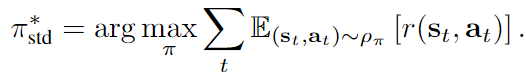
# 最大熵强化学习

《强化学习：原理与Python实现》读书笔记

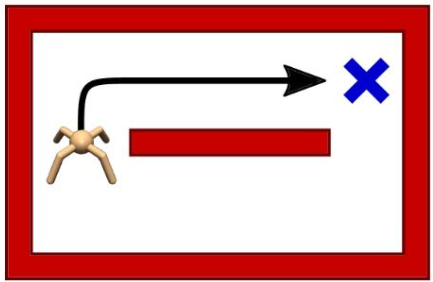
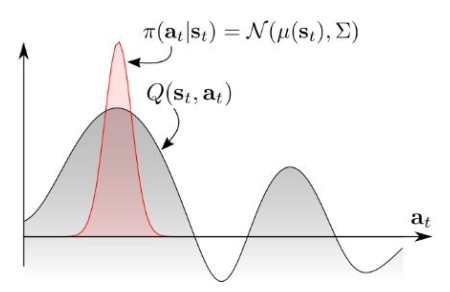
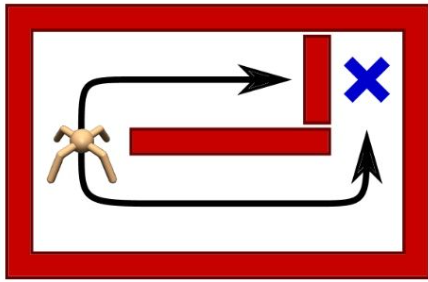
近年来，强化学习发展火热，最大熵强化学习类算法是强化学习的主流算法之一。本文的写作受启发于《强化学习：原理与Python实现》一书，这是一本不可多得的介绍强化学习的好书。特此记录一番。

## 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：

****

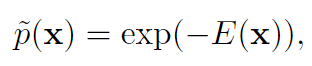
A1 B1 A2

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（0），这样智能体的探索能力和抗噪声能力都会得到提升。一种满足我们要求的方法是基于Energy-Based Model（EBM）制定策略。

## Energy-Based Model(EBM)

对于这样的假设，EBM表示为：

（2）

**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。基于EBM的策略正是利用这种特性来定义新的策略：

我们可以将能量函数定义为Q函数的负数：

（3）

其中为用于控制探索程度的参数，策略表示为：

（4）

具体的：

（5）

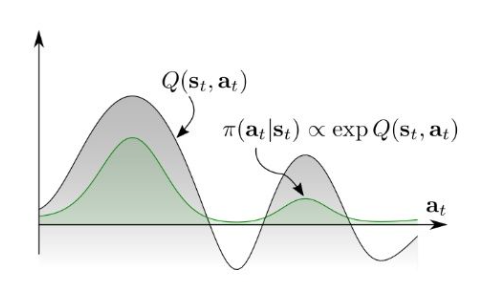
其中：

（6）

我们称作配分函数，它是仅与状态s有关的常数。

所以：

（7）

B2

图B2绿色部分展示了Energy-Based策略基于Q函数所形成的分布。可以直观地看出Energy-Based策略能够很好地反应出Q值的分布形态，表达出Q值的multimodal特性。这正是我们想要的结果。

还有个疑问，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，即对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，这里可以说是soft max。

我们策略的优化方向从B2就能看出——策略尽可能逼近Q函数的分布,我们可以用KL散度来表示：

（8）

那这样的策略优化的目标是什么呢？接着这个式子推下去:

（9）

推到这一步，我们发现，Energy-Based策略的基于KL散度的优化方向实际上就是在优化最大熵RL的目标（是仅和状态有关的常数，与优化目标无关）——使得每个状态的奖励和该状态下动作的熵最大化。那什么是最大熵RL？最大熵RL起什么作用？

## 最大熵RL

我们定义最大熵RL的最优策略如下。

（10）

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，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。可以看到最大熵RL的策略优化目标和Energy-Based策略是等价的。

根据最大熵RL的思想，将一个状态动作的熵作为奖励的一部分，我们可以定义最大熵Q函数：

（11）

依据最大熵Q函数的定义，给出策略优化目标：

（12）

有了策略的定义，策略的表示，策略的优化目标以及Q函数的定义，接下来我们给出策略改进定理并证明：

其中策略的定义：

策略的具体表示：

Q函数的定义：

策略的优化目标：

注意前文已经推导等价于最大化

策略改进定理：

对于两个策略，若（），则有

即。

证：

其中是仅与状态有关的常数，在此省略：

因此：

所以有

然后：

=

得证

## SQL的原理和伪代码

### 简介

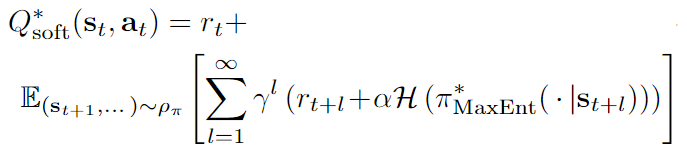
SQL即Soft Q-Leaening算法，它的主要思想基于我们上文介绍的最大熵RL。

