电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Logbook

From: 8/January/2021   To: 23/4/2021

Month	**List the main activities** (only few words per activity)	**Interaction with the supervisor**			**Any other form of supervisory interaction** (second supervisor, industry, fellows etc.)
		Number of meetings	Mode of meeting (face- to-face, online e.g., Skype, WeChat etc.)	Number of emails exchanged	
Jan	1.5G UDN model 2.Python Code 3. may expand, if needed	3	Zoom online	10	Email with Attai Abubakar 15
Feb	1.reinfocement learning 2. 3. may expand, if needed	3	Zoom online	10	Email with Attai Abubakar 15
Mar	1. Clustering based method 2. 3. may expand, if needed	4	Zoom online	7	Email with Attai Abubakar 20

```
#Epsilon Greedy Method

import csv
import np as np
import pandas as pd
import numpy as np
import math
import random
from numpy import genfromtxt
np.set_printoptions(threshold=np.inf)

class Q_learning:
    def __init__(self,num_sc=12, epsilon=0.1):
        self.J = np.array((1009,4096))
        self.R = 0;
        self.num_sc = num_sc;
        self.TimeSlots = 1008;
        self.R_list = []

        self.N_arg = np.zeros(4096)
        self.T_opt = np.array(1009)

        self.Q_e_greedy = np.zeros(pow(2, num_sc))



    def training(self):
        self.generate_table()

        self.Q_e_greedy = self.J[1,:]
    # useless
    def generate_table(self):
        self.J = genfromtxt(r'C:\Users\yanglianrui\Desktop\J_value.csv',
delimiter=',');

    def best_result(self):
        self.T_opt = np.max(self.J, 1);



    def get_rewards(self, arg_time_slot_j, arg_sc_i):
        return self.J[arg_time_slot_j, arg_sc_i + 1];
    # get_rewards from J, where j is the argument of timeslot, i is one of 4096
combinations


    def e_greedy(self,epsilon):
        self.best_result()
        self.training()
        for j in range(1, self.TimeSlots + 1):#j是time slot角标 # j is time slot
            if np.random.random() > epsilon:# 如果比epsilon大，则选取已知最好的sc组合 # if
ramdom is larger that epsilon, choose the best choice the we current know
                if j == 1:
                    arg_sc_i = np.random.randint(1,pow(2,self.num_sc))
                else:
                    arg_sc_i = np.argmax(self.Q_e_greedy) + 1
            else:# 如果比epsilon小，则随机选择sc组合
             arg_sc_i = np.random.randint(1,pow(2,self.num_sc))
            reward = self.get_rewards(j, arg_sc_i);
            print("e-greedy")
            print(arg_sc_i)
            print(reward)
            a = reward / self.T_opt[j]
            print(a)
            self.N_arg[arg_sc_i - 1] += int(1);

                #self.R += (reward - self.R) / j
            self.R = reward
                # for A in range(1, int(self.N_arg[arg_sc_i - 1])): # 计算Q(n+1) AAA是
一个表，sum(i=1:n) a(1-a)^(n-i)R(i) n是该arm已经被选过的次数
             # self.AAA[A] = alph * pow((1 - alph), (self.N_arg[arg_sc_i - 1] - A)) *
self.R

            if self.Q_e_greedy[arg_sc_i - 1] == -float('inf'):
                self.Q_e_greedy[arg_sc_i - 1] = 2 * self.Q_e_greedy[arg_sc_i - 1] / 3
                #print("Q")
                # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
            elif self.R == -float('inf'):
                self.Q_e_greedy[arg_sc_i - 1] = 2 * self.Q_e_greedy[arg_sc_i - 1] /3
                #print("Q")
                # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
            else:
                self.Q_e_greedy[arg_sc_i - 1] = self.Q_e_greedy[arg_sc_i - 1] +
(np.array(self.R) - self.Q_e_greedy[arg_sc_i - 1]) / self.N_arg[arg_sc_i - 1]
                #print("Q")
                    # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
                #self.Q_e_greedy[arg_sc_i - 1] = pow((1 - alph), self.N_arg[arg_sc_i -
1]) * self.Q_e_greedy[arg_sc_i - 1] + np.sum(self.AAA)
            #print(self.R)

    def printQ(self): # print the Q table of e-greedy
        print(self.Q_e_greedy)

    def random(self): # randomly choose arms
        self.best_result()
        for j in range(1, self.TimeSlots + 1):
            arg_sc_i = np.random.randint(1, pow(2, self.num_sc))
            reward= self.get_rewards(j, arg_sc_i)
            print("random")
            print(reward)
            print(reward / self.T_opt[j])
    # def generate_argmaxTable(self,N_times ,Q_table ):
    #     argmaxTable = self.num_sc
    # def UCB(self):
```

```
    #     for t in range(1, self.TimeSlots + 1):
    #
    #         At = np.argmax()



A = Q_learning()
A.generate_table()
A.e_greedy(0.5)
A.random()
A.printQ()
```

```
#Upper confidence method
import csv
import np as np
import pandas as pd
import numpy as np
import math
import random
from numpy import genfromtxt

np.set_printoptions(threshold=np.inf)


class Q_learning:
    def __init__(self):
        self.bandits = pow(2, 12)
        self.timeslots = 1008
        self.J = np.array((1009, self.bandits + 1))
        self.Q_UCB = np.zeros(self.bandits + 1)
        self.lr = 0.99 #
        self.N = np.zeros(self.bandits + 1)
        self.ATt = np.zeros((1010, 4097))
        self.num_sc = 12
        self.c = 2 #
        self.parameter_a = 2 / 5
        self.R_training = np.zeros(self.bandits + 1)
        self.Q_training = np.zeros(self.bandits + 1)
        self.sum_R = 0

    def best_result(self):
        self.T_opt = np.max(self.J, 1);

    def generate_table(self):
        self.J = genfromtxt(r'C:\Users\yanglianrui\Desktop\J_value.csv',
delimiter=',');

    def training(self):  # success
        self.generate_table()
        for t in range(1, 504):
            reward = self.J[t, :]
            # print(reward)
            for act in range(1, self.bandits + 1):
                if reward[act] == -float('inf'):
                    self.Q_training[act] = self.parameter_a * self.Q_training[act]

                else:
                    self.Q_training[act] = self.Q_training[act] + self.lr *
(reward[act] - self.Q_training[act])
        self.N[1:] = 504
        self.ATt[504, :] = self.Q_training + self.c * math.sqrt(math.log(504) / 504)
        print(self.Q_training)  # see Q_training table
        max_value = np.max(self.Q_training)
        argument_max_value = np.argmax(self.Q_training)
        print(max_value, argument_max_value)  # see the max value of Q_training table
and its argument

    def UCB(self):
        self.generate_table()
        self.Q_UCB = self.Q_training
        self.best_result()
        for t in range(505, self.timeslots + 1):

            action = np.argmax(self.ATt[t, :])
            R = self.J[t, action]
            self.N[action] += 1
            if R == -float('inf'):
                self.Q_UCB[action] = self.parameter_a * self.Q_UCB[action]
            else:
                self.Q_UCB[action] = self.Q_UCB[action] + self.lr * (R -
self.Q_UCB[action])
            for act in range(1, self.bandits + 1):
                self.ATt[t + 1, act] = self.Q_UCB[act] + self.c * math.sqrt(math.log(t)
/ self.N[act])
            print("time,action,reward,       Percentage Compared with Reward of the
Best choice(PCRB)")
            print(t, action, R,R / self.T_opt[t])
            if R == -float('inf'): #
                self.sum_R = self.sum_R
            else:
                self.sum_R += R / self.T_opt[t]
        print(self.sum_R)


A = Q_learning()
A.training()
A.UCB()
