70

CO424H Coursework

Jinsung Ha

November 21, 2018

Question 1

1.1 1

My personalised trace from my CID (00935517) is $\tau = s_1 1 s_1 1 s_0 1 s_0 1 s_2 1 s_2 1$. good 100%

1.2 2

1.2.1 a

80% - Incomplete answer. You need to actually estimate the coefficient in the transition matrix here and explain how. You should also explain how you obtain the graph from the trace

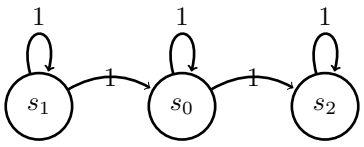


Figure 1: MDP

$$\text{transition matrix} = \begin{pmatrix} \beta & 0 & 1-\beta \\ 1-\alpha & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$\text{reward function} = \begin{pmatrix} 1 & . & 1 \\ 1 & 1 & . \\ . & . & 1 \end{pmatrix} \tag{2}$$

1. Regarding the transition matrix, there is only one action, thus one transition matrix of 3x3 shape where each row and column represents each state from s_0, s_1 and s_2. Since we are given one trace only (i.e. one possible list of executed actions), we can only derive a transition matrix from the sample (refer to the Figure 1: MDP). For the transition matrix, we assume that the sample MDP represents the population. Therefore, for example, there can be no transition from s_0 to s_1 as it did not happen in the trace.
2. The reward function is a 3x3 matrix. As before, we assume that the sample represents the population. Therefore, the reward function contains 1s and undefined rewards only (i.e. no 0s).

1.2.2 b

There are two s_0 in the trace. The values of each of s_0 are 3 and 2 rewards respectively. Therefore the value of s_0 is $\frac{(3+2)}{2} = 2.5$. Again, this only works if we assume that the trace (i.e. sample) represents the population.

Correct, but you should name your method. 80%

Question 2

2.1 1

100%

From the CID of mine, I could derive the reward state s_3, $p = 0.5$ and $\gamma = 0.35$.

2.2 2

I have used Q-Learning algorithm to compute optimal value function and the optimal policy (please refer to Appendix A). For the algorithm, I used learning rate of 0.95, number of episode of 10^5 and all other parameters are as stated in section 2.1.

For each episode of the learning process, I initialised the starting state randomly for exploration. Then the agent starts its journey from the state and for each step of the agent (i.e. from each state), it chooses an action which maximises the value for that state (i.e. policy). After the action is chosen (i.e. desired action), the agent takes a true action as the environment is stochastic. Then update the value function every time each true action is performed. After all episodes is done (or there is no value change in q function) we assume that the final value function contains the optimal values thus the optimal policy.

2.3 3

80%. Good
You mean p(Left | s9)=1
More discussion on gamma and p. E.g. as p drops below 0.25, it actually becomes better to pick the worst action, rather than the best action, because it actually has less chance of bringing you to your desired state than if you had picked another action.

At s_9, the optimal policy I obtained directs the agent to execute the action of Left. Therefore $p(a|s_9) = 1$ as the agent will always execute the action but there is 0.5 probability that the desired action will be performed. Taking the Left action at s_9 is sensible because the agent tries to get away from the penalty terminal state (i.e. s_{11}). I believe this result is derived from my personalised p and γ as the agent will take the desired action to get away from s_{11} with the highest feasible probability. Furthermore, the agent goes left since the.

