STA4001 Course Project Report

Introduction

In the first question, I first used Matlab to generate the estimate of value function $v(x) = E[\sum_1^\infty \beta^n g(X_n)|X_0 = x]$ based on Monte Carlo Method. The initial state is (0, 0, 0, 0, 0, 0). The result found the estimate of value function and its 95% confidence interval with respect to (0, 0, 0, 0, 0, 0).

In the second question, using similar way to get the Monte Carlo estimate and 95% confidence interval first. Then generate K states at specific time, which will be used as training set and test set. Then modify demo code to read the data I got to generate the estimate $\widetilde{v(x)}$ based on Neural Network. The conclusion from graph is obvious. Neural Network 's estimate generally is more accurate and efficient than the only using Monte Carlo method.

Project Set up & solution

1. For question 1, starting from the initial state (0, 0, 0, 0, 0, 0), we need to generate a Markov Chain and use Monte Carlo Method to compute its value function, i.e. $V(x) = E(\sum_1^\infty \beta^n g(X_n)|X_0 = x)$. The following are my results.

   Case 1:

   k=20; N=100

   The result is:

d	17
episode	100
epsilon	0.8139
j	20
k	20
m	0
M	[]
profit	6.2000
T	20000
v	-1.1438
V	1x100 double
vmean	-4.8496

Fig 1-1

From above, the mean value function's estimator is -4.8496 (Starting from initial sate (0,0,0,0,0,0). The 95% confidence interval is: (-5.6635, -4.0357)

Case2:

k=200; N=100

episode	100
epsilon	0.7842
j	200
k	200
m	0
M	[]
profit	3.7000
T	200000
v	-2.6495
V	1x100 double
vmean	-5.5658

Fig 1-2

From above, the mean value function's estimator of state (0,0,0,0,0,0) is -5.5658, the 95% confidence interval is: (-6.35, -4.7816).

Case 3

k=2000; N=100

d	19
episode	100
epsilon	0.8095
j	2000
k	2000
m	0
M	[]
profit	4.3000
T	2000000
v	-6.4405
V	1x100 double
vmean	-5.1702

Fig 1-3

From above, the mean value function's estimator of state (0,0,0,0,0,0) is -5.1702, the 95% confidence interval is: (-5.9797, -4.3607)

Case 4

k=20; N=1000

count	1000
d	15
episode	1000
epsilon	0.2558
j	20
k	20
m	0
M	[]
profit	6
T	20000
v	0.8689
V	1x1000 double
vmean	-4.9093

Fig 1-4

From above, the mean value function's estimator of state (0,0,0,0,0,0) is -4.9093, the 95% confidence interval is (-5.1651, -4.6535).

Case 5

K=200; N=1000

count	1000
d	29
episode	1000
epsilon	0.2526
j	200
k	200
m	0
M	[]
profit	15.5000
T	200000
v	0.9721
V	1x1000 double
vmean	-4.8637

Fig1-5

For case 5, the mean value function of state (0,0,0,0,0,0) is -4.8637. The 95% confidence interval is (-5.1163, -4.6111)

Case 6

K=2000; N=1000

count	1000
d	16
episode	1000
epsilon	0.2486
j	2000
k	2000
m	0
M	[]
profit	7
T	2000000
v	-0.7172
V	1x1000 double
vmean	-4.6517

Fig 1-6

For case 6, the mean value function of state (0,0,0,0,0,0) is -4.6517, the 95% confidence interval is (-4.9003, -4.4031).

Note: The code for Monte Carlo simulation implemented will be submitted as document for reference. