D_case

Bongseokkim

2021-02-10

차 례

introduction : Finding Shortest path using TD agent

In this case, I am trying to solve a path planning problem using TD agent.

Let's say there are 100 states. one of the way to find the optimal path is to find all the cases in trial and error, which will be about $100!$ it is impossible to compute 100! even if we use all the computer in the world. To solve the real world problem, I would like to learn the agent in a short time to find the optimal path even if it is not strictly optimal

Problem discription

State : set of X,Y coordiantes {$S_0, S_1, \ldots S_{10}$}, each coordinates are randomly created

Action : agent can do three action at each state, Select and move one of three paths that can go from each node. {go fisrt node, go second node, go third node}

$P^a_{ss'}$: transition probability with certain action in state is deterministic. It is set randomly and will be explained in detail in matrix form below

reward: Euclidean distance from s to s'

Goal : find the shortest path state 0 to 8 (just randomly selected)

Preparation

```
import numpy as np
import pandas as pd
```

making X,Y coordinate

The coordinates of each 11 state were randomly created.

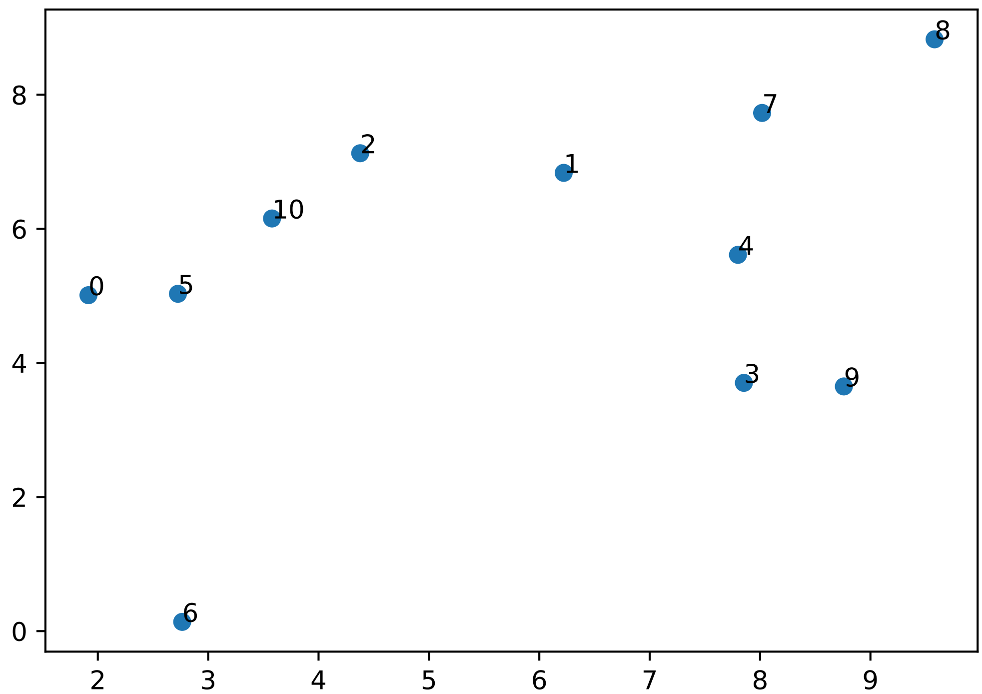
```
np.random.seed(1234)
coordinate= 10*np.random.rand(2,11)
data=pd.DataFrame(coordinate.T, columns=['X','Y'])
data
```

```
##            X         Y
## 0   1.915195  5.009951
## 1   6.221088  6.834629
## 2   4.377277  7.127020
## 3   7.853586  3.702508
## 4   7.799758  5.611962
## 5   2.725926  5.030832
## 6   2.764643  0.137684
## 7   8.018722  7.728266
## 8   9.581394  8.826412
## 9   8.759326  3.648860
## 10  3.578173  6.153962
```

```
import matplotlib.pyplot as plt
states = np.arange(0,11).astype(str)
y = data['Y']
x = data['X']
n = states

fig, ax = plt.subplots()
ax.scatter(x, y)

for i, txt in enumerate(n):
    ax.annotate(n[i], (x[i], y[i]))
plt.show()
```



