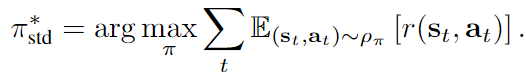
# 最大熵强化学习

《强化学习：原理与Python实现》读书笔记

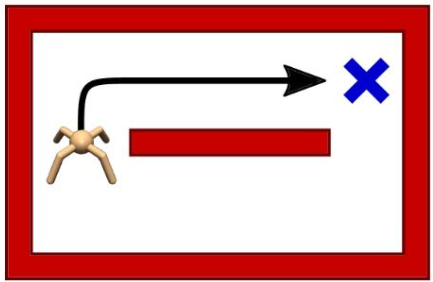
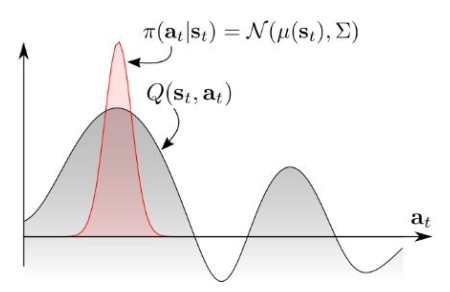
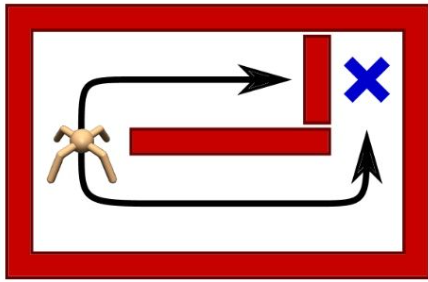
近年来，强化学习发展火热，最大熵强化学习类算法是强化学习的主流算法之一。本文的写作受启发于《强化学习：原理与Python实现》一书，这是一本不可多得的介绍强化学习的好书。特此记录一番。

## 问题引入

传统的强化学习的目标是最大化来自环境的奖励：

（1）

典型的代表就是Q-learning算法，它定义了一个Q(s,a)函数，表示在当前状态s下采取动作a所获得的累计奖励的期望值。而这样的目标设置存在一定的局限性。请看下图：

****

A1 B1 A2

A1图中，智能体可以通过上下两条路径到达目标点，向上的路径比向下的路径短，对应的奖励也稍高。这时候的Q函数可以用A2图的灰色曲线表示。传统的基于Q-learning的强化学习的策略比如DDPG等会选择Q值最大的action，再加上exploration noise构成一个高斯分布，如图A2的红色曲线部分。这样的策略是single modal，而原来的Q函数的分布是multimodal。传统Q-learning对应的策略无法表达这种multimodal，进而在测试中如果遇到B1图所示的障碍时候，智能体会一直选择向上的那条路径，无法到达目标点。

那如何解决上诉问题呢？一种策略是让智能体在选择动作时，不仅要考虑最优动作，还要考虑其他动作，这样智能体的探索能力和抗噪声能力都会得到提升。为了实现这样的策略一种想法是利用最大熵的思想来定义我们的策略。

## 最大熵RL

我们定义最大熵RL的策略如下。

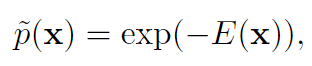
（2）

其中的是平衡探索程度的参数。传统的策略是最大化环境给予的奖励，最大熵RL的策略是在最大化环境给予奖励的基础上，还要最大化动作的熵。这样的策略实际上是一种stochastic policy, 即让策略随机化，即输出的每一个action的概率尽可能分散，而不是集中在一个action上。

有了以上定义，接下来我们的目标就是优化最大熵目标，在此之前，需要一种形式将最大熵策略表示出来以便我们优化。一种符合我们对策略的定义（0）的方法是Model-Based Model(EBM) 。

