悬崖漫步——强化学习实验报告

201250205 郭慕凡

1.悬崖漫步情景中策略迭代和价值迭代的伪代码

为了较好地说明策略迭代与价值迭代地伪代码，我将在1.1和1.2中首先概述环境描述、策略评估与策略迭代，最后在1.3说明策略迭代；在1.4说明价值迭代。

1.1悬崖漫步场景描述

4*12的网格世界，每个网格代表一个状态，智能体的起始位置是左下角，目标位置是右下角，起始位置到悬崖位置的直线路径经过的网格为悬崖。在每个网格（状态）可以采取四种行动：向上，向下，向左，向右。目标是学习到从起始位置达到目标位置的策略（路径）

将以上情景转换为数学表达，其中强化学习用到的符号定义如下：

$$E =< X, A, P, R >$$

E:环境；X:状态集；A:行动集；P:转移概率；R:奖赏; 其中P，R：$X \times A \times X \to \mathbb{R}$

策略：

$$\pi : X \times A \to \mathbb{R}$$

累计奖赏计算方式采用γ折扣累计奖赏，即：

$$E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1}]$$

状态值函数及状态-动作函数定义为：

$$V_\gamma^\pi(x) = E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x]$$

$$Q_\gamma^\pi(x, a) = E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x, a_0 = a]$$

下文用到的符号均与此处保持一致。

在本情景中，环境具体描述如下：

1. X是4*12的网格世界，每个网格代表一个状态,如果设irow为行，icol为列,则X的可以用irow*12+icol的整数值表示
2. A={向左，向右，向上，向下}
3. P可以表示为一个矩阵：P[status][action] = [(p, new_status)]；需处理边界情况，如地图边界，悬崖，终点
4. R可以表示为一个矩阵：R[status][action][new_status] = r；需处理边界情况，如地图边界，悬崖，终点
5. 策略π可以表示为一个矩阵：Pi[status][action] = p

1.2 策略评估与策略改进

策略评估:

对于给定的策略π，可以使用**Bellman等式**计算出每个状态的状态值函数和状态动作函数。

在实际计算中，并不会采用解线性方程组的形式计算每个状态的状态值函数，而是将状态值函数初始化为0后迭代计算，当计算值改变不大（即小于一个收敛因子θ时，迭代结束。这是一种类似"动态规划"的算法。

结合悬崖漫步的背景，计算方法的伪代码如下：其中参数θ代表收敛阈值。

```
def policy_evaluation:
    for x in X: V[x]= 0;//初始化为0
    max_diff = 0;
    while 1:
        for x in X:
            V_new[x] = 0
            qsa_list = [];
            for a in A:
                //计算qsa
                qsa = sum(P[x][x_new][a]*(R[x][x_new][a]+gamma*V(x_new)) for x_new in X);
                qsa_list append qsa;
            V_new = sum(qsa * pi[s][a] for qsa in qsa_list[]);
            max_diff = max(max_diff, abs(V(x)-V_new));
            V_new[x] = V_new
        if max_diff < theta:满足迭代终止条件
            break;
        for x in X: V[x] = V_new[x]
```

策略改进:

我们希望最优策略能够最大化累积奖赏，即

$$\pi^* = argmax \sum_{x \in X} V^{\pi}(x)$$

记

$$V^*(x) = V^{\pi^*}(x)$$

由最优策略的累积奖赏值最大的特性，**我们可以将前面Bellman等式的对动作的求和变为取最优**，即

$$V^*(x) = max_{a \in A} Q^{\pi^*}(x, a)$$

这告诉我们非最优策略的改进方式：将策略选择的动作改变为策略评估环节得出的最优的动作（对应策略-动作值函数最大）

因此结合悬崖漫步的背景，伪代码可以表述为：

```
def policy_improvement:
    for x in X:
        qsa_list = [];
        for a in A:
            //计算qsa
            qsa = sum(P[x][x_new][a]*(R[x][x_new][a]+gamma*V(x_new)) for
x_new in X);
            qsa_list append qsa;
        max_q = max(qsa in qsa_list[])
        //如果有多个qsa=max_q，则在新策略中等可能地执行对应的动作
        cnt_q = qsa_list.count(max_q)
        pi[x][a] = 1 / cnt_q if q==max_q else 0
```

1.3 策略迭代的伪代码

在前述策略评估和策略改进的伪代码的基础上，给出策略迭代的伪代码：

```
def policy_iteration:
    policy_evaluation();
    old_pi;
    new_pi = policy_improvemrnt();
    if old_pi == new_pi: break;
```

1.4 值迭代的伪代码

策略迭代在每次改进策略后都要重新进行**策略评估，这非常耗时。**

注意到策略改进与值函数的改进是一致的，如果我们将改进终止的指标从策略不变变为值函数基本不发生改变（仍然需要收敛因子），**并且修改迭代更新值函数的计算公式**，那么我们就得到了值迭代的方法。本质上仍然在利用：

$$V^*(x) = max_{a \in A} Q^{\pi^*}(x, a)$$

伪代码阐述如下。

```
def value_iteration:
    while 1:
        for x in X: V[x], V_new[x] = 0;//初始化为0
        max_diff = 0;
        for x in X:
            qsa_list = [];
            for a in A:
                //计算qsa
                qsa = sum(P[x][x_new][a]*(R[x][x_new][a]+gamma*V(x_new)) for
x_new in X);
                qsa_list append qsa;
            //注意！更新公式发生改变
            V_new = max(qsa for qsa in qsa_list[]);
            max_diff = max(max_diff, abs(V(x)-V_new));
            V_new[x] = V_new
        V = V_new
        if max_diff < theta:满足迭代终止条件
            break;
```

