Machine Learning

Tugas 3 : Q-learning

Agung Nursatria Banyuwiguna

1301150073

IF-39-03

Program Studi S1 Teknik Informatika – Fakultas Informatika

Universitas Telkom

Jalan Telekomunikasi Terusan Buah Batu, Bandung

Indonesia

Problem Statement

Bangunlah 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 dalam Figure 1 berikut ini. Data pada Figure 1 dapat dilihat di file DataTugasML3.txt. Pada kasus ini, Agent hanya bisa melakukan empat aksi: N, E, S, dan W yang secara berurutan menyatakan North (ke atas), East (ke kanan), South (ke bawah), dan West (ke kiri). Anda boleh menggunakan skema apapun dalam mengimplementasikan sebuah episode.

10	-1	-3	-5	-1	-3	-3	-5	-5	-1	100
9	-2	-1	-1	-4	-2	-5	-3	-5	-5	-5
8	-3	-4	-4	-1	-3	-5	-5	-4	-3	-5
7	-3	-5	-2	-5	-1	-4	-5	-1	-3	-4
6	-4	-3	-3	-2	-1	-1	-1	-4	-3	-4
5	-4	-2	-5	-2	-4	-5	-1	-2	-2	-4
4	-4	-3	-2	-3	-1	-3	-4	-3	-1	-3
3	-4	-2	-5	-4	-1	-4	-5	-5	-2	-4
2	-2	-1	-1	-4	-1	-3	-5	-1	-4	-1
1	-5	-3	-1	-2	-4	-3	-5	-2	-2	-2
	1	2	3	4	5	6	7	8	9	10

Figure 1: Sebuah grid world ukuran 10 x 10, di mana angka-angka dalam kotak menyatakan reward. Agent berada di posisi Start (1,1) dan Goal di posisi (10,10)

Langkah Pengerjaan

- Learning

  The Q-Learning algorithm goes as follows:

  1. Set the gamma parameter, and environment rewards in matrix R.

  2. Initialize matrix Q to zero.

  3. For each episode:

     Select a random initial state.

Do While the goal state hasn't been reached.

- Select one among all possible actions for the current state.
- Using this possible action, consider going to the next state.
- Get maximum Q value for this next state based on all possible actions.
- Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
- Set the next state as the current state.

End Do

End For

- Running/Testing

Algorithm to utilize the Q matrix:

1. Set current state = initial state.
2. From current state, find the action with the highest Q value.
3. Set current state = next state.
4. Repeat Steps 2 and 3 until current state = goal state.

Screenshot Program

- Library / Dependencies:

```
import numpy as np
import random
```

- Function
  1. Fungsi **possible_direction** ini berguna untuk menentukan arah (*direction*) mana saja yang bisa diambil untuk berpindah tempat

```
def possible_direction(row,column):
    direction = ['N','E','S','W']
    if (row == 0):
        direction.remove('N')
    if (row == len(R)-1):
        direction.remove('S')
    if (column == 0):
        direction.remove('W')
    if (column == len(R)-1):
        direction.remove('E')
    return direction
```

2. Fungsi **nextState** berguna untuk berpindah tempat dari suatu state ke state lain

```
def nextState(row,column, direction):
    if(direction == 'N'):
        row = row -1
    if(direction == 'E'):
        column = column + 1
    if(direction == 'S'):
        row = row + 1
    if(direction == 'W'):
        column = column -1
    return [row,column]
```

3. Fungsi **possible_action_Q** berguna untuk mencari aksi yang dapat diambil berdasarkan Q

```
def possible_action_Q(Q,qRow,qColumn,direction):
    valueNextAction = []
    for d in direction:
        [nRow,nColumn] = nextState(qRow,qColumn,d)
        valueNextAction.append([d,Q[nRow,nColumn]])
    return valueNextAction
```

