深度强化学习方法第一次作业说明文件：**多臂老虎机问题**

一、问题重述

基于强化学习的课程背景，我们学习了"k臂老虎机"问题，玩家每次拉动老虎机k个控制杆的其中一个控制杆，希望能够得到最大化奖励。动作可以描述为玩家需要重复在k个选项中做出选择，收益为动作选择带来的一个R_t的收益。为了最大化奖励，玩家需要学习的最优策略是能一直选择期望收益最大的动作。其中的几种常见算法包括：贪婪算法、乐观初始值算法、置信度上界 UCB 算法和偏好函数梯度算法。

在本次实验的多臂老虎机问题设置中，设定臂数$k=15$，每个臂的 reward 服从高斯分布，其中高斯均值μ从 1 取到 15 为$[1，15]$，标准差σ为1。在本次实验中编写 python 程序，实现 UCB 算法和$\varepsilon-$Greedy 算法，绘制图像比较两种算法性能。实验结果发现，相较于贪婪算法只能专注于某个特定结果，获得局部极小值，可以发现$\varepsilon-$贪心算法能够取得优于贪婪算法的结果，因为其能以一定的概率搜索可能的最优策略，但也容易陷入局部极值，在较大的ε值下能期待较好的结果，而 UCB 算法因为其特性能以很大概率探索得到最优结果。实验也尝试了 Sutton 书中提到乐观初始值算法，根据理论其估量结果是有偏的

二、实验环境

硬件环境 PC 机,CPU Intel(R)Core (TM) I7-9750H@2.60GHz 内存 16GB

软件环境 PyCharm Community Edition 2021.2.2，python 版本 3.7.6

三、问题假设及符号说明

假设 1：每个控制杆的收益都服从一定均值方差的正态分布。

假设 2：相邻控制杆的收益相互独立，且不随时间变化，为平稳过程。

假设 3：策略的改变依靠历史，用户理性遵从特定算法。

符号	**说明**
$\boldsymbol{q_*(a)}$	动作 a 对应的价值均值
$\boldsymbol{Q_t(a)}$	动作 a 在时刻 t 时的价值的估计
$\boldsymbol{v_{q_*}(a)}$	动作 a 对应的价值方差
$\boldsymbol{actiontime(a)}$	动作 a 被选择的次数
$\boldsymbol{A_t}$	时刻 t 的动作
$\boldsymbol{R_t}$	时刻 t 的收益

四、实验方法

1、 k臂老虎机问题建模：

在"k臂老虎机"问题中，k个动作中的每一个在被选择时都有一个期望价值$Q_t(a)$，在实际中与程序编写中为给定的常数：$q_*(a)$，由于我们假设收益遵循正态分布，可以认为前面的动作价值是实际分布的均值，方差定义为$v_{q_*}(a)$，给定为 1。由于在实验之前没有先验知识，那么我们只有根据历史数据对t时刻每个动作的价值进行估计,定义为$Q_t(a)$。在过程中我们需要记录每个动作出现的次数，定义为 $actiontime(a)$，通过出现次数在如 UCB 算法中来更新估计价值。以上四个关于动作的变量均为维度为 k 的向量，对应着 k 个不同的杆。我们将t选择的动作记作A_t，收益记作R_t，其为从高斯分布中抽样得到的 t 时刻收益。 由于我们不知道动作的真实价值$q_*(a)$，只有通过$Q_t(a)$来近似的估计动作的真实价值，由大数定律可知，当尝试足够多的次数以后，$Q_t(a)$在收敛的前提下能一致收敛到$q_*(a)$。

$$q_*(a) = E(R_t | A_t = a)$$

2、 问题策略与解决：

1). 贪婪策略

编程与书中内容保持一致，在选择杆的时候，我们按照贪婪策略（可以理解$\varepsilon=0$）或

者$\varepsilon-$贪婪策略，进行臂的选择，当随机数生成不在ε范围内，即满足$1-\varepsilon$，此时选择开发；选择当前价值最大的策略，如果随机数在ε范围内，此时选择试探，通过随机数生成一个$0-15$之间的整数，表示从 15 个杆中选择的随机选择的杆。在程序中为函数 selectAnArm()；同时每一个杆生成的奖励满足正态分布，那么也需要在满足均值为$q_*(a)$，方差为$v_{q_*}(a)$的正态分布中随机生成奖励，在程序中为 getReward(selectedAction):

```
# use a random choice
def selectAnArm():
    temp = np.random.randint(0, 15)
    return int(temp)

# get Reward value via a gauss distribution
def getReward(selectedAction):
    meanQa = qa_star[selectedAction]
    print("meanQa=", meanQa)
    varQa = var_qa[selectedAction]
    print("varQa=", varQa)
    temp = np.random.normal(meanQa, varQa, 1)
    return temp[0]
```

