Report of Assignment 3

717030210005 Zhexin Jin

1. Environment

```
class Grid:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.action = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        if self.x == 3 and 0 < self.y < 11:
            self.reward = -100.0
        else:
            self.reward = -1.0
        self.gamma = 1.0
        self.epsilon = 0.1
        self.alpha = 0.3

    def nextState(self, gridworld, action):
        next_x, next_y = self.x + action[0], self.y + action[1]
        if next_x > 3 or next_x < 0 or next_y > 11 or next_y < 0:
            next_x, next_y = self.x, self.y
        return gridworld[next_x][next_y]
```

I use a class named Grid to describe every grid in gridworld which is a 4*12 array.

This class has some attributes.

Self.x and self.y are the coordinates of the grid in gridworld.

Self.action contains four movements: (1,0) : east; (0,1) : south; (-1,0) : west; (0,-1) : north.

Self.reward is always -1 except the region marked "The Cliff". This region incurs a reward of -100.

Self.gamma, self.epsilon and self.alpha are parameters related to Q's iterative formula.

The method nextState() will return the next Grid after taking the action.

2. Sarsa

```
def sarsa():
    # init gridworld
    gridworld = [[Grid(i, j) for j in range(12)] for i in range(4)]
    # init Q
    stateList = [gridworld[i][j] for j in range(12) for i in range(4)]
    actionList = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    Q = {state:{action:0.0 for action in actionList} for state in stateList}
```

```
    for k1 in range(10000):     # for each episode
        state = gridworld[3][0]
        # get action using e_greedy
        action = e_greedy(gridworld, Q, state)
        while state != gridworld[3][11]:    # for each step
            # take action, get next_state, consider cliff region
            if state.x == 3 and 0 < state.y < 11:
                next_state = gridworld[3][0]
            else:
                next_state = state.nextState(gridworld, action)
            # get next_action using e_greedy
            next_action = e_greedy(gridworld, Q, next_state)
            if next_action != (0,0):
                # update Q using Q's iterative formula
                Q[state][action] += state.alpha * (state.reward + state.gamma *
Q[next_state][next_action] - Q[state][action])
            # S <= S', A <= A'
            state = next_state
            action = next_action

    # show policy, if failed, print "failed!" five times
    print('Sarsa:')
    state = gridworld[3][0]
    episode = [state]
    while state != gridworld[3][11]:
        allQ = [Q[state][action] for action in state.action]
        maxQ = max(allQ)
        maxA = state.action[allQ.index(maxQ)]
        state = state.nextState(gridworld, maxA)
        if state in episode:
            print('failed!\t'*5)
            break
        episode.append(state)
    if state == gridworld[3][11]:
        for state in episode[:-1]:
            print((state.x,state.y),end='->')
        state = episode[-1]
        print((state.x,state.y))
```

Process:

1. Initialize gridworld and Q.
2. Initialize state.
3. Choose action from state using e_greedy().
4. Take action to get next_state.
5. Choose next_action from next_state using e_greedy().
6. Update Q using Q's iterative formula: Q(S,A) <= Q(S,A) + alpha * [R + gamma * Q(S',A') - Q(S,A)].
7. S <= S', A <= A'.
8. Go to 4 and repeat until state is terminal.
9. Go to 2. Repeat 10000 times.
10. Show final policy. If this policy can't reach to terminal state, print "failed!" five times.

```
def e_greedy(gridworld, Q, state):
    if state == gridworld[3][11]:
        return (0,0)
    allQ = [Q[state][action] for action in state.action]
    maxQ = max(allQ)
    maxA = state.action[allQ.index(maxQ)]
    pi_maxA = 1 - state.epsilon
    pi_each = state.epsilon/4
    randnum = random.random()
    if randnum < pi_maxA:
        return maxA
    else:
        return state.action[int((randnum - pi_maxA) // pi_each)]
```

The function above will return an action using e_greedy improvement.

If the state is terminal return action (0, 0).

3. Q-learning

```
def q_learning():
    # init gridworld
    gridworld = [[Grid(i, j) for j in range(12)] for i in range(4)]
    # init Q
    stateList = [gridworld[i][j] for j in range(12) for i in range(4)]
    actionList = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    Q = {state:{action:0.0 for action in actionList} for state in stateList}

    for k1 in range(10000):     # for each episode
        state = gridworld[3][0]
        while state != gridworld[3][11]:    # for each step
            # choose action using e_greedy
            action = e_greedy(gridworld, Q, state)
            # take action considering cliff region
            if state.x == 3 and 0 < state.y < 11:
                next_state = gridworld[3][0]
            else:
                next_state = state.nextState(gridworld, action)
            # choose next_action using greedy
            next_action = greedy(gridworld, Q, next_state)
            if next_action != (0,0):
                # update Q using Q's iterative formula
                Q[state][action] += state.alpha * (state.reward + state.gamma *
Q[next_state][next_action] - Q[state][action])
            state = next_state

    # show policy
    print('Q-learning:')
    state = gridworld[3][0]
    print((state.x,state.y),end='->')
    while state != gridworld[3][11]:

        allQ = [Q[state][action] for action in state.action]
```

```
        maxQ = max(allQ)
        maxA = state.action[allQ.index(maxQ)]
        state = state.nextState(gridworld, maxA)
        if state == gridworld[3][11]:
            print((state.x,state.y))
        else:
            print((state.x,state.y),end='->')
```

The process of q_learning() is very similar to sarsa(), but it uses greedy improvement when choosing next_action and doesn't really take next_action at next time.

```
def greedy(gridworld, Q, state):
    if state == gridworld[3][11]:
        return (0,0)
    allQ = [Q[state][action] for action in state.action]
    maxQ = max(allQ)
    maxA = state.action[allQ.index(maxQ)]
    return maxA
```

The function above will return an action using greedy improvement.

4. Conclusion

```
for k in range(10):
    # Sarsa
    sarsa()

    # Q-learning
    q_learning()
```

Set epsilon to 0.1 and repeat 10 times to see the results: (the results of sarsa may be different)

```
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(0, 0)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(1, 5)->(1, 6)->(1, 7)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(0, 0)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)

Sarsa:
```

```
(3, 0)->(2, 0)->(1, 0)->(0, 0)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(1, 1)->(1, 2)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(1, 8)->(1, 9)->(1, 10)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(1, 1)->(1, 2)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(2, 1)->(1, 1)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(1, 1)->(1, 2)->(1, 3)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(0, 0)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Sarsa:
(3, 0)->(2, 0)->(1, 0)->(0, 0)->(0, 1)->(0, 2)->(0, 3)->(0, 4)->(0, 5)->(0, 6)->(0, 7)->
(0, 8)->(0, 9)->(0, 10)->(0, 11)->(1, 11)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
```

Set epsilon to 0.000001 and repeat 10 times to see the results.

Always:

```
Sarsa:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
Q-learning:
(3, 0)->(2, 0)->(2, 1)->(2, 2)->(2, 3)->(2, 4)->(2, 5)->(2, 6)->(2, 7)->(2, 8)->(2, 9)->
(2, 10)->(2, 11)->(3, 11)
```