第二次作业（强化学习）

崔士强　PB22151743

2024 年 4 月 30 日

本次作业需独立完成，不允许任何形式的抄袭行为，如被发现会有相应惩罚。在上方修改你的姓名学号，说明你同意本规定。

问题 1：热身（10 分）

a. 计算（5 分）

s	-2	-1	0	1	2
$V^{(0)}(s)$	0	0	0	0	0
$V^{(1)}(s)$	0	7.5	-10	20	0
$V^{(2)}(s)$	0	2.5	5	16	0

b. 计算（5 分）

$$\pi(s) = arg\ max_a \sum_{s' \in \mathcal{S}} \gamma P_{sas'}[R(s,a,s') + \gamma V(s')]$$

可以得到三个 $(s, \mu(s))$ 数值对：$(-1, a_2), (0, a_1), (1, a_1)$

问题 2：Q-Learning（15 分）

a. 回答问题（2 分）

MDP 具有时间齐性. $v(s)$ 和 $q(s,a)$ 和状态转移概率有关，而状态转移概率不随时间变化而变化.

b. 计算（8 分）

更新结果如下：

1. $q(0, a_1) = 2$
2. $q(1, a_1) = 1$
3. $q(0, a_2) = -1$
4. $q(1, a_1) = 0$

c. 回答问题（5 分）

首先考虑 Q 函数与最大 Q 函数的差值. 由 Bellman 公式可以推导出，对一个状态-动作对，第 n+1 个 Q 函数最多是前一个的 γ 倍. 而在每一个遍历区间，Q 函数最大误差不超过 γ 倍的前一个最大误差，这样一来随着遍历区间数目的增长，最大误差趋近于零.

问题 3：Gobang Programming（55 分）

a. 回答问题（2 分）

由前面的分析过程，首先 Q^* 一定存在，因此经过足够多的迭代能够收敛. 再加上本例中状态空间并不大，应当可以在相对有限的轮数内收敛.

b. 代码填空（33 分）

```
class Gobang(UtilGobang):

    def get_next_state(self, action: Tuple[int, int, int], noise: Tuple[int, int, int])
        -> np.array:

        # BEGIN_YOUR_CODE (our solution is 3 line of code, but don't worry if you
            deviate from this)
        next_state = self.board.copy()
        black, x_black, y_black = action
        next_state[x_black][y_black] = black
        # END_YOUR_CODE

        if noise is not None:
            white, x_white, y_white = noise
            next_state[x_white][y_white] = white
        return next_state

    def sample_noise(self) -> Union[Tuple[int, int, int], None]:

        if self.action_space:
            # BEGIN_YOUR_CODE (our solution is 2 line of code, but don't worry if you
                deviate from this)
            (x, y) = random.choice(self.action_space)
            self.action_space.remove((x, y))
            # END_YOUR_CODE
            return 2, x, y
        else:
            return None

    def get_connection_and_reward(self, action: Tuple[int, int, int],
                                  noise: Tuple[int, int, int]) -> Tuple[int, int, int,
                                      int, float]:
```

```
        # BEGIN_YOUR_CODE (our solution is 4 line of code, but don't worry if you
            deviate from this)
        (black_1, white_1) = self.count_max_connections(self.board)
        (black_2, white_2) = self.count_max_connections(self.get_next_state(action,
            noise))
        reward = pow(black_2, 2)-pow(white_2, 2)-pow(black_1, 2)+pow(white_1, 2)
        # END_YOUR_CODE

        return black_1, white_1, black_2, white_2, reward

    def sample_action_and_noise(self, eps: float) -> Tuple[Tuple[int, int, int], Tuple[
        int, int, int]]:

        # BEGIN_YOUR_CODE (our solution is 8 line of code, but don't worry if you
            deviate from this)
        state_hashable = self.array_to_hashable(self.board)

        if(random.random() < eps or state_hashable not in self.Q.keys()):
            action_ = random.choice(self.action_space)
            action = (1, action_[0], action_[1])
        else:
            actions = [(1, action[0], action[1]) for action in self.action_space if (1,
                action[0], action[1]) in self.Q[state_hashable].keys()]
            if(not actions):
                action_ = random.choice(self.action_space)
                action = (1, action_[0], action_[1])
            else:
                action = max(actions, key=self.Q[state_hashable].get)
        self.action_space.remove((action[1], action[2]))
        # END_YOUR_CODE
        return action, self.sample_noise()

    def q_learning_update(self, s0_: np.array, action: Tuple[int, int, int], s1_: np.
        array, reward: float,
                          alpha_0: float = 1):

        s0, s1 = self.array_to_hashable(s0_), self.array_to_hashable(s1_)
        self.s_a_visited[(s0, action)] = 1 if (s0, action) not in self.s_a_visited else
            \
            self.s_a_visited[(s0, action)] + 1
        alpha = alpha_0 / self.s_a_visited[(s0, action)]

        # BEGIN_YOUR_CODE (our solution is 18 line of code, but don't worry if you
            deviate from this)
        available_actions_s1 = [(1, action_[0], action_[1]) for action_ in self.
            action_space]
        have_Q = False
```

```
        for action_ in available_actions_s1:
            if s1 in self.Q.keys() and action_ in self.Q[s1].keys():
                have_Q = True
        if not have_Q :
            value_at_s1 = 0
        else:
            max_Q = -float("inf")
            for action_ in available_actions_s1:
                if(action_ not in self.Q[s1].keys()):
                    self.Q[s1][action_] = 0
                if(self.Q[s1][action_]>max_Q):
                    max_Q = self.Q[s1][action_]
            value_at_s1 = max_Q

        if(s0 not in self.Q.keys()):
            self.Q[s0] = {}
            self.Q[s0][action] = 0
        elif(action not in self.Q[s0].keys()):
            self.Q[s0][action] = 0
        self.Q[s0][action] -= alpha*(self.Q[s0][action]-reward-self.gamma*value_at_s1)
        # END_YOUR_CODE
```

