

暑假实训报告

学 院： 信息科学与工程学院

专 业 班 级： 计科1803

姓 名： 苏浩天

学 号： 201808010327

完 成 时 间： 2021 年 8 月 6 日

飞桨链接：https://aistudio.baidu.com/aistudio/projectdetail/2260494?forkThirdPart=1

**一、实验内容**

用 PaddlePaddle 框架设计一个基于 DDPG 算法的股票量化交易

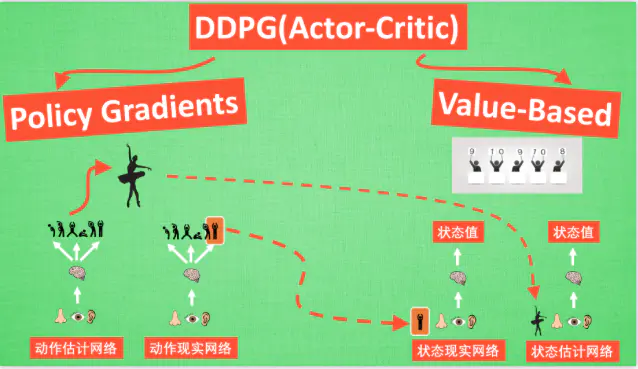
**二、实验原理**

DDPG的全称是Deep Deterministic Policy Gradient。

我们首先来看Deep，正如Q-learning加上一个Deep就变成了DQN一样，这里的Deep即同样使用DQN中的经验池和双网络结构来促进神经网络能够有效学习。

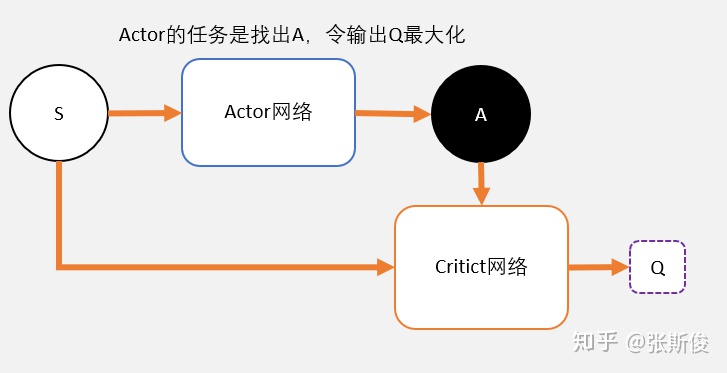
再来看Deterministic，即我们的Actor不再输出每个动作的概率，而是一个具体的动作，这更有助于我们连续动作空间中进行学习。之前不太理解这个连续动作空间是什么意思，既然policy gradient和dqn都是输出每个动作的概率和q值，那么我们为什么还要用policy gradient呢？这个连续动作空间的例子可以举一个么？既然已经诚心诚意的发问了，那么我就班门弄斧回答一下。假如想要通过强化学习得到一个词的32维词向量，哇，这个词向量的动作空间可是无限大的呀，[1,0....0]是一个动作，[0,1...0]是一个动作，如果加上小数，那更是数不过来啦，这时候我们根本不可能去计算每个动作的概率或者q值，我们只能给定状态即一个单词，直接输出一个合适的词向量。类似于这种情况，DDPG就可以大显神威了。

盗用莫烦老师的一张图片来形象的表示DDPG的网络结构，同图片里一样，我们称Actor里面的两个网络分别是动作估计网络和动作现实网络，我们称Critic中的两个网络分别是状态现实网络和状态估计网络：



我们采用了类似DQN的双网络结构，而且Actor和Critic都有target-net和eval-net。我们需要强调一点的事，我们只需要训练动作估计网络和状态估计网络的参数，而动作现实网络和状态现实网络的参数是由前面两个网络每隔一定的时间复制过去的。

与传统的DQN不同的是，传统的DQN采用的是一种被称为'hard'模式的target-net网络参数更新，即每隔一定的步数就将eval-net中的网络参数赋值过去，而在DDPG中，采用的是一种'soft'模式的target-net网络参数更新，即每一步都对target-net网络中的参数更新一点点，这种参数更新方式经过试验表明可以大大的提高学习的稳定性。'soft'模式到底是如何更新网络的？我们可以通过代码更好的理解。



