Laporan Pembelajaran Mesin
Semester Genap Tahun Akademik 2017-2018
Tugas 3: *Q - Learning*

Raden Muhammad Imam
1301154106
IF-39-08

April 26, 2018

Q - Learning merupakan salah satu metode pembelajaran yang ada di dalam pembelajaran mesin. Teknik ini tidak membutuhkan model lingkungan. Pada Q - *Learning* diberikan hukum alam yaitu jika melakukan sebuah aksi maka akan mendapatkan reward atau punishment. Komponen dalam *Q - Learning* ada empat yaitu *player, state, actions,* dan *reward*. Ketika seorang player menentukan akan kearah mana dia bergerak. Probabilitasnya dapat di hitung dengan menggunakan rumus

$$Q(state, action) = R(state, action) + Gamma * Max[Q(next\ state, all\ actions)]$$

Setelah di dapat Q yang paling baik, maka aksi dari *learner* selanjutnya akan mengikuti hasil sebelumnya. Pada rumus terdapat gamma yang berupa sebuah konstanta agar tidak *stuck* di suatu titik saja. Inti dari *Q - Learning* adalah agar *player* dapat menuju ke *final state* dengan skor tertinggi. *Player* berhenti jika skor yang di dapatkan telah konvergen alias skornya sudah tidak berubah lagi.

Masalah yang ada pada tugas ini, membangun aplikasi yang dapat belajar sendiri. Aplikasi tersebut merupakan sebuah *grid* dengan ukuran 10x10 dimana inisial *state* berada pada pojok kiri bawah dan *final state* nya berada pada pojok kanan atas atau dapat dilihat pada gambar berikut

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

Design Program pada masalah ini terdiri dari dua buah file, yaitu world.py dan learner.py