## Model-Based Model(EBM)

对于这样的假设，EBM表示为：

（3）

**E(x)**被称作是能量函数（energy function）。对所有的z，exp(z)都是正的，这保证了没有一个能量函数会使得某一个状态x的概率为0。

我们可以将能量函数定义为与Q相关的函数：

（4）

，这样我们就可以把Q值函数和策略联系起来，利用Q函数的迭代来优化我们的策略，策略表示如下：

（5）

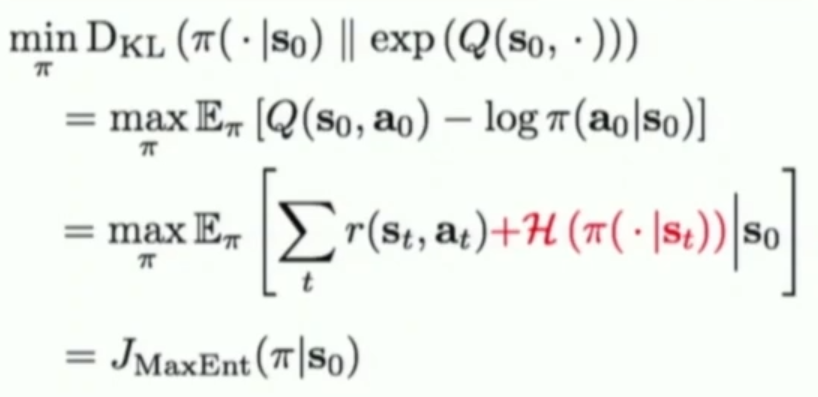
具体的：

其中：

我们称作配分函数，它是仅与状态s有关的常数。

所以：

基于此，我们可以采用KL散度来优化策略，并且证明其等价于最大化最大熵目标：



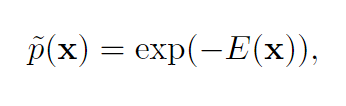
有了策略的表示以及优化方式，接着我们给出策略改进定理：

对于两个策略和，其中是依据优化得来，

如果，则，.

## 柔性RL

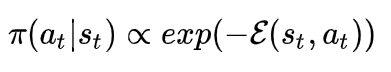
在介绍柔性RL之前，我们先介绍Energy-Based Model(EBM):



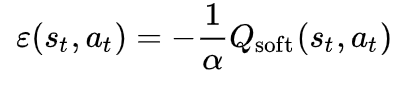
其中E(x)被称为能量函数。对于所有的z , exp(z)都是正，这保证了没有一个能量函数会使状态x的概率为0(引用1[[1]](#footnote-1))。同时服从上式定义的分布都是玻尔兹曼分布的一个实例。

回想一下我们的要求：让智能体在选择动作时，不仅要考虑最优动作，还要考虑其他动作。

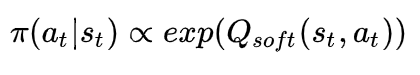
能量模型的定义正好符合我们的要求。在此我们定义Energy-Based policy:



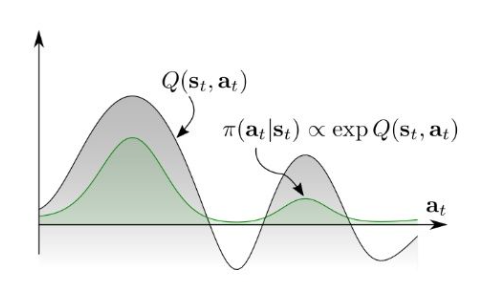
上面的形式就是玻尔兹曼分布。ε是能量函数。我们设定：



因此：

。

这样的policy能够为每个action赋值一个特定符合Q值的分布。



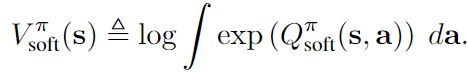
B2

如B2图所示，Energy-Based Policy可以很好的反应出Q值的分布形态，表达出Q值的multimodal特性。我们通过KL散度来优化这样的策略。

## SQL的原理和伪代码

SQL即Soft Q-Leaening算法，它的基本原理基于我们上文介绍的最大熵RL。

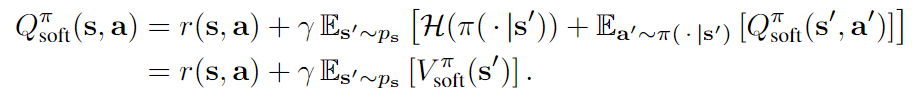
在SQL中，V函数的Bellman Equation定义如下：



假定：

****

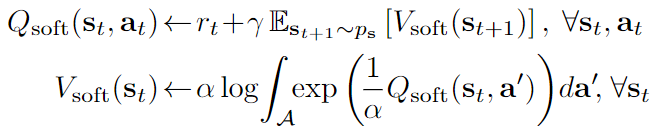
则Q函数的Bellman Equation定义如下:



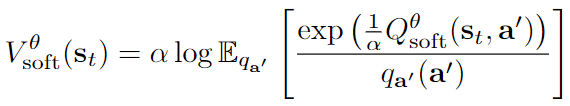
为什么在这里叫做soft呢？回想一下一般的RL每次都是选择Q值最大的动作作为策略，而此处的策略是

上式分母其实是LogSumExp的积分形式。如果非常大，同时由于exp()的存在，分母的值将主要取决于非常大时相对应的action，这时策略将以接近1的概率选择这个动作。而当各个动作的都相似或者没有特别突出大的值的时候，策略就将趋向于对各个动作赋予近似均匀的概率，每个动作都有被选择的机会。这就非常soft，即当当前局面的动作明确时，果断选择，而当动作不明确时候，随机探索。对比一般Q-Learning的Hard max策略，这里便是soft max。

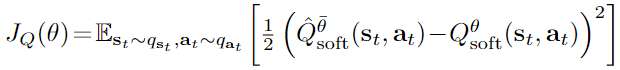
有了上述Q，V函数的定义，接下来给出SQL的值函数迭代方程：



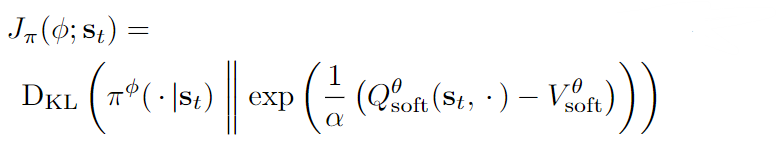
在SQL算法中，作者利用神经网络来近似Q函数，同时利用重要性采样的方法来近似V函数的积分形式。



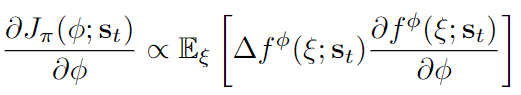
其中可以可以是任意分布，这里为了方便采用均匀分布。loss函数为

。

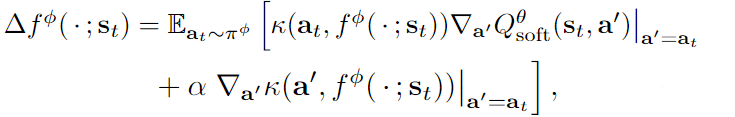
对于策略的优化，由于基于EBM模型的策略分布的采样十分困难，作者训练了一个神经网络*Φ* ，训练目标是使这个神经网络和energy-based policy之间的KL最小：



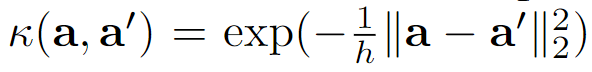
同时为了加速推理，SQL利用SVGD技术来生成近似样本。SQL利用策略的目标函数和用于从策略中采样的神经网络的如下关系进行优化：

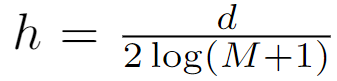


其中：



к函数为核函数，论文中采用的是径向基函数。

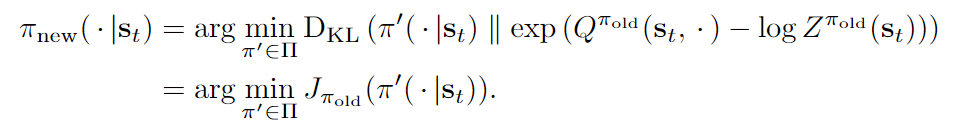


****

此处d是

## SAC算法的原理和伪代码

证明如下：

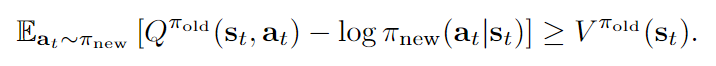


对于pi new和pi old 总有，，因为我们总能选择pi new = pi old

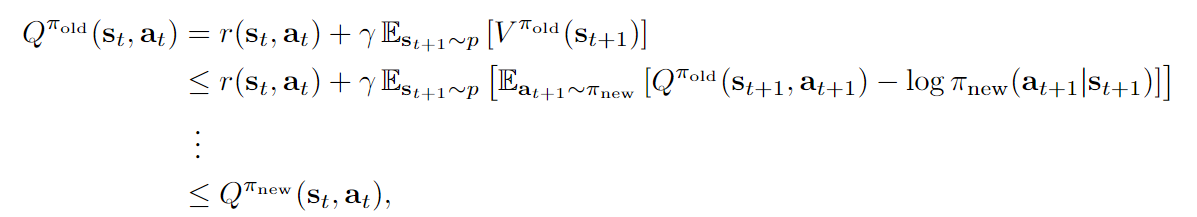
因此

****

因为配分函数Z只和状态有关，所以这个不等式可以写成



最后利用贝尔曼方程进行迭代：



我们重复利用贝尔曼方程拓展Q pi old，同时利用不等式迭代，式子就能迭代收敛到Q pi new

## SQL实现MountainCarContinuous-v0

要求最新稳定版python ,gym ,TF,满足PEP8要求

算法主要逻辑写在称为Agent的类里面，Agent类有个成员函数叫learn(),还有个成员函数叫做decide。需要输出训练过程中的每个episode的rewards，并且有效训练

## SAC实现MountainCarContinuous-v0

1. 《深度学习》 [↑](#footnote-ref-1)