2.4 4

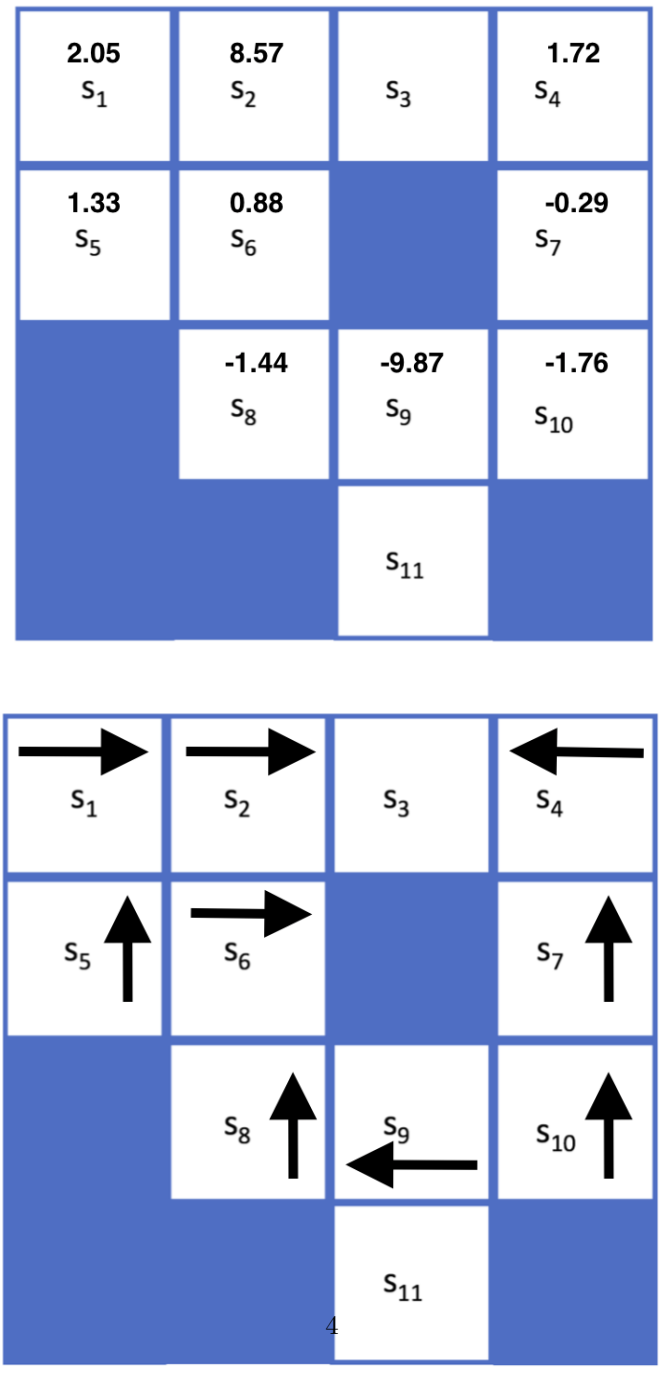
As shown in the figure 2, the optimal policy of mine directs the agent to RIGHT at s_6. Intuitively, this is not the optimal as human will choose to go UP to reach the rewarding state. This non-ideal action does not appear when I set my p-value to 1 (i.e. deterministic, always choose an optimal action from each state). Therefore, we can conclude that this abnormality comes from the environmental

20%
Little/no discussion of how gamma and p effect this. No discussion of what happens if you change these.

characteristic, the stochastic world.

Appendices

The result of Q Learning



Values should be
-0.274, 4.736, 0, 5.075,
-1.213, -0.411, -0.289,
-2.481, -19.194, -2.457 and 0
respectively.
Policy in s6 should be N, in
s9 should be E. 30%

Figure 2: Optimal value function and optimal policy obtained

Code for question 1.1

```
cid = [int(n) for n in '935517']

def CID(t):
    return cid[t-1]

def s(t):
    return (CID(t) + 1) % 3

def r(t):
    return CID(t) % 2

def trace(cid):
    return [(s(t), r(t)) for t in range(1, len(cid)+1)]

if __name__ == '__main__':
    # q1.1
    print(trace(cid))
```

