中山大学数据科学与计算机学院

计算机科学与技术专业-人工智能

本科生实验报告

（2018-2019学年秋季学期）

课程名称：**Artificial Intelligence**

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | 计科2班 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337341 | 姓名 | 朱志儒 |

## 实验题目

**期末项目**

## 实验内容

* **算法原理**

1. **Epsilon-greedy算法**

Epsilon-greedy算法是在“exploit”（利用）和“explore”（探索）之间权衡以尽可能实现收益的最大化。假设当前有n台老虎机，每台老虎机吐钱的概率不一样，也不清楚每台老虎机吐钱的概率分布，如何实现收益最大化呢？通常有两种方案，第一种就是在已经探索过的老虎机中找到吐钱最多的那台老虎机，然后坚持摇它，这就是“exploit”（利用）；第二种就是尝试不断探索新的老虎机，在探索过程中可能会找到更好的老虎机，也可能找到吐钱很少的老虎机，这就是“explore”（探索）。Epsilon-greedy算法是每次以epsilon概率进行“explore”（探索），也就是说在所有的老虎机中选择一个进行探索；每次以1 – epsilon概率进行“exploit”（利用），也就是在已经探索过的老虎机中选择收益最大的那个。

Epsilon设置的越高，收敛到最佳收益的速度越快，因为随机搜索的概率越大，越能更快地探索到最好的老虎机。当0.1 < Epsilon < 0.5时，Epsilon设置的越低，最终的平均收益越高，因为每次以较大概率随机探索，将不能有效利用收益最大的老虎机。

1. **Q-learning**

1） 初始化状态S1和表Q。

2） 第一次循环，根据Epsilon-greedy算法在状态S1中选择动作，选择并执行动作A1后得到状态S2和S2的即时回报R2，然后更新表Q，即

其中表示在状态下的动作中找到具有最大Q值的动作所对应的Q值。

3） 第二次循环，根据Epsilon-greedy算法在状态S2中选择动作，选择并执行动作A2后得到状态S3和S3的即时回报R3，然后更新表Q，即

其中表示在状态下的动作中找到具有最大Q值的动作所对应的Q值。

4） 不断循环直至S到达最终状态。

1. **Sarsa**

1） 初始化状态S1和表Q。

2） 第一次循环，根据Epsilon-greedy算法在状态S1中选择动作，选择并执行动作A1后得到状态S2和S2的即时回报R2，再根据Epsilon-greedy算法在状态S2中选择动作A2，然后更新表Q，即

3） 第二次循环，根据Epsilon-greedy算法在状态S2中选择动作，选择并执行动作A2后得到状态S3和S3的即时回报R3，再根据Epsilon-greedy算法在状态S3中选择动作A3，然后更新表Q，即

4） 不断循环直至S到达最终状态。

1. **遗传算法（队友实现）**

队友使用的是遗传算法，我并未参与实现，所以在此只介绍相关原理。

1. **初始化**

对于某个问题，随机生成该问题的一些解，每个解是一个向量代表一条染色体，一个向量中的不同元素代表一条染色体中的不同基因。根据具体的应用场景设计相应的适应度函数，每条染色体可通过该适应度函数计算出其自身的适应度。

1. **复制**

模仿自然界基因复制和染色体复制，对问题解的复制就是复制其向量，以进入下一次迭代，过程如下：

1. **交叉**

模仿自然界的一对染色体的等位基因相互交换，在两个解向量之间，随机选择几对相同位置的元素相互交换，再进行归一化后就可以进入下一次迭代，过程如下：

1. **突变**

模仿自然界的基因突变，在每次迭代中，随机选择一个解向量，在该解向量上随机选择一个位置的元素，将其值改变为一个随机数，再进行归一化后得到一个新的解向量，然后进入下一次迭代，过程如下：

