策略梯度方法

在策略梯度方法(Policy Gradient Method) 之前，我们的策略 都是以表格形式表示的。然而，当状态-动作对的数量急剧增加，或者是在处理连续状态或连续动作空间的问题时，用表格来表示策略就行不通了。因此，在策略梯度方法中，我们将策略用一个函数 来近似，其中是函数的参数。这样一来，我们就可以只存储这个函数的参数就可以表示不同状态下的策略，极大地降低了对存储的要求。

因此，我们的目标就是通过优化这个策略函数的参数，通过**梯度上升**来最大化我们所制定的某个目标。这就是策略梯度方法的基本思想，对比DQN等方法，DQN是直接让神经网络输出每个动作的action value，而策略梯度方法则是让神经网络直接输出各个动作选择的概率，即让神经网络输出策略 。

我们希望在本文结束时能够回答下面的几个问题：

1. 如何定制策略梯度方法的目标函数?
2. 如何求解策略梯度?
3. 策略梯度法的基本流程是什么?
4. 策略梯度法有何优势? 有何不足?

首先，我们来回答第一个问题：如何定制策略梯度方法的目标函数?

1. **平均状态价值 (Average state value)**

设立该目标函数就是希望能够找到某一种策略，使得所有状态所对应的状 态价值的加权平均最大。其中，可以根据状态的重要程度来设置。

1. **平均单步奖励 (Average one-step reward)**

单步奖励的含义是在某一状态下，采取各种可能的动作所能获得的奖励的期望。那么平均单步奖励就是将所有状态下对应的单步奖励加权平均。

1. **起始状态价值函数**

对于一些回合制的问题，若Agent每次都会从某个特定的起始状态开始，那么我们就可以让策略去最大化起始状态的状态价值函数。

明确了目标函数，我们就可以通过梯度上升的方法优化从而最大化我们的目标函数：

因此，就需要直面我们所提出的第二个问题：如何计算目标函数的梯度? 我们在本文中以最后一个目标函数 为例：

首先，根据状态价值函数的定义，我们有：

如果我们从状态 出发，按照策略和环境进行交互，能够得到一条轨迹：

那么我们可以求出这条轨迹下的discounted return：

此外，我们还可以求出这条轨迹出现的概率，如下：

因此，根据期望的定义我们可知：

下面，我们对该目标函数求导：

其中， 的计算过程如下：(由于 和 是环境模型所决定的，和策略无关，因此求导为0)

所以我们最终得到目标函数的导数如下：

我们发现，在该目标函数的导数中存在求期望的过程，期望往往难以计算，因此我们采用蒙特卡洛的思想，用采样的方式获得N个轨迹，求出各个轨迹下的 再做平均，即可近似目标函数的梯度：

我们得到了策略梯度法非常重要的一个式子，这个式子可以指导我们的代码实现：我们首先产生了一条轨迹 ，首先知道累计折扣奖励的形式为：，因此我们就倒序便利整个episode，对于state-action pair ，将 输入进策略网络，根据我们所选择的动作 ，选择对应 的概率值 。并取对数得到 ，与当前时刻的累计折扣奖励 相乘，即 ，求梯度得：。同理，在遍历到state-action pair 时，将前面的梯度累加，因此得到：。以此类推，最终得到策略梯度并进行梯度下降。如此重复直到该episode结束。计算多条episode并取平均最后就能够实现对目标函数的优化。

通过上面的讨论，我们回答了文章开篇的第2、3个问题，最后我们再来看一下为什么要使用策略网络? 之前的DQN不好吗?

1. 之前的value-based的方法会学到确定性策略 (即会选择action value最大的action作为当前环境的策略) ，这在某些场合会导致策略退化，即时采用了-greedy方法也会导致Agent的大量徘徊。而policy-based的方法输出的是随机性策略，因此不会存在策略退化的问题。
2. 在DQN中，神经网络的输出是各个动作的action value，这对于动作空间有限的问题是可行的，然而在面对像机械臂控制之类的问题时，机械臂是连续运动的，因此该问题的动作空间是连续的，尽管将连续动作离散化可解决燃眉之急，但毕竟治标不治本，在对操作精度有严格要求的场合，DQN就不再适用。然而在policy-based的方法中我可以让神经网络直接输出一个连续的 函数，直接反映了所有动作的分布，完美地解决了连续动作空间的问题。