Code for question 2.2

```
import random
import numpy as np

def get_transition_matrices():
    # Each transition matrix is 11x11
    # Resulting tm is 4x(11x11)

    # Transition for Left (0)
    TL = np.array([
        [1,0,0,0,0,0,0,0,0,0,0],
        [1,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,0,0,0,0,0,0,0,0],
        [0,0,0,0,1,0,0,0,0,0,0],
        [0,0,0,0,1,0,0,0,0,0,0],
        [0,0,0,0,0,0,1,0,0,0,0],
        [0,0,0,0,0,0,0,1,0,0,0],
        [0,0,0,0,0,0,0,1,0,0,0],
        [0,0,0,0,0,0,0,0,1,0,0],
        [0,0,0,0,0,0,0,0,0,0,0],
    ])

    # Transition for Down (1)
```

```
TD = np.array([
    [0,0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,0,0],
    [0,0,0,0,0,0,0,0,0,0,1],
    [0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,0,0,0],
])

# Transition for Right (2)
TR = np.array([
    [0,1,0,0,0,0,0,0,0,0,0],
    [0,0,1,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,0,0,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,0,0],
    [0,0,0,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,0,0,0],
])

# Transition for Up (3)
TU = np.array([
    [1,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,0,0,0,0,0,0,0],
    [1,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,0,0,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,0,0],
    [0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0],
])

return [TL, TD, TR, TU]
```

```
def get_true_action(p, action):
    action_space = list(range(4))
    if random.random() < p:
        return action
    else:
        action_space.remove(action)
        return random.choice(action_space)

def step(state, action, p, tms):
    # Evaluate true action from this stochastic world
    true_action = get_true_action(p, action)

    # Get corresponding transition matrix
    tm = tms[true_action]

    try:
        # Find next state when performing the true_action at current state
        next_state = list(tm[state]).index(1)
    except ValueError:
        print('current state: ', state)
        print('current action: ', true_action)
        print('current matrix: \n', tm)

    # Evaluate reward and its termination
    if next_state == 2:
        reward = 10
        done = True
    elif next_state == 10:
        reward = -100
        done = True
    else:
        reward = -1
        done = False

    return next_state, reward, done

def init_state():
    non_terminal_states = list(range(11))
    non_terminal_states.remove(2)
    non_terminal_states.remove(10)
    return random.choice(non_terminal_states)

def learn(Q, tms, alpha, gamma, p, nb_episodes):
    print('Running the Q-Learning Algorithm for {} times'.format(nb_episodes))

    for i in range(nb_episodes):
```

```
        state = init_state()    # Init initial state of the agent (exploration)
        done = False            # Is required to enter the loop

        # The Q-Table learning algorithm with discounted reward
        while not done:
            # Choose an action which will maximise the state-action value func
            action = np.argmax(Q[state, :])

            # Receive a feedback after taking a desired action with prob p
            new_state, reward, done = step(state, action, p, tms)

            # In stochastic world, we need not to rely on previous Q-Table
            # This can be done by adding the learning rate , alpha
            prev_Q = np.copy(Q)
            Q[state, action] = (
                    (1 - alpha) * Q[state, action]
                    + alpha * (reward + gamma * np.max(Q[new_state, :]))
                )

            if np.allclose(prev_Q, Q, 1e-15):
                return Q

            # Update state to the next one
            state = new_state

    return Q

if __name__ == '__main__':
    # Initialise Q-Table with zeros representing rewards
    nb_states = 11
    nb_actions = 4
    Q = np.zeros([nb_states, nb_actions])

    # Get transition matrices
    tms = get_transition_matrices()

    # Validate the transition matrices
    assert len(tms) == 4

    # Set learning rate, alpha
    alpha = .95

    # Set discount reward factor, gamma
    gamma = .35

    # Set probability of performing the desired action
```

```
p = .5

# Set number of iterations
nb_episodes = int(1e5)

# Run the q-learning algorithm
Q = learn(Q, tms, alpha, gamma, p, nb_episodes)

# Print the result
print('    Left         Down         Right        Up')
print(Q)
```