1. world.py merupakan file python untuk membuat *grid-grid. Source code* dapat dilihat sebagai berikut:

```
from tkinter import *
master = Tk()
```

```
import pandas as pd

triangle_size  = 0.1
cell_score_min = -10
cell_score_max = 10
Width   = 50
( x , y ) = ( 12 , 12 )
actions   = [ "up" , "down" , "left" , "right" ]

board       = Canvas ( master, width = x * Width , height = y * Width )
player      = ( 1 , y - 2 )
score       = 0
restart     = False
data        = pd.read_csv ( "DataTugasML3.txt" , delimiter = '\t' )
totalReward = []
hasilReward = []
asd         = []
for i in range ( data.shape [ 0 ] ):
    reward  = []
    rewardd = []
    for j in range(data.shape[1]):
        reward.append  ( float ( data [ str ( i ) ][ j : j + 1 ] ) / 100000
)
        rewardd.append ( float ( data [ str ( i ) ][ j : j + 1 ] ) )
    totalReward.append ( reward )
    hasilReward.append ( rewardd )

def walkReward ( totalReward , x , y ):
    if ( x >= 0 and x <= 9 ) and ( y >= 0 and y <= 9 ):
        return totalReward [ x ] [ y ]
    else:
        return -999
walls       =
[(0,0),(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(0,9),(0,10),(0,11),(
1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(10,0),(11,0),(11,1),(1
1,2),(11,3),(11,4),(11,5),(11,7),(11,6),(11,8),(11,9),(11,10),(11,11),(10,11
),(9,11),(8,11),(7,11),(6,11),(5,11),(4,11),(3,11),(2,11),(1,11)]
specials    = [ ( 10 , 1 , "green" , 100 ) ]
cell_scores = {}



def create_triangle ( i , j , action ):
    if action == actions [ 0 ]:
        return board.create_polygon ( ( i + 0.5 - triangle_size ) * Width ,
( j + triangle_size ) * Width ,
                                      ( i + 0.5 + triangle_size ) * Width , (
j + triangle_size ) * Width ,
                                      ( i + 0.5 ) * Width , j * Width,
                                      fill = "white" , width=1)
    elif action == actions [ 1 ]:
        return board.create_polygon ( ( i + 0.5 - triangle_size ) * Width ,
( j + 1 - triangle_size ) * Width ,
                                      ( i + 0.5 + triangle_size ) * Width , (
j + 1 - triangle_size ) * Width ,
                                      ( i + 0.5 ) * Width , ( j + 1 ) * Width
,
                                      fill = "white" , width = 1 )
    elif action == actions [ 2 ]:
        return board.create_polygon ( ( i + triangle_size ) * Width , ( j +
0.5 - triangle_size ) * Width ,
                                      ( i + triangle_size ) * Width , ( j +
0.5 + triangle_size ) * Width ,
                                      i * Width , ( j + 0.5 ) * Width ,
                                      fill = "white" , width = 1 )
    elif action == actions [ 3 ]:
        return board.create_polygon ( ( i + 1 - triangle_size ) * Width , (
j + 0.5 - triangle_size ) * Width ,
```

```
                                    ( i + 1 - triangle_size ) * Width , ( j
+ 0.5 + triangle_size ) * Width ,
                                    ( i + 1 ) * Width , ( j + 0.5 ) * Width
,
                                    fill = "white" , width = 1 )

def render_grid():
    global specials , walls , Width , x , y , player
    for i in range ( x ):
        for j in range ( y ):
            board.create_rectangle ( i * Width , j * Width , ( i + 1 ) *
Width , ( j + 1 ) * Width , fill = "white" , width = 1 )
            temp = {}
            for action in actions:
                temp [ action ] = create_triangle ( i , j , action )
            cell_scores [ ( i , j ) ] = temp
    for ( i , j , c , w ) in specials:
        board.create_rectangle ( i * Width , j * Width , ( i + 1 ) * Width ,
( j + 1 ) * Width , fill = c , width = 1 )
    for ( i , j ) in walls:
        board.create_rectangle ( i * Width , j * Width , ( i + 1 ) * Width ,
( j + 1 ) * Width , fill = "black" , width = 1 )

render_grid()


def set_cell_score ( state , action , val ):
    global cell_score_min , cell_score_max
    triangle = cell_scores [ state ] [ action ]
    green_dec = int ( min ( 255 , max ( 0 , ( val - cell_score_min ) * 255.0
/ ( cell_score_max - cell_score_min ) ) ) )
    green = hex ( green_dec ) [ 2 : ]
    red = hex ( 255 - green_dec ) [ 2 : ]
    if len ( red ) == 1:
        red += "0"
    if len ( green ) == 1:
        green += "0"
    color = "#" + red + green + "00"
    board.itemconfigure ( triangle , fill = color )


def findScore ( data , hasilReward ):
    sum = 0
    for i in range ( len ( data ) ):
        if ( data [ i ] [ 0 ] >= 0 and data [ i ] [ 0 ] <= 9 ) and ( data [
i ] [ 1 ] >= 0 and data [ i ] [ 1 ] <= 9 ):
            sum += hasilReward [ data [ i ] [ 0 ] ] [ data [ i ] [ 1 ] ]
    return sum

def try_move ( dx , dy ):
    global player , x , y , score , totalReward , me, restart , asd ,
hasilReward
    if restart == True:
        restart_game()
    if ( player [ 0 ] + dx != -1 or player [ 0 ] + dx != 10 or player [ 1 ]
+ dy != -1 or player [ 1 ] + dy != -1 ):
        new_x = player [ 0 ] + dx
        new_y = player [ 1 ] + dy
        asd.append ( [ new_x - 1 , new_y - 1 ] )
        if ( new_x == 9 and new_y == 8 ):
            score += -0.03
        score += walkReward ( totalReward , new_x -1 , new_y -1 )
        if ( new_x > -1 ) and ( new_x < x ) and ( new_y > -1 ) and ( new_y <
y ) and not ( ( new_x , new_y ) in walls ):
            board.coords ( me , new_x * Width + Width * 2 / 10 , new_y *
Width + Width * 2 / 10, new_x * Width + Width * 8 / 10 , new_y * Width +
Width * 8 / 10 )
            player = ( new_x , new_y )
```

```
        for ( i , j , c , w ) in specials:
            if new_x == i and new_y == j:
                score += walkReward ( totalReward , i - 1 , j - 1 ) #find
score nya do pake index dijumlahin
                print ("Success! score: ", findScore( asd , hasilReward ) )
                restart = True
                asd     = []
                return
        #print "score: ", score


def call_up  ( event  ):
    try_move ( 0 , -1 )


def call_down ( event ):
    try_move  ( 0 , 1 )


def call_left ( event  ):
    try_move  ( -1 , 0 )


def call_right ( event ):
    try_move   ( 1 , 0 )


def restart_game():
    global player , score , me , restart
    player  = ( 1 , y - 2 )
    score   = 1
    restart = False
    board.coords ( me , player [ 0 ] * Width + Width * 2 / 10 , player [ 1 ]
* Width + Width * 2 / 10 , player [ 0 ] * Width + Width * 8 / 10 , player [
1 ] * Width + Width * 8 / 10 )

def has_restarted():
    return restart

master.bind ( "<Up>"    , call_up   )
master.bind ( "<Down>"  , call_down )
master.bind ( "<Right>" , call_right)
master.bind ( "<Left>"  , call_left )

me = board.create_rectangle ( player [ 0 ] * Width + Width * 2 / 10 , player
[ 1 ] * Width + Width * 2 / 10 ,
                              player [ 0 ] * Width + Width * 8 / 10 , player [
1 ] * Width + Width * 8 / 10 , fill = "orange" , width = 1 , tag = "me" )

board.grid ( row = 0 , column = 0 )


def start_game():
    master.mainloop()
```

