

程序报告

学号：2112492

姓名：刘修铭

一、问题重述

机器人自动走迷宫：分别使用基础搜索算法和 Deep QLearning 算法，完成机器人自动走迷宫。

游戏规则为:从起点开始，通过错综复杂的迷宫，到达目标点(出口)。

在任一位置可执行动作包括：向上走 'u'、向右走 'r'、向下走 'd'、向左走 'l'。

- 执行不同的动作后，根据不同的情况会获得不同的奖励，具体而言，有以下几种情况。
- 撞墙
- 走到出口
- 其余情况
- 需要实现基于基础搜索算法和 Deep QLearning 算法的机器人，使机器人自动走到迷宫的出口。

对问题的理解

- 自己实现基础算法和强化学习算法。其中，基础算法主要是搜索算法，强化学习算法则更加侧重于根据智能体与环境的交互来学习。

二、设计思想

2.1 基础算法

- **广度优先搜索**
 1. 首先以机器人起始位置建立根节点，并入队；
 2. 接下来不断重复以下步骤直到判定条件：
 - 将队首节点的位置标记已访问；判断队首是否为目标位置(出口)，若是则终止循环并记录回溯路径；
 - 判断队首节点是否为叶子节点，若是则拓展该叶子节点
 - 如果队首节点有子节点，则将每个子节点插到队尾
 - 将队首节点出队
- **深度优先搜索**
 1. 首先将根节点放入stack中。
 2. 从stack中取出第一个节点，并检验它是否为目标。
如果找到目标，则结束搜寻并回传结果。
否则将它某一个尚未检验过的直接子节点加入stack中。
 3. 重复步骤2。
 4. 如果不存在未检测过的直接子节点。
将上一级节点加入stack中。
重复步骤2。
 5. 重复步骤4。
 6. 若stack为空，表示整张图都检查过了——亦即图中没有欲搜寻的目标。结束搜寻并回传“找不到目标”。

2.2 Deep QLearning算法

QLearning 算法要点:

- Q-Learning 算法是值迭代算法。
- Q-Learning 算法在执行过程中会计算每个“状态”或“状态-动作”的 Value，然后在执行动作的时候，会设法最大化这个值。算法（值迭代）的核心在于对每个状态值的准确估计。
- 考虑最大化动作的长期奖励——考虑当前动作带来的奖励 + 考虑动作长远的奖励。

QLearning 算法步骤:

1. 获取机器人所处迷宫位置

2. 对当前状态，检索Q表，如果不存在则添加进入Q表

3. 选择动作

- 为了防止出现 因机器人每次都选择它认为最优的路线而导致的路线固定（缺乏有效探索） 的现象，通常采用 *epsilon-greedy* 算法：
 - 在机器人选择动作的时候，以一部分的概率随机选择动作，以一部分的概率按照最优的 Q 值选择动作。
 - 同时，这个选择随机动作的概率应当随着训练的过程逐步减小。

4. 以给定的动作（移动方向）移动机器人

5. 获取机器人执行动作后所处的位置

6. 对当前 next_state，检索Q表，如果不存在则添加进入Q表

7. 更新 Q 表 中 Q 值以及其他参数

- Q表 (Q_table) :
 - Q-learning 算法将状态和动作构建成一张 Q_table 表来存储 Q 值;
 - Q-Learning 算法中，长期奖励记为 Q 值，其中会考虑每个“状态-动作”的 Q 值，计算公式为：

$$Q(s_t, a) = R_{t+1} + \gamma \times \max_a Q(a, s_{t+1})$$

- (St, a) : 当前的“状态-动作”
- Rt+1: 执行动作 a 后的环境奖励
- maxQ(a, St+1): 执行任意动作能够获得的最大的Q值
- γ: 折扣因子
- 然而，计算得到新的 Q 值之后，一般会使用更为保守地更新 Q 表的方法，即引入松弛变量 alpha，按如下的公式进行更新，使得 Q 表的迭代变化更为平缓。

$$Q(s_t, a) = (1 - \alpha) \times Q(s_t, a) + \alpha \times (R_{t+1} + \gamma \times \max_a Q(a, s_{t+1}))$$

