Coursework 1

Xiaonan Chong
co424H Reinforcement learning

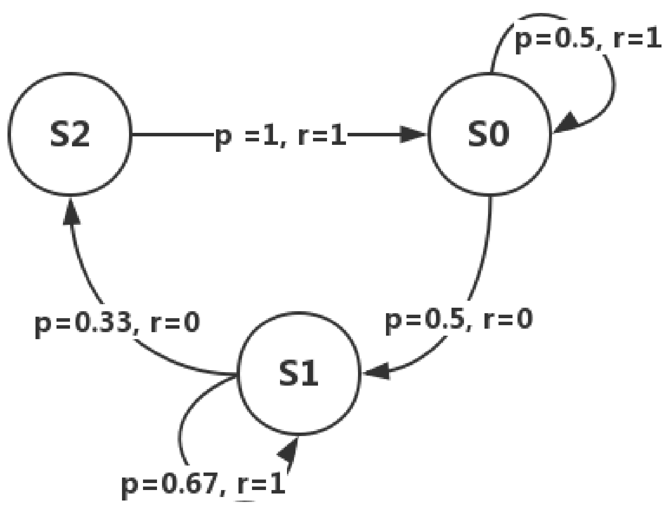
November 10, 2018

My CID is: 1529904

Exercise 1. understanding of MDPs

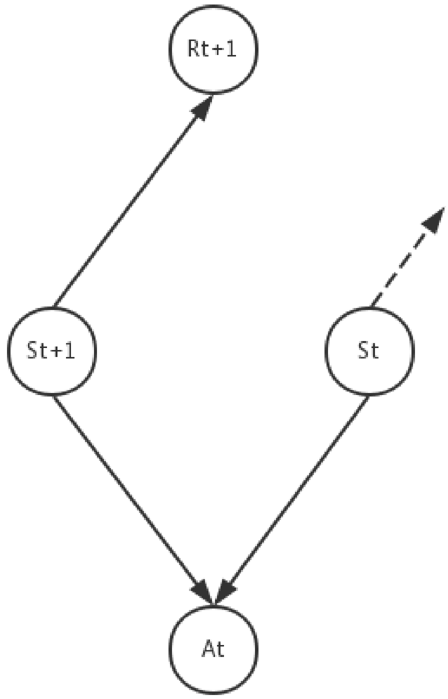
1. τ =s$_2$ 1 s_0 1 s_0 0 s_1 1 s_1 1 s_1 0 s_2 0

2 (a) **transition matrix and reward function**



Assume we do not know the transition matrix, we can draw this graph above according to the trace. It shows that the transition matrix is not deterministic and we can only specify the probability of states transaction based on the experiences. In terms of reward function, in my case, there is no conflicts, that is given initial state, action and next state, the reward is a certain value. And according to only the trace, we do not know that rewards are actually not dependent on the action nor the next state. Therefore, I can assume the reward function is deterministic as shown on the graph.

The following graph shows an abstract version of the real Monte Carlo process, where the succinct rewards rely only on the current state. The actions are determined by states and actually keeps the same value in all cases: 'no choice'. And the next state is not dependent on the current state.



2 (b) **value of states**

If we adopt every visit Monte Carlo method, then during the whole episode, we observe that the state S_0 appears twice: once return reward 1 and once return reward 0. Therefore, the average of the value of state S_0 is 0.5.

Or if we use the first visit Monte Carlo method, and only count on the first appearance of each state. Then, the state value of S_0 is 1.

Exercise 2. understanding of grid world

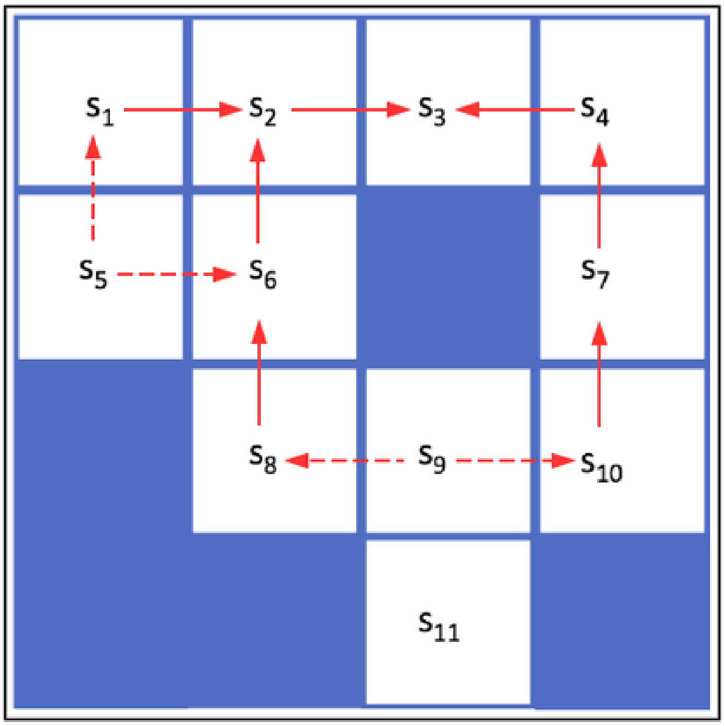
1. reward state is S_3, p is 0.7, γ is 0.3.

2. I adopt Q-learning method to find the optimal value function and policy. It is assumed that the learning rate $\alpha = 0.5$ and I decide the learning iteration to be 1000000. After running my codes, which is attached in the appendix 1.1, I have the final Q-value for each state and action pairs.

