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$s \in S$:

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

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

   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

```
import numpy as np
import gym

def policy_eval(enviroment,policy,discount_factor=1.0,theta=0.1):
    # 引用环境
     env = enviroment

    # 初始化值函数
     V = np.zeros(env.nS)

    # 开始迭代
     for _ in range(500):
         delta = 0
         # 扫描所有状态
         for s in range(env.nS):
             v=0
             # 扫描动作空间
             for a,action_prob in enumerate(policy[s]):
```

```
                # 扫描下一状态
                for prob,next_state,reward,done in env.P[s][a]:
                    # 更新值函数
                    v += action_prob * prob * ( reward + discount_factor * V[next_state])
            # 更新最大的误差值
            delta=max(delta,np.abs(v-V[s]))
            V[s] =v

        if delta < theta:
            break
    return np.array(V)

# 定义策略生产函数
def generate_policy(env,input_policy):
    policy=np.zeros([env.nS,env.nA])
    for _ , x in enumerate(input_policy):
        policy[_][x] = 1
    return policy


def policy_iteration(env,policy,discount_factor=1.0,endstate=15):
    while True:
        # 策略评估
        V=policy_eval(env,policy,discount_factor)

        policy_stable = True
        # 策略改进
        for s in range(env.nS):
            # 在策略中找到概率最大的动作
            old_action = np.argmax(policy[s])

            # 在当前策略和状态的基础上找到最优动作
            action_value = np.zeros(env.nA)
            for a in range(env.nA):
                for prob, next_state, reward, done in env.P[s][a]:
                    action_value[a] += prob * (reward + discount_factor * V[next_state])
                    if done and next_state != endstate:
                        action_value[a] = float( "-inf" )

            # 进行贪婪更新
            best_action = np.argmax(action_value)

            if old_action != best_action:
                policy_stable = False
            policy[s] = np.eye(env.nA)[best_action]

        # 稳定后退出
        if policy_stable:
            return policy, V

if __name__=="__main__":
    env=gym.make("FrozenLake-v0")
    random_policy = np.ones([env.nS, env.nA])/env.nA
    finalpolicy, Value = policy_iteration(env, random_policy)
    print("格式化最终的策略 (0 = up, 1 = right, 2= down, 3 =left ):\n")
    print(np.reshape(np.argmax(finalpolicy, axis = 1), [4,4]))
```

```
    print("最终的值函数:\n")
    print(Value.reshape([4,4]))

```

运行后的最终策略和值函数如下：

```
格式化最终的策略 (0 = up, 1 = right, 2= down, 3 =left ):

[[0 3 2 3]
 [0 0 0 0]
 [3 1 0 0]
 [0 2 1 0]]
最终的值函数e:

[[0.         0.         0.01234568 0.00411523]
 [0.         0.         0.06995885 0.        ]
 [0.02469136 0.14814815 0.26200274 0.        ]
 [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

  ```
  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：

  ```
  [[0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]
   [0.25 0.25 0.25 0.25]]
  ```

- 策略更新过程

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

```
            for a in range(env.nA):
                for prob, next_state, reward, done in env.P[s][a]:
                    action_value[a] += prob * (reward + discount_factor *
V[next_state])
                    if done and next_state != endstate:
                        action_value[a] = float( "-inf" )

            # 进行贪婪更新
            best_action = np.argmax(action_value)
```