

强化学习基础篇（十三）策略改进在FrozenLake中的实现

本节将主要基于gym环境中的FrozenLake-v0进行策略改进的实现。

1. 策略改进的伪代码

算法（迭代策略评估算法），用于估计 $\pi = \pi_*$

1. 初始化

对 $s \in S$ ，任意设定 $V(s) \in R$ 以及 $\pi(s) \in A(s)$

2. 策略评估

循环：

$\Delta \leftarrow 0$

对每一个 $s \in S$ 循环：

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

直到 $\Delta < \theta$ （一个决定估计精度的小正数）

3. 策略改进

$policy - stable \leftarrow true$

对每一个 $s \in S$ ：

$old - action \leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

如果 $old - action \neq \pi(s)$ ，那么 $policy - stable \leftarrow false$

如果 $policy - stable$ 为true，那么停止并返回 $V \approx v_*$ 以及 $\pi \approx \pi_*$ ；否则跳转2。

2. 源代码

```
1 import numpy as np
2 import gym
3
4 def policy_eval(enviroment, policy, discount_factor=1.0, theta=0.1):
5     # 引用环境
6     env = enviroment
7
8     # 初始化值函数
9     v = np.zeros(env.ns)
10
11     # 开始迭代
12     for _ in range(500):
13         delta = 0
14         # 扫描所有状态
15         for s in range(env.ns):
16             v=0
17             # 扫描动作空间
18             for a, action_prob in enumerate(policy[s]):
```

```

19         # 扫描下一状态
20         for prob,next_state,reward,done in env.P[s][a]:
21             # 更新值函数
22             v += action_prob * prob * ( reward + discount_factor *
v[next_state])
23             # 更新最大的误差值
24             delta=max(delta,np.abs(v-v[s]))
25             v[s] =v
26
27         if delta < theta:
28             break
29         return np.array(v)
30
31 # 定义策略生产函数
32 def generate_policy(env,input_policy):
33     policy=np.zeros([env.nS,env.nA])
34     for _ , x in enumerate(input_policy):
35         policy[_][x] = 1
36     return policy
37
38
39 def policy_iteration(env,policy,discount_factor=1.0,endstate=15):
40     while True:
41         # 策略评估
42         v=policy_eval(env,policy,discount_factor)
43
44         policy_stable = True
45         # 策略改进
46         for s in range(env.nS):
47             # 在策略中找到概率最大的动作
48             old_action = np.argmax(policy[s])
49
50             # 在当前策略和状态的基础上找到最优动作
51             action_value = np.zeros(env.nA)
52             for a in range(env.nA):
53                 for prob, next_state, reward, done in env.P[s][a]:
54                     action_value[a] += prob * (reward + discount_factor *
v[next_state])
55
56                 if done and next_state != endstate:
57                     action_value[a] = float( "-inf" )
58
59             # 进行贪婪更新
60             best_action = np.argmax(action_value)
61
62             if old_action != best_action:
63                 policy_stable = False
64                 policy[s] = np.eye(env.nA)[best_action]
65
66         # 稳定后退出
67         if policy_stable:
68             return policy, v
69
70 if __name__=="__main__":
71     env=gym.make("FrozenLake-v0")
72     random_policy = np.ones([env.nS, env.nA])/env.nA
73     finalpolicy, value = policy_iteration(env, random_policy)
74     print("格式化最终的策略 (0 = up, 1 = right, 2= down, 3 =left):\n")
75     print(np.reshape(np.argmax(finalpolicy, axis = 1), [4,4]))

```

```

75     print("最终的值函数:\n")
76     print(value.reshape([4,4]))
77

```

运行后的最终策略和值函数如下:

```

1  格式化最终的策略 (0 = up, 1 = right, 2= down, 3 =left ):
2
3  [[0 3 2 3]
4   [0 0 0 0]
5   [3 1 0 0]
6   [0 2 1 0]]
7  最终的值函数e:
8
9  [[0.          0.          0.01234568 0.00411523]
10 [0.          0.          0.06995885 0.          ]
11 [0.02469136 0.14814815 0.26200274 0.          ]
12 [0.          0.3127572  0.62688615 0.          ]]

```

3.代码解析

- 初始策略选择为了一个随机策略

```

1  random_policy = np.ones([env.nS, env.nA])/env.nA

```

该策略如下所示, 在每个状态下四个动作都具有相同的概率:

```

1  [[0.25 0.25 0.25 0.25]
2   [0.25 0.25 0.25 0.25]
3   [0.25 0.25 0.25 0.25]
4   [0.25 0.25 0.25 0.25]
5   [0.25 0.25 0.25 0.25]
6   [0.25 0.25 0.25 0.25]
7   [0.25 0.25 0.25 0.25]
8   [0.25 0.25 0.25 0.25]
9   [0.25 0.25 0.25 0.25]
10  [0.25 0.25 0.25 0.25]
11  [0.25 0.25 0.25 0.25]
12  [0.25 0.25 0.25 0.25]
13  [0.25 0.25 0.25 0.25]
14  [0.25 0.25 0.25 0.25]
15  [0.25 0.25 0.25 0.25]
16  [0.25 0.25 0.25 0.25]]

```

- 策略更新过程

策略更新过程遵循 $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

```
1         for a in range(env.nA):
2             for prob, next_state, reward, done in env.P[s][a]:
3                 action_value[a] += prob * (reward + discount_factor *
V[next_state])
4                 if done and next_state != endstate:
5                     action_value[a] = float( "-inf" )
6
7             # 进行贪婪更新
8             best_action = np.argmax(action_value)
```