三、代码内容

• 基础算法

```
1  # 导入相关包
2  import os
3  import random
4  import numpy as np
5  from Maze import Maze
6  from Runner import Runner
7  from QRobot import QRobot
8  from ReplayDataSet import ReplayDataSet
9  from torch_py.MinDQNRobot import MinDQNRobot as TorchRobot # PyTorch版本
10 from keras_py.MinDQNRobot import MinDQNRobot as KerasRobot # Keras版本
11 import matplotlib.pyplot as plt
```

```

12
13 # 机器人移动方向
14 move_map = {
15     'u': (-1, 0), # up
16     'r': (0, +1), # right
17     'd': (+1, 0), # down
18     'l': (0, -1), # left
19 }
20
21 # 迷宫路径搜索树
22 class SearchTree(object):
23
24     def __init__(self, loc=(), action='', parent=None):
25         """
26         初始化搜索树节点对象
27         :param loc: 新节点的机器人所处位置
28         :param action: 新节点的对应的移动方向
29         :param parent: 新节点的父节点
30         """
31
32         self.loc = loc # 当前节点位置
33         self.to_this_action = action # 到达当前节点的动作
34         self.parent = parent # 当前节点的父节点
35         self.children = [] # 当前节点的子节点
36
37     def add_child(self, child):
38         """
39         添加子节点
40         :param child: 待添加的子节点
41         """
42         self.children.append(child)
43
44     def is_leaf(self):
45         """
46         判断当前节点是否是叶子节点
47         """
48         return len(self.children) == 0
49
50     def expand(maze, is_visit_m, node):
51         """
52         拓展叶子节点，即为当前的叶子节点添加执行合法动作后到达的子节点
53         :param maze: 迷宫对象
54         :param is_visit_m: 记录迷宫每个位置是否访问的矩阵
55         :param node: 待拓展的叶子节点
56         """
57         can_move = maze.can_move_actions(node.loc)
58         for a in can_move:
59             new_loc = tuple(node.loc[i] + move_map[a][i] for i in range(2))
60             if not is_visit_m[new_loc]:
61                 child = SearchTree(loc=new_loc, action=a, parent=node)
62                 node.add_child(child)
63
64     def back_propagation(node):
65         """
66         回溯并记录节点路径
67         :param node: 待回溯节点
68         :return: 回溯路径
69         """
70         path = []
71         while node.parent is not None:
72             path.insert(0, node.to_this_action)
73             node = node.parent
74         return path
75
76     def my_search(maze):
77         """
78         对迷宫进行广度优先搜索
79         :param maze: 待搜索的maze对象

```

```

80     """
81     start = maze.sense_robot()
82     root = SearchTree(loc=start)
83     queue = [root] # 节点队列，用于层次遍历
84     h, w, _ = maze.maze_data.shape
85     is_visit_m = np.zeros((h, w), dtype=np.int) # 标记迷宫的各个位置是否被访问过
86     path = [] # 记录路径
87     while True:
88         current_node = queue[0]
89         is_visit_m[current_node.loc] = 1 # 标记当前节点位置已访问
90
91         if current_node.loc == maze.destination: # 到达目标点
92             path = back_propagation(current_node)
93             break
94
95         if current_node.is_leaf():
96             expand(maze, is_visit_m, current_node)
97
98         # 入队
99         for child in current_node.children:
100             queue.append(child)
101         # 出队
102         queue.pop(0)
103
104     return path

```

• Q-learning

```

1  import random
2  from QRobot import QRobot
3
4  class Robot(QRobot):
5
6      valid_action = ['u', 'r', 'd', 'l']
7
8      def __init__(self, maze, alpha=0.5, gamma=0.9, epsilon=0.5):
9          """
10             初始化 Robot 类
11             :param maze: 迷宫对象
12             """
13             self.maze = maze
14             self.state = None
15             self.action = None
16             self.alpha = alpha
17             self.gamma = gamma
18             self.epsilon = epsilon # 动作随机选择概率
19             self.q_table = {}
20
21             self.maze.reset_robot() # 重置机器人状态
22             self.state = self.maze.sense_robot() # state为机器人当前状态
23
24             if self.state not in self.q_table: # 如果当前状态不存在，则为 Q 表添加新列
25                 self.q_table[self.state] = {a: 0.0 for a in self.valid_action}
26
27     def train_update(self):
28         """
29             以训练状态选择动作，并更新相关参数
30             :return :action, reward 如: "u", -1
31             """
32             self.state = self.maze.sense_robot() # 获取机器人当初所处迷宫位置
33
34             # 检索Q表，如果当前状态不存在则添加进入Q表
35             if self.state not in self.q_table:
36                 self.q_table[self.state] = {a: 0.0 for a in self.valid_action}
37