Computing Euclidean distance

```
distance = np.zeros(shape=(11,11))
for i in range(len(data)):
    for j in range(len(data)):
        distance[i,j]=np.sqrt(np.sum(data.iloc[i]**2+data.iloc[j]**2))
distance = pd.DataFrame(distance,index=states, columns= states)

distance
```

```
##             0          1          2  ...          8          9         10
## 0    7.585194  10.685582   9.935923  ...  14.088159  10.899888   8.913032
## 1   10.685582  13.070126  12.464713  ...  15.972562  13.245908  11.665704
## 2    9.935923  12.464713  11.828354  ...  15.481073  12.648911  10.983148
## 3   10.205633  12.680752  12.055801  ...  15.655543  12.861856  11.227731
## 4   11.004450  13.332083  12.739125  ...  16.187618  13.504455  11.958466
## 5    7.842673  10.869868  10.133850  ...  14.228443  11.080610   9.133154
## 6    6.035709   9.647606   8.810061  ...  13.318065   9.884441   7.637851
## 7   12.360970  14.472045  13.927705  ...  17.138689  14.630994  13.217434
## 8   14.088159  15.972562  15.481073  ...  18.423281  16.116719  14.845310
## 9   10.899888  13.245908  12.648911  ...  16.116719  13.419387  11.862316
## 10   8.913032  11.665704  10.983148  ...  14.845310  11.862316  10.067231
```

```
##
## [11 rows x 11 columns]
```

transition prob

transition prob matrix is created randomly, It can be adjusted later.

```
P_go_first= pd.DataFrame(np.matrix([[0,1,0,0,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,0,0,0,0,0,0,1,0,0,0],
                                    [0,0,0,0,0,0,0,0,1,0,0],
                                    [0,0,0,0,0,0,0,0,0,1,0],
                                    [0,0,0,0,0,0,0,0,0,0,1],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                                        ]),index= states, columns=states )

P_go_second= pd.DataFrame(np.matrix([[0,0,0,0,0,0,0,0,0,1,0],
                                    [0,0,0,0,0,0,0,0,1,0,0],
                                    [0,0,0,0,0,0,0,1,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                    [1,0,0,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,0,0,0,0,1,0],
                                                        ]),index= states, columns=states )

P_go_third= pd.DataFrame(np.matrix([[0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
```

```
                                    [0,0,0,0,0,0,0,0,0,0,1],
                                    [0,0,0,0,0,0,0,0,0,0,1]
                                                      ]),index= states, columns=states )

P_go_third
```

```
##     0  1  2  3  4  5  6  7  8  9  10
## 0   0  0  0  1  0  0  0  0  0  0   0
## 1   0  0  0  0  0  1  0  0  0  0   0
## 2   0  0  1  0  0  0  0  0  0  0   0
## 3   0  0  0  0  0  0  1  0  0  0   0
## 4   0  1  0  0  0  0  0  0  0  0   0
## 5   0  0  0  1  0  0  0  0  0  0   0
## 6   0  0  1  0  0  0  0  0  0  0   0
## 7   0  0  0  0  0  1  0  0  0  0   0
## 8   0  0  0  0  1  0  0  0  0  0   0
## 9   0  0  0  0  0  0  0  0  0  0   1
## 10  0  0  0  0  0  0  0  0  0  0   1
```