随机选择一个解向量：

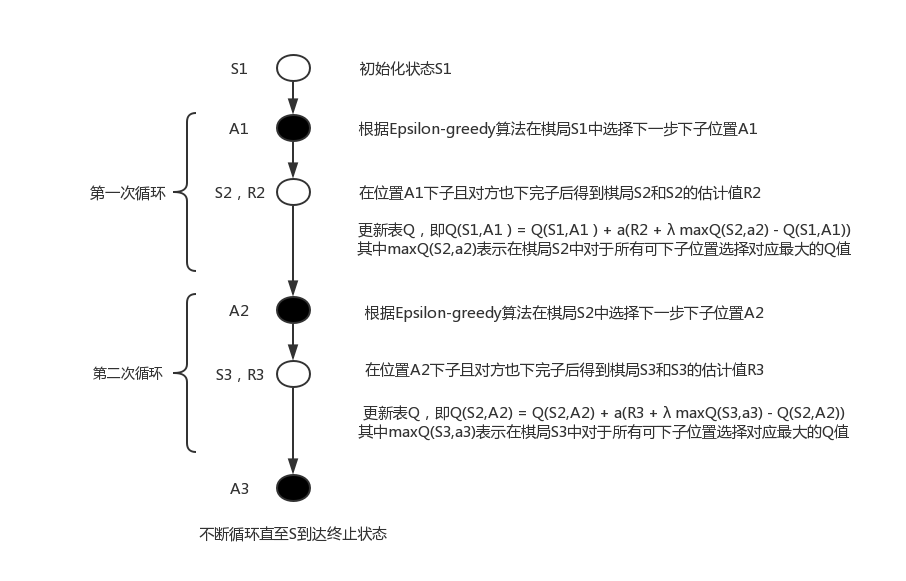
在该向量中随机选择一个元素，将其改为一个随机数，再进行归一化：

1. **结束**

当解向量收敛时，迭代就可以结束，当然也可以直接限制迭代次数。

* **流程图**

**1、Q-learning**

****

**2、Sarsa**

****

* **关键代码**

**1）Q-learning**

Q-learning在黑白棋中的应用中，每个不同的状态S表示每个不同的棋局，动作a表示下一步下子的位置，动作集A表示可下子位置的集合，状态S的即时回报R表示S所对应棋局的估计值。Q-learning在黑白棋中的步骤为：

1. 初始化棋局S1和表Q。
2. 进入循环，根据Epsilon-greedy算法在棋局S1中选择下一步下子位置A1，在位置A1下子且对方也下完子后得到棋局S2和S2的估计值R2，然后更新表Q，即

其中表示在棋局中对于所有可下子位置选择对应最大的Q值。

1. 不断循环直至游戏结束。
2. **void** Q\_learning(**char** board[8][8], **char** color, **void**(\*function)(**char** board[8][8], **char**)) {
3. **char** opp = color == '@' ? 'O' : '@';
4. **double** alpha = 0.8, gama = 0.5;
5. state pre = state(board);
6. default\_random\_engine e;
7. uniform\_real\_distribution<**double**> u(0.0, 1.0);
8. **double** epsilon = u(e);
9. auto next\_step = show\_places(board, color);
10. // 根据Epsilon-greedy算法选择动作
11. **if** (1 - epsilon < 0.5) {
12. // 以1 – epsilon概率进行“exploit”（利用），在已经探索过的动作中选择收益最大的
13. **double** best\_value = -10000000;
14. pre.action = next\_step[0];
15. **for** (**int** i = 0; i < next\_step.size(); ++i) {
16. auto index = state(pre.board, next\_step[i]);
17. **if** (Q.find(index) == Q.end())
18. **continue**;
19. **else** **if** (Q[index] > best\_value) {
20. best\_value = Q[index];
21. pre.action = next\_step[i];
22. }}}
23. **else**
24. // 以epsilon概率进行“explore”（探索），在所有的动作种随机选择一个
25. pre.action = next\_step[rand() % next\_step.size()];
26. //进入循环
27. **while** (!is\_over(board)) {
28. // 执行选好的动作
29. move(board, pre.action, color);
30. **if** (!show\_places(board, opp).empty() && !show\_places(board, opp).empty())
31. function(board, opp);
32. **while** (!is\_over(board) && show\_places(board, color).empty())
33. function(board, opp);
34. // 得到下一个状态
35. state S = state(board);
36. // 得到下一个状态的即时回报
37. **double** reward = evaluate(board, color);
38. next\_step = show\_places(board, color);
39. // 在下一个状态下找到具有最大Q值的动作
40. **if** (!next\_step.empty()) {
41. **double** best\_value = -10000000;
42. S.action = next\_step[0];
43. **for** (**int** i = 0; i < next\_step.size(); ++i) {
44. auto index = state(board, next\_step[i]);
45. **if** (Q.find(index) == Q.end())
46. **continue**;
47. **else** **if** (Q[index] > best\_value) {
48. best\_value = Q[index];
49. S.action = next\_step[i];
50. }}}
51. **double** new\_q = 0, old\_q = 0;
52. **if** (Q.find(pre) != Q.end())
53. old\_q = Q[pre];
54. **if** (Q.find(S) != Q.end())
55. new\_q = Q[S];
56. // 更新表Q
57. Q[pre] = old\_q + alpha \* (reward + gama \* new\_q - old\_q);
58. // 以epsilon概率进行“explore”（探索）
59. epsilon = u(e);
60. **if** (!next\_step.empty() && epsilon < 0.5)
61. S.action = next\_step[rand() % next\_step.size()];
62. pre = S;}}

**2）Sarsa**

Sarsa在黑白棋的应用中，状态S、动作a、动作集A、即时回报R所表示的意义与Q-learning在黑白棋的应用中的相同，Sarsa在在黑白棋中的步骤为：

a. 初始化棋局S1和表Q。

b. 进入循环，根据Epsilon-greedy算法在棋局S1中选择下一步下子位置A1，在位置A1下子且对方也下完子后得到棋局S2和S2的估计值R2，再根据Epsilon-greedy算法在棋局S2中选择下一步下子位置A2，然后更新表Q，即

