
第一章 贝尔曼最优方程

1.1 设计思想

1. 网格环境定义

网络大小: $\text{NUM_STATES} = \text{GRID_ROW} * \text{GRID_COL}$

网格结构: 使用一维向量 `grid` 定义网格世界, 包含不同的状态和对应的奖励。

状态可以是普通状态、禁区、目标或边界。

奖励设置:

OTHERSTEP: 普通状态的奖励。

FORBIDDEN: 禁区的惩罚。

TARGET: 目标状态的奖励。

BORDER: 越界惩罚。

设置 `gamma`。

2. 动作定义

动作数 $\text{NUM_ACTION} = 5$ 。

定义五种可能的动作: RIGHT、DOWN、UP、LEFT、STAY。每个动作对应一个整数值。

3. 初始化策略 `policy`, 状态值 `V`

`policy`: 采用的策略, 定义为大小 $[\text{NUM_STATES}, \text{NUM_ACTION}]$ 的矩阵, 其中 `policy[s][a]` 表示在状态 `s` 时, 选择动作 `a` 的概率。

`V`: 状态值向量, 长度与状态个数相同, 初始化为 $v_i = 0$ 。

4. 贝尔曼方程求解

使用迭代法: 使用第 k 次迭代的 V_k , $Q(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v(s')$, 求得动作值 Q , 带入 `policy`, $v_{k+1}(s) = \sum_a \pi(a|s)q_k(s, a)$ 求得 V_{k+1} , 若 $\|V_{k+1} - V_k\| < \alpha$, 则取得结果, 并保留最优策略, 若不满足, 则更新 V_k , 继续迭代。

5. 贝尔曼最优方程求解

1) 值迭代: 循环每一个状态 `s`, 求 `s` 在每种动作 `a` 下的动作值 $Q(s, a) = R + \gamma PV_k$ 。取 Q 最大的动作, 更新 $V_{k+1} = \max Q_k(s, a)$, 若 $\|V_{k+1} - V_k\| < \alpha$, 则取得结果, 并保留最优策略, 若不满足, 则更新 V_k , 继续迭代。

2) 策略迭代: 对于第 k 步 V_k , 首先进行策略评估: 使用当前策略求解贝尔曼方程得到 V_{k+1} , 再进行策略改进, 循环每一个状态 `s`, 求 `s` 在每种动作 `a` 下的动作

值 $Q(s, a) = R + \gamma PV_k$ 。取 Q 最大的动作，更新策略 policy。若 $\|V_{k+1} - V_k\| < \alpha$ ，则取得结果，并保留最优策略，若不满足，则更新 V_k ，继续迭代。

3) 截断式策略迭代：在策略迭代基础上，修改策略评估阶段迭代次数为固定值 MAX_ITER，其他与策略迭代相同。

1.2 伪码描述

1. 值迭代

初始化：所有(s,a)概率模型已知。初始猜测 V_0

目标：求解贝尔曼最优方程，搜索最优状态值和最优策略

对于第 k+1 次迭代，while $\|V_{k+1} - V_k\| > \alpha$ ，do

 对于每个状态 s, do

 对于每个动作 a, do

 计算 q 值 $Q_k(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s')$

 取最大 q 值对应动作 $a_k^*(s) = \arg \max_a Q_k(s, a)$

 策略更新：如果 $a = a_k^*(s)$ ，则 $\pi_{k+1}(a|s) = 1$ ，否则 $\pi_{k+1}(a|s) = 0$

 值更新： $\pi_{k+1}(s) = \max_a Q_k(a, s)$

2. 策略迭代

初始化：所有(s,a)概率模型已知，初始猜测 π_0

目标：搜索最优状态值和最优策略

对于第 k 次迭代，while $\|V_{\pi_k} - V_{\pi_{k-1}}\| > \alpha$ ，do

 策略评估：

 初始化：任意初始猜测 $v_{\pi_k}^{(0)}$

 对于第 k 次迭代，当 $v_{\pi_k}^{(j)}$ 未收敛，即 $\|v_{\pi_k}^{(j)} - v_{\pi_k}^{(j-1)}\| > \alpha$ ，do

 对于每个状态 s, do

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) [\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}^{(j)}(s')]$$

