17 Reinforcement Learning - Q-Learning

本节实验目标：在网格世界中实现和体会Q-Learning算法。

报告说明（重要）：本部分“马尔可夫决策和强化学习”包含三周内容，本节课为第二周内容，请留意该文档目录中标有“报告应实现任务[分值]”的部分，请在实现后及时保存文件和运行结果截图，在下一周课程报告中一并提交。

Introduction

在该堂实验课，你将**实现一系列Q函数。**

实验代码包括一个autograder，以下命令对所有问题运行以测试你的实现：

```
python autograder.py
```

你也可以通过以下形式的命令做特定case的测试：

```
python autograder.py -t test_cases/q2/1-bridge-grid
```

实验代码

你应该要编辑的文件：	
valueIterationAgents.py	Q1所需：用于解决已知 MDP 的价值迭代agent
qlearningAgents.py	Q3-6所需：Q-learning agents for Gridworld, Crawler and Pacman
analysis.py	Q2所需
你应该看但不要编辑的文件：	
mdp.py	Defines methods on general MDPs. 里面的methods可以调用。
learningAgents.py	Defines the base classes `ValueEstimationAgent` and `QLearningAgent`, which your agents will extend.
util.py	Utilities, including `util.Counter`, which is particularly useful for Q-learners.
gridworld.py	The Gridworld implementation.
featureExtractors.py	Classes for extracting features on (state, action) pairs. Used for the approximate Q-learning agent (in `qlearningAgents.py`).
你可以忽略的文件：	
environment.py	Abstract class for general reinforcement learning environments. Used by `gridworld.py`.
graphicsGridworldDisplay.py	Gridworld graphical display.
graphicsUtils.py	Graphics utilities.
textGridworldDisplay.py	Plug-in for the Gridworld text interface.
crawler.py	The crawler code and test harness. You will run this but not edit it.
graphicsCrawlerDisplay.py	GUI for the crawler robot.
autograder.py	Project autograder

你应该要编辑的文件:	
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes
test_cases/	Directory containing the test cases for each question
reinforcementTestClasses.py	Specific autograding test classes

注意事项

- 请***不要***更改代码中提供的任何函数或类的名称，否则会对 autograder 造成严重破坏。
- **正确使用数据集**：你在此项目中的部分分数取决于你训练的模型在自动评分器的测试集上的表现。我们不提供任何 API 供你直接访问测试集。
- 如果你发现自己卡在某事上，请联系老师、助教或同学寻求帮助。我们希望这些项目是有益的和指导性的，而不是令人沮丧和沮丧的。

MDPs in Grid World

首先，试着用键盘上下左右箭头在手动控制模式下运行 Gridworld：

```
python gridworld.py -m
```

你将看到课堂上所讲Grid World的两个出口布局。 蓝点是agent。 请注意，**当你按下 *up* 时，agent实际上只有 80% 的时间向北移动。 这就是这个世界中agent的生活，总有不如意的时候！**

你可以控制模拟器的许多方面。 运行以下命令可获得完整的选项列表：

```
python gridworld.py -h
```

默认agent随机移动

```
python gridworld.py -g MazeGrid
```

你应该看到随机agent在网格上来回弹跳，直到它到达了exit的网格（右上角）时。对于 AI agent来说，这不是最好的情况。

- 注意：Gridworld MDP 是这样的，你首先必须进入预终止状态（**GUI 中显示的右上角双框），然后在一段经历（episode）实际结束之前采取特殊的"exit"操作（真正的终止状态称为 `TERMINAL_STATE` ，未在 GUI 中显示）**。如果你手动运行一次 episode，你的总回报可能会低于你的预期，这是由于**折扣因子（可通过 `-d` 更改；默认为 0.9）**的存在。
- 查看跟图形界面同时输出的**控制台(Terminal)输出，你可以看到agent经历的每次转换**（要关闭此功能，请在上述命令中使用 `-q` ）。
- 与 Pacman 中一样，agent 位置由 `(x,y)` 笛卡尔坐标表示，数组由 `[x][y]` 索引， `'north'` 动作增加 `y` ，等等。默认情况下，大多数动作将获得零回报。**你可以使用实时奖励选项 (`-r`) 更改回报值。**