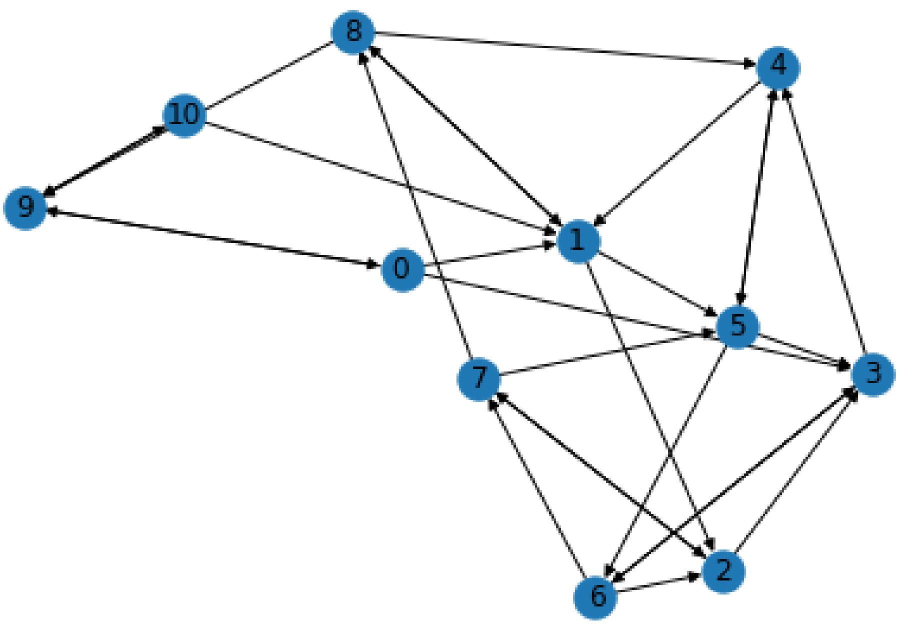


그림 1: nodes

R_s_a

get R_s_a as the distance between the states that arrive when agent act in each state

```python
import numpy as np
import pandas as pd
R_s_a = pd.DataFrame(np.c_[np.repeat( 0, len( states ) ), np.repeat( 0, len( states ) ),np.repeat( 0, len( st

R_s_a=R_s_a.astype('float')
```

```python
for i in range(11):
    R_s_a['first'][i]=-distance.iloc[i,P_go_first.iloc[i][P_go_first.iloc[i].values==1][0]].astype(float)

for i in range(11):
    R_s_a['second'][i]=-distance.iloc[i,P_go_second.iloc[i][P_go_second.iloc[i].values==1][0]].astype(float)

for i in range(11):
    R_s_a['third'][i]=-distance.iloc[i,P_go_third.iloc[i][P_go_third.iloc[i].values==1][0]].astype(float)

R_s_a
```

```
##        first     second      third
## 0  -10.685582 -10.685582 -10.685582
## 1  -13.070126 -13.070126 -13.070126
## 2  -12.464713 -12.464713 -12.464713
## 3  -12.680752 -12.680752 -12.680752
## 4  -13.332083 -13.332083 -13.332083
## 5  -10.869868 -10.869868 -10.869868
## 6   -9.647606  -9.647606  -9.647606
## 7  -14.472045 -14.472045 -14.472045
## 8  -15.972562 -15.972562 -15.972562
## 9  -13.245908 -13.245908 -13.245908
## 10 -11.665704 -11.665704 -11.665704
```

policy

```python
pi_50 = pd.DataFrame( np.c_[np.repeat( 1/3, len( states ) ), np.repeat( 1/3, len( states ) ),np.repeat( 1/3,
                      columns = ['first', 'second','third'] )

pi_50
```

```
##       first    second     third
## 0   0.333333  0.333333  0.333333
## 1   0.333333  0.333333  0.333333
```

```
## 2   0.333333  0.333333  0.333333
## 3   0.333333  0.333333  0.333333
## 4   0.333333  0.333333  0.333333
## 5   0.333333  0.333333  0.333333
## 6   0.333333  0.333333  0.333333
## 7   0.333333  0.333333  0.333333
## 8   0.333333  0.333333  0.333333
## 9   0.333333  0.333333  0.333333
## 10  0.333333  0.333333  0.333333
```