# CONCLUSION:
# The performance is not always good. In fact, compared with the best reward at each
timeslot
# the reward of UCB (PCRB) is around 50% - 80%. Rarely, -inf occurs. Sometimes, the
PCRB is lower
# than 15%. The total performance depends on the value of self.c and self.parameter_a.
# We can say that this algorithm can save energy while maximize revenue.
```

```
#Clustering method
import csv
import np as np
import pandas as pd
import numpy as np
import math
import random

from matplotlib import ticker
from numpy import genfromtxt
import matplotlib.pyplot as plt
np.set_printoptions(threshold=np.inf)

class Q_learning:
    def __init__(self,num_sc=12, epsilon=0.1):
        self.J = np.array((1009,4096))
        self.R = 0;
        self.num_sc = num_sc;
        self.TimeSlots = 1008;
        self.R_list = []

        self.N_arg = np.zeros(4096)
        self.T_opt = np.array(1009)

        self.Q_e_greedy = np.zeros(pow(2, num_sc))



    def training(self):
        self.generate_table()

        self.Q_e_greedy = self.J[1,:]
    # useless
    def generate_table(self):
        self.J = genfromtxt(r'C:\Users\yanglianrui\Desktop\J_value.csv',
delimiter=',');

    def best_result(self):
        self.T_opt = np.max(self.J, 1);



    def get_rewards(self, arg_time_slot_j, arg_sc_i):
        return self.J[arg_time_slot_j, arg_sc_i + 1];
    # get_rewards from J，where j is the argument of timeslot，i is one of 4096
combinations


    def e_greedy(self,epsilon):
        self.best_result()
        self.training()
        aaa = np.zeros(self.TimeSlots + 1)
        for j in range(1, self.TimeSlots + 1):#j 是time slot角标 # j is time slot
            if np.random.random() > epsilon:# 如果比epsilon大，则选取已知最好的sc组合 # if
ramdom is larger that epsilon, choose the best choice the we current know
                if j == 1:
                    arg_sc_i = np.random.randint(1,pow(2,self.num_sc))
                else:
                    arg_sc_i = np.argmax(self.Q_e_greedy) + 1
            else:# 如果比 epsilon 小，则随机选择 sc 组合
             arg_sc_i = np.random.randint(1,pow(2,self.num_sc))
            reward = self.get_rewards(j, arg_sc_i);
            #print("e-greedy")
            #print(arg_sc_i)
            #print(reward)
            aaa[j] = reward / self.T_opt[j]


            self.N_arg[arg_sc_i - 1] += int(1);

                #self.R += (reward - self.R) / j
            self.R = reward
                # for A in range(1, int(self.N_arg[arg_sc_i - 1])): # 计算 Q(n+1) AAA 是
一个表，sum(i=1:n) a(1-a)^(n-i)R(i) n 是该 arm 已经被选过的次数
             # self.AAA[A] = alph * pow((1 - alph), (self.N_arg[arg_sc_i - 1] - A)) *
self.R

            if self.Q_e_greedy[arg_sc_i - 1] == -float('inf'):
                self.Q_e_greedy[arg_sc_i - 1] = 2 * self.Q_e_greedy[arg_sc_i - 1] / 3
                #print("Q")
                # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
            elif self.R == -float('inf'):
                self.Q_e_greedy[arg_sc_i - 1] = 2 * self.Q_e_greedy[arg_sc_i - 1] /3
                #print("Q")
                # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
            else:
                self.Q_e_greedy[arg_sc_i - 1] = self.Q_e_greedy[arg_sc_i - 1] +
(np.array(self.R) - self.Q_e_greedy[arg_sc_i - 1]) / self.N_arg[arg_sc_i - 1]
                #print("Q")
                    # print(arg_sc_i)
                #print(self.Q_e_greedy[arg_sc_i - 1])
                #self.Q_e_greedy[arg_sc_i - 1] = pow((1 - alph), self.N_arg[arg_sc_i -
1]) * self.Q_e_greedy[arg_sc_i - 1] + np.sum(self.AAA)
            #print(self.R)
        print("egreedy")
        print(aaa)
        record = np.arange(1, self.TimeSlots + 2)
        plt.plot(record,aaa, alpha = 1)
        plt.xlabel("Time slots / 10 min")
        plt.ylabel("Revenue / Best Revenue")
        plt.gca().yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1))
        plt.show()

    #def printQ(self): # print the Q table of e-greedy
        #print(self.Q_e_greedy)


    def random(self): # randomly choose arms
        self.best_result()
        aaa = np.zeros(self.TimeSlots + 1)
        for j in range(1, self.TimeSlots + 1):
            arg_sc_i = np.random.randint(1, pow(2, self.num_sc))
            reward= self.get_rewards(j, arg_sc_i)

            #print(reward)
            #print(reward / self.T_opt[j])
            aaa[j] = reward / self.T_opt[j]
        print("random")
        print(aaa)
        record = np.arange(1, self.TimeSlots + 2)
        plt.plot(record,aaa )
        plt.xlabel("Time slots / 10 min")
        plt.ylabel("Revenue / Best Revenue")
        plt.gca().yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1))
        plt.show()
    # def generate_argmaxTable(self,N_times ,Q_table ):
    #     argmaxTable = self.num_sc
    # def UCB(self):
    #     for t in range(1, self.TimeSlots + 1):
    #
    #         At = np.argmax()



A = Q_learning()
A.generate_table()
A.e_greedy(0.5)
A.random()
#A.printQ()
```

```
# 5G Ultra Dense Network Model with 12 small BSs and 1 Macro BS


