Q-learning机器学习实验——171250574 杨逸存

实验简介

- 本实验使用python语言以及numpy，pandas等库实现了Q-learning寻宝小游戏

算法/代码详解

ε-greedy Q-Learning算法伪代码

```
Q_Learning(Actions, ϵ, α, γ):
    Initialize Q_table arbitrarily
    For each episode:
        Initialize s: s = s0
        Repeat:
            Select action by ϵ-greedy policy: a* ← Choose_best_action(Actions, ϵ)
            Take action a* and observe s', r: s', r ← get_feedback(s, a*)
            Q(s, a) ← Q(s, a) + α[r + γ·MAXa' (Q(s', a')) - Q(s, a)]
            s ← s'
        Until s is terminal state
    return Q_table
```

游戏规则

在一维空间中，agent起点为0，宝藏位置为N（末尾），每次agent向左或右移动收益为0，但若达到终点则收益为1

```python
def get_env_feedback(s: int, a: str):
    # 向右
    if a == 'R':
        s_ = s + 1
        reward = 1 if s_ == N_STATES - 1 else 0
    # 向左
    else:
        if s == 0:
            s_ = s
        else:
            s_ = s - 1
        reward = 0
    return s_, reward
```

Q表设计

panda.Dataframe数据结构，index代表状态，columns代表行动（详见实验结果展示部分）

```python
def init_Q_table(n_state, actions):
    q_table = pd.DataFrame(
        data=np.zeros((n_state, len(actions))),
        columns = actions
    )
    return q_table
```

ε-greedy策略选择过程

即获取下一动作的过程。ε-greedy测量略模拟退火算法思想，在ε概率下查找Q表选择当前最优动作，在1 - ε概率下随机选择动作。

```python
def get_next_action(curr_state, Q_table):
    actions_list = Q_table.iloc[curr_state, : ]
    # 1-eps概率随机选择动作（模拟退火），或初始情况下随机选择动作
    if np.random.uniform() > EPS or (actions_list == 0).all():
        action = np.random.choice(ACTIONS)
    else:
        # 对多个最优值进行随机选择
        action = np.random.choice(actions_list[actions_list == np.max(actions_list)].index)
    return action
```

Q值更新（主过程）描述

见代码注释:

```python
def Q_learning():
    # 初始化Q表
    Q_table = init_Q_table(N_STATES, ACTIONS)
    for episode in range(MAX_EPISODES):
        step_counter = 0  # 步数计数
        s = 0  # 初始状态
        update_env(s, episode, step_counter)
        while s != N_STATES - 1:
            # 获取最有动作
            a = get_next_action(s, Q_table)
            # 获取下一状态和收益（环境反馈）
            s_, r = get_env_feedback(s, a)

            #Q值更新过程如下
            Q_predict = Q_table.loc[s, a]
            if s_ != N_STATES - 1:  # 如果没走到终点
                Q_actual = r + LAMBDA * Q_table.iloc[s_, :].max()
            else:  # 走到终点
                Q_actual = r
            # 更新Q表中Q(s, a)值
            # Q(s, a) ← Q(s, a) + α[r + γ·MAXa' (Q(s', a')) - Q(s, a)]
```

```
            # i.e. Q(s, a) += α[r + γ·MAXa' (Q(s', a')) - Q(s, a)]
            Q_table.loc[s, a] += ALPHA * (Q_actual - Q_predict)
            # 更新状态、步数、展示
            s = s_
            step_counter += 1
            update_env(s, episode, step_counter)

    return Q_table
```

实验结果分析

动图展示

见附件中的视频**mv_1d_1.mp4**

运行结果及Q表

- 该agent可大致在10轮探索中收敛。
- Q表中'R'一列的值均非负，说明其学习到向右走的期望收益是正的，即有希望获得宝藏。

```
E:\Anaconda3\envs\untitled\python.exe
Episode 1: total_steps = 50
Episode 2: total_steps = 24
Episode 3: total_steps = 19
Episode 4: total_steps = 55
Episode 5: total_steps = 51
Episode 6: total_steps = 12
Episode 7: total_steps = 10
Episode 8: total_steps = 7
Episode 9: total_steps = 7
Episode 10: total_steps = 7

Q-table:

          L         R
0  0.000000  0.000005
1  0.000000  0.000092
2  0.000000  0.001230
3  0.000000  0.010897
4  0.000000  0.062085
5  0.001873  0.237511
6  0.000000  0.651322
7  0.000000  0.000000
```

思维发散

在上述实验中，由于Q表中的值初始化为0，因此agent在探索初期是纯随机的试错，导致收敛很慢（50步完成）。因此，可对向左走这一动作施加很小的惩罚值（-0.01，合情合理）：

```
# 向左
else:
    if s == 0:
        s_ = s
    else:
        s_ = s - 1
    reward = -0.01
```

运行动画见视频**mv_1d_2.mp4**

运行结果如下，可以看到初期使用步数明显减少Q表中'L'列也出现了负值：

```
E:\Anaconda3\envs\untitled\python.exe F:/Desktop/study/Grade3-2/机
Episode 1: total_steps = 15
Episode 2: total_steps = 10
Episode 3: total_steps = 7
Episode 4: total_steps = 9
Episode 5: total_steps = 12
Episode 6: total_steps = 9
Episode 7: total_steps = 7
Episode 8: total_steps = 7
Episode 9: total_steps = 11
Episode 10: total_steps = 7

Q-table:

          L         R
0 -0.001900  0.000008
1 -0.002710  0.000121
2 -0.001000  0.001479
3 -0.001900  0.011685
4 -0.001000  0.062224
5 -0.000706  0.237511
6 -0.001000  0.651322
7  0.000000  0.000000
```

二维探索尝试

在一维的基础上改进代码，实现了二维的寻宝游戏。（代码详见**QLearning_2d.py**）

在二维游戏中：

- 宝藏处于右下方
- 采用元组表示agent的坐标（状态），地图为4×4大小
- 动作有上下左右四种
- 对向上和左给予-0.01的惩罚，对向下和右给予0的收益，对终点宝藏给予1的收益

运行动画见**mv_2d.mp4**，运行结果Q表如下：

```
Q-table:

                L         R         U         D
(0, 0) -0.01900  0.000000 -0.034361  0.000800
(0, 1)  0.00000  0.000000  0.000000  0.000000
(0, 2)  0.00000  0.000000  0.000000  0.000000
(0, 3)  0.00000  0.000000 -0.010000  0.000000
(1, 0) -0.01000  0.005845 -0.010000  0.000000
(1, 1) -0.01000  0.000000 -0.010000  0.031814
(1, 2)  0.00000  0.000000  0.000000  0.000000
(1, 3)  0.00000  0.000000 -0.019000  0.000000
(2, 0)  0.00000  0.000204  0.000000  0.000000
(2, 1) -0.01000  0.000000 -0.010000  0.128017
(2, 2) -0.01000  0.000000 -0.010000  0.017100
(2, 3)  0.00000  0.000000 -0.010000  0.100000
(3, 0) -0.01000  0.000000 -0.019000  0.000000
(3, 1)  0.00000  0.373834 -0.010000  0.006598
(3, 2) -0.01819  0.771232 -0.010000  0.000000
(3, 3)  0.00000  0.000000  0.000000  0.000000

Process finished with exit code 0
```

可以看到在(3, 1)、(3, 2)处向右的Q值较高，(2, 1)、(2,3)处向下的Q值也很高，因为还差一步就是宝藏了。