2.代码详述

2.1悬崖漫步环境设计

```
"""
        (0, 0) -- -- -- -- -- -- -- -- --  --  --  > col
        |   .  .  .  .  .  .  .  .  .  .  .  .   |
        |   .  .  .  .  .  .  .  .  .  .  .  .   |
        |   .  .  .  .  .  .  .  .  .  .  .  .   |
        begin  x  x  x  x  x  x  x  x  x  x  target
        v
        row
"""
```

以上是悬崖漫步的场景“图示”

在代码设计中，我们定义一个`CliffWalkingEnv`类来表示悬崖漫步的环境。其构造函数如下：

```
    def __init__(self, nrow=4, ncol=12):
        self.nrow = nrow
        self.ncol = ncol
        self.P = self.creatEnv()
```

其中nrow，ncol代表悬崖平台的长和宽，而P则是状态转移矩阵，是E中P（状态-动作转移概率）与R（奖赏函数）的统一表达。P的定义如下：

```
P[state][action] = [(p, next_state, reward, done)]
# p:在state采取action能够转移到next_state的概率；done：boolean，当在悬崖或者target位置时取1，否则取0
```

环境设计的关键在于P的初始化，而P的初始化的关键在于边界情况：设s为当前状态（位置）

1. 如果s是悬崖状态或者目标位置状态，那么done=True，reward=0，next_state=s；
2. 对于s是其他状态，分别向上、下、左、右移动，计算移动后位置：
    - 如果移动后达到目标位置，奖励100
    - 如果移动后掉到悬崖里，奖励-100
    - 如果移动后超出目标边界，则next_state仍为s，且奖励为0
    - 其他：奖励为0

以上规则明确后的代码化就不再赘述。

可以看到，我的设计与助教给的实例代码在奖励设计上略有不同，但得到的结果是相同的。

2.2策略迭代

策略迭代的实现代码和伪代码接近。实现如下：

```
def policy_iteration(self):
    while 1:
        self.policy_evaluation()
        old_pi = copy.deepcopy(self.pi)
        new_pi = self.policy_improvement()
        if old_pi == new_pi: break
```

其中，policy_evaluation时策略评估函数，policy_improvement时策略提升函数。此外，需要注意在比对时需要使用深拷贝。

策略评估代码，说明见代码注释：

```
def policy_evaluation(self):
    cnt = 1
    while 1:
        max_diff = 0
        #值函数初始化为0
        new_v = [0] * self.env.ncol * self.env.nrow
        for s in range(self.env.ncol * self.env.nrow):
            qsa_list = []
            for a in range(4):
                # 计算qsa
                qsa = 0
                for res in self.env.P[s][a]:
                    p, next_state, r, done = res
                    # part of Bellman
                    qsa += p*(r + self.gamma*self.v[next_state]*(1-done))
                qsa_list.append(self.pi[s][a] * qsa)
            #更新值函数，依据是Bellman等式
            new_v[s] = sum(qsa_list)
            max_diff = max(max_diff, abs(new_v[s] - self.v[s]))
        self.v = new_v
        # 判断是否循环终止
        if max_diff < self.theta: break
        cnt += 1
    print("策略评估进行%d轮后完成" % cnt)
```

策略提升代码，说明见代码注释：

```
def policy_improvement(self):
    for s in range(self.env.nrow * self.env.ncol):
        qsa_list = []
        for a in range(4):
            # 计算qsa
            qsa = 0
            for res in self.env.P[s][a]:
                p, next_state, r, done = res
                qsa += p*(r + self.gamma*self.v[next_state]*(1-done))
            qsa_list.append(qsa)
        # 获取最大qsa，并且将策略更改为采取qsa最大的动作
        maxq = max(qsa_list)
        # 如果qsa最大的有多个，则它们等分概率
        cntq = qsa_list.count(maxq)
        # 更新策略
        self.pi[s] = [1 / cntq if q == maxq else 0 for q in qsa_list]
    print("策略提升完成")
    return self.pi
```

2.3值迭代

```
def value_iteration(self):
    cnt = 0
    while 1:
```

```
        max_diff = 0
        # 值函数初始化为0
        new_v = [0] * self.env.ncol*self.env.nrow
        for s in range(self.env.ncol * self.env.nrow):
            qsa_list = []
            for a in range(4):
                # 计算qsa
                qsa = 0
                for res in self.env.P[s][a]:
                    p, next_state, r, done = res
                    qsa += p * (r+self.gamma * self.v[next_state]*(1 - done))
                qsa_list.append(qsa)
            # 注意，这里是与策略提升的最大不同，我们直接更新值函数，而不是通过更新策略进行迭代
            new_v[s] = max(qsa_list)
            max_diff = max(max_diff, abs(new_v[s]-self.v[s]))
        self.v = new_v
        # 判断迭代是否应当终止
        if max_diff < self.theta:break
        cnt += 1
    print("价值迭代一共进行 %d轮" % cnt)
    # 由值迭代的结果得到策略，逻辑和之前一样
    self.get_policy()
```

3.结果

```
策略评估进行58轮后完成
策略提升完成
策略评估进行73轮后完成
策略提升完成
策略评估进行56轮后完成
策略提升完成
策略评估进行10轮后完成
策略提升完成
策略评估进行1轮后完成
策略提升完成
状态价值：
25.419 28.243 31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000
28.243 31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000 90.000
31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000 90.000 100.00
28.243  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
策略：
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE

价值迭代一共进行 14轮
状态价值：
25.419 28.243 31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000
28.243 31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000 90.000
31.381 34.868 38.742 43.047 47.830 53.144 59.049 65.610 72.900 81.000 90.000 100.00
28.243  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
策略：
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE

Process finished with exit code 0
```