# All print functions in this file is for test. You can remove them as you wish.

# Unlike Matlab, index of array in Python starts from 0, which is inconvenient. So, I add

# a zero column and a zero row in each matrix so that the index of any useful value starts from 1, which

# is the same as matlab


# Next point, because I did not find a measure to generate x matrix here. A file that contains value of x is used.



import csv

import np as np

import pandas as pd

import numpy as np

import math

from numpy import genfromtxt


np.set_printoptions(threshold=np.inf)


num_sc = 12;

num_time_slot = 1008;

R_insert_col = np.zeros(14, dtype=float);

R_insert_row = np.zeros(1008, dtype=float);

# Load milan_data set as Primary traffic in numpy

# R_load=csv.reader(open(r'C:\Users\yangl\Desktop\milan_data.csv', 'r'));

R_load = genfromtxt(r'C:\Users\yanglianrui\Desktop\milan_data.csv', delimiter=',');


# Secondary traffic S

R1 = np.array(R_load);

RS1 = np.array(R_load);

# This part is to guarantee that the corner mark of each element in python is the same as matlab. Start from 1
```

```
R2 = np.c_[R_insert_row.T, R1];  # add zeros to column 0
R = np.row_stack((R_insert_col, R2));  # add zeros to row 0
RS2 = np.c_[R_insert_row.T, RS1];
RS = np.row_stack((R_insert_col, RS2));

S_S = 1 - RS[:, :];  # S is 2D please modify to S=1-RS[:,:,:] if S is 3D
S = np.delete(S_S, 1, axis=1);  # remove col 1 in SS to get S   axis = 1 means col; 0
means row.

# backup values
S_bkp = np.array(S);
R_bkp = np.array(R);

# power model parameters, ma macrocell  mi microcell pi picocell fe femocell
# rh remote radio head
p_mao = 130;
k_ma = 4.7;
p_matx = 20;
p_mio = 56;
k_mi = 2.6;
p_mitx = 6.3;
P_mis = 39.0;
p_pio = 6.8;
k_pi = 4.0;
p_pitx = 0.13;
P_pis = 4.3;
p_feo = 4.8;
k_fe = 8.0;
p_fetx = 0.05;
P_fes = 2.9;
p_rho = 84;
k_rh = 2.8;
p_rhtx = 20;
```

```
P_rhs = 56;

# additional parameters
Ntx_ma = 1;
Ntx_sc = 1;
C_RB = 0.5;
N_RB_rh = 75;  # 15MHz
N_RB_mi = 50;  # 10MHz
N_RB_pi = 25;  # 5MHz
N_RB_fe = 15;  # 3MHz

x_load = genfromtxt(r'C:\Users\yanglianrui\Desktop\binary_array_x.csv', delimiter=',');
x = np.array(x_load);

# initialize values
P_ON = np.zeros((num_time_slot + 1, 1), dtype=float);
S_opt = np.zeros((num_time_slot + 1, 1), dtype=float);
T_opt = np.zeros((num_time_slot + 1, 1), dtype=float);
P_t = np.zeros((num_time_slot + 1, pow(2, num_sc) + 1), dtype=float);
Rev_t = np.zeros((num_time_slot + 1, pow(2, num_sc) + 1), dtype=float);
Rev_sc = np.zeros((2, 13));
P_sc = np.zeros((2, 13));
shabi = np.zeros((num_time_slot +1, (pow(2, num_sc) + 1)));

# TOTAL POWER CONSUMPTION WHEN ALL SCs ARE ON
P_sc_1 = np.zeros((1, 13), dtype=float);
for k in range(1, num_time_slot + 1):
    P_ma_1 = np.array(Ntx_ma * (p_mao + k_ma * R_bkp[k, 1] * p_matx));
    # print("P_ma_1");