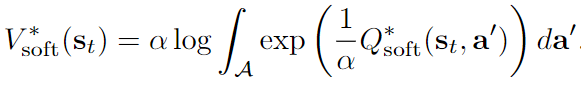
具体来说SQL的策略目标从最大熵出发定义为：



同样基于EBM来定义策略：

在最优化情形下，有如下定义的函数和函数：

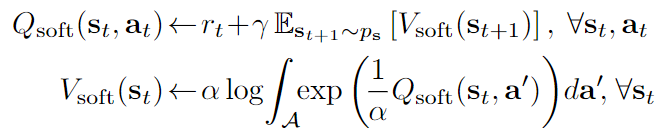




可以看出，函数相当于是对函数的softmax操作，在EBM的表示中，函数相当于配分函数。最大熵策略目标可以通过,函数表示如下：



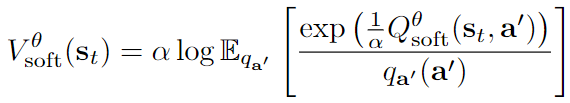
接下来定义，函数的迭代形式：



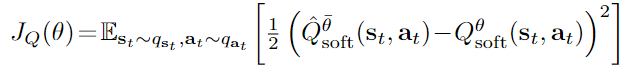
可以证明，函数经过迭代可以收敛至,。

### 优化

对于这里采用神经网络来近似。上述的积分在连续空间难以被求解，解决方法是利用重要性采样来求积分。

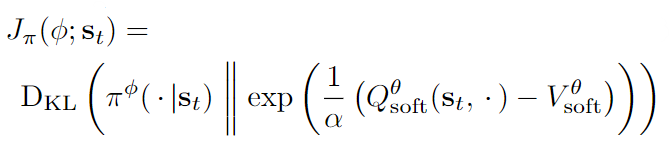


其中可以是任意分布。在训练初期进行随机均匀采样，在训练后期根据policy采样。为了简单起见，整个训练过程可以只进行随机均匀采样。通过MSE loss进行优化：

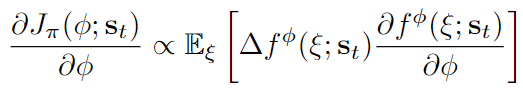


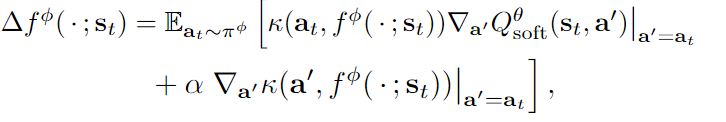
由于策略的分布过于复杂，无法采样进而难以进行优化。这里考虑使用一个神经网络，

,输入state和从标准正太分布中采样的，输出action。这里输入的 ，是为了对同一个state采样时得到不同的输出，模拟从分布中采样。考虑缩小该网络产生的分布和实际分布的差别，即最小化

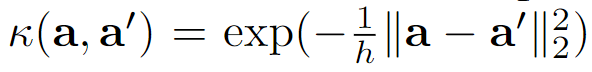


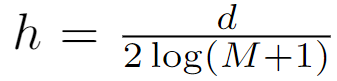
由此我们可以计算得到梯度





к是kernel，此处采用径向基核函数。



****

d是action两两之间的距离(pairwise distance)的中间数。

### 伪代码

输入：环境（无数学描述）

输出：最优策略

参数：优化器，折扣因子，目标网络学习率

1.(初始化)，。

2.循环执行以下操作。

2.1（累计经验）从初始状态S出发，执行以下操作，直到满足终止条件：

2.1.1对于状态,，其中

2.1.2执行动作，观测到奖励和下一状态。

2.1.3将经验存储在经验存储空间

2.2（更新）在更新的时机，执行一次或多次以下操作：

2.2.1（经验回放）从存储空间采样出一批经验

2.2.2（更新

2.2.2.1对每个采样

2.2.2.2计算

2.2.2.3计算

2.2.2.4利用优化器更新

2.2.3（更新策略）

2.2.3.1对每个，采样

2.2.3.2计算动作

2.2.3.3计算∆fφ

2.2.3.4计算∇φJ(π)

2.2.3.5利用优化器更新参数φ

2.2.4（更新目标）在恰当的时机更新目标网络。

## SAC算法的原理和伪代码

本文介绍的SAC算法来自论文Soft Actor-Critic Algorithms and Applications，即省略V网络以及加入了自适应参数的版本。

### 简介

SAC同样基于最大熵框架，策略基于EBM。



最大熵框架是在奖励中加入了entropy项，传统的V函数可以据此改为：

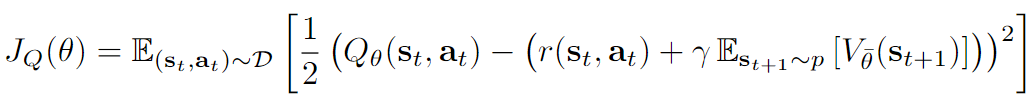
依据贝尔曼期望方程，函数为：



### 优化

SAC用含参函数和来分别近似近似函数和策略。特别的对于函数，采用了双重学习类似的技术以及目标网络，引入两套更新的参数，以及对应的目标网络参数，。为了消除最大偏差，在估计目标是选取值中较小的那个，即 。

在学习 时，试图最小化：

**

在学习时，试图最小化：

值得一提的是，在策略优化时采用的是reparameterize技巧，即：



其中是噪声，可以自标准正态分布采样得到。

而这篇工作最大的创新点来自自动化调节，在学习时试图优化：

其中。

### 伪代码

输入：环境（无数学描述）

输出：最优策略

参数：，优化器，折扣因子，目标网络学习率

1.(初始化)，。

2.循环执行以下操作。

2.1（累计经验）从初始状态S出发，执行以下操作，直到满足终止条件：

2.1.1用策略;

2.1.2执行动作，观测到奖励和下一状态。

2.1.3将经验存储在经验存储空间

2.2（更新）在更新的时机，执行一次或多次以下操作：

2.2.1（经验回放）从存储空间采样出一批经验

2.2.2（计算代价函数）为每条经验计算(，，

2.2.3（更新参数）用优化器更新

2.2.4（更新目标）在适当的时机更新目标网络