

第二次作业（强化学习）

彭煜峰 PB22051087

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）

考虑 Value Iteration:

$$V^{(i+1)}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p_{sas'} [\mathcal{R}(s, a, s') + \gamma V^{(i)}(s')]$$

带入数据得：

	-2	-1	0	1	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

b. 计算（5 分）

考虑最优策略为决定性策略：

$$\mu(s) = \arg \max_{a \in \mathcal{A}} Q_{opt}(s, a)$$

因此有：

$$\begin{cases} \mu(-1) = a_2 \\ (-1, a_2) \end{cases} \quad \begin{cases} \mu(0) = a_1 \\ (0, a_1) \end{cases} \quad \begin{cases} \mu(1) = a_1 \\ (0, a_1) \end{cases}$$

问题 2: Q-Learning (15 分)

a. 回答问题 (2 分)

$v(s)$ 是在给定状态 s 下的预期回报，它是从状态 s 开始的所有可能轨迹的期望值。根据马尔可夫性质，状态 s 的未来发展只取决于当前的状态 s ，而与到达状态 s 的过程历史无关； $q(s, a)$ 是在给定状态 s 和动作 a 下的预期回报，它也是从状态 s 和执行动作 a 开始的所有可能轨迹的期望值，根据马尔可夫性质，状态 s 和执行动作 a 后的未来发展只取决于当前的状态 s 和执行的动作 a ，而与到达状态 s 的过程历史无关。

b. 计算 (8 分)

采用 Monte Carlo Q-Learning 的 Q 值更新算法：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(\mathcal{R}(s, a) + \gamma \max_a Q(s', a) - Q(s, a) \right)$$

取 $\gamma = 1, \alpha = 1$ ：

$t = 1$ 时：

$$\hat{q}(0, a_1) = \hat{q}(0, a_1) + \alpha \left(\mathcal{R}(0, a_1) + \gamma \max_a \hat{q}(1, a) - \hat{q}(0, a_1) \right) = 0 + (1 + 0 - 0) = 1$$

$t = 2$ 时：

$$\hat{q}(1, a_1) = \hat{q}(1, a_1) + \alpha \left(\mathcal{R}(1, a_1) + \gamma \max_a \hat{q}(0, a) - \hat{q}(1, a_1) \right) = 0 + (2 + 1 - 0) = 3$$

$t = 3$ 时：

$$\hat{q}(0, a_2) = \hat{q}(0, a_2) + \alpha \left(\mathcal{R}(0, a_2) + \gamma \max_a \hat{q}(1, a) - \hat{q}(0, a_2) \right) = 0 + (-1 + 3 - 0) = 2$$

$t = 4$ 时：

$$\hat{q}(1, a_2) = \hat{q}(1, a_2) + \alpha \left(\mathcal{R}(1, a_2) + 0 - \hat{q}(1, a_2) \right) = 0$$

这里由于轨迹 τ 在 $t = 4$ 之后没有再观测到下一个状态，因此在计算 $\hat{q}(1, a_2)$ 时将 $\gamma \max_a \hat{q}(s', a)$ 设为 0。

c. 回答问题 (5 分)

首先对于更新步长 $\alpha = 1$ 的情况，设最优情况 Q^* 与迭代过程中的 \hat{Q}_n 之间有误差

$$\Delta_n = \max_{s, a} \left| \hat{Q}_n(s, a) - Q^*(s, a) \right|$$

可以证明 $\Delta_{n+1} \leq \gamma \Delta_n$ 。假设最多经过 m 次迭代就能够使得每一个状态动作对 (s, a) 被访问至少一次，将这样的 m 次迭代作为一个遍历区间。则根据前面的分析，在第 k 个遍历区间之后，系统的最大误差不超过 $\gamma^k \Delta_0$ 。又由于 $0 < \gamma \leq 1$ ，可以推出随着迭代次数增加， \hat{Q}_n 最终收敛于 Q^* 。

对于 $0 < \alpha \leq 1$ 的情况，可以由贝尔曼最优算子 \mathcal{T} 推出迭代误差：

$$\Delta_{n+1} \leq [\alpha\gamma + (1 - \alpha)] \Delta_n$$

同样的，考虑经过 k 个遍历区间后，系统最大误差不超过 $[\alpha\gamma + (1 - \alpha)]^k \Delta_0$ ，并由此得出算法收敛的结论。

问题 3: Gobang Programming (55 分)

a. 回答问题 (2 分)

对于 $n = 3$ 的情况，状态空间大小 $|S| = 3^9$ ，动作空间大小 $|\mathcal{A}| = 9$ ，因此可以得出 $|\{Q^*\}| = 177147$ ，对于这个数量级的计算，所需时间是比较长的，因此我认为不可行。

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)
        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)
    state = self.array_to_hashable(self.board)
    q_values = self.Q.get(state, {})
    if random.random() < eps or not q_values:
        act = random.choice(self.action_space)
        action = (1, act[0], act[1])
    else:
        action = max(q_values, key=q_values.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):

