**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 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 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。具体要求如下所示：

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。
3. 阅读vi\_and\_pi.py文件，然后补全policy\_evaluation, policy\_improvement和 policy\_iteration函数。tolerance 定义为tol = 10^{-3}。设置参数gamma=0.9. 返回最优值函数和最佳策略。
4. 补全value\_iteration函数。tolerance 为tol = 10^{-3}. 设置参数gamma=0.9. 返回最优值函数和最佳策略。

# 实验内容

1. 算法原理

本次任务运用到的主要算法为强化学习。强化学习是一种机器学习方法，旨在通过智能体与环境的交互来学习最优的行为策略。强化学习的算法原理主要包括以下几个要素：环境、状态、动作、奖励、策略、值函数。而在本次实验中需要完成的策略评估、策略提升、策略迭代和值迭代函数是强化学习中的重要一环，是用于优化策略函数和值函数的重要步骤。

* **策略评估（Policy Evaluation）**：策略评估的目标是计算给定策略下的值函数。值函数表示在特定状态S下，智能体从当前状态开始，按照策略选择行动A后，未来可以获得的累积奖励reward。通过策略评估可以估计每个状态的值函数，从而了解策略在不同状态下的效果。
* **策略提升（Policy Improvement）**：策略提升的目标是根据值函数改进策略。在策略提升中，根据值函数的估计结果，选择在每个状态S下具有最大值的行动A作为新的策略。
* **策略迭代（Policy Iteration）**：策略迭代是一种交替进行策略评估和策略提升的过程。在策略迭代中，会首先进行策略评估，计算当前策略下的值函数，然后进行策略提升，根据值函数改进策略。这个过程会一直迭代，直到策略不再改变，即达到最优策略。
* **值迭代（Value Iteration）**：值迭代是一种通过迭代更新值函数来学习最优策略的方法。在值迭代中，首先初始化值函数，然后通过迭代更新值函数，直到值函数收敛。在每次迭代中，根据当前值函数的估计值Q来更新值函数，使用贪心策略选择在每个状态下具有最大值的行动。

1. 伪代码

策略迭代：

1. Initialization:

V(s)∈R and π(s)∈ A(s) arbitrarily for all s∈S

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)

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：

algorithm parameter：a small threshold θ>0 determining accuracy of estimation

initialize V（s）， for all s ∈ S+，arbitrarily except that V（terminal）=0

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＜θ

Output a deterministic policy,π ≈π\*,such that

π(s)=argmax∑s'rp(s,r|sa)[r+V(s')]

1. 关键代码展示（带注释）

**策略评估部分**。代码中的循环会迭代更新值函数，直到值函数收敛。在每次迭代中，对于每个状态，会计算其对应的值函数，并选出最大值作为新的值函数。

def policy\_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):

# 策略评估

    value\_function = np.zeros(nS)

    # start implement here

    while True:

        delta=0 # 记录总值看收敛情况

        for state in range(nS): # 遍历状态

            v=value\_function[state] #保存当前状态的值函数

            q\_values=np.zeros(nA) #计算每个动作的值函数

            for action in range(nA): #遍历动作

                for prob, next\_state, reward, \_ in P[state][action]:

                    q\_values[action]+=prob\*(reward+gamma\*value\_function[next\_state]) #update

            value\_function[state]=np.max(q\_values) #选出最大值

            delta=max(delta, np.abs(v-value\_function[state]))

        if delta<tol: #小于阈值则认为收敛

            break

    return value\_function

**策略提升部分**。思路基本同策略评估部分，会循环然后选出具有最大值的动作作为新的策略，最后返回。新的策略是一个整数数组，其中每个元素表示在对应状态下应该选择的动作。

def policy\_improvement(P, nS, nA, value\_from\_policy, policy,gamma=0.9):

#策略提升

    new\_policy = np.zeros(nS, dtype='int')