```

```

38         action = random.choice(self.valid_action) if random.random() < self.epsilon else
max(self.q_table[self.state], key=self.q_table[self.state].get) # action为机器人选择的动作
39         reward = self.maze.move_robot(action) # 以给定的方向移动机器人, reward为迷宫返回的奖励值
40         next_state = self.maze.sense_robot() # 获取机器人执行指令后所处的位置
41
42         # 检索Q表, 如果当前的next_state不存在则添加进入Q表
43         if next_state not in self.q_table:
44             self.q_table[next_state] = {a: 0.0 for a in self.valid_action}
45
46         # 更新 Q 值表
47         current_r = self.q_table[self.state][action]
48         update_r = reward + self.gamma * float(max(self.q_table[next_state].values()))
49         self.q_table[self.state][action] = self.alpha * self.q_table[self.state][action] +(1 -
self.alpha) * (update_r - current_r)
50
51         self.epsilon *= 0.5 # 衰减随机选择动作的可能性
52
53         return action, reward
54
55     def test_update(self):
56         """
57         以测试状态选择动作, 并更新相关参数
58         :return :action, reward 如: "u", -1
59         """
60         self.state = self.maze.sense_robot() # 获取机器人现在所处迷宫位置
61
62         # 检索Q表, 如果当前状态不存在则添加进入Q表
63         if self.state not in self.q_table:
64             self.q_table[self.state] = {a: 0.0 for a in self.valid_action}
65
66         action = max(self.q_table[self.state],key=self.q_table[self.state].get) # 选择动作
67         reward = self.maze.move_robot(action) # 以给定的方向移动机器人
68
69         return action, reward

```

四、实验结果

平台测试结果

OCS-4.4.15

详情页 >

File Edit View Run Kernel Tabs Help

结果提交

测试结果

生成 py 文件

选择主文件中的相关 Cell, 系统会自动生成支持测试的 py 文件

重新生成

main.py

系统测试

系统会随机给出测试用例并显示结果, 你可以根据测试结果测试文件

重新测试

查看结果

提交结果

自测通过后, 你还有1次提交机会, 提交后系统会给出评分, 请耐心等待公布。

提交结果

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

Playing Atari with Deep Reinforcement Learning.pdf

QRobot.py

ReplayDataSet.py

Robot.py

Runner.py

main.ipynb

keras.py

torch.py

main.py

接口测试

接口测试通过。

用例测试

展示迷宫

测试点	状态	时长	结果
测试基础搜索算法	✓	1s	恭喜, 完成了迷宫
测试强化学习算法(初级)	✓	1s	恭喜, 完成了迷宫
测试强化学习算法(高级)	✓	1s	恭喜, 完成了迷宫
测试强化学习算法(中级)	✓	1s	恭喜, 完成了迷宫

提交结果

第六次实验课 - 强化学习 pd

边栏收起 按钮介绍 功能介绍 常见问题

main.py

GPU: 3.0/53 min CPU: 0.00% RAM: 0.05 MB / 4.0 GB (main.py)

OCS-4.4.15

详情页 >

File Edit View Run Kernel Tabs Help

结果提交 测试结果

Launch

生成 py 文件

选择主文件中的相关 Cell，系统会自动生成支持测试的 py 文件

重新生成 main.py

系统测试

系统会输出测试用例并显示结果，你可以根据测试结果测试文件

重新测试 查看结果

提交结果

自测通过后，你只有1次提交机会，提交后系统会给出评分，请等待分数公布。

提交结果

Robot.py

Runner.py

main.ipynb

第六次实验课 - 强化学习.pdf

keras.py

torch.py

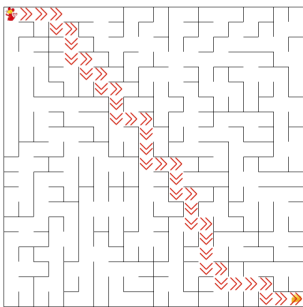
main.py

接口测试

接口测试通过。

用例测试

基础搜索算法 (Victory)




1 / 4

提交结果

重置文件 帮助

边栏收起 接口介绍 功能介绍 常见问题

deep Reir X 第六次实验课 - 强化学习.pdf X



GPU: 3 h 53 min CPU: 0.00 % RAM: 0.05 MB / 4.0 GB memory

OCS-4.4.15

详情页 >

File Edit View Run Kernel Tabs Help

结果提交 测试结果

Launch

生成 py 文件

选择主文件中的相关 Cell，系统会自动生成支持测试的 py 文件

重新生成 main.py

系统测试

系统会输出测试用例并显示结果，你可以根据测试结果测试文件

重新测试 查看结果

提交结果

自测通过后，你只有1次提交机会，提交后系统会给出评分，请等待分数公布。

提交结果

Robot.py

Runner.py

main.ipynb

第六次实验课 - 强化学习.pdf

keras.py

torch.py

main.py

接口测试

接口测试通过。

用例测试

强化学习level3 (Victory)




