

中山大學

本科实验报告

课程名称:	人工智能
实验名称:	值迭代与策略迭代
专业名称:	保密管理
学生姓名:	武自厚
学生学号:	20336014
实验地点:	东校园实验楼 D502
实验成绩:	
报告时间:	2022 年 6 月 14 日

一、 实验要求

基于 gym 库中的 Frozen Lake 环境在 `vi_and_pi.py` 中实现策略迭代和值迭代算法，并输出算法收敛后的路径。

- 实现 `vi_and_pi.py` 中的策略评估、策略提升、策略迭代函数。
- 实现 `vi_and_pi.py` 中的值迭代函数。
- 需要提交一份简要报告 + 代码。

二、 实验过程

1. 强化学习原理

强化学习, 是机器学习的范式和方法论之一. 用于描述和解决智能体 (agent) 在与环境的交互过程中通过学习策略以达成回报最大化或实现特定目标的问题. 强化学习的常见模型是标准的 Markov 决策过程. Markov 决策过程可以用六元组 $(S, A, R, T, P_0, \gamma)$ 描述, 其中

- S 表示状态空间.
- A 表示行为空间.
- $R = R(s, a)$ 表示在特定状态下进行特定行为的奖励值函数.
- $T: S \times A \times S \mapsto [0, 1]$ 表示状态变化函数.
- P_0 表示初始状态的分布.
- γ 表示折扣因子.
- 目标是找到一个能使奖励值期望最大化的策略.

2. 策略迭代原理

从一个初始化的策略出发, 先进行策略评估, 然后改进策略, 评估改进的策略, 再进一步改进策略. 经过不断迭代更新, 直达策略收敛, 这种算法被称为 “策略迭代”.

此算法可以用伪代码表示:

Algorithm 1: 策略评估算法 `evaluate()`**输入:** Markov 六元组 $(S, A, R, T, P_0, \gamma)$, 临界误差 θ **输出:** 评估价值映射 V

```

1 从  $P_0$  中获取价值映射  $V$ 
2 do
3    $\Delta := 0$ 
4   for  $s \in S$  do
5      $v := V(s)$ 
6      $V(s) := R(s, \pi(s)) + \gamma \sum_{s'} T(s, a, s') V(s')$ 
7      $\Delta := \max\{\Delta, |v - V(s)|\}$ 
8 until  $\Delta < \theta$ ;
9 return  $V$ 

```

Algorithm 2: 策略改进算法 `improve()`**输入:** Markov 六元组 $(S, A, R, T, P_0, \gamma)$, 现有策略 π , 已有价值 V **输出:** 策略迭代结果 π'

```

1 for  $s \in S$  do
2    $\pi(s) := \arg \max_{a \in A} \{R(s, a) + \gamma \sum_{s'} T(s, a, s') V(s')\}$ 
3 return  $\pi$ 

```

Algorithm 3: 策略迭代算法**输入:** Markov 六元组 $M = (S, A, R, T, P_0, \gamma)$, 临界误差 θ **输出:** 价值评估结果 V , 策略迭代结果 π

```

1 随机初始化策略  $\pi$ 
2 do
3    $V := \text{evaluate}(M, \theta)$ 
4    $\pi := \text{improve}(M, \pi, V)$ 
5 until  $\pi$  收敛;
6 return  $V, \pi$ 

```

3. 价值迭代原理

对每一个当前状态 s , 对每个可能的动作 a 都计算一下采取这个动作后到达的下一个状态的期望价值. 看看哪个动作可以到达的状态的期望价值函数最大, 就将这个最大的期望价值函数作为当前状态的价值函数 $V(s)$, 循环执行这个步骤, 直到价值函数收敛.

此算法可以用伪代码表示:

Algorithm 4: 价值迭代算法**输入:** Markov 六元组 $M = (S, A, R, T, P_0, \gamma)$, 临界误差 θ **输出:** 价值评估结果 V , 策略迭代结果 π

```

1  随机初始化策略  $\pi$ , 零初始化价值函数  $V$ 
2  do
3       $V' := V$ 
4       $\Delta := 0$ 
5      for  $s \in S$  do
6           $Q_\pi(a) := R(s, a) + \gamma \sum_{s'} T(s, a, s') V(s')$ 
7           $V(s) := \max_{a \in A} Q_\pi(a)$ 
8           $\pi(s) := \arg \max_{a \in A} Q_\pi(a)$ 
9           $\Delta := \max\{\Delta, |V(s) - V'(s)|\}$ 
10 until  $\Delta < \theta$ ;
11 return  $V, \pi$ 