c. 结果复现（10 分）

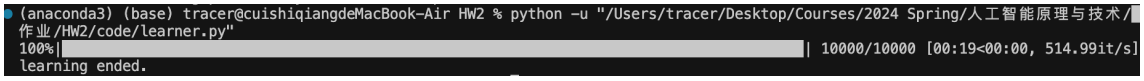

图 1: 训练结果

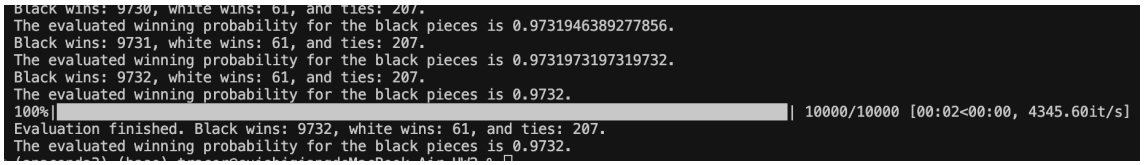

图 2: 评估结果

d. 回答问题（10 分）

图 3: $n = 4$ 的结果

该结果符合预期. 对于更大的状态和动作空间，遍历区间更长，收敛到 Q^* 更慢，所获取的 Q 函数离最优的距离更远，测试胜率更低.

问题 4：Deeper Understanding（10 分）

a. 回答问题（5 分）

$$\mathcal{T}_\mu = \max_a \left(r_{sa} + \gamma \cdot \sum_{s' \in \mathcal{S}} p_{sas'} \cdot v(s') \right)$$

b. 回答问题（5 分）

由于概率之和为 1，我们有

$$\sum_{s' \in \mathcal{S}} p_{sas'} \cdot v(s') \leq \max_{s' \in \mathcal{S}} v(s')$$

因此

$$|\mathcal{T}_{v_1}(s) - \mathcal{T}_{v_2}(s)| \leq \gamma \cdot \max_{s' \in \mathcal{S}} |v_1(s') - v_2(s')|$$

便可得到待证结论.

反馈（10 分）

- 本次作业相对第一次简单一些. 工作时间大约 10h. 体验比较好.
- 另外由于不清楚赋值特性，不了解应当使用`copy()`方法，导致 debug 时间很长.