Question 1-2 请查看上节课的实验文档并先完成

Question 3: Q-Learning 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q3 的运行结果，获得评分器给的5分中，这道题才算得它占的1分。**

请注意，你前面的**价值迭代agent实际上并没有从经历中学习**。相反，它会在与真实环境交互之前考虑其 MDP 模型以达成完整的策略。当它与环境交互时，它只是遵循预先计算的策略（例如，它成为一个**反射agent**）。这种区别在像 Gridword 这样的模拟环境中可能很微妙，但在现实世界中非常重要，因为**在现实世界中不知道真正的 MDP**。

现在，你**将在 `qlearningAgents.py` 实现一个 Q-learning agent**，它通过其 `update(state, action, nextState, reward)` 方法从其与环境的交互中通过反复试验来学习。 Q-learner 在 `qlearningAgents.py` 的 `QLearningAgent` 中指定，你可以使用选项 `'-a q'` 在命令行中选择它作为agent。

对于这个问题，你必须实现 `update` 、 `computeValueFromQValues` 、 `getQValue` 和 `computeActionFromQValues` 方法。

注意： 对于`computeActionFromQValues`，**你应该随机打破平局（即两个或以上Q值相同时）以获得更好的行为。**`random.choice()` **函数会有所帮助。**在特定状态下，你的agent之前未看到的操作仍然具有 Q 值，特别是为零的 Q 值，并且如果你的agent之前看到的所有行动都具有负 Q 值，一个还没试过的行动可能是最优的。

***重要提示*：确保在你的** `computeValueFromQValues` **和** `computeActionFromQValues` **函数中，你只能通过调用** `getQValue` **来访问 Q 值。**当你重写`getQValue`以使用 状态-动作对 的功能而不是直接使用 状态-动作对时，此抽象对于后面 Question 6 很有用。

随着 Q-learning 更新到位，你可以使用键盘在**手动控制**下观看你的 Q-learner 学习过程：

```
python gridworld.py -a q -k 5 -m
```

`-k` 将控制你的agent的episodes数量。 观察agent如何了解它刚刚所处的状态，而不是它移动到的状态，并“让学习随波逐流 (leaves learning in its wake)”。 提示：为了帮助调试，你可以使用 `--noise 0.0` 参数关闭噪声（尽管这显然会降低 Q-learning 的趣味性）。

评分: Autograder 将运行你的 Q-learning agent，并检查当提供相同的示例集时，agent 是否学习了与我们的参考实现相同的 Q 值和策略。要对你的实现进行评分，请运行autograder：

```
python autograder.py -q q3
```

Question 4: Epsilon Greedy 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q4 的运行结果，获得评分器给的2分中，这道题才算得它占的1分。**

通过在 `getAction` **中实现 epsilon-greedy 行动**选择来完成你的 Q-learning agent，这意味着它会在一小部分时间内选择随机动作，在其它时间遵循其当前的最佳 Q 值。 请注意，选择随机行动可能也会导致选择最佳行动 - 也就是说，你不应选择随机的次优行动，而应选择任何随机的符合条件的行动。

你可以通过调用 `random.choice` 函数从列表中均匀随机选择一个元素。 你可以使用 `util.flipCoin(p)` 来模拟具有成功概率 `p` 的二进制变量，它返回具有概率 `p` 的 `True` 和具有概率 `1-p` 的 `False`。

实现 `getAction` 方法后，观察网格世界中agent的以下行为（默认epsilon = 0.3）。

```
python gridworld.py -a q -k 100
```

你的最终 Q 值应该类似于你的价值迭代agent的值，尤其是在比较好的路径上。 但是，由于随机行动和处于初始学习阶段，你的平均回报将低于 Q 值预测。

你还可以观察以下针对不同 epsilon 值的模拟。 agent的行为是否符合你的预期?

```
python gridworld.py -a q -k 100 --noise 0.0 -e 0.1
```

```
python gridworld.py -a q -k 100 --noise 0.0 -e 0.9
```