simul_step

```
def simul_step(pi, s_now, P_go_first,P_go_second, P_go_third, R_s_a):
    if np.random.uniform() < pi_50.loc[s_now].cumsum()[0] :
        a_now ='first'
        P = P_go_first

    elif  pi_50.loc[s_now].cumsum()[0]< np.random.uniform() < pi_50.loc[s_now].cumsum()[1]:
        a_now ='second'
        P = P_go_second

    else :
        a_now = 'third'
        P = P_go_third

    r_now = R_s_a.loc[s_now , a_now]
    s_next = states[np.argmin( P.loc[s_now].cumsum() < np.random.uniform() )]

    if np.random.uniform() < pi_50.loc[s_now].cumsum()[0] :
        a_next ='first'

    elif  pi_50.loc[s_now].cumsum()[0]< np.random.uniform() < pi_50.loc[s_now].cumsum()[1]:
        a_next ='second'



    else :
        a_next = 'third'

    sarsa = [s_now, a_now, r_now, s_next, a_next]

    return sarsa
```

```
sample_step = simul_step(pi_50, '0', P_go_first,P_go_second, P_go_third, R_s_a )

print( sample_step )
```

```
## ['0', 'first', -10.685582447534157, '1', 'second']
```

test simul step

```
for i in range(10):
    test_state = str(i)
    sample_step = simul_step(pi_50, test_state, P_go_first,P_go_second, P_go_third, R_s_a )
    print( sample_step )
```

```
## ['0', 'third', -10.685582447534157, '3', 'third']
## ['1', 'third', -13.070125529303853, '5', 'third']
## ['2', 'first', -12.464712830709807, '3', 'third']
## ['3', 'first', -12.680751685899999, '4', 'first']
## ['4', 'second', -13.332082835266162, '5', 'first']
## ['5', 'third', -10.869868014054747, '3', 'first']
## ['6', 'second', -9.647605720253912, '3', 'third']
## ['7', 'third', -14.472045062589983, '5', 'third']
## ['8', 'second', -15.97256209643203, '1', 'third']
## ['9', 'third', -13.245907551439055, '10', 'first']
```

q_s_a

```
q_s_a_init= pd.DataFrame( np.c_[np.repeat(0, len( states ) ), np.repeat(0, len( states ) ),np.repeat(0, len(
                      columns = ['first', 'second','third'] ).astype(float)

q_s_a_init
```

```
##     first  second  third
## 0     0.0     0.0    0.0
## 1     0.0     0.0    0.0
## 2     0.0     0.0    0.0
## 3     0.0     0.0    0.0
## 4     0.0     0.0    0.0
## 5     0.0     0.0    0.0
## 6     0.0     0.0    0.0
## 7     0.0     0.0    0.0
## 8     0.0     0.0    0.0
```

```
## 9      0.0      0.0    0.0
## 10     0.0      0.0    0.0
```

TD contol

```
def pol_eval_TD(sample_step, q_s_a, alpha):
    q_s_a_copy= q_s_a.copy()
    s = sample_step[0]
    a = sample_step[1]
    r = sample_step[2]
    s_next = sample_step[3]
    a_next = sample_step[4]

    q_s_a_copy.loc[s,a] +=alpha*(r+q_s_a_copy.loc[s_next, a_next]-q_s_a_copy.loc[s,a])

    return q_s_a_copy


q_s_a=pol_eval_TD(sample_step, q_s_a_init, alpha = 0.1)
q_s_a
```

```
##      first  second     third
## 0      0.0     0.0  0.000000
## 1      0.0     0.0  0.000000
## 2      0.0     0.0  0.000000
## 3      0.0     0.0  0.000000
## 4      0.0     0.0  0.000000
## 5      0.0     0.0  0.000000
## 6      0.0     0.0  0.000000
## 7      0.0     0.0  0.000000
## 8      0.0     0.0  0.000000
## 9      0.0     0.0 -1.324591
## 10     0.0     0.0  0.000000
```

```
def pol_imp(pi, q_s_a, epsilon): # epsilon = exploration_rate
    pi_copy =pi.copy()
    for i in range(pi.shape[0]):
        # exploitation
        if np.random.uniform() > epsilon:
            pi_copy.iloc[i] = 0
            pi_copy.iloc[i, np.argmax(q_s_a.iloc[i,])] = 1
```

```
        else:
            # exploration
            pi_copy.iloc[i] = 1/q_s_a.shape[1]

    return pi_copy

pol_imp(pi_50, q_s_a, epsilon=0)
```

```
##      first  second  third
## 0     1.0     0.0    0.0
## 1     1.0     0.0    0.0
## 2     1.0     0.0    0.0
## 3     1.0     0.0    0.0
## 4     1.0     0.0    0.0
## 5     1.0     0.0    0.0
## 6     1.0     0.0    0.0
## 7     1.0     0.0    0.0
## 8     1.0     0.0    0.0
## 9     1.0     0.0    0.0
## 10    1.0     0.0    0.0
```

