Laporan Tugas 3 Machine Learning : Q-Learning

Nama : Septian Dwi Indradi

Kelas : IF-39-10

NIM : 1301154164

1. Analisis Masalah

Diketahui sebuah grid world berukuran 10x10, dimana angka-angka dalam kotak menyatakan reward. Sebuah agent berada di posisi start (1,1) dan goal berada di posisi(10,10). Agent dapat melakukan empat aksi, yaitu N (North), S (South), W (West), dan E (East) yang menyatakan arah ke atas, bawah, kiri dan kanan.

In [1]:
```
import pandas as pd
import numpy as np
import time

reward_table = []
for data in open('DataTugasML3.txt'):
    reward_table.append(data.split())

reward_table = pd.DataFrame(pd.DataFrame(reward_table).astype(int))
reward_table
```

Out[1]:

	0	1	2	3	4	5	6	7	8	9
0	-1	-3	-5	-1	-3	-3	-5	-5	-1	100
1	-2	-1	-1	-4	-2	-5	-3	-5	-5	-5
2	-3	-4	-4	-1	-3	-5	-5	-4	-3	-5
3	-3	-5	-2	-5	-1	-4	-5	-1	-3	-4
4	-4	-3	-3	-2	-1	-1	-1	-4	-3	-4
5	-4	-2	-5	-2	-4	-5	-1	-2	-2	-4
6	-4	-3	-2	-3	-1	-3	-4	-3	-1	-3
7	-4	-2	-5	-4	-1	-4	-5	-5	-2	-4
8	-2	-1	-1	-4	-1	-3	-5	-1	-4	-1
9	-5	-3	-1	-2	-4	-3	-5	-2	-2	-2

Akan dibangun sebuah sistem Q-Learning untuk menemukan optimum policy sehingga agent yang berada di posisi Start (1,1) mampu menemukan goal yang berada di posisi (10,10) dengan mendapatkan total reward maksimum pada grid world.

2. Desain

Beberapa parameter yang digunakan pada sistem Q-Learning yang dibangun diantaranya :

1. α (Alpha) : Merupakan learning rate yang berpengaruh terhadap seberapa cepat sistem akan konvergen.
2. ɣ (Gamma) : Merupakan discount factor yang akan berpengaruh terhadap nilai reward pada step-step yang akan dilalui dan menentukan seberapa besar pengaruh reward di step-step yang akan datang. Semakin besar gamma maka future reward akan semakin diperhitungkan.
3. ϵ (Epsilon) : Disebut juga greedy policy merupakan parameter yang menentukan seberapa besar peluang agent akan melakukan eksplorasi dibanding eksploitasi. Semakin besar epsilon maka peluang eksploitasi semakin besar, semakin kecil epsilon maka peluang eksplorasi semakin besar. Epsilon juga digunakan untuk menghindari overfit.

Sistem Q-Learning yang dibangun memiliki parameter sebagai berikut :

In [2]:
```
ACTIONS = ['up','down','left','right']
N_STATES = 100
ALPHA = 0.5
GAMMA = 0.9
EPSILON = 0.4
```

Optimum policy akan didapatkan dari Q-Table yang akan terupdate selama dilakukan iterasi sebanyak n episode. Q-Table berukuran 100x4 dimana 100 merupakan jumlah state dan 4 merupakan jumlah action yang dapat dilakukan. Q-Table diinisialisasi menggunakan fungsi build_q_table(n_states,actions) berikut.

In [3]:
```
def build_q_table(n_states, actions) :
    table = pd.DataFrame(
        (np.zeros((n_states,len(actions)))),
        columns=actions,
    )
    index = []
    for i in range(10) :
        for j in range(10) :
            index.append(str(i)+','+str(j))

    table['state'] = pd.DataFrame(index)
    table.set_index('state',inplace=True)
    return table
```

Setiap episode, action diambil secara random atau mencari nilai Q maksimum dari next_state, jika menggunakan epsilon maka di random suatu angka dan akan dibandingkan dengan nilai epsilon, jika hasil random lebih besar maka action diambil secara random. Action dipilih menggunakan fungsi choose_action(state,q_table,testMode=False,EPSILON=0) berikut.

In [4]:
```
def choose_action(state, q_table,testMode=False,EPSILON=0) :
    state_actions = q_table.loc[state]
    if testMode :
        action = state_actions.idxmax()
    else:
        if (state_actions.all == 0) or (np.random.uniform() > EPSILON) :
            action = np.random.choice(ACTIONS)
        else:
            action = state_actions.idxmax()
    return action
```

Dibuat fungsi get_env_feedback untuk mendapatkan reward dan next_state berdasarkan current state dan action yang dipilih.

