第二次作业（强化学习）

高茂航　PB22061161

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）

i	$s=-2$	$s=-1$	$s=0$	$s=1$	$s=2$
0	0	0	0	0	0
1	0	7.5	-10	20	0
2	0	2.5	5	16	0

表 1: Value Iteration for $i \in \{0,1,2\}$

b. 计算（5 分）

$(-1, a_2), (0, a_1), (1, a_1)$

问题 2：Q-Learning（15 分）

a. 回答问题（2 分）

状态值函数 v(s) 表示在状态 s 下的长期回报的期望，而动作值函数 q(s, a) 表示在状态 s 下采取动作 a 后的长期回报的期望，它们的计算基于整个 MDP 过程，而不依赖于任何特定的时间步。

b. 计算（8 分）

$q^*(s,a) = (1-\alpha)q^*(s,a) + \alpha(r + \gamma q^*(s1, a1))$，
$t=1$ 时 $s=0$ 转移到 $s=1$，此时 $q^*(0,a1)=0, q^*(1,a)=0, \alpha=1$，$\therefore q^*(0,a1)=r_1=1$;
$t=2$ 时 $s=1$ 转移到 $s=0$，此时 $q^*(1,a1)=0, q^*(0,a)=1, \alpha=1$，$\therefore q^*(1,a1)=\alpha(r_2+q^*(0,a))=3$;
$t=3$ 时 $s=0$ 转移到 $s=1$，此时 $q^*(0,a2)=0, q^*(1,a)=3, \alpha=1$，$\therefore q^*(0,a2)=\alpha(r_3+q^*(1,a))=2$。

c. 回答问题（5 分）

在确定性环境中：

(1)Q-Learning 算法基于贝尔曼方程，通过不断地更新 Q 值 (在每个时间步都会选择当前最优的动作) 来逼近贝尔曼方程的解，这个解就是最优的 Q 值。

(2)Q-Learning 算法满足了所有状态-动作对的无限访问性条件，例如 $\epsilon - greedy$ 策略。

(3)Q-Learning 算法引入折扣因子 $\gamma \in [0, 1)$，在学习率 $\alpha \in (0, 1]$ 时保证 $\hat{Q}_n$ 在第 k 个遍历区间的误差小于 $(\alpha\gamma + (1 - \alpha))^k \Delta_0$。

问题 3：Gobang Programming（55 分）

a. 回答问题（2 分）

可行的，原因如下：

在 3×3 的棋盘上，每个格子有三种可能的状态：空、玩家 1 的棋子、玩家 2 的棋子。因此，总的状态空间大小是 $3^{(3*3)} = 19683$，这是一个相对较小的状态空间，我们可以在合理的时间内遍历所有的状态。

对于每个状态，可能的动作是在空的格子上放一个棋子，因此动作空间的大小最多是 9，这也是一个相对较小的动作空间。

由于状态空间和动作空间都相对较小，因此我们可以在合理的时间内计算出每个状态-动作对的 Q 值。

b. 代码填空（33 分）

```python
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)
        black, x_black, y_black = action
        next_state = copy.deepcopy(self.board)
        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)
        next_state = self.get_next_state(action, noise)
        black_2, white_2 = self.count_max_connections(next_state)
        reward = (black_2 ** 2 - white_2 ** 2) - (black_1 ** 2 - 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)
        if random.random() < eps or self.array_to_hashable(self.board) not in self.Q:
            x, y = random.choice(self.action_space)
        else:
            state = self.array_to_hashable(self.board)
            if self.Q[state]:
                _, x, y = max(self.Q[state], key=self.Q[state].get)
            else:
                x, y = random.choice(self.action_space)
        # 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)]
```

```python
# BEGIN_YOUR_CODE (our solution is 18 line of code, but don't worry if you
    deviate from this)
if s0 not in self.Q:
    self.Q[s0] = {}
if action not in self.Q[s0]:
    self.Q[s0][action] = 0
if s1 not in self.Q:
    self.Q[s1] = {}
Q_x1_a1 = max(self.Q[s1].values(), default=0) if max(self.Q[s1].values(),
    default=0) > 0 else 0
self.Q[s0][action] = (1 - alpha) * self.Q[s0][action] + alpha * (reward + self.
    gamma * Q_x1_a1)
# END_YOUR_CODE
```

c. 结果复现（10 分）

```
 96%|#########5| 9557/10000 [00:43<00:01, 261.56it/s]
 96%|#########5| 9584/10000 [00:43<00:01, 260.34it/s]
 96%|#########6| 9614/10000 [00:43<00:01, 262.46it/s]
 96%|#########6| 9644/10000 [00:44<00:01, 262.90it/s]
 97%|#########6| 9671/10000 [00:44<00:01, 263.39it/s]
 97%|#########7| 9700/10000 [00:44<00:01, 260.40it/s]
 97%|#########7| 9727/10000 [00:44<00:01, 262.62it/s]
 98%|#########7| 9754/10000 [00:44<00:00, 263.82it/s]
 98%|#########7| 9781/10000 [00:44<00:00, 264.56it/s]
 98%|#########8| 9809/10000 [00:44<00:00, 257.85it/s]
 98%|#########8| 9837/10000 [00:44<00:00, 262.47it/s]
 99%|#########8| 9866/10000 [00:44<00:00, 260.26it/s]
 99%|#########8| 9893/10000 [00:45<00:00, 261.41it/s]
 99%|#########9| 9922/10000 [00:45<00:00, 258.93it/s]
 99%|#########9| 9949/10000 [00:45<00:00, 261.95it/s]
100%|#########9| 9978/10000 [00:45<00:00, 267.00it/s]
100%|##########| 10000/10000 [00:45<00:00, 220.04it/s]
learning ended.
```

```
Black wins: 9656, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.
9660830415207604.
Black wins: 9657, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.
9660864345738295.
Black wins: 9658, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.
9660898269480844.
Black wins: 9659, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.
9660932186437288.
Black wins: 9660, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.
9660966096609661.
Black wins: 9661, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.9661.
Evaluation finished. Black wins: 9661, white wins: 63, and ties: 276.
The evaluated winning probability for the black pieces is 0.9661.

100%|##########| 10000/10000 [00:05<00:00, 1776.90it/s]
```