```

4. 代码实现

```

43 def policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):
44     """Evaluate the value function from a given policy.
45
46     Parameters
47     -----
48     P, nS, nA, gamma:
49         defined at beginning of file
50     policy: np.array[nS]
51         The policy to evaluate. Maps states to actions.
52     tol: float
53         Terminate policy evaluation when
54         max |value_function(s) - prev_value_function(s)| < tol
55     Returns
56     -----
57     value_function: np.ndarray[nS]
58         The value function of the given policy, where value_function[s] is
59         the value of state s
60     """
61
62     value_function = np.zeros(nS)
63     i = 0
64     while True:
65         prev_value_function = np.copy(value_function)
66         for state in range(nS):
67             action = policy[state]

```

```
68         value_function[state] = sum(
69             [prob * (reward + gamma * prev_value_function[next_state])
70              for prob, next_state, reward, done in P[state][action]]
71         )
72     if np.max(np.fabs(value_function - prev_value_function)) < tol:
73         break
74     else:
75         i += 1
76
77     return value_function
78
79
80 def policy_improvement(P, nS, nA, value_from_policy, policy, gamma=0.9):
81     """Given the value function from policy improve the policy.
82
83     Parameters
84     -----
85     P, nS, nA, gamma:
86         defined at beginning of file
87     value_from_policy: np.ndarray
88         The value calculated from the policy
89     policy: np.array
90         The previous policy.
91
92     Returns
93     -----
94     new_policy: np.ndarray[nS]
95         An array of integers. Each integer is the optimal action to take
96         in that state according to the environment dynamics and the
97         given value function.
98     """
99
100     new_policy = np.zeros(nS, dtype='int')
101
102     for state in range(nS):
103         q_value = np.zeros(nA)
104         for action in range(nA):
105             for next_sr in P[state][action]:
106                 prob, next_state, reward, done = next_sr
107                 q_value[action] += prob * (reward + gamma *
108                     ↪ value_from_policy[next_state])
109         new_policy[state] = np.argmax(q_value)
```

```
110     return new_policy
111
112
113 def policy_iteration(P, nS, nA, gamma=0.9, tol=10e-3):
114     """Runs policy iteration.
115
116     You should call the policy_evaluation() and policy_improvement() methods to
117     implement this method.
118
119     Parameters
120     -----
121     P, nS, nA, gamma:
122         defined at beginning of file
123     tol: float
124         tol parameter used in policy_evaluation()
125     Returns:
126     -----
127     value_function: np.ndarray[nS]
128     policy: np.ndarray[nS]
129     """
130
131     policy = np.zeros(nS, dtype=int)
132     i = 0
133     while True:
134         value_function = policy_evaluation(P, nS, nA, policy)
135         next_policy = policy_improvement(P, nS, nA, value_function, policy)
136         if np.all(policy == next_policy):
137             print(f'Policy Iteration converged as step {i}')
138             break
139         else:
140             policy = next_policy
141             i += 1
142     return value_function, policy
143
144
145 def value_iteration(P, nS, nA, gamma=0.9, tol=1e-3):
146     """
147     Learn value function and policy by using value iteration method for a given
148     gamma and environment.
149
150     Parameters:
151     -----
152     `P`, `nS`, `nA`, `gamma`:
```

```
153         defined at beginning of file
154     tol: float
155         Terminate value iteration when
156         max |value_function(s) - prev_value_function(s)| < tol
157     Returns:
158     -----
159     value_function: np.ndarray[nS]
160     policy: np.ndarray[nS]
161     """
162
163     value_function = np.zeros(nS)
164     policy = np.zeros(nS, dtype=int)
165     i = 0
166     while True:
167         prev_value_function = value_function.copy()
168
169         for state in range(nS):
170             q_values = np.zeros(nA)
171             for action in range(nA):
172                 for next_sr in P[state][action]:
173                     prob, next_state, reward, done = next_sr
174                     q_values[action] += prob * (reward + gamma *
175                     ↪ prev_value_function[next_state])
176                 value_function[state] = np.max(q_values)
177                 policy[state] = np.argmax(q_values)
178
179         i += 1
180
181         if np.max(np.fabs(value_function - prev_value_function)) < tol:
182             print(f"Convergence at iteration {i}")
183             break
184
185     return value_function, policy
```

三、 实验结果

在 Frozen Lake 问题应用价值迭代和策略迭代得到结果:

