

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 24, 2020

目录

- 1. 引言
- 2. 简例
- 4. 结论

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 共性分析
- 2.4 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

└ 2. Simple examples

└ 2.1 Maze

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	<u>10.0</u>	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

└ 2. Simple examples

└ 2.1 Maze

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	<u>0.7190</u>	-100.0	0.0
15	-0.19	-0.19	0.0	-0.271
16	-0.19	-100.0	-0.19	<u>2.4280</u>
17	-100.0	-0.1	-0.1	<u>27.1</u>
18	0.0	0.0	0.0	0.0
19	-0.1	0.0	<u>19.0</u>	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

└ 2. Simple examples

└ 2.1 Maze

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	<u>85.232</u>
1	0.0	<u>91.246</u>	-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	<u>93.697</u>	-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	<u>95.933</u>	-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	<u>97.991</u>
17	-100.0	2.0289	95.708	<u>99.995</u>
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)



2.2.1 非对抗方案

Environment建模

- 棋盘状态: 各个格子为黑/白/空. 9个格子各有3种选择, 导致 3^9 种状态, 虽然很多不可达
- 见stateToCheckerboard()与checkerboardToState()方法
- 棋局状态: 黑/白胜, 平局, 未结束
- 代码: environment/TicTacToe.java

Agent建模

- 与迷宫游戏共用...amazing
- 如果预先为Q矩阵分配空间, 则为 $3^9 \times 9 = 3^{11}$ (因为下一步棋最多有9种选择), 绝大多数浪费掉
- 对于大些的棋盘, 不可以采用这种方式

2.2.2 对抗方案

■ 未写

2.3 Commonality analysis

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

9. 总结

■ 9.1

其它

- properties
- Help