Reinforcement learning
强化学习

Fan Min

Lab of machine learning, Southwest Petroleum University
www.fansmale.com
minfan@swpu.edu.cn
minfanphd@163.com
Code: https://github.com/FanSmale/MFReinforcement

August 25, 2020

目录

1. 引言

- 1.1 动机
- 1.2 简史
- 1.3 应用
- 1.4 讨论与作业

1.1 动机

- Learning from trial

1.2 简史

- Stage 1:

1.3 典型应用

- 游戏
- 自动驾驶

1.4 讨论与作业

- Questions

2. 简例

- 2.1 迷宫
- 2.2 井字棋
- 2.3 相同点分析
- 2.4 不同点分析
- 2.5 讨论与作业

2.1 迷宫

- 输入: 迷宫(入口、出口、墙)
- 输出: 从入口到出口的路径
- 优化目标: 路径最短

2.1.1 单源最短路径

- 构造无向图
  - 每个非墙坐标点为一个节点, 墙为无效节点;
  - 如果节点A与(上、下、左、右)B直接相邻, 则其距离为1
  - 以入口为源(出口也行)
- 计算单源最短路径, 同时获得路径长度与路径本身
- 时间复杂度仅为$O(N^2)$, 其中N为节点个数
- 代码见independent/maze/ShortestPathLearning.java
- 该方案用于获得标准答案

迷宫例

Table: 迷宫(s入口, +表示出口, -表示墙)

s				
−				
		−	−	
			+	
	−		−	

单源最短路径步骤1: 节点编号

Table: 编号

0	1	2	3	4
–	6	7	8	9
10	11	–	–	14
15	16	17	**18**	19
20	–	22	–	24

寻找节点0到18的单源最短路径

2.1.2 强化学习的方案

- Agent与Environment的交互
- Agent试错, Environment反馈, Agent越来越聪明

Environment建模

- 令二维迷宫大小为$n \times n$ (容易扩展到三维及其它形状)
- 用矩阵$M = m_{ij}$表示迷宫, $m_{ij} = 10$表示出口, $m_{ij} = -10$表示墙, $m_{ij} = 0$表示其它节点
- 代码见environment/Maze.java

Agent建模

- Agent到达出口获得奖励(如100), 撞墙获得惩罚(如-100), 向前走一步获得惩罚(如-1)
- Agent可以训练很多轮(如1000 episodes), 每轮均从入口到出口
- Agent边走边记, 如这次在位置A向右走撞墙, 则下次到同一位置知道该信息

Q-learning

- Agent边走边训练Q矩阵, 其行数为有效状态数(位置数、节点数). 简便起见, 可以假设Agent知道迷宫大小
- Agent列数为4, 即向上、下、左、右4个方向走获得的奖励值
- Agent根据Q矩阵的值决定下一步怎么走: 训练阶段用随机或带权随机方式, 测试阶段用贪心方式
- 随机方式代码: agent/SimpleQAgent.java
- 带权随机方式代码: agent/WeightedRandomQAgent.java

Q矩阵的训练方式

- 如果从节点(i,j)向右走碰墙, 则可以将$q_{(i-1)n+j,3}$设置为-100或$-\infty$, 避免下次犯错. 其中3对应于RIGHT
- 如果从节点(i,j)向右走不碰墙且到$(i,j+1)$, 则根据$(i,j+1)$四个方向最大的Q值来更新$q_{(i-1)n+j,3}$
- 代码见agent/QAgent.java中的learn()方法
- 不同的策略、参数影响了学习的效率、效果. 可以继承该类, 覆盖learn()方法获得其它算法

Q矩阵跟踪(episode 0, 已经有奖励为10的行为)

State	UP	DOWN	LEFT	RIGHT
0	0.0	-100.0	0.0	-0.19
1	0.0	-0.19	-0.1	-0.19
2	0.0	-0.19	-0.1	-0.271
3	0.0	-0.19	-0.19	-0.271
4	0.0	-0.271	-0.19	0.0
5	0.0	0.0	0.0	0.0
6	-0.19	-0.1	-100.0	-0.19
7	-0.19	-100.0	-0.19	-0.19
8	-0.19	-100.0	-0.19	-0.19
9	-0.19	-0.271	-0.19	0.0
10	-100.0	-0.1	0.0	0.0
11	-0.1	0.0	-0.1	-100.0

State	UP	DOWN	LEFT	RIGHT
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	-0.19	-0.19	-100.0	0.0
15	0.0	-0.1	0.0	-0.19
16	-0.1	-100.0	-0.1	-0.1
17	-100.0	0.0	-0.1	0.0
18	0.0	0.0	0.0	0.0
19	-0.1	0.0	10.0	0.0
20	-0.1	0.0	0.0	-100.0
21	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0

Q矩阵跟踪(episode 4, 奖励范围不断扩散)

State	UP	DOWN	LEFT	RIGHT
0	0.0	-100.0	0.0	-0.7458
1	0.0	-0.6125	-0.5695	-0.6125
2	0.0	-0.4685	-0.5217	-0.4685
3	0.0	-0.4095	-0.4095	-0.4095
4	0.0	-0.4095	-0.3439	0.0
5	0.0	0.0	0.0	0.0
6	-0.4685	-0.5217	-100.0	-0.4685
7	-0.4095	-100.0	-0.4685	-0.4685
8	-0.4095	-100.0	-0.4095	-0.4095
9	-0.3439	-0.3439	-0.3439	0.0
10	-100.0	-0.271	0.0	-0.271
11	-0.3439	-0.2727	-0.3439	-100.0