 策略改进：

 对于每个状态 s, do

 对于每个动作 a, do

$$q_{\pi_k}(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s')$$

 取最大 q 值对应动作 $a_k^*(s) = \arg \max_a q_{\pi_k}(s, a)$

 如果 $a = a_k^*(s)$ ，则 $\pi_{k+1}(a|s) = 1$ ，否则 $\pi_{k+1}(a|s) = 0$

3. 截断式策略迭代

初始化：所有(s,a)概率模型已知，初始猜测 π_0

目标：搜索最优状态值和最优策略

对于第 k 次迭代，while $\|V_{\pi_k} - V_{\pi_{k-1}}\| > \alpha$ ，do

策略评估：

初始化：任意初始猜测 $v_k^{(0)} = v_{k-1}$ ，最大迭代次数设置为 $j_{truncate}$ 。

当 $j < j_{truncate}$ ，do

对于每个状态 s, do

$$v_k^{(j+1)}(s) = \sum_a \pi_k(a|s) [\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k^{(j)}(s')]$$

设置 $v_k = v_k^{(j_{truncate})}$

策略改进：

对于每个状态 s, do

对于每个动作 a, do

$$q_k(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s')$$

取最大 q 值对应动作 $a_k^*(s) = \arg \max_a q_k(s, a)$

如果 $a = a_k^*(s)$ ，则 $\pi_{k+1}(a|s) = 1$ ，否则 $\pi_{k+1}(a|s) = 0$

1.3 时间复杂度分析

设初始网格世界为 $n \times m$ ，令 $N = n \times m$ ，每个状态动作值个数 A

1) 值迭代

迭代部分：求 Q 时间复杂度 $O(N * A)$ ，更新 V 时间复杂度 $O(N * A)$ ，迭代次数 K，相加有 $O(K * A * N) + O(K * N * A)$ ，综合时间复杂度 $O(K * A * N)$ 。

2) 策略迭代

策略评估部分：设迭代次数 J，时间复杂度 $O(J * N * A)$ ；策略改进部分：时间复杂度 $O(N * A)$ ，总迭代次数为 K，时间复杂度 $O(K * J * N * A) + O(K * N * A)$ ，综合时间复杂度 $O(K * J * N * A)$ 。