2. learner.py merupakan file python untuk pembelajaran *state-state* selanjutnya. *Source code* dapat dilihat sebagai berikut:

```
import World
import threading
import time

#1
discount = 1
actions = World.actions
states = []
Q = {}
```

```
for i in range ( World.x ):
    for j in range ( World.y ):
        states.append ( ( i , j ) )

for state in states:
    temp = {}
    for action in actions:
        temp [ action ] = 0.1
        World.set_cell_score ( state , action , temp [ action ] )
    Q[state] = temp

for ( i , j , c , w ) in World.specials:
    for action in actions:
        Q [ ( i , j ) ] [ action ] = w
        World.set_cell_score ( ( i , j ) , action , 100 )


def do_action ( action ):
    s =  World.player
    r = -World.score
    if action == actions [ 0 ]:
        World.try_move ( 0 , -1 )
    elif action == actions [ 1 ]:
        World.try_move ( 0 , 1 )
    elif action == actions [ 2 ]:
        World.try_move ( -1 , 0 )
    elif action == actions [ 3 ]:
        World.try_move ( 1 , 0 )
    else:
        return
    s2 = World.player
    r += World.score
    return s , action , r , s2


def max_Q ( s ):
    val = None
    act = None
    for a , q in Q [ s ].items():
        if val is None or ( q > val ):
            val = q
            act = a
    return act , val


def inc_Q ( s , a , alpha , inc ):
    Q [ s ] [ a ] *= 1 - alpha
    Q [ s ] [ a ] += alpha * inc
    World.set_cell_score ( s , a , Q [ s ] [ a ] )


def run():
    global discount
    time.sleep ( 1 )
    #0.1  = 59
    alpha = 0.1
    t     = 1
    while True:
        # Pick the right action
        s = World.player
        max_act , max_val  = max_Q(s)
        ( s , a , r , s2 ) = do_action ( max_act )

        # Update Q
        max_act , max_val = max_Q ( s2 )
        inc_Q ( s , a , alpha , r + discount * max_val )

        # Check if the game has restarted
```

```
        t += 1.0
        if World.has_restarted():
            World.restart_game()
            time.sleep ( 0.01 )
            t = 1.0

        # Update the learning rate
        alpha = pow ( t , -0.1 )

        # MODIFY THIS SLEEP IF THE GAME IS GOING TOO FAST.
        time.sleep ( 0.1 )


t = threading.Thread ( target = run )
t.daemon = True
t.start()
World.start_game()
```

Evaluasi dari *Q – Learning* di dapatkan *score* sebesar 59.0. Skor setiap *episode* dapat dilihat sebagai berikut:

Success! score: -824.0
Success! score: -836.0
Success! score: -572.0
Success! score: -601.0
Success! score: -601.0
Success! score: -354.0
Success! score: -176.0
Success! score: -207.0
Success! score: -241.0
Success! score: 15.0
Success! score: -130.0
Success! score: -135.0
Success! score: -7.0
Success! score: -20.0
Success! score: -44.0
Success! score: -71.0
Success! score: -86.0
Success! score: 31.0
Success! score: -63.0
Success! score: 39.0
Success! score: -26.0
Success! score: -5.0
Success! score: 36.0
Success! score: -1.0
Success! score: -5.0
Success! score: 19.0
Success! score: -9.0
Success! score: 50.0
Success! score: 46.0
Success! score: 11.0
Success! score: -42.0
Success! score: 47.0
Success! score: 35.0

```
Success! score:  3.0
Success! score:  33.0
Success! score:  49.0
Success! score:  -4.0
Success! score:  22.0
Success! score:  39.0
Success! score:  56.0
Success! score:  51.0
Success! score:  43.0
Success! score:  39.0
Success! score:  54.0
Success! score:  48.0
Success! score:  44.0
Success! score:  58.0
Success! score:  58.0
Success! score:  55.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
Success! score:  59.0
```

Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0
Success! score: 59.0

Hasil running pada program ini dapat dilihat pada:
https://www.youtube.com/channel/UCvXpk4grhnHFuiEPBfBcvtg