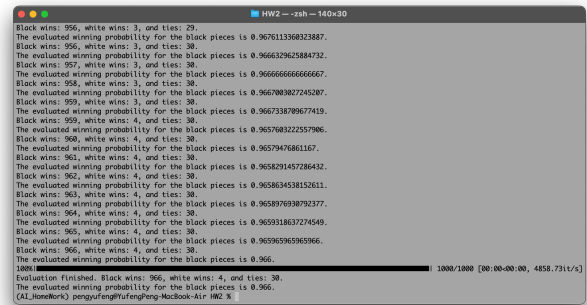
    s0, s1 = self.array_to_hashable(s0_), self.array_to_hashable(s1_)
    self.s_a_visited[(s0, action)] = 1 if (s0, action) not in self.s_a_visited else
    \
        self.s_a_visited[(s0, action)] + 1
    alpha = alpha_0 / self.s_a_visited[(s0, action)]

    # BEGIN_YOUR_CODE (our solution is 18 line of code, but don't worry if you
    deviate from this)
    q_value = self.Q.get(s0, {}).get(action, 0.0)
    new_q_value = q_value + alpha * (reward + self.gamma * max(self.Q.get(s1, {}).
        values(), default=0.0) - q_value)
    if s0 not in self.Q:
        self.Q[s0] = {}
    self.Q[s0][action] = new_q_value
    # END_YOUR_CODE
```

c. 结果复现 (10 分)



(a) 训练




(b) 评估

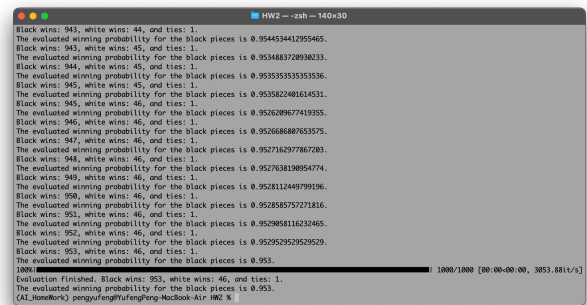
图 1: 复现结果

图 1 是训练过程和评估结果的截图。训练轮次为: 10000, 训练棋盘大小为 3×3 。可以看到训练得到模型的黑棋胜率为 96.6% 左右。

d. 回答问题 (10 分)



(a) 训练



(b) 评估

图 2: 复现结果

图 2 为将棋盘大小改为 4 后的训练结果。可以看到训练的时间有所增加, 模型胜率有所降低, 符合预期。训练时间增多是因为棋盘大小增大, 状态空间增大, 需要循环处理的次数增多, 胜率下降是因为状态空间变大, 计算的 Q 值与理论最优值的 Q^* 值之间的误差增大。

问题 4: Deeper Understanding (10 分)

a. 回答问题 (5 分)

对于确定性策略 μ , 其贝尔曼算子 T_μ 可以表示为:

$$(\mathcal{T}_\mu v)(s) = \mathbb{E}_{a \sim \mu(s)} \{r_{sa} + \gamma v(s')\} = r_{s\mu(s)} + \gamma v(s')$$

b. 回答问题 (5 分)

根据 \mathcal{T}_{v_1} 和 \mathcal{T}_{v_2} 的定义:

$$\begin{aligned}(\mathcal{T}_{v_1} v)(s) &= \max_{a \in A} \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_1(s') \right) \\(\mathcal{T}_{v_2} v)(s) &= \max_{a \in A} \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_2(s') \right)\end{aligned}$$

因此:

$$(\mathcal{T}_{v_1} - \mathcal{T}_{v_2})(v)(s) = \max_{a \in A} \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_1(s') \right) - \max_{a \in A} \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_2(s') \right)$$

估计 $\|\mathcal{T}_{v_1} - \mathcal{T}_{v_2}\|_\infty = \sup_{s \in \mathcal{S}} |(\mathcal{T}_{v_1} - \mathcal{T}_{v_2})(s)|$ 。注意到对于任意的 a :

$$\left| \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_1(s') \right) - \left(r_{sa} + \gamma \sum_{s' \in \mathcal{S}} p_{sas'} v_2(s') \right) \right| = \gamma \left| \sum_{s' \in \mathcal{S}} p_{sas'} (v_1(s') - v_2(s')) \right|$$

因此:

$$|(\mathcal{T}_{v_1} - \mathcal{T}_{v_2})(s)| \leq \gamma \max_{a \in A} \left| \sum_{s' \in \mathcal{S}} p_{sas'} (v_1(s') - v_2(s')) \right|$$

根据最大范数的性质, 有

$$\|\mathcal{T}_{v_1} - \mathcal{T}_{v_2}\|_\infty \leq \gamma \max_{s \in \mathcal{S}} \max_{a \in A} \left| \sum_{s' \in \mathcal{S}} p_{sas'} (v_1(s') - v_2(s')) \right| = \gamma \|v_1 - v_2\|_\infty$$

因此, \mathcal{T} 是一个压缩映射。

反馈 (10 分)

在每次实验报告的最后欢迎反馈你上这门课的感受, 你可以写下任何反馈, 包括但不限于以下几个方面: 课堂、作业难度和工作量、助教工作等等。

- 感觉 PPT 比较过于简略?
- 希望可以多给一些参考阅读资料!