人工智能导论: **Assignment #4**

张知行 **2015012018**

May 26, 2018

Value Iteration

Description: Write a value iteration agent in ValueIterationAgent, which has been partially specified in valueIterationAgents.py. ValueIterationAgent agent is an offline planner, not a reinforcement learning agent, and so the relevant training option is the number of iterations of value iteration it should run (option -i) in its initial planning phase.

由于实现的 $ValueIterationAgent$ 是一个离线的算法，因此需要在初始化的时候对后续所有情况进行计算并进行存储。在每次迭代中，计算所有状态下的 $QValue$，使用最后的 $QValue$ 以及记录下的每个状态下最好的操作方式作为最终的结果来对 $agent$ 进行操作。每个格点上的 $Value$ 是在该格点所有可能动作下 $QValue$ 的最大值，同样，在该格点处选择的操作也是 $QValue$ 最大的操作对应的值。使用 100 次迭代计算得到的结果如 Figure1和 Figure2所示。

Figure 1: Values after 100 iterations

Figure 2: QValues after 100 iterations

其余设定不变，分别进行 100 次和 5 次迭代之后进行 10 次测试得到的结果，如 Figure3和 Figure4所示。可以看出增加迭代次数会减少错误率，更好地实现路径规划。

Figure 3: Returns after 100 iterations

Figure 4: Returns after 5 iterations

Parameter Analysis in Value Iteration

Description: Change parameters so that the optimal policy causes the agent to attempt to cross the bridge.

这一问中需要深入理解第一问中实现的 $ValueIterationAgent$ 的具体的参数的功能。在初始的参数设定下，agent 会通过局部较好的情况走向分数较小的最终结果，需要改变参数中的 $Discount$ 或者 $Noise$ 的数值来实现 Agent 更为长远的考虑。从 $ValueIterationAgent$ 中使用的计算 $QValue$ 的公式来看，对于越远的位置，由于 discount 造成的影响越大，造成下一步中计算得到向该方向前进的可能性越小，但是初始的 $discount$ 的数值为 0.9 可以向上调整的幅度不大，并且经过测试，过大的 $discount$ 会造成选择特别差的路

径。因此对 $Noise$ 进行修改。通过修改，发现将 $Noise$ 降低到 0.02 之后会很大程度上降低向左的数值，但是仍然会有走向低分的规划。最终测试结果发现 $Noise = 0.001$ 可以很好地实现要求的结果，最终结果如 Figure5。

Figure 5: Values after 100 iterations

Q-Learning

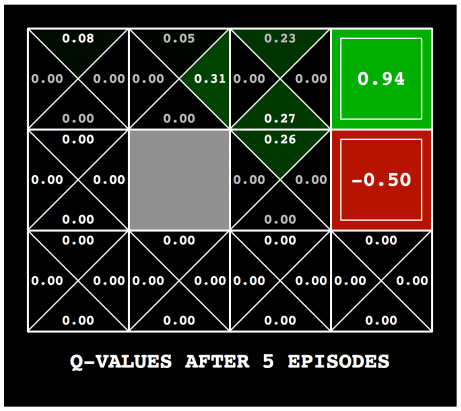

Figure 6: Values after 5 episodes

Figure 7: Values after 100 episodes

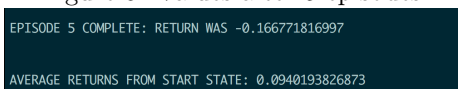

Figure 8: Results after 5 episodes

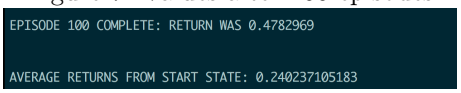

Figure 9: Results after 5 episodes

Description: Implement the update, computeValueFromQValues, getQValue, computeActionFromQValues methods in QLearningAgent in qlearningAgents.py.

Q-Learning continued on next page...

这一问中需要对 *QLearning* 进行实现，是一个通过不断的实践进行学习的算法，核心主要在 *update* 过程中，当前状态进行一个动作对应的 $QValue$ 是之前同样情况下进行相同操作的 $QValue$ 的数值和下一个状态下之前经验的一个组合。分别使用 5 次训练和 100 次训练查看结果如 Figure6,7,8,9。可以看出通过大量的训练测试结果有了很好地提升。

Q-Learning and Pacman

这一问中要求修改 *QLearningAgent* 使其可以适配于 PacMan 的操作，通过测试，发现上一问中实现的代码可以很好地适用于这一问中的测试中，*autograder* 测试结果如 Figure 10。同时，通过测试可以发现，在较小的 Grid 下 PacMan 可以有很好地表现。但是，如果使用较大的 Grid，PacMan 则会出现静止不动的现象，最终会造成失败，主要原因可能是状态过多，学习效果较差，PacMan 无法分辨 Ghost 的危险。因此这种算法并不具有大规模下使用的能力。

```
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q7: 1/1 ###
```

Figure 10: Autograder result

Approximate Q-Learning

Description: Implement an approximate Q-learning agent that learns weights for features of states.

这一问需要使用特征提取函数进行学习，从而可以实现更好的学习效果，和上一问中最大的区别是使用提取出来的特征作为字典的 key 值而不是 (state, action) 的二元组。使用这种学习方法可以看出有着较好的学习效果，很好地解决了上一问中 PacMan 在较大的地图下无法进行判断的问题。最终结果使用各种地图进行测试几乎都会得到胜利的结果，*autograder* 测试结果如 Figure 11。

```
*** PASS: test_cases/q8/1-tinygrid.test
*** PASS: test_cases/q8/2-tinygrid-noisy.test
*** PASS: test_cases/q8/3-bridge.test
*** PASS: test_cases/q8/4-discountgrid.test
*** PASS: test_cases/q8/5-coord-extractor.test

### Question q8: 3/3 ###
```

Figure 11: Autograder result