在每一次选择动作并且得到奖励后，需要更新在 t 时刻选择动作 a 的估计价值$Q_t(a)$，在此更新$Q_t(a)$的过程也是通过之前的旧的$Q_t(a)$，并加上学习了与修正的乘积得到的，公式为：

$$Q_t(a)=Q_{t-1}(a)+\alpha(R_a-Q_{t-1}(a))$$

其中R_a为 t 时刻选择动作 a 的收益R_t，在程序中为函数 updatQa：

```
def updateQa(selectedAction, t, Ra):
    Qa[selectedAction] = Qa[selectedAction] + alpha * (Ra - Qa[selectedAction])
    actionTimes[selectedAction] = actionTimes[selectedAction] + 1
```

在我们的实验中，根据实验要求选择$k=15$十五臂老虎机，选择的迭代次数$steps=20000$，步长$\alpha=0.1$，在一般的更新中，也可能会用到$\alpha=\frac{1}{n}$，这样用的参数会使得 reward 时间越早，权重越小，使得模型更多考虑近期情况，会在非平稳过程中使用。

2). UCB 策略

相较于贪婪算法容易陷入局部极小值以及乐观初始值的有偏估计, UCB 算法的思路为：计算每个动作的 reward 置信区间上界, 将上界值作为动作估计值, 选择上界值最大的动作。其中动作选择算法为：

$$A_t=argmax_a[Q_t(a)+c\sqrt{\frac{\log t}{actiontime(a)}}]$$

上式带来的代码变化为计算一个新的动作评价值：

```
temp = Qa + c * np.sqrt(np.log(t+1)/actionTimes)
```

其核心思想是，会随着时间的推移重新考虑各个不同动作的可能潜在价值，以及更多试探没有试探过的策略，减少试探过很多次的策略，同时考虑动作的固有价值。

3). 乐观初始值策略

相较于贪婪算法，在乐观初始值中，我们通过对$Q(a)$在迭代最开始时给予一个不为 0 的正数价值估计，这样就可以在开始的阶段尽可能多的探索更多的策略。在程序编写中通过更

改$Q(a)$即可实现。

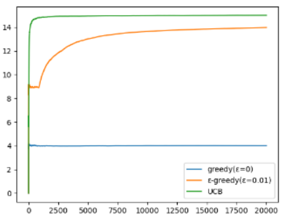
3、 实验结果:

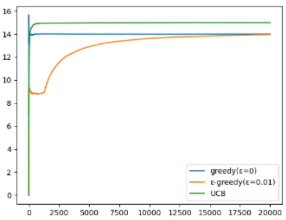
1). 贪婪策略

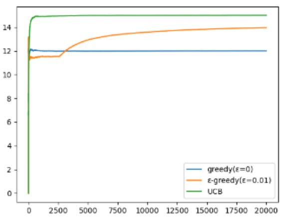
在上述的建模和参数给定情况下，运行 20000 次迭代选择，平均的收益情况在下图中所示，可以看到由于最初几步的随机选择动作，因此无论在$\varepsilon = 0$或$\varepsilon = 0.01$的结果，都容易陷入局部极小值，只有$\varepsilon \neq 0$才有一定可能逃出局部极值从而逼近最优选择，但是当迭代步有限的条件下平均收益总是低于最优收益（15），也低于 UCB 算法的平均收益。

2). UCB 策略

当如《强化学习第 2 版》我们将c超参数的值选为 2，编写程序得到结果，发现我们的结果与书中图 Figure2.6 类似，UCB 的结果能显著优于贪婪策略，将所有的实验结果作图如下，我们随机运行多次，可以发现 UCB 的结果更鲁棒，能在每次接近最优结果，而贪婪算法则会在每次实验的不同随机选择中得到不同结果。

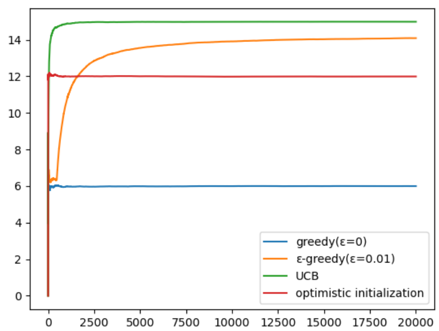


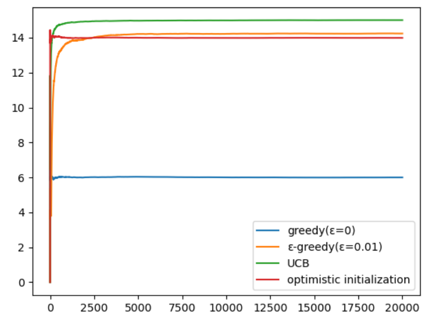




3). 乐观初始值

设置$Q(a)$在迭代最开始时为 5 时，可以观察到在最开始时能大范围地寻找最优策略，而且实验结果显示能以一定概率选中最优选项，但是结果是有偏的估计。而且相对于上述的方法，能看见更为明显的初始时的尖峰，这就是最开始广泛进行策略搜索时所带来的特性





。