    # start implement here

    for state in range(nS):

        q\_values=np.zeros(nA) # 算Q值

        for action in range(nA):

            for prob, next\_state, reward, \_ in P[state][action]:

                q\_values[action]+=prob\*(reward+gamma\*value\_from\_policy[next\_state])# update

        new\_policy[state]=np.argmax(q\_values) #选出Q值最大的策略当新策略

    return new\_policy

**策略迭代部分**。循环并不断进行策略评估和提升，如果新策略与旧策略相同，则说明已达到最优策略，循环结束。

def policy\_iteration(P, nS, nA, gamma=0.9, tol=10e-3):

# 策略迭代

    value\_function = np.zeros(nS)

    policy = np.zeros(nS, dtype=int)

    # start implement here

    while True:

        value\_function=policy\_evaluation(P, nS, nA, policy, gamma, tol)

        new\_policy = policy\_improvement(P, nS, nA, value\_function,gamma)

        if np.array\_equal(policy, new\_policy): #如果策略不再改变，说明达到最优策略

            break

        policy = new\_policy

    return value\_function, policy

**值迭代部分**。仍然是通过不断循环，选出具有最大值的动作作为当前状态的值函数。在每次迭代中，通过比较当前值函数与上次迭代的值函数的变化来判断值函数收敛。若小于给定的阈值，则认为值函数收敛。最后，根据更新后的值函数，计算每个状态下具有最大值的动作，作为最优策略。将最优策略保存在policy数组中，并返回最终的值函数和最优策略。

def value\_iteration(P, nS, nA, gamma=0.9, tol=1e-3):

#值迭代

    value\_function = np.zeros(nS)

    policy = np.zeros(nS, dtype=int)

    # start implement here

    while True:

        delta=0 #记录变化，来判断收敛情况

        for state in range(nS): # 遍历状态

            v=value\_function[state] #保存当前状态的值函数

            q\_values=np.zeros(nA) #动作的值函数

            for action in range(nA):

                for prob, next\_state, reward, \_ in P[state][action]:

                    q\_values[action]+=prob\*(reward+gamma\*value\_function[next\_state])

            value\_function[state]=np.max(q\_values) #取最大的Q值 然后更新

            delta=max(delta, np.abs(v-value\_function[state]))

        if delta<tol: #判断收敛

            break

    for state in range(nS):

        q\_values = np.zeros(nA)

        for action in range(nA):

            for prob, next\_state, reward, \_ in P[state][action]:

                q\_values[action]+=prob\*(reward+gamma\*value\_function[next\_state])

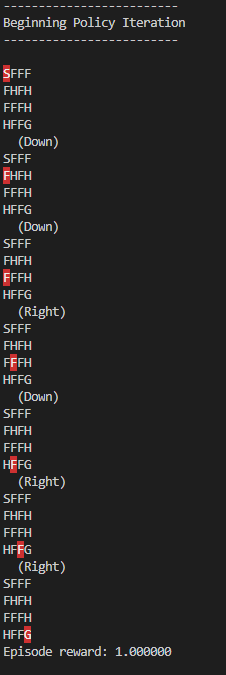
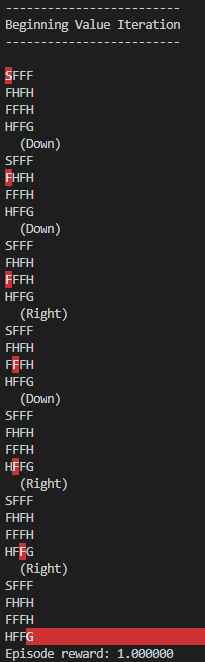
        policy[state] = np.argmax(q\_values)

    return value\_function, policy

# 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

**策略迭代算法： 值迭代算法：**

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

# 参考资料

[Frozen Lake问题参数释义以及一种实现方式](https://blog.csdn.net/njshaka/article/details/89237941)

**超算习堂-课时32-第十七讲实验课**

**超算习堂-课时27-第十四讲**