Reinforcement Learning Assignment Report

516030910373 XueyanLi

Part 1: Policy Valuation:

Policy Valuation calculates the value of a given policy.

✧ Code:

Figure 1 shows the code of Policy Valuation.

```
def valueUpdate(V, policy):
    # update the v to Vs
    delta = 2
    while (delta > 0.001):
        delta = 0
        v = []
        for line in V:
            tmpv = []
            for Vs in line:
                tmpv.append(Vs)
            v.append(tmpv)


        #calculate new V
        for i in range(4):
            for j in range(4):
                if (i == 0 and j == 0 or (i == 3 and j == 3)):
                    V[i][j] = 0.0
                    continue
                im = i - 1
                ip = i + 1
                jm = j - 1
                jp = j + 1
                if (im < 0): im = 0
                if (ip > 3): ip = 3
                if (jm < 0): jm = 0
                if (jp > 3): jp = 3
                V[i][j] = -1.0 + v[im][j] * policy[2][i][j] + v[ip][j] * policy[3][i][j] + v[i][jm] * policy[0][i][j] + v[i][jp] * policy[1][i][j]
                delta = delta + abs(V[i][j] - v[i][j])
    for line in V:
        print(line)
```

Figure 1 the code of Policy Valuation

✧ Result:

Figure 2 shows the policy valuation of the optimal policy calculated in Policy Iteration, which is shown in Part 2.

```
[0.0, -13.999099182742686, -19.99866515103998, -21.99850622507236]
[-13.999099182742688, -17.998824077007605, -19.998674080280555, -19.998665151039983]
[-19.99866515103998, -19.998674080028055, -17.998824077007605, -13.999099182742688]
[-21.998506225072354, -19.99866515103998, -13.999099182742684, 0.0]
```

Figure 2 the result of Policy Valuation

Part 2: Policy Iteration:

Policy Iteration improve the policy according to the value calculated in Policy Valuation. For every grid, Policy Iteration will update the policy greedily which choose the action with the highest value.

Code:

Figure 3 shows the code of Policy Iteration.

```
def generatePolicy(V, policy):
    p = []
    for i in range(4):
        for j in range(4):
            if (i == 0 and j == 0 or i == 3 and j == 3):
                continue
            im = i - 1
            ip = i + 1
            jm = j - 1
            jp = j + 1
            if (jm < 0):
                west = MINX
            else:
                west = V[i][jm]
            if (jp > 3):
                east = MINX
            else:
                east = V[i][jp]
            if (im < 0):
                north = MINX
            else:
                north = V[im][j]
            if (ip > 3):
                south = MINX
            else:
                south = V[ip][j]

            maxp = max(west, east, north, south)
            count = 0
            if (abs(maxp - west) < 0.0001):
                count = count + 1
            if (abs(maxp - east) < 0.0001):
                count = count + 1
            if (abs(maxp - north) < 0.0001):
                count = count + 1
            if (abs(maxp - south) < 0.0001):
                count = count + 1
            if (abs(maxp - west) < 0.0001):
                policy[0][i][j] = 1.0 / count
            else :
                policy[0][i][j] = 0.0
            if (abs(maxp - east) < 0.0001):
                policy[1][i][j] = 1.0 / count
            else :
                policy[1][i][j] = 0.0
            if (abs(maxp - north) < 0.0001):
                policy[2][i][j] = 1.0 / count
            else :
                policy[2][i][j] = 0.0
            if (abs(maxp - south) < 0.0001):
                policy[3][i][j] = 1.0 / count
            else :
                policy[3][i][j] = 0.0
```

Figure 3 the code of Policy Iteration

Result:

Figure 4 shows the result of Policy Iteration, which is the optimal policy. Every matrix represents the probability of a grid to choose this action. For example, the 0.5 in first matrix[1][2] means when in this grid, the probability to go west is 0.5. This result is equal to the policy in the Lecture, which is shown in Figure 5.

```
The matrix of policy is:
The probability of going west in the optimal policy:
[0.0, 1.0, 1.0, 0.5]
[0.0, 0.5, 0.5, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
goint east:
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.5, 0.5, 0.0]
[0.5, 1.0, 1.0, 0.0]
goint north:
[0.0, 0.0, 0.0, 0.0]
[1.0, 0.5, 0.0, 0.0]
[1.0, 0.5, 0.0, 0.0]
[0.5, 0.0, 0.0, 0.0]
goint south:
[0.0, 0.0, 0.0, 0.5]
[0.0, 0.0, 0.5, 1.0]
[0.0, 0.0, 0.5, 1.0]
[0.0, 0.0, 0.0, 0.0]
```

Figure 4 the result of Policy Iteration

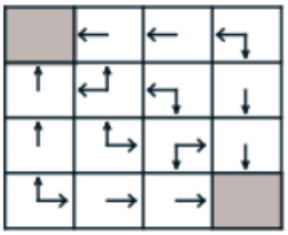

Figure 5 the policy in Lecture

Part 3: Value Iteration:

Value Iteration updates the policy every time after the value of policy is calculated.

Code:

```
def valueUpdateOnce(V, policy):
    # update the v to Vs
    delta = 0
    v = []
    for line in V:
        tmpv = []
        for Vs in line:
            tmpv.append(Vs)
        v.append(tmpv)


    #calculate new V
    for i in range(4):
        for j in range(4):
            if (i == 0 and j == 0 or (i == 3 and j == 3)):
                V[i][j] = 0.0
                continue
            im = i - 1
            ip = i + 1
            jm = j - 1
            jp = j + 1
            if (im < 0): im = 0
            if (ip > 3): ip = 3
            if (jm < 0): jm = 0
            if (jp > 3): jp = 3
            V[i][j] = -1.0 + v[im][j] * policy[2][i][j] + v[ip][j] * policy[3][i][j] + v[i][jm] * policy[0][i][j] + v[i][jp] * policy[1][i][j]
            delta = delta + abs(V[i][j] - v[i][j])
    #print("v")
    #print(v)
    #print(V)
    if (delta < 0.001):
        return 1
    else:
        return 0


for i in range(10):
    valueUpdateOnce(V, policy)
    generatePolicy(V, policy)
```

Figure 6 the code of Value Iteration

Result:

Figure 7 shows the result of Value Iteration. Like part 2, every matrix represents the probability of a grid to choose this action. It can be found that the optimal policy of Policy Iteration is different from the one of Value Iteration. However, they are both optimal policies.

```
Using value iteration:
The probability of going west in the optimal policy:
[0.0, 1.0, 1.0, 0.5]
[0.0, 0.5, 0.25, 0.0]
[0.0, 0.25, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
goint east:
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.25, 0.0]
[0.0, 0.25, 0.5, 0.0]
[0.5, 1.0, 1.0, 0.0]
goint north:
[0.0, 0.0, 0.0, 0.0]
[1.0, 0.5, 0.25, 0.0]
[1.0, 0.25, 0.0, 0.0]
[0.5, 0.0, 0.0, 0.0]
goint south:
[0.0, 0.0, 0.0, 0.5]
[0.0, 0.0, 0.25, 1.0]
[0.0, 0.25, 0.5, 1.0]
[0.0, 0.0, 0.0, 0.0]
```

Figure 7 the result of Value Iteration