中山大学计算机学院人工智能本科生实验报告

2022学年春季学期

课程名称：Artificial Intelligence

教学班级	人工智能（陈川）	专业（方向）	计算机科学与技术人工智能与大数据
学号	20337025	姓名	崔璨明

一、实验题目

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。

实验要求

1. 实现vi_and_pi.py中的策略评估、策略提升、策略迭代函数。
2. 实现vi_and_pi.py中的值迭代函数。

二、实验内容

1、算法原理

强化学习的学习过程，个人理解就是通过不断的尝试，去更新每个状态的值函数(每个状态的值代表了当前状态的优劣，如果状态值很大，从其他状态选择一个动作，转移到该状态便是一个正确的选择)，然后通过更新后的值函数去动态的调整策略，在调整策略后，又去更新值函数，不断的迭代更新，最后训练完成一个满足要求的策略。在这个过程中，有两个主要的过程：策略评估和策略提升。

策略评估:

策略评估就是通过某种方式，计算状态空间中每个状态的值函数。由于状态空间之间存在很多转移关系，要直接计算某个状态的值函数，是很困难的，一般采用迭代方法。

策略提升:

通过当前拥有的信息，对当前策略进行优化，修改当前策略。如改变某个状态时选取每个动作的概率等等，在本实验中修改策略是通过更新Q表，即更新动作对应的Q值，然后选取最大的Q值对应的动作作为策略的动作。

值迭代

值迭代算法对每一个当前状态 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。

期望值计算公式:$V_{k+1}(s) = maxa \sum s', rP(s', r \mid s, a)(r + \gamma V_k(st))$

策略迭代不同的是值迭代是根据状态期望值选择动作，而策略迭代是先估计状态值然后修改策略

2、算法伪代码

策略迭代算法伪代码

```
1. Initialization:
V(s)∈R and π(s)∈ A(s) arbitrarily  for all s∈S

//策略评估
2. Policy Evaluation
        Loop:
                Δ←0
                Loop for each s S:
                        v←V(s)
                        V(s)←∑s'rp(s',r|s,π(s))[r+γV(s')]
                        Δ←max(Δv-V(s)I)
        until Δ<0(a small positive number determining the accuracy of
estimation)
//策略提升
3.Policy Improvement
        policy-stable ←true
        For each s S:
                old-action ←π(s)
                π(s)←argmaxa∑s'rp(s',r|s,a)[r+vV(s')]
                If old-action≠π(s),then policy-stable←false
        If policy-stable, then stop and return V≈v*and π ≈π*; else go to 2
```

值迭代算法伪代码

```
输入：可以接受的误差值0>0，状态价值函数V(s)
Loop:Δ←0
        Loop for each s S:
                v←V(s)
                V(s)←maxa∑s'rp(s',r|s,a)[r+rV(s')]
                Δ←max(Δ,|v-V(s)|)
until Δ< 0

输出：
Output a deterministic policy,π ≈π*,such that
        π(s)=argmax∑s'rp(s,r|sa)[r+V(s')]
```

3、关键代码展示

根据代码中的提示，完成`vi_and_pi.py`中的`policy_evaluation`函数、`policy_improvement`函数和`policy_iteration`函数，具体代码和解析如下：

```python
#策略评价
def policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):
    value_function = np.zeros(nS)
    ##############################
    # YOUR IMPLEMENTATION HERE #
    while True:
        #更新前的价值函数
        update_value_fun=np.copy(value_function)
        #遍历每一个状态
        for state in range(nS):
            #选取动作
            action=policy[state]
            #更新
            value_function[state]=sum([trans_prob*(reward+gamma*update_value_fun[next_state])
                                       for trans_prob, next_state, reward, done in P[state][action]])
        #判断价值函数是否收敛
        if (np.sum((np.fabs(update_value_fun-value_function))) <=tol):
            break
    ##############################
    return value_function
```

```python
#策略提升函数
def policy_improvement(P, nS, nA, value_from_policy, policy, gamma=0.9):
    new_policy = np.zeros(nS, dtype='int')
    ##############################
    # YOUR IMPLEMENTATION HERE #
    #遍历每一个状态
    for state in range(nS):
        #计算Q值
        Q_table=np.zeros(nA)
        for action in  range(nA):
            for next_sr in P[state][action]:
                trans_prob, next_state, reward, done = next_sr
                # 更新动作对应的Q值
                Q_table[action]+=(trans_prob *(reward+gamma*value_from_policy[next_state]))
        #选取Q值最大的策略
        new_policy[state]=np.argmax(Q_table)
    ##############################
    return new_policy
```

```python
#策略迭代算法
def policy_iteration(P, nS, nA, gamma=0.9, tol=10e-3):

    value_function = np.zeros(nS)
    #随机策略全部为0（往左）
    policy = np.zeros(nS, dtype=int)

    ############################
    #迭代次数
    number_iteration=200000
    for i in range(number_iteration):
        #新的价值函数
        new_val_fun=policy_evaluation(P,nS,nA,policy)
        #新的策略
        new_policy=policy_improvement(P,nS,nA,new_val_fun,policy)

        #策略收敛是退出
        if (np.all(policy==new_policy)):
            print("no change between policy\n")
            break
        policy=new_policy
        value_function=new_val_fun


    ############################
    return value_function, policy
```

`value_iteration`函数的实现代码和解析如下：

```python
#值迭代
def value_iteration(P, nS, nA, gamma=0.9, tol=1e-3):


    value_function = np.zeros(nS)
    #随机策略全部为0（往左）
    policy = np.zeros(nS, dtype=int)
    ############################
    #迭代次数
    number_iteration=100000
    for i in range(number_iteration):
        #更新前的价值函数
        update_val_fun=np.copy(value_function)
        #遍历每一个状态
        for state in range(nS):
            Q_value=[]
            for action in range(nA):
                next_sta_reward=[]
                for next_sr in P[state][action]:
                    # done判断是否为终止状态
                    trans_prob,next_state, reward, done = next_sr
                    # 计算next_states_reward
                    next_sta_reward.append(
                        (trans_prob*(reward+gamma*update_val_fun[next_state])))
                    # 计算Q值
                    Q_value.append(np.sum(next_sta_reward))
                    # 取最大Q值更新V表，即更新当前状态的V值
                    value_function[state] = max(Q_value)
        #判断是否收敛
        if(np.sum(np.fabs(update_val_fun-value_function)) <= tol):
            print("no change between value function\n")
            break
    ############################
    policy=policy_improvement(P,nS,nA,value_function,policy)
    #返回价值函数、策略
    return value_function, policy
```

三、实验结果

运行程序，可以得到相应的Frozen Lake问题的结果：
值迭代算法：

```
------------------------
Beginning Policy Iteration
------------------------
no change between policy


SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
Episode reward: 1.000000
```

策略迭代算法：

```
Beginning Value Iteration
-------------------------
no change between value function


SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
Episode reward: 1.000000
PS E:\VSCODE\py>
```

四、参考资料

- 策略迭代和值迭代.pdf