- Learning

```
##========== Learning
# Setting Gamma
Gamma = 0.9
# Initialize matrix R
R = np.loadtxt('Data Tugas 3 RL.txt')
# Initialize matrix Q to zero
Q = np.zeros((10,10), dtype=int)

for episode in range(1,1001):
    # initial state
    [row,column] = [9,0]
    current_state = R[row,column]

    while current_state != 100:
        # selecting one of possible direction
        direction = random.choice(possible_direction(row,column))
        [nextRow,nextColumn] = nextState(row,column,direction)

        # get possible action from next state
        valueNextAction = possible_action_Q(Q,nextRow,nextColumn,possible_direction(nextRow,nextColumn)

        # get maximum Q value for next state and insert to Q next state
        Q[nextRow,nextColumn] = R[nextRow,nextColumn] + (Gamma * max(valueNextAction, key=lambda x:x[1]

        # set next state as the current state
        [row,column] = [nextRow,nextColumn]
        current_state = R[row,column]
```

Hal yang dilakukan pada tahapan learning yaitu :

1. Setting gamma, dengan rentang nilai 0 <= gamma > 1. Semakin mendekati 0, maka semakin tidak bergantung future reward dan semakin bergantung pada immediate reward, dan sebaliknya. Disini penulis menggunakan gamma 0.9.
2. Me-*load* data matrix reward dari file txt ke dalam variable R.
3. Membuat matrix Q dengan ukuran 10x10.
4. Melakukan learning sebanyak 1000 episode, dengan 1 episode yaitu terjadinya pergerakan dari state awal ke goal state yang mana goal state ditunjuk berupa elemen pada matrix R dengan reward = 100. Yang dapat dilihat pada langkah nomor 5 – 9.
5. Memilih secara acak state selanjutnya dari state saat ini.
6. Mencari daftar aksi yang dapat dilakukan dari state selanjutnya.
7. Memilih aksi terbaik berdasarkan maximum Q dari aksi yang dapat dilakukan dari state selanjutnya.
8. Berpindah ke state selanjutnya.
9. Ulangi kembali dari langkah 5 sampai state saat ini = goal state.

- Running

```
##========== Running
# initial state
[row,column] = [9,0]
direction = []
current_state = R[row,column]
Totalscore = Q[row,column]
ListReward = [current_state]

while current_state != 100:
    # get possible action direction
    direction = possible_direction(row,column)
    listNextAction = possible_action_Q(Q,row,column,direction)

    # get best direction
    bestDirection = max(listNextAction, key=lambda x:x[1])[0]

    # set next state as the current state
    [nextRow,nextColumn] = nextState(row,column,bestDirection)
    [row,column] = [nextRow,nextColumn]
    current_state = R[row,column]
    Totalscore = Totalscore + Q[row,column]
    ListReward.append(current_state)

print('======================================')
print('Reward = ', sum(ListReward))
print('Number of Action = ', len(ListReward)-1)
print('Total Score = ', Totalscore)
print('======================================')
```

Hal yang dilakukan pada running yaitu :

1. Input posisi awal, dan jadikan state saat ini (current state), juga dapatkan score dan reward dari posisi tersebut
2. Lakukan looping hingga state saat ini = goal state yang bernilai 100
3. Dapatkan daftar state kemana saja dari posisi sekarang yang bisa dipilih
4. Pilih state terbaik berdasakan possible action Q terbaik untuk menjadi state selanjutnya
5. State selanjutnya dijadikan state saat ini
6. ulangi dari langkah 3 hingga state saat ini = goal state
7. keluarkan hasil running

Evaluasi Hasil Running Program

Berikut daftar perpindahan state yang dilakukan oleh program untuk mencapai goal state

	0	1	2	3	4	5	6	7	8	9
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

	0	1	2	3	4	5	6	7	8	9
0	179	200	226	257	287	323	363	409	461	514
1	160	181	203	227	256	285	323	363	409	457
2	141	158	178	203	227	252	286	324	365	406
3	123	137	158	177	203	226	256	291	325	361
4	108	125	143	163	184	206	230	257	289	320
5	95	110	124	144	161	180	206	230	258	284
6	82	96	111	126	143	159	181	204	231	252
7	71	84	94	110	127	139	157	179	205	222
8	66	76	86	97	113	122	139	161	180	198
9	54	65	76	85	97	106	122	142	160	176

```
========================================
Reward =  60.0
Number of Action =  18
Total Score =  4165
========================================
```

Dari hasil running program, didapat reward sebesar **60** dengan jumlah aksi berpindah tempat sebanyak **18x** dan total score Q dari start state ke goal state sebesar **4165**.

Referensi

http://mnemstudio.org/path-finding-q-learning-tutorial.htm