TD iteration

goal is find shortest path S_0 to S_8

```
import time
num_ep = 100
beg_time =time.time()
q_s_a = q_s_a_init
pi=pi_50
exploration_rate = 1


for epi_i in range(1,num_ep) :
    s_now="0"
    while s_now != "8":
        sample_step = simul_step(pi_50, s_now, P_go_first,P_go_second, P_go_third, R_s_a )
        q_s_a = pol_eval_TD(sample_step, q_s_a, alpha = 1/epi_i)
        pi = pol_imp(pi, q_s_a, epsilon= exploration_rate)
```

```
        s_now = sample_step[3]
        exploration_rate *=0.9995

end_time =time.time()

print("Time difference of {} sec".format(end_time- beg_time))
```

```
## Time difference of 9.77786111831665 sec
```

```
print(pi.T)
```

```
##            0         1    2    3    4    5         6    7    8    9        10
## first   0.0  0.333333  0.0  1.0  0.0  0.0  0.333333  1.0  1.0  1.0  0.333333
## second  1.0  0.333333  1.0  0.0  1.0  1.0  0.333333  0.0  0.0  0.0  0.333333
## third   0.0  0.333333  0.0  0.0  0.0  0.0  0.333333  0.0  0.0  0.0  0.333333
```

```
print(q_s_a.T)
```

```
##                  0          1          2  ...    8         9         10
## first  -28.522001 -45.831120 -56.359056  ...  0.0 -7.952490 -15.344732
## second  -9.824207 -13.070126 -42.553108  ...  0.0 -8.958847  -6.815551
## third  -41.068752 -43.766223 -64.930868  ...  0.0 -9.461634 -10.617875
##
## [3 rows x 11 columns]
```

TD iteration 2

This time, I started with State 0 and looked for the best route to visit all states. aka 한붓그리기

```
import time
num_ep = 1000
beg_time =time.time()
q_s_a = q_s_a_init
pi=pi_50
exploration_rate = 1
```

```
import time
num_ep = 1000
beg_time =time.time()
q_s_a = q_s_a_init
```

```
pi=pi_50
exploration_rate = 1


for epi_i in range(1,num_ep) :
    s_now="0"
    history = [int(s_now)]
    while not np.array_equal(history, states.astype(int)):
        sample_step = simul_step(pi_50, s_now, P_go_first,P_go_second, P_go_third, R_s_a )
        q_s_a = pol_eval_TD(sample_step, q_s_a, alpha = 1/epi_i)
        pi = pol_imp(pi, q_s_a, epsilon= exploration_rate)
        s_now = sample_step[3]
        history.append(int(s_now))
        my_set = set(history) #집합set으로 변환
        history = list(my_set) #list로 변환
        exploration_rate *=0.9995
        #print(history)


end_time =time.time()

print("Time difference of {} sec".format(end_time- beg_time))
```

```
## Time difference of 439.9066162109375 sec
```

```
print(pi.T)
```

```
##           0    1    2    3    4    5    6    7    8    9   10
## first   0.0  0.0  0.0  1.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0
## second  1.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0
## third   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
print(q_s_a.T)
```

```
##                  0           1           2  ...           8          9          10
## first   -166.848426 -234.213716 -241.224002  ...  -78.832439 -67.549757 -144.633757
## second   -47.990571 -156.676012 -219.220331  ... -190.007449 -98.287234  -39.586080
## third   -191.485243 -232.123415 -246.140014  ... -222.407756 -73.269189  -60.129308
##
## [3 rows x 11 columns]
```

Limitation

1. **Calculations at about 11 states also take longer time than expected (about 10~20 miniutes)** It may be because my computer is not good, and the code is not perfect
2. **It is a randomly generated coordinate, not a coordinate of the real world, and a process set to a task.** if the process I have done so far is valid, it would be fun to solve it using the coordinates and distances of the real world.

```
"Done "
```

```
## [1] "Done "
```