要测试你的实现，请运行autograder:

```
python autograder.py -q q4
```

无需额外代码，你现在应该能够运行 Q-learning crawler robot：

```
python crawler.py
```

如果这不起作用，你可能编写了一些过于针对 `GridWorld` 具体问题的代码，你应该使其对所有 MDP 更通用。

这将使用你的 Q-learner 从调用crawler robot。 玩转各种学习参数，看看它们如何影响agent的策略和行动。 请注意，步长延迟是模拟的参数，而学习率和 epsilon 是你的学习算法的参数，折扣因子是环境的属性。

Question 5: Q-Learning and Pacman 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q5 的运行结果，获得评分器给的1分中，这道题才算得它占的1分。**

注意：本题无需实现，如果Q3,Q4实现的好，本题应也能顺利通过autograder评分

是时候玩吃豆人了！ Pacman 将分两个阶段进行游戏。 在第一阶段，训练中，吃豆人将开始了解位置和动作的价值。 因为即使对于微小的网格也需要很长时间才能学习准确的 Q 值，所以 Pacman 的训练游戏默认以安静模式运行，没有 GUI（或控制台）显示。 吃豆人的训练完成后，他将进入测试模式。 测试时，Pacman 的 `self.epsilon` 和 `self.alph` 将被设置为 0.0，有效地停止 Q-learning 并禁用exploration，以便让 Pacman 能够利用它学习到的策略。 测试游戏默认显示在 GUI 中。 **无需任何代码更改，你应该能够为非常小的网格运行 Q-learning Pacman，如下所示：**

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

请注意，已经根据你已经编写的 `QLearningAgent` 为你定义了 `PacmanQAgent` **（本题无需新的实现）。** `PacmanQAgent` 的不同之处仅在于它具有对 Pacman 问题更有效的默认学习参数（`epsilon=0.05, alpha=0.2, gamma=0.8`）。如果上面的命令没有例外地顺利运行并且你的agent至少有 80% 的时间能获胜，那么你将获得此问题的全部分数。在 2000 场训练游戏之后，autograder将运行 100 场测试游戏。

提示： 如果你的 `QLearningAgent` 适用于 `gridworld.py` 和 `crawler.py`，但**似乎没有在** `smallGrid` **上为 Pacman 学习一个好的策略，这可能是因为你的** `getAction` **和/或** `computeActionFromQValues` **方法在某些情况下不能正确考虑看不见的动作。**特别是，因为根据定义，看不见的动作的 Q 值为零，如果所有*已经*见过的动作都具有负 Q 值，那么看不见的动作可能是最优的。当心 util.Counter 的 argmax 函数！

注意： 要为你的实现评分，请运行：

```
python autograder.py -q q5
```

注意：如果你**试验不同的学习参数（玩一下）**，可以使用选项 `-a`，例如

`-a epsilon=0.1,alpha=0.3,gamma=0.7`。 然后，这些值将可以在agent内部以 `self.epsilon, self.gamma` 和 `self.alpha` 的形式访问。

注意： 虽然总共会进行 2010 场比赛，但前 2000 场比赛不会显示，因为选项 `-x 2000` 指定前 2000 场比赛进行训练（无输出）。 因此，你只会看到Pacman玩这些游戏的最后 10 场。 训练游戏的数量也会作为选项 `numTraining` 传递给你的agent。

注意： 如果你想观看 10 场训练比赛以了解发生了什么，请使用以下命令：

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

在训练期间，你将看到每 100 场游戏的输出，其中包含有关 Pacman 表现的统计信息。 **Epsilon 在训练中是正数**，所以即使在学习了一个好的策略之后，Pacman也会玩得很差：这是因为他偶尔会做一个随机的探索动作靠近一个幽灵。作为一个参考，Pacman 对 100 个episode的奖励应该需要 1000 到 1400 场游戏才能变为正数，这反映了他开始赢多于输。到训练结束时，它应该保持相当高的正值（在 100 到 350 之间）。

确保你理解这里发生的情况：MDP 状态是 Pacman 所面临的切确的环境配置(board configuration)，现在复杂的转移描述了对该状态的整个更改。Pacman已经移动但幽灵还没有移动的中间游戏配置***不是*** MDP 状态，而是捆绑在转移中。