2 / 4

提交结果

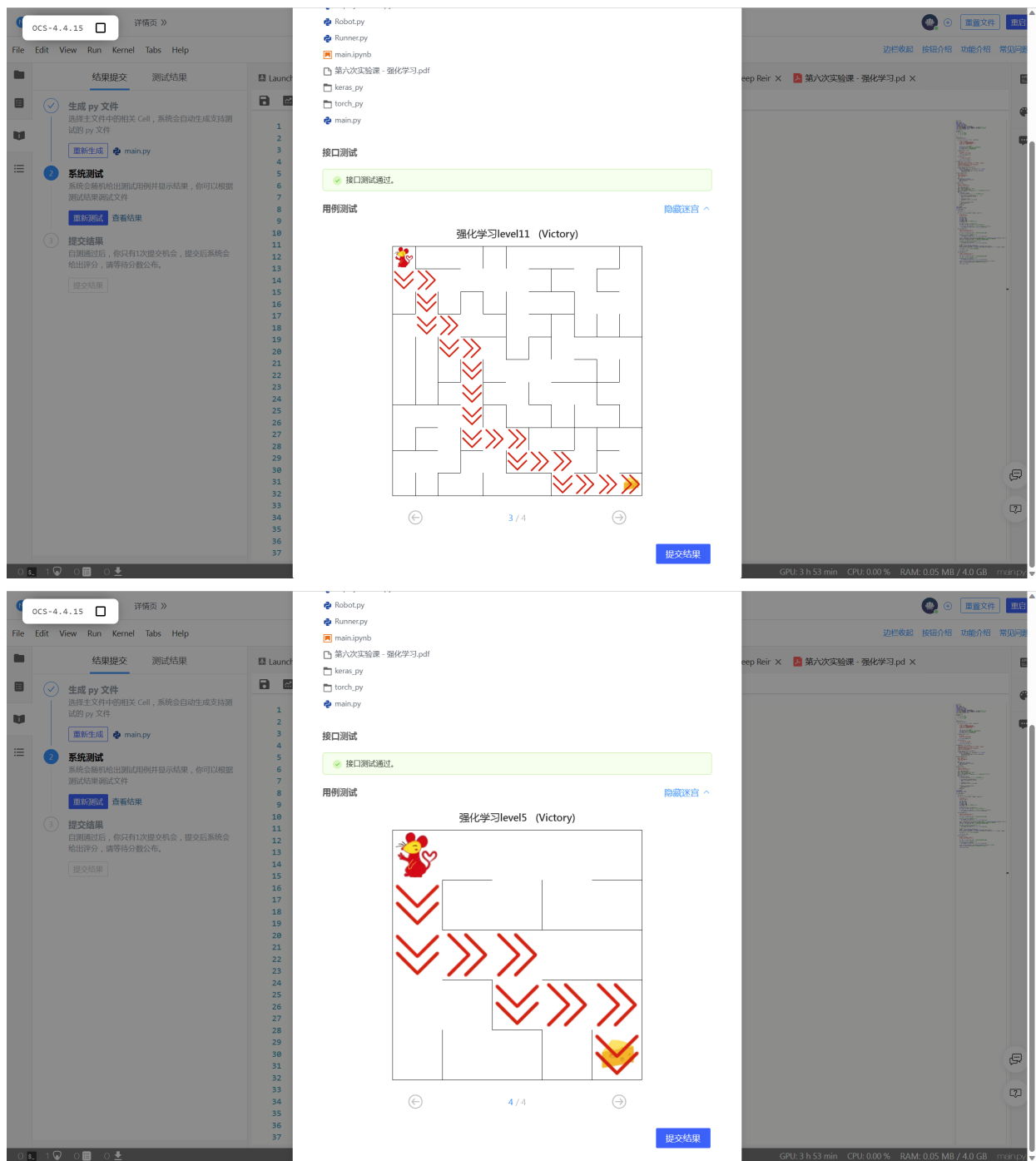
重置文件 帮助

边栏收起 接口介绍 功能介绍 常见问题

deep Reir X 第六次实验课 - 强化学习.pdf X



GPU: 3 h 53 min CPU: 0.00 % RAM: 0.05 MB / 4.0 GB memory



五、总结

1. 通过本次实验，加深了对强化学习的认识。
2. 学习了 Q-Learning 算法，对其代码框架有了更深的认识。