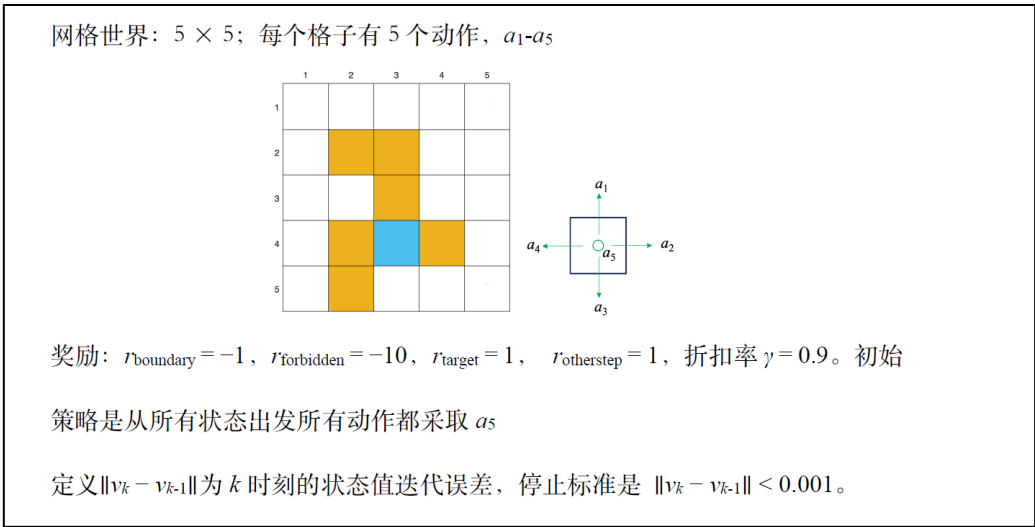
3) 截断式策略迭代

时间复杂度分析与策略迭代相同，在策略评估部分迭代次数 J 已知，综合时间复杂度 $O(K * J * N * A)$ 。

1.4 结果分析

动作方向以箭头表示，保持原地不动表示为“S”。

环境设置如下：

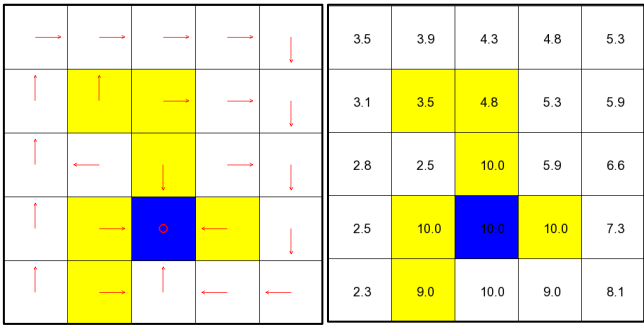


1. 比较三种算法的收敛速度

设置参数如上图，分别使用值迭代、策略迭代与截断式策略迭代方法求解贝尔曼最优方程，记录 TARGET 状态（图中蓝色方框）随迭代次数增加的 v 值变化。

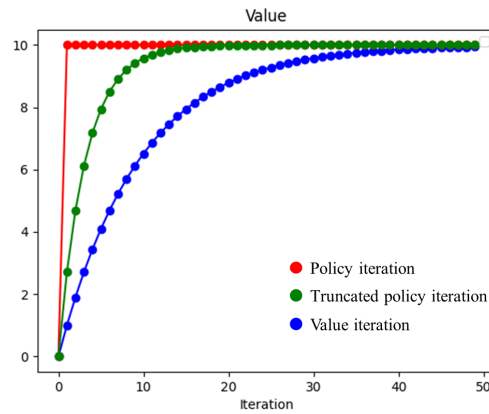
首先，三种迭代方法得到最优策略与 V 值如下(S 表示保持不动)：

Value Iteration	Policy Iteration	Truncated Policy Iteration
Optimal State Values (V^*):	Optimal State Values (V^*):	Optimal State Values (V^*):
3.5 3.9 4.3 4.8 5.3	3.5 3.9 4.3 4.8 5.3	3.5 3.9 4.3 4.8 5.3
3.1 3.5 4.8 5.3 5.9	3.1 3.5 4.8 5.3 5.9	3.1 3.5 4.8 5.3 5.9
2.8 2.5 10.0 5.9 6.6	2.8 2.5 10.0 5.9 6.6	2.8 2.5 10.0 5.9 6.6
2.5 10.0 10.0 10.0 7.3	2.5 10.0 10.0 10.0 7.3	2.5 10.0 10.0 10.0 7.3
2.3 9.0 10.0 9.0 8.1	2.3 9.0 10.0 9.0 8.1	2.3 9.0 10.0 9.0 8.1
Optimal Policy (Actions):	Optimal Policy (Actions):	Optimal Policy (Actions):