1. 不断循环直至游戏结束。
2. **void** Sarsa(**char** board[8][8], **char** color, **void** (\* function)(**char** board[8][8], **char**)) {
3. **char** opp = color == '@' ? 'O' : '@';
4. **double** alpha = 0.8, gama = 0.5;
5. state pre = state(board);
6. default\_random\_engine e;
7. uniform\_real\_distribution<**double**> u(0.0, 1.0);
8. **double** epsilon = u(e);
9. auto next\_step = show\_places(board, color);
10. // 根据Epsilon-greedy算法选择动作
11. **if** (1 - epsilon < 0.5) {
12. // 以1 – epsilon概率进行“exploit”（利用），在已经探索过的动作中选择收益最大的
13. **double** best\_value = -10000000;
14. pre.action = next\_step[0];
15. **for** (**int** i = 0; i < next\_step.size(); ++i) {
16. auto index = state(pre.board, next\_step[i]);
17. **if** (Sarsa\_Q.find(index) == Sarsa\_Q.end())
18. **continue**;
19. **else** **if** (Sarsa\_Q[index] > best\_value) {
20. best\_value = Sarsa\_Q[index];
21. pre.action = next\_step[i];
22. }}}
23. **else**
24. // 以epsilon概率进行“explore”（探索），在所有的动作种随机选择一个
25. pre.action = next\_step[rand() % next\_step.size()];
26. // 进入循环
27. **while** (!is\_over(board)) {
28. // 执行已选好的动作
29. move(board, pre.action, color);
30. **if** (!show\_places(board, opp).empty() && !show\_places(board, opp).empty())
31. function(board, opp);
32. **while** (!is\_over(board) && show\_places(board, color).empty())
33. function(board, opp);
34. // 得到下一个状态
35. state S = state(board);
36. // 得到下一个状态的即时回报
37. **double** reward = evaluate(board, color);
38. epsilon = u(e);
39. next\_step = show\_places(board, color);
40. // 根据Epsilon-greedy算法选择动作
41. **if** (!next\_step.empty())
42. **if** (1 - epsilon < 0.5) {
43. // 以1 – epsilon概率进行“exploit”（利用）
44. **double** best\_value = -10000000;
45. S.action = next\_step[0];
46. **for** (**int** i = 0; i < next\_step.size(); ++i) {
47. auto index = state(board, next\_step[i]);
48. **if** (Sarsa\_Q.find(index) == Sarsa\_Q.end())
49. **continue**;
50. **else** **if** (Sarsa\_Q[index] > best\_value) {
51. best\_value = Sarsa\_Q[index];
52. S.action = next\_step[i];
53. }}}
54. **else**
55. // 以epsilon概率进行“explore”（探索）
56. S.action = next\_step[rand() % next\_step.size()];
57. **double** new\_q = 0, old\_q = 0;
58. **if** (Sarsa\_Q.find(pre) != Sarsa\_Q.end())
59. old\_q = Sarsa\_Q[pre];
60. **if** (Sarsa\_Q.find(S) != Sarsa\_Q.end())
61. new\_q = Sarsa\_Q[S];
62. // 更新表Q
63. Sarsa\_Q[pre] = old\_q + alpha \* (reward + gama \* new\_q - old\_q);
64. pre = S;}}

## 实验结果及分析

* **实验结果展示**

**1、Q-learning**

将设置为0.5，学习速率α设置为0.8，折扣因子γ设置为0.5，与随机数对局训练50万局，Q表中存储了12559838个不同的棋局S和下子位置A以及对应的Q值。与随机数对战100局，胜率为68%。

**2、Sarsa**

将设置为0.5，学习速率α设置为0.8，折扣因子γ设置为0.5，与随机数对局训练100万局，Q表中存储了24834550个不同的棋局S和下子位置A以及对应的Q值。与随机数对战100局，胜率为69%。

**3、Alphabeta剪枝**

Alphabeta剪枝使用的是实验八中的代码，只不过在评价函数中新增了内部稳定子的估计值和靠近边角的估计值，设置搜索深度为6，与随机数对战100局，胜率为92%。

* **评测指标展示**

从实验结果展示可以看出，强化学习中的Q-learning和Sarsa并不适合黑白棋，因为黑白棋具有太多的棋局和对应每个棋局不同的下子位置，这导致实际运用中根本无法存储这么多种情况，再者它与随机数对局训练，这导致每次训练的棋局也会有所不同，从而Q表中对应的值收敛较慢，即时训练100万次，但对于1亿多个棋局和下子位置，这些训练次数无法使得它们对应的Q值收敛。

与强化学习中的Q-learning和Sarsa相反，Alphabeta剪枝适合黑白棋，从实验结果展示可以看出它的胜率远远高于另外两个。