d. 回答问题（10 分）

```
 97%|#########7| 9720/10000 [01:15<00:01, 140.92it/s]
 97%|#########7| 9735/10000 [01:15<00:01, 138.38it/s]
 98%|#########7| 9753/10000 [01:15<00:01, 143.29it/s]
 98%|#########7| 9768/10000 [01:16<00:01, 138.86it/s]
 98%|#########7| 9784/10000 [01:16<00:01, 138.37it/s]
 98%|#########8| 9800/10000 [01:16<00:01, 138.20it/s]
 98%|#########8| 9817/10000 [01:16<00:01, 146.52it/s]
 98%|#########8| 9835/10000 [01:16<00:01, 149.01it/s]
 98%|#########8| 9850/10000 [01:16<00:01, 141.77it/s]
 99%|#########8| 9867/10000 [01:16<00:00, 143.38it/s]
 99%|#########8| 9882/10000 [01:16<00:00, 139.41it/s]
 99%|#########8| 9899/10000 [01:16<00:00, 146.20it/s]
 99%|#########9| 9914/10000 [01:17<00:00, 135.96it/s]
 99%|#########9| 9931/10000 [01:17<00:00, 144.54it/s]
 99%|#########9| 9946/10000 [01:17<00:00, 145.71it/s]
100%|#########9| 9964/10000 [01:17<00:00, 154.88it/s]
100%|#########9| 9980/10000 [01:17<00:00, 143.43it/s]
100%|#########9| 9996/10000 [01:17<00:00, 144.88it/s]
100%|##########| 10000/10000 [01:17<00:00, 128.76it/s]
learning ended.
```

```
The evaluated winning probability for the black pieces is 0.949359487590072.
Black wins: 9487, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.949364555188632.
Black wins: 9488, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493696217730638.
Black wins: 9489, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493746873436718.
Black wins: 9490, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493797519007603.
Black wins: 9491, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493848154446334.
Black wins: 9492, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493898779755952.
Black wins: 9493, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9493949394939494.
Black wins: 9494, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9494.

100%|##########| 10000/10000 [00:09<00:00, 1097.88it/s]
Evaluation finished. Black wins: 9494, white wins: 504, and ties: 2.
The evaluated winning probability for the black pieces is 0.9494.
```

最终的胜率达到 95% 左右，这个结果基本符合预期，因为通过合理地更新 Q 值使得 agent 能较好判断在每种状态下的最佳策略，但应该还可以继续微调以继续提高胜率。

问题 4：Deeper Understanding（10 分）

a. 回答问题（5 分）

确定性策略 μ 可以被表示为某个随机策略 π，其中当 $a=\mu(s)$ 时 $\pi(s,a)=1$ ，否则 $\pi(s,a)=0$，$\mathcal{T}_\mu$ 的定义如下：

$$(\mathcal{T}_\mu v)(s) = r_{s,\mu(s)} + \gamma \sum_{s' \in S} p_{s,\mu(s),s'} v(s')$$

b. 回答问题（5 分）

对 $\forall v1, v2 \in \mathbb{R}^{|\mathcal{S}|}$ ，我们有

$$
\begin{aligned}
\|\mathcal{T}v_1 - \mathcal{T}v_2\|_\infty &= \max_{s\in S} |(\mathcal{T}v_1)(s) - (\mathcal{T}v_2)(s)| \\
&= \max_{s\in S} \left| \max_{a\in A} \left\{ r_{s,a} + \gamma \sum_{s'\in S} p_{s,a,s'} v_1(s') \right\} - \max_{a\in A} \left\{ r_{s,a} + \gamma \sum_{s'\in S} p_{s,a,s'} v_2(s') \right\} \right| \\
&\le \max_{s\in S} \max_{a\in A} \left| \gamma \sum_{s'\in S} p_{s,a,s'} (v_1(s') - v_2(s')) \right| \\
&\le \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} p_{s,a,s'} |v_1(s') - v_2(s')| \\
&= \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} p_{s,a,s'} \|\mathcal{T}v_1 - \mathcal{T}v_2\|_\infty \\
&= \gamma \|v_1 - v_2\|_\infty
\end{aligned}
$$

所以得证。

反馈（10 分）

- 大概是两天时间，感觉比第一次的代码难度低一些，但在基础知识的学习理解上花了更多时间，特别是数学公式的推导感觉比较困难。