我们现在整理：

**Critic：**

Critic网络的作用是预估Q，虽然它还叫Critic，但和AC中的Critic不一样，这里预估的是Q不是V；

注意Critic的输入有两个：动作和状态，需要一起输入到Critic中；

Critic网络的loss其还是和AC一样，用的是TD-error。这里就不详细说明了，我详细大家学习了那么久，也知道为什么了。

**Actor：**

和AC不同，Actor输出的是一个动作；

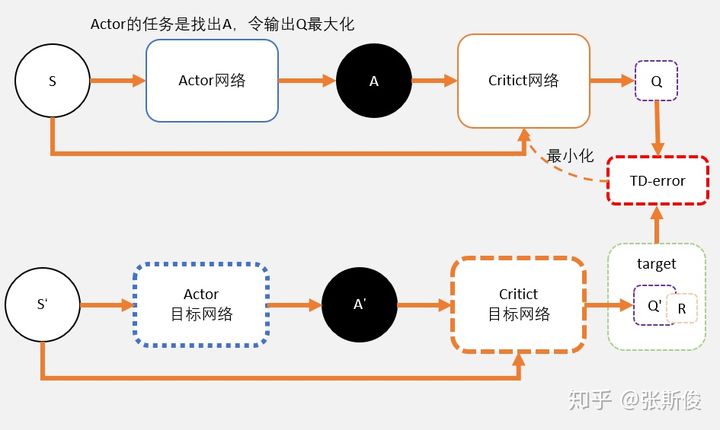
Actor的功能是，输出一个动作A，这个动作A输入到Crititc后，能够获得最大的Q值。

所以Actor的更新方式和AC不同，不是用带权重梯度更新，而是用梯度上升。

弄清楚怎么来的，就不会和PPO混淆在一起了。也就明白为什么说DDPG是源于DQN而不是AC了。

所以，和DQN一样，更新的时候如果更新目标在不断变动，会造成更新困难。所以DDPG和DQN一样，用了固定网络(fix network)技术，就是先冻结住用来求target的网络。在更新之后，再把参数赋值到target网络。

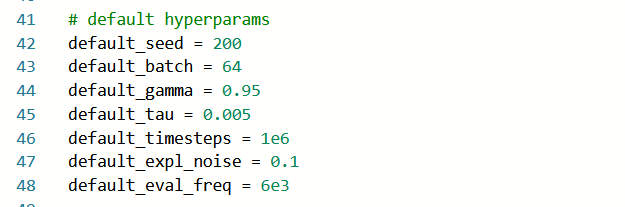
所以在实做的时候，我们需要4个网络。actor, critic, Actor\_target, cirtic\_target。



**三、算法实现**

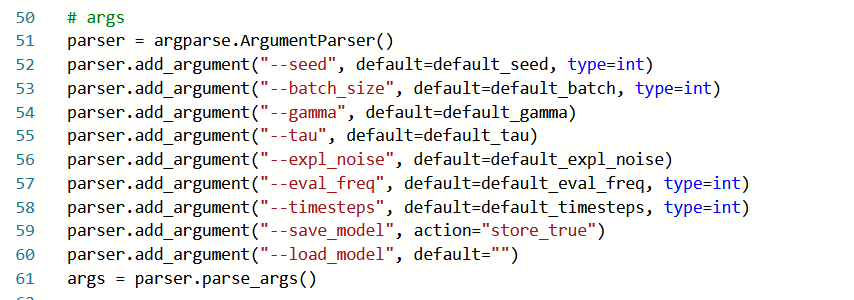
**1、定义超参数**

我们首先定义网络中的超参数，比如经验池的大小，两个网络的学习率等等:



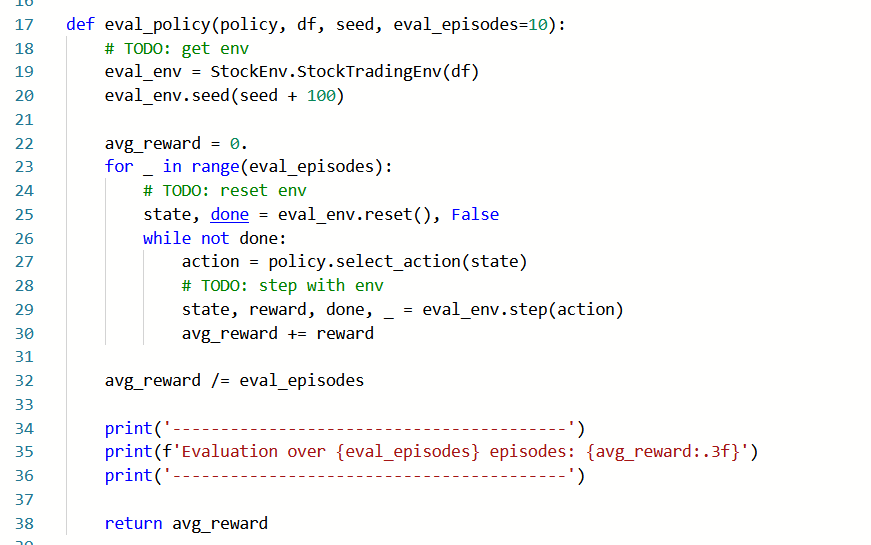
2、**定义网络输入**

我们需要定义的placeholder包括当前的状态S，下一时刻的状态S',以及对应的奖励R，而动作A由Actor得到，因此不需要再定义：



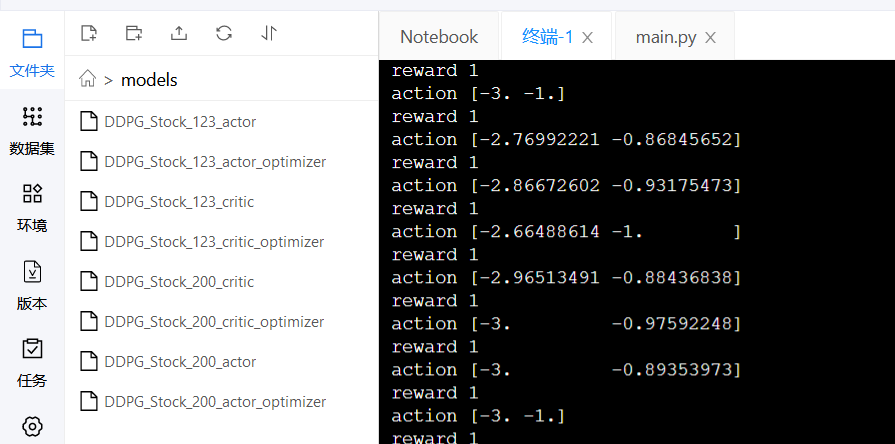
3、**构建两个网络**

两个网络都是两层全链接的神经网络，Actor输出一个具体的动作，而Critic网络输出一个具体的Q值。

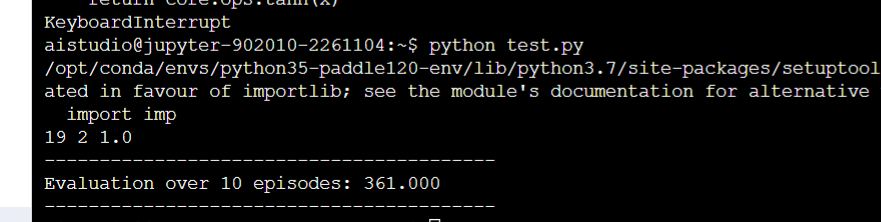


**四、运行结果**

1、在终端输入指令python main.py --save\_model，训练得到模型文件



2、利用main文件训练的模型，用相同的种子进行测试，结果为361.000



**五、实验心得**

DDPG 通过 off-policy 的方式来训练一个确定性策略。因为策略是确定的，如果 agent 使用同策略来探索，在一开始的时候，它会很可能不会尝试足够多的 action 来找到有用的学习信号。为了让 DDPG 的策略更好地探索，我们在训练的时候给它们的 action 加了噪音。DDPG 的原作者推荐使用时间相关的 OU noise，但最近的结果表明不相关的、均值为 0 的 Gaussian noise 的效果非常好。由于后者更简单，因此我们更喜欢使用它。为了便于获得更高质量的训练数据，你可以在训练过程中把噪声变小。

在测试的时候，为了查看策略利用它学到的东西的表现，我们不会在 action 中加噪音。