In [5]:
```
def get_env_feedback(state, action, reward_table) :
    state = state.split(',')
    state = [int(state[0]),int(state[1])]
    if action == 'up' :
        if state == [1,9] :
            return 'terminal',100
        elif state[0] == 0 :
            next_state = state
            reward = -100
        else:
            next_state = [state[0]-1,state[1]]
            reward = reward_table.at[next_state[0],next_state[1]]
    elif action == 'down' :
        if state[0] == 9 :
            next_state = state
            reward = -100
        else:
            next_state = [state[0]+1,state[1]]
            reward = reward_table.at[next_state[0],next_state[1]]
    elif action == 'left' :
        if state[1] == 0 :
            next_state = state
            reward = -100
        else:
            next_state = [state[0],state[1]-1]
            reward = reward_table.at[next_state[0],next_state[1]]
    else:
        if state == [0,8] :
            return 'terminal',100
        elif state[1] == 9 :
            next_state = state
            reward = -100
        else:
            next_state = [state[0],state[1]+1]
            reward = reward_table.at[next_state[0],next_state[1]]

    next_state = str(next_state[0]) + ',' + str(next_state[1])
    return next_state,reward
```

Fungsi getPath untuk mendapatkan optimum policy dari Q-Table yang sudah dibangun. Fungsi akan mengoutputkan action-action yang dilakukan dan total reward yang didapatkan.

In [6]:
```
def getPath(q_table) :
    step_counter = 0
    state = '9,0'
    is_terminated = False

    finalReward = 0.0
    steps = []

    while not is_terminated:
        action = choose_action(state, q_table,testMode=True)
        next_state, reward = get_env_feedback(state, action,reward_table)
        finalReward += reward
        steps.append(action)
        if next_state == 'terminal' :
            is_terminated = True
        state = next_state
        step_counter += 1

    print(steps)
    print('Total reward =',finalReward)
```

Fungsi q_learning merupakan algoritma utama dari sistem Q-Learning yang dibangun. Pada proses learning, setiap iterasi / episode agent akan berjalan sesuai action yang dipilih, Q-Table kemudian akan di update dan agent akan berhenti ketika telah mencapai goal.

In [7]:
```
def q_learning(q_table,MAX_EPISODES=100,EPSILON=0) :

    start = time.time()

    for episode in range(MAX_EPISODES):
        step_counter = 0
        state = '9,0'
        is_terminated = False

        finalReward = 0.0

        while not is_terminated:
            action = choose_action(state, q_table, testMode=False, EPSILON=EPSI
LON)
            next_state, reward = get_env_feedback(state, action,reward_table)
            finalReward += reward
            q_predict = q_table.loc[state,action]
            if next_state != 'terminal' :
                q_target = reward + GAMMA * q_table.loc[next_state].max()
            if next_state == 'terminal' :
                is_terminated = True
                q_target = reward
            q_table.loc[state,action] += ALPHA * (q_target - q_predict)
            state = next_state
            step_counter += 1

        if episode % 10 == 0 :
            print('\rBuilding Q-Table : %i of %i episode'%(episode,MAX_EPISODES
),end='')
    print('\rBuilding Q-Table : %i of %i episode'%(episode+1,MAX_EPISODES))
    print('Finished in %.4f seconds'%(time.time()-start))
```

3. Hasil Eksperimen

Eksperimen dilakukan dengan menginisialisasi Q-Table, kemudian melakukan learning sebanyak 100 episode dengan parameter-parameter yang telah ditentukan sebelumnya. Setelah dilakukan q-learning, didapatkan optimum policy yaitu :

['up', 'right', 'right', 'right', 'right', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'up', 'right', 'up', 'up', 'up', 'right']

Total reward = 65

In [8]:
```
q_table = build_q_table(N_STATES,ACTIONS)
q_learning(q_table,MAX_EPISODES=100,EPSILON=EPSILON)
getPath(q_table)
```

```
Building Q-Table : 100 of 100 episode
Finished in 4.3046 seconds
['up', 'right', 'right', 'right', 'right', 'up', 'up', 'up', 'up', 'right', 'r
ight', 'right', 'up', 'right', 'up', 'up', 'up', 'right']
Total reward = 65.0
```

Q-Table kemudian disimpan dalam file q-table.xlsx

In [9]:
```
writer = pd.ExcelWriter('q-table.xlsx')
q_table.to_excel(writer,'sheet1',index=False)
writer.save()
```