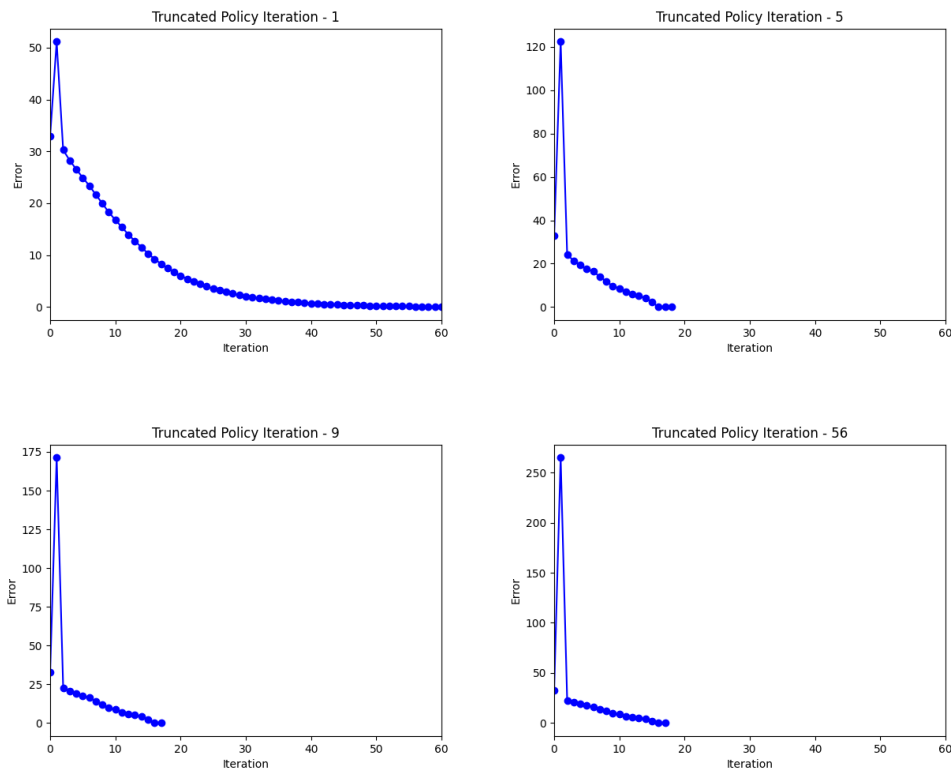
从图中可以看出三种迭代方式均能得到最优策略，且最优策略与最优状态值相同。

绘制不同迭代方式下迭代速度，横坐标表示迭代次数，纵坐标表示 $V[17]$ (TARGET 状态) 的状态值：



从图中结果可以看出，策略迭代下，状态值 v 到达最优状态值 v^* 收敛速度最快，值迭代收敛速度最慢，截断式策略迭代收敛速度位于二者中间，与设计逻辑相符。

2. 对于截断式策略迭代-x 算法，给出 $x=1, 5, 9, 56$ ，描述观测到的实验结果，并绘制结果图，其中横轴表示迭代次数，纵轴表示状态值误差



从图中结果可以看出，策略评估时选择不同的迭代次数 x ，值估计收敛速度不同： x 的值越大，值估计收敛得越快，如 $x=1$ 与 $x=5$ ，收敛速度相差较大， $x=1$ 时，迭代 60 次仍未达到阈值，当 $x=5$ 时，迭代次数小于 20 次则达到阈值；然而，当 x 很大时，增加 x 的好处很快就会下降，如 $x=9$ 与 $x=56$ ，收敛速度相差较小，均在

迭代次数 18 左右达到阈值。

第二章 MC ϵ -贪心策略算法

2.1 设计思想

1. 网格环境定义

网络大小: $\text{NUM_STATES} = \text{GRID_ROW} * \text{GRID_COL}$

网络结构: 使用一维向量 `grid` 定义网格世界, 包含不同的状态和对应的奖励。
状态可以是普通状态、禁区、目标或边界。

奖励设置:

OTHERSTEP: 普通状态的奖励。

FORBIDDEN: 禁区的惩罚。

TARGET: 目标状态的奖励。

BORDER: 越界惩罚。

设置 γ 。

设置 EPSILON。

2. 动作定义

动作数 $\text{NUM_ACTION} = 5$ 。

定义五种可能的动作: RIGHT、DOWN、UP、LEFT、STAY。每个动作对应一个整数值。

3. 初始化策略 `policy`, 状态值 `V`

`policy`: 采用的策略, 定义为大小 $[\text{NUM_STATES}, \text{NUM_ACTION}]$ 的矩阵, 其中 `policy[s][a]` 表示在状态 `s` 时, 选择动作 `a` 的概率, 由 `epsilon` 决定。

`V`: 状态值向量, 长度与状态个数相同, 初始化为 $v_i = 0$ 。

4. 蒙特卡洛 ϵ -贪心策略

初始化动作值 $Q[\text{NUM_STATES}, \text{NUM_ACTION}]$; 回报 $R[\text{NUM_STATES}, \text{NUM_ACTION}]$: 用于记录全部 episodes 状态 `s` 下动作 `a` 的总回报;
 $R_count[\text{NUM_STATES}, \text{NUM_ACTION}]$: 记录全部 episodes 到达 `s` 状态 `a` 动作的次数。

初始化策略为状态选择每个动作概率相同。

当迭代次数少于设定次数:

根据初始 `policy` 每个状态动作的概率 `policy[s][a]`, 连续生成一条长度为 `T` 的

episode。

从 $T-1$ 时刻开始，倒序记录每个 (s,a) 对的 reward，加入对应的 $R[s][a]$ ，并且 $R_count[s][a]+1$ ， $Q[s][a] = R[s][a] / R_count[s][a]$ 。

