差损失

E = [Q(s,a)-Q(s,a;θ)]² 最小化

根据梯度下降算法，求梯度可得

2\*Q(s,a;θ)对θ梯度，[Q(s,a)-Q(s,a;θ)]

更新法则 θ = θ + 2 α\*Q(s,a;θ)对θ梯度\*[Q(s,a)-Q(s,a;θ)]

= θ - 2 α\*Q(s,a;θ)对θ梯度\*[Q(s,a;θ)-Q(s,a)]

根据对神经网络的了解，该更新过程，即参数更新，只会让模拟的值函数更加接近已经给出的

状态-动作-值函数估计，表明我们仍然需要先获取对于每个状态动作对的值函数估计值

线性值函数或者神经网络的好处是面对没有出现过的状态动作对，已经能够做出合理的推断

可以理解为原先的矩阵，规律全被神经网络积聚学习了，面对未知情况，也能模拟

再给出估计值公式 Q(s,a) = Q(s,a) + 1/k (r+γQ(s’,a’)-Q(s,a))

1. 给出估计值的参数更新公式

θ = θ - 2 α\*Q(s,a;θ)对θ梯度\*[Q(s,a;θ)-(r+γQ(s,a;θ))]

不是直接使用平均值，输入有多个相同的，但是输出有多个不同的

跟以往监督学习不同，并非直接使用准确值，而是多个参差不齐的值都会考虑

这样省去了 TD 算法的累计 求均值的过程，直接使用了每一条链的结果即可

其中 Q(s’,a’) 是对下一次的估计，因此我们需要转换为 Q(s,a;θ)

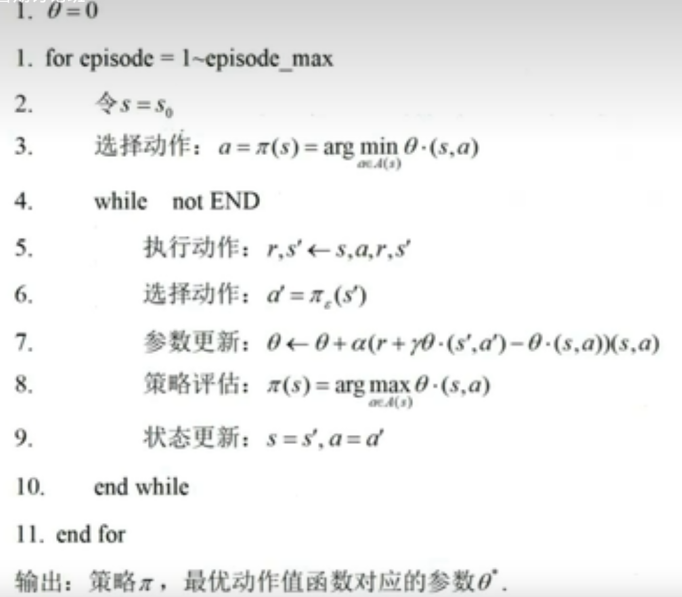
核心思想使用到了 时序差分算法的一步估计，但是这次并不需要累加进行更新均值

可以直接将其放在 值函数近似 参数更新中，总体效果是一样的，但是该步骤

将两步并作一步

1. 线性值函数 Sarsa 算法

输入， E S A 初始状态s0 折扣系数γ，学习率α



Max

对于该算法分析，首先初始化值函数与策略

1. 策略评估

执行动作，然后按策略选择，直接获取到了下一步 状态-动作-奖励，就可以更新了

此时直接用公式参数更新

1. 策略迭代

更新一步参数后，就直接更新策略

并且准备执行下一步动作

这种再获取到数据后，直接策略评估与迭代了

并且以往的获取多个数据，完善评估后

才进行策略更新

理解为一步更新，急躁的更新，理论上有效，实际如此

细节

1 选择动作为 ε 贪心策略，策略迭代也是贪心策略

2 此时选择策略执行的动作，与参数更新选择的动作为同一个动作

同策略算法，应该理解为同动作，从表现上

如果理解为同策略，就是动作选择策略与更新参数选择的动作为同一个动作

3 还有一个细节，每一个episode 而言

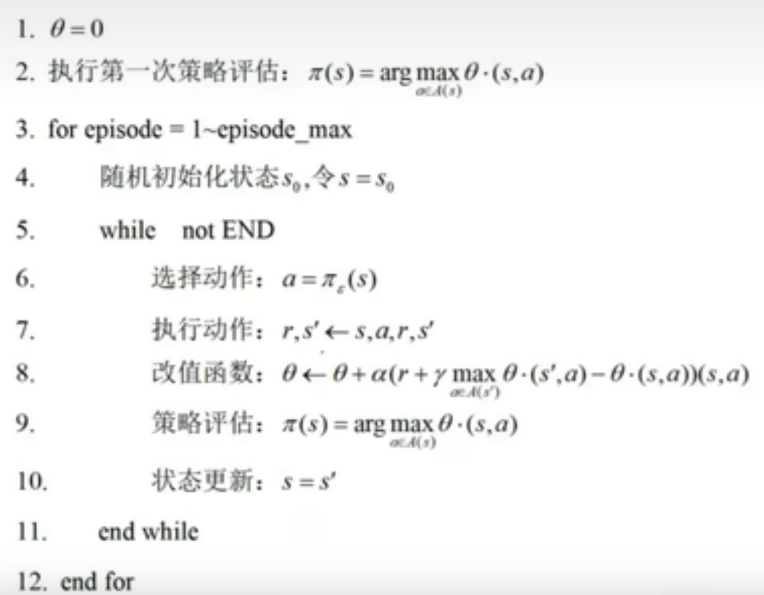
我们对于给予的初始状态s0，我们选择动作既不是随机，而是选择值最大的动作

在最开始状态，我们就让agent努力朝利润最大的方向前进

1. 线性值函数Q-learning算法

输入， E S A 初始状态s0 折扣系数γ，学习率α

1. learning 是异策略算法



同样是随机初始化，一步更新，但是在参数更新时候，也就是策略评估

过程中，采用的动作并不一定是状态策略选定的动作，而是使得下一步状态价值最大的动作

这样还要不仅要记录当前状态动作奖励，还要下一步状态与动作

但是此时，动作对于更新就不需要记录，从给定状态中选取最大的动作值即可

该END表明未达到终止状态时，因此对于初始化状态的s0也变得可以理解