The following is a table directly showing the result Q-values:

	s1	**s2**	**s3**	**s4**	**s5**	**s6**	**s7**	**s8**	**s9**	**s10**	**s11**
N	-0.4	2.0	0	2.0	-0.4	2.0	2.0	-0.4	-1.34	-0.4	0
E	2.0	10.0	0	2.0	-0.4	-0.4	-0.4	-1.34	-1.12	-1.12	0
S	-1.12	-0.4	0	-0.4	-1.12	-1.12	-1.12	-1.12	-100	-1.12	0
W	-0.4	-0.4	0	10.0	-1.12	-1.12	-0.4	-1.12	-1.12	-1.34	0

Then I can derive the optimal policy from the above table, that is pick the action with largest value for each state. Dashed arrow indicates that there are more than one actions for that state have the same Q-value.



3. My learned policy choose to go East or West with equal possibility.

1 appendix

1.1 python codes

```
##coding of the four direction
#        0
# 3              1
#        2
## 0-up, 1-right, 2-down, 3-left

import numpy as np
import random as R

gamma = 0.3
alpha = 0.5

# s1-I[1-2,4-10] a-I[0,3], return: s2-I[1,10]
def takeaction(s1, a):
    #-1 means there is a wall in that direction
    transition_matrix = [[-1, 2, 5, -1], #1
                         [-1, 3, 6, 1],  #2
```

```
                        [0, 0, 0, 0], #3 terminal state
                        [-1, -1, 7, 3], #4
                        [1, 6, -1, -1], #5
                        [2, -1, 8, 5],  #6
                        [4, -1, 10, -1],#7
                        [6, 9, -1, -1], #8
                        [-1, 10, 11, 8],#9
                        [7, -1, -1, 9], #10
                        [0, 0, 0, 0]]   #11 terminal state
    if(s1 != 3 and s1 != 11):
        index=s1-1
        s2 = transition_matrix[index][a]
        if (s2==-1):
            return s1
        else:
            return s2
    else:
        print('warning: you are in terminal state.')

def checkreward(s):
    if(s == 11):
        r = -100
    elif(s==3):
        r=10
    else:
        r=-1
    return r

Q = [ [ int(round(R.random()*10)) for i in range(4) ] for j in range(11) ]
#[11.4]
Q[10] = [0 for i in range(4)]
#Q(terminal-state, .)=0
Q[2] = [0 for i in range(4)]
print(Q)

for i in range(1000000):
    Q_privious =[ [Q[b][a] for a in range(4)] for b in range(11)]

    start_state = round(R.random()*10)+1 #randomly choose the initial state
    s = int(start_state)
    print('')
    print('episode: '+str(i)+' start with state: '+str(s))

    while (s != 11 and s != 3):

        desired_action = np.argmax(Q[s-1]) ##greedy method
```

```
        actions = [0,1,2,3]
        #corresponding to N, E, S, W- up, right, down, left
        actions.remove(desired_action)
        random_number = R.random()
        if(random_number <= 0.7):
            action_taken = desired_action
        elif(random_number <=0.8):
            action_taken = actions[0]
        elif(random_number <=0.9):
            action_taken = actions[1]
        else:
            action_taken = actions[2]

        print('desired action: '+ str(desired_action)
        +' | action taken: '+ str(action_taken))

        next_state = takeaction(s, action_taken)
        reward = checkreward(next_state)

        #update Q-value
        m = max(Q[next_state-1])

        Q[s-1][action_taken] = Q[s-1][action_taken]
        + alpha*(reward + 0.3*m - Q[s-1][action_taken])

        print(str(s)+' -> '+str(next_state)+ ' reward= '+str(reward) )
        print('update Q[' + str(s)+'],[' + str(action_taken) +']'
        + ' = '+ str(Q[s-1][action_taken]))

        s= next_state

    ##condition for converge
    change = [[Q[j][i]-Q_privious[j][i] for i in range(4)] for j in range(11)]
    print(change)

print('final value function')
print(Q)
```

1.2 running result

```
episode: 999996 start with state: 3
[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0, 0, 0], [0.0, 0.0, 0.0
, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0,
 0, 0]]

episode: 999997 start with state: 9
desired action: 1 | action taken: 1
9 -> 10 reward= -1
update Q[9],[1] = -1.12
desired action: 0 | action taken: 0
10 -> 7 reward= -1
update Q[10],[0] = -0.4
desired action: 0 | action taken: 0
7 -> 4 reward= -1
update Q[7],[0] = 2.0
desired action: 3 | action taken: 3
4 -> 3 reward= 10
update Q[4],[3] = 10.0
[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0, 0, 0], [0.0, 0.0, 0.0
, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0,
 0, 0]]

episode: 999998 start with state: 7
desired action: 0 | action taken: 0
7 -> 4 reward= -1
update Q[7],[0] = 2.0
desired action: 3 | action taken: 3
4 -> 3 reward= 10
update Q[4],[3] = 10.0
[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0, 0, 0], [0.0, 0.0, 0.0
, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0,
 0, 0]]

episode: 999999 start with state: 4
desired action: 3 | action taken: 3
4 -> 3 reward= 10
update Q[4],[3] = 10.0
[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0, 0, 0], [0.0, 0.0, 0.0
, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0, 0,
 0, 0]]
final value function
[[-0.4, 2.0, -1.12, -0.4], [2.0, 10.0, -0.4, -0.4], [0, 0, 0, 0], [2.0, 2
.0, -0.4, 10.0], [-0.4, -0.4, -1.12, -1.12], [2.0, -0.4, -1.12, -1.12], [
2.0, -0.4, -1.12, -0.4], [-0.4, -1.3359999999999999, -1.12, -1.12], [-1.3
359999999999999, -1.12, -100.0, -1.12], [-0.4, -1.12, -1.12, -1.335999999
9999999], [0, 0, 0, 0]]
xc4718@edge08:Desktop$
```