遍历完整条 episode 后，选择每个状态 s 下，最大的 $Q[s][a]$ 更新策略，设置 $policy[s][best_action] = 1.0 - EPSILON + EPSILON / NUM_ACTIONS$ ，该状态下其他 $policy[s][a] = EPSILON / NUM_ACTIONS$ 。

利用新的 policy 重复上述步骤，直到迭代结束。

2.2 伪码描述

初始化：初始猜测 π_0 和 $\epsilon \in [0,1]$ 的值，对于所有的 (s, a) , $Returns(s, a)=0$ 。

目的：寻找最优策略。

对于每个 episode，do

episode 生成：随机选择一个起始“状态-动作对” (s_0, a_0) 并确保所有对都可能被选择。按照当前策略，生成一个长度为 T 的 episode: $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$ 。

策略评估和策略改进：

初始化： $g \leftarrow 0$

对于 episode 的每步， $t = T-1, T-2, \dots, 0$, do

$g \leftarrow \gamma g + r_{t+1}$

使用每次访问方法：

$Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$

策略评估: $q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$

策略改进: 令 $a^* = \arg \max_a q(s_t, a)$ 并且

$$\pi(a|s_t) = \begin{cases} 1 - \frac{|A(s_t)|-1}{|A(s_t)|} \epsilon & \text{if } a = a^* \\ \frac{\epsilon}{|A(s_t)|} & \text{if } a \neq a^* \end{cases}$$

2.3 时间复杂度分析

令总状态数 N ，动作数 A 。

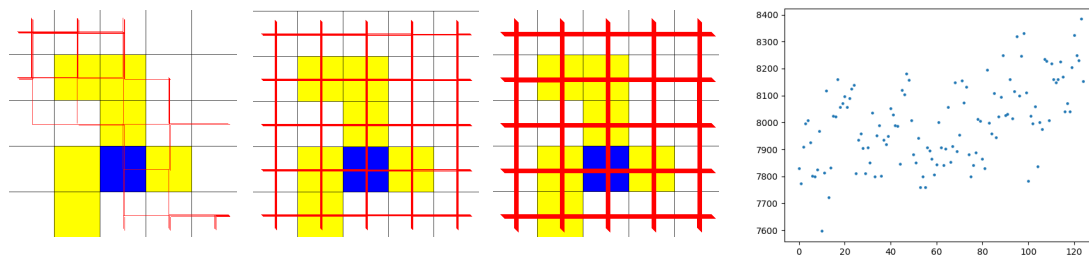
单条 episode 生成: $O(T)$ ，策略评估 $O(T)$ ，策略改进 $O(N * A)$ ，共有 K 条 episode，时间复杂度 $O(K * T) + O(K * T) + O(K * N * A)$ ，一般设置 $T > N * A$ ，综合时间复杂度 $O(K * T)$ 。

2.4 结果分析

环境参照第一章。

1. 分析 $\varepsilon = 1$ 、 $\varepsilon = 0.5$ 时，单个 episode 可以访问的“状态-动作对”情况：
episode 的长度分别为 100 步、1000 步、10000 步、100 万步的情况。

- (1) $\varepsilon = 1$ ，选择 state = 0 为起点，每个动作概率相同



(a) 100 步

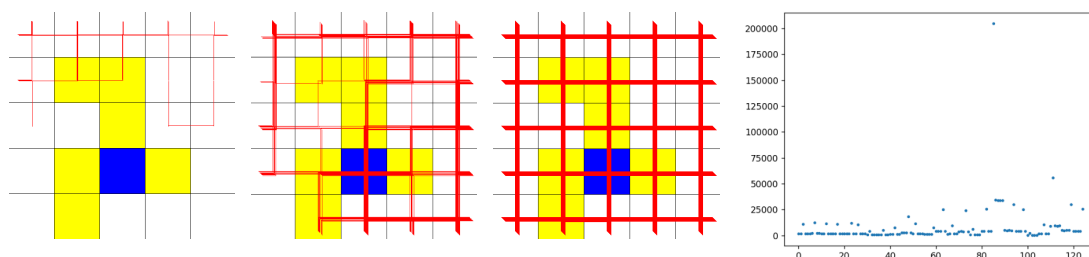
(b) 1000 步

(c) 10000 步

(d) 100 万步

当模拟 100 步时，一部分状态与动作未能访问到；模拟 1000 步时，状态与动作均访问到；模拟 10000 步时，每个状态与动作访问次数增加；统计模拟 100 万步情况，对于 125 个状态动作对，访问次数均超过 7600 次，低于 8400 次，访问次数较为均衡，符合 $\varepsilon = 1$ 时，选择各个动作概率相同要求。