    # print(P_ma_1);
    for n in range(1, num_sc + 1):
        # for remote radio head (RRH)
        if n <= round(num_sc / 4):
            P_sc_1[:, n] = np.array(Ntx_sc * (p_rho + k_rh * R_bkp[k, n + 1] *
p_rhtx));
        # for micro cell(mi)
        elif (n > round(num_sc / 4) and n <= round(2 * num_sc / 4)):
            P_sc_1[:, n] = np.array(Ntx_sc * (p_mio + k_mi * R_bkp[k, n + 1] *
p_mitx));
        # for pico cell(pi)
        elif (n > round((2 * num_sc) / 4) and n <= round(3 * num_sc / 4)):
            P_sc_1[:, n] = np.array(Ntx_sc * (p_pio + k_pi * R_bkp[k, n + 1] *
p_pitx));
        # for femto cell(fe)
        elif (n > round((3 * num_sc) / 4) and n <= round(4 * num_sc / 4)):
            P_sc_1[:, n] = np.array(Ntx_sc * (p_feo + k_fe * R_bkp[k, n + 1] *
p_fetx));

        P_ON[k, :] = np.array(P_ma_1 + np.sum(P_sc_1));
# for remote radio head (RRH)



for j in range(1, num_time_slot + 1):
    l_max = 1;
    # maximum normilized capacity of the macro cell is set to 1, as the traffoc load of
all cells are btw 0 and 1
    # back up values
    R2 = np.array(R[j, 2:5]);
    R3 = np.array(R[j, 5:8]);
    R4 = np.array(R[j, 8:11]);
    R5 = np.array(R[j, 11:14]);

    for i in range(1, pow(2, num_sc) + 1):
        # Checking to see that the maximum normlized capacity of the macro
        # cell is not exceeded

        # Please notice that some value of l_ma is different from
        # those in matlab
        R2x = np.array(1 - x[i, 1:4]);

        R3x = np.array(1 - x[i, 4:7]);

        R4x = np.array(1 - x[i, 7:10]);

        R5x = np.array(1 - x[i, 10:13]);

        l_ma = np.array(R[j, 1] + np.sum(R2.dot(R2x)) * 0.75 + (np.sum(R3.dot(R3x))) * 0.5 + (np.sum(R4.dot(R4x))) * 0.25 + (np.sum(R5.dot(R5x))) * 0.15);


        shabi[j][i] = l_ma;

        if l_ma > l_max:

            P_t[j, i] = float('inf');  # float('inf') means infinity in python

            Rev_t[j, i] = -float('inf');

            continue;

        else:

            # update traffic load of small cells after traffic offloading

            S_bkpx = np.array(1 - x[i, 1:]);

            R_bkpx = np.array(x[i, 1:]);

            S_S_bkp1 = np.array(S_bkp[j, 1:]);

            R_R_Bkp1 = np.array(R_bkp[j, 2:]);

            S[j, 1:] = np.array(S_S_bkp1 * (S_bkpx));

            R[j, 2:] = np.array(R_R_Bkp1 * (R_bkpx));


            P_ma = Ntx_ma * (p_mao + k_ma * l_ma * p_matx);

            for sc_idx in range(1, num_sc + 1):

                # Rev_sc(:, sc_idx) =  S(j, sc_idx) * C_RB * N_RB;


                # RRH POWER CONSUMPTION AND LEASING REVENUE

                if sc_idx <= round(num_sc / 4):

                    S_Rev_sc1 = np.array(S[j, sc_idx]);

                    Rev_sc[1:, sc_idx] = np.array(S_Rev_sc1 * C_RB * N_RB_rh);

                    if R[j, sc_idx + 1] == 0:

                        P_sc[1:, sc_idx] = Ntx_sc * P_rhs;

                    else:

                        S_P_sc = np.array(R[j, sc_idx + 1]);
                        P_sc[1:, sc_idx] = np.array(Ntx_sc * (p_rho + k_rh * S_P_sc *
p_rhtx));

                if (sc_idx > round(num_sc / 4) and sc_idx <= round(2 * num_sc / 4)):
                    S_Rev_sc2= np.array(S[j, sc_idx]);
                    Rev_sc[1:, sc_idx] = np.array(S_Rev_sc2 * C_RB * N_RB_mi);
                    if R[j, sc_idx + 1] == 0:
                        P_sc[1:, sc_idx] = Ntx_sc * P_mis;
                    else:
                        R_P_sc1=np.array(R[j, sc_idx + 1]);
                        P_sc[1:, sc_idx] = np.array(Ntx_sc * (p_mio + k_mi * R_P_sc1 *
p_mitx));

                if (sc_idx > round(2 * num_sc / 4) and sc_idx <= round(3 * num_sc /
4)):
                    S_Rev_sc3 = np.array(S[j, sc_idx]);
                    Rev_sc[1:, sc_idx] = np.array(S_Rev_sc3 * C_RB * N_RB_pi);
                    if R[j, sc_idx + 1] == 0:
                        P_sc[1:, sc_idx] = Ntx_sc * P_pis;
                    else:
                        R_P_sc2 = np.array(R[j, sc_idx + 1]);
                        P_sc[1:, sc_idx] = np.array(Ntx_sc * (p_pio + k_pi * R_P_sc2 *
p_pitx));

                if (sc_idx > round(3 * num_sc / 4) and sc_idx <= round(4 * num_sc /
4)):
                    S_Rev_sc4 = np.array(S[j, sc_idx])
                    Rev_sc[1:, sc_idx] = np.array( S_Rev_sc4 * C_RB * N_RB_fe);
                    if R[j, sc_idx + 1] == 0:
                        P_sc[1:, sc_idx] = Ntx_sc * P_fes;
                    else:
                        R_P_sc3 = np.array(R[j, sc_idx + 1]);
                        P_sc[1:, sc_idx] = np.array(Ntx_sc * (p_feo + k_fe * R_P_sc3 *
p_fetx));
            P_t1 = np.sum(P_sc, axis=1)
            # print("P_t1");

            # print(P_t.shape);


            P_t[j, i] = np.array(P_ma + P_t1[1]);


            # print("Rev_sc");

            # print(Rev_sc);


            Rev_t1 = np.sum(Rev_sc, axis=1)

            Rev_t[j, i] = np.array(Rev_t1[1]);

P_save = P_ON - P_t;


# total revenue per time slot for each switching combination

J = P_save + Rev_t;

# print("P_save");

# print(P_save);

# print("J");

# print(J);

J[:, 0] = 0

T_opt = np.max(J, 1);

# print(T_opt)

# print(T_opt.shape)

# Please notice that the value of S_opt here is

# different from that in matlab

S_opt = np.argmax(J, axis=1)

print(S_opt)

#

T = np.array(T_opt);

S_opt_bin = x[S_opt, :];  # optimal switching pattern that gives maximum revenue in
binary

# S_opt_dec = bi2de[S_opt_bin]; # optimal switching pattern that gives maximum revenus
in decimal

# Rev_Max_dataset = [R, S_bkp, S_opt_bin, S_opt_dec, T]; # saving the results
```

电子科技大学
格拉斯哥学院
Glasgow College, UESTC