State	UP	DOWN	LEFT	RIGHT
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	-0.19	0.7190	-100.0	0.0
15	-0.19	-0.19	0.0	-0.271
16	-0.19	-100.0	-0.19	2.4280
17	-100.0	-0.1	-0.1	27.1
18	0.0	0.0	0.0	0.0
19	-0.1	0.0	19.0	0.0
20	-0.19	0.0	0.0	-100.0
21	0.0	0.0	0.0	0.0
22	-0.1	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0

Q矩阵(episode 99, 各状态最大奖励值指明最短路径)

State	UP	DOWN	LEFT	RIGHT
0	0.0	-100.0	0.0	85.232
1	0.0	91.246	-0.8905	-0.5791
2	0.0	-0.7941	9.9009	-0.7712
3	0.0	-0.6861	-0.6861	-0.6125
4	0.0	-0.5217	-0.5217	0.0
5	0.0	0.0	0.0	0.0
6	81.201	93.697	-100.0	-0.7175
7	-0.7175	-100.0	8.6326	-0.7175
8	-0.6513	-100.0	-0.6125	-0.6338
9	-0.4095	0.8108	-0.4685	0.0
10	-100.0	-0.271	0.0	8.7082
11	62.372	95.933	-0.3439	-100.0
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	-0.271	10.790	-100.0	0.0
15	-0.271	-0.271	0.0	9.3469
16	43.866	-100.0	-0.3439	97.991
17	-100.0	2.0289	95.708	99.995
18	0.0	0.0	0.0	0.0
19	-0.1	-0.1	46.855	0.0
20	-0.271	0.0	0.0	-100.0
21	0.0	0.0	0.0	0.0
22	24.899	0.0	-100.0	-100.0
23	0.0	0.0	0.0	0.0
24	-0.1	0.0	0.0	0.0

0	**1**	2	3	4
–	**6**	7	8	9
10	**11**	–	–	14
15	**16**	**17**	**18**	19
20	–	22	–	24

2.2 井字棋

- 输入: 3 * 3矩阵, 黑白子
- 输出: 获胜判断(某方在同一行、同一列、同一斜向三种情况下有三子即为获胜)

CompetitionEnvironment建模

- 获胜方: winner
- 棋盘状态: $3*3$整数矩阵
- 棋局状态: 平局、白胜、黑胜、未完
- 当前玩家: currentPlayer
- 代码: environment/CompetitionEnvironment.java, environment/TicTacToe.java,

CompetitionQAgent建模

- 玩家编号(表示黑、白): player
- 对手(的引用): competitor
- 仅走一步: step()

Umpire建模

- 一个CompetitionEnvironment
- 两个QCompetitionAgent, 有相互的引用, 也有对环境的引用
- 玩家轮流出招, 直到棋局结束. 训练若干轮: train()

棋局例

- 训练10^6次的最后几局, 可以看出有一定学习效果
- 平: 511497, 白胜: 313343, 黑胜: 175160, 当前学习策略不好

Table: 落子位置为0到8, 结果0为平局, 1为白胜, 2为黑胜

落子过程	结果
3, 6, 7, 4, 2, 8, 0, 1, 5	0
3, 2, 5, 4, 6, 0, 1, 8	2
0, 3, 4, 8, 7, 1, 2, 6, 5	0
6, 5, 0, 3, 4, 2, 8	1
6, 1, 0, 3, 7, 8, 4, 2, 5	0
5, 1, 6, 3, 4, 2, 0, 8, 7	0
8, 7, 6, 1, 4, 0, 2	1

2.3 相同点分析

Environment

- 有限数量的状态: numStates
- 有限数量的行为: numActions
- 开始状态: startState
- 当前所处状态: currentState
- 状态迁移: transitionMatrix
- 状态下有的效行为: validActions
- 状态对应的奖励: getStateRewardValue()
- 对行为的反应: step(), 需要在子类中实现
- 代码: environment/Environment.java

Agent

- 对环境的引用: environment
- 有限数量的状态(同Environment): numStates
- 有限数量的行为(同Environment): numActions
- 学习: learn(), 需要在子类中实现
- 代码: agent/Agent.java

Analysis

- step()被子类覆盖, 使得Agent可以适用于不同的Environment, 正如迷宫问题与井字棋问题那样
- learn()被子类覆盖, 使得我们支持不同的学习方法
- 另外, QAgent.selectAction()为抽象方法, 也是为了支持不同的行为选择策略, 见SimpleRandomQAgent.java与WeightedRandomQAgent.java中代码

2.4 不同点分析

- 有竞争双方时, 每个Agent需要自己的Q矩阵
- 有竞争双方时, 每个Agent需要借助对方的Q矩阵来更新自己的, 且对方的奖励即己方的惩罚
- 有竞争双方时, 每个Agent只能控制当前步, 可使用step()但不能使用learn()
- 有竞争双方时, 需要一个Umpire控制整个的训练过程, 见umpire.Umpire.train()

2.5 讨论与作业

- Questions

9. 总结

- 9.1

其它

- properties
- Help