- (2) $\varepsilon = 0.5$ ，选择 state = 0 为起点，最优动作概率大于其他动作



(a) 100 步

(b) 1000 步

(c) 10000 步

(d) 100 万步

当模拟 100 步时，只有少部分状态与动作被访问到；当模拟 1000 步时，仍然存在部分状态动作对没有访问，与(1)中模拟 1000 步存在差异；模拟 10000 步时，全部状态动作对被访问；模拟 100 万步，少数状态动作对访问次数极多，超过 200000，部分访问次数接近 25000，大部分状态动作对访问次数极少。

2. 分析 $\varepsilon = 0$ 、 $\varepsilon = 0.1$ 、 $\varepsilon = 0.2$ 、 $\varepsilon = 0.5$ 时，最优的 ε 贪心策略及其状态值。

- (1) $\varepsilon = 0$

					3.5	3.9	4.3	4.8	5.3
					3.1	3.5	4.8	5.3	5.9
					2.8	2.5	10.0	5.9	6.6
					2.5	10.0	10.0	10.0	7.3
					2.3	9.0	10.0	9.0	8.1

当 $\varepsilon = 0$ 时，策略为贪心策略，策略与状态值如图。

(2) $\varepsilon = 0.1$

					0.4	0.5	0.9	1.3	1.4
					0.1	0.0	0.5	1.3	1.7
					0.1	-0.4	3.4	1.4	1.9
					-0.1	3.4	3.3	3.7	2.2
					-0.3	2.8	3.7	3.1	2.7

当 $\varepsilon = 0.1$ 时，最优策略与贪心策略一致，但最优状态值与贪心策略差异较大， ε 策略的最优状态值整体小于贪心策略。并且，在贪心策略中，TARGET 的状态值最大（为 10），而 ε 策略 TARGET 的状态值（3.3）小于四周网格状态值（3.7），主要由于周围存在禁区，而 TARGET 状态存在 $\varepsilon/||A||$ 的概率进入禁区，获得负的收益。


























(3) $\varepsilon = 0.2$

					-1.1	-1.5	-1.1	-0.6	-0.6
					-1.5	-2.2	-2.3	-1.0	-0.6
					-1.1	-2.4	-2.2	-1.5	-0.6
					-1.6	-2.2	-2.6	-1.4	-1.1
					-2.0	-2.5	-1.8	-1.4	-1.0

当 $\varepsilon = 0.2$ 时，最优策略与贪心策略不一致，且 TARGET 下状态值最小。由于 ε 增大，选择非最优动作的可能性增加，以 TARGET 状态为例，选择 UP、LEFT、

RIGHT 策略均会走到禁区而获得负的收益， ϵ 增加使其收益为负的概率增大，因此 TARGET 周围禁区最多，状态值最低。同时由于不确定性增大，负收益使其他动作判断存在偏差， ϵ 最优策略与贪心策略不一致。

(4) $\epsilon = 0.5$

					-4.3	-5.5	-4.5	-2.6	-2.3
					-5.6	-7.7	-7.7	-4.1	-2.4
					-5.4	-8.9	-8.0	-5.6	-2.8
					-6.7	-8.7	-9.3	-5.4	-4.2
					-7.7	-8.7	-6.5	-5.1	-3.7

当 $\epsilon = 0.5$ 时，最优策略与贪心策略差异巨大，并且全部状态下状态值均为负，分析原因与 $\epsilon = 0.2$ 相同。 ϵ 增大，虽然探索性增大，但最优性受损，难以获得与贪心策略一致的最优策略，蒙特卡洛方法失效。因此，在选择 ϵ 大小时，需要平衡探索性与最优性，选择较小的 ϵ 。