一旦 Pacman 完成训练，他应该在测试游戏中非常可靠地获胜（至少 90% 的时间），因为现在他正在利用他学到的策略。

但是，你会发现在看似简单的 `mediumGrid` 上训练相同的agent效果并不好。在我们的参考实现中，Pacman 的平均训练奖励在整个训练过程中都是负数。在测试时，他玩得不好，可能输掉了所有的测试游戏。尽管效果不佳，但**训练也将需要很长时间。**

Pacman 未能在更大的布局上获胜，因为每个board configuration都是具有不同 Q 值的独立状态。他无法就“在所有位置遇到鬼都不利”这事一概而论。显然，**这种Q-Learning方法难以扩展。**

怎么办呢？我们试试Q6.

Question 6: Approximate Q-Learning 报告应实现任务[2分]

- **说明：该小题截图只需截 python autograder.py -q q6 的运行结果，获得评分器给的3分中，这道题才算得它占的2分。**

实现一个Approximate Q-Learning agent，学习状态特征的权重，其中许多状态可能共享相同的特征。 **在** `qlearningAgents.py` **的** `ApproximateQAgent` **类中编写你的实现，**它是 `PacmanQAgent` 的子类。

注意：近似 Q-learning 假设在状态和动作对上存在特征函数 $f(s,a)$，从而产生向量 $[f_1(s,a),...,f_i(s,a),...,f_n(s,a)]$的特征值。 我们**在** `featureExtractors.py` **中为你提供了特征函数。** 特征向量是包含非零特征和值对的 `util.Counter`（像字典）对象； **所有省略的特征的值为零。**

近似 Q 函数采用以下形式：

$Q(s,a) = \sum_{i=1}^{n} f_i(s,a)w_i$

其中每个权重 w_i都与特定特征 $f_i(s,a)$ 相关联。 在你的代码中，你应该将权重向量实现为将特征（特征提取器将返回的）映射到权重值的字典。 你将更新权重向量，类似于更新 Q 值的方式：

$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$

difference $=(r + \gamma max_{a'} Q(s',a')) - Q(s,a)$

请注意，difference 项与正常 Q-learning 中的相同，r 是经历的奖励。

默认情况下，`ApproximateQAgent` 使用 `IdentityExtractor`，**它为每个 `(state,action)` 对分配一个特征。** 使用此特征提取器，你的近似 Q-learning agent应该与 `PacmanQAgent` 一样工作。 你可以使用以下命令对此进行测试：

```
python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
```

重要提示： `ApproximateQAgent` 是 `QLearningAgent` 的子类，因此它**共享多个方法，例如 `getAction`。 确保你在 `QLearningAgent` 中的方法调用 `getQValue` 而不是直接访问Q值，这样当你在近似agent中覆盖实现 `getQValue` 时，新的近似q值将用于计算动作。**

一旦你确信你的近似学习器可以正确使用默认的identity features，请使用我们的**自定义特征提取器**(SimpleExtractor)运行你的近似 Q-learning agent，它可以轻松学会取胜：

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid
```

对于你的 `ApproximateQAgent` 来说，**更大的布局应该都没有问题**（警告：这可能需要几分钟的时间来训练）：

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic
```

如果你没有错误，那么你的近似 Q-learning agent应该几乎每次都能通过这些简单的features获胜，即使只有 50 次训练游戏。

评分：我们将运行approximate Q-learning agent，并检查它是否在当每个都提供相同的示例集时，学习了与我们的参考实现相同的 Q 值和特征权重。**（建议成功实现后也测试一下上面说的三个代码命令！）** 要对你的实现进行评分，请运行autograder：

```
python autograder.py -q q6
```

本节课实现内容保存说明 - 为提交第七次上机报告做准备

- **第18周上机课结束后，一并提交本部分"马尔可夫决策-强化学习"的报告，该报告包含本节课的Q3,Q4,Q5,Q6的代码和运行截图，请按Q3,Q4,Q5,Q6大标题下的说明进行截图，保存。**
- **第16周的Q1,Q2也别忘了哟**
- **本节课实现的代码文件也请保存，你将需要把其中实现的代码加到第18周相应的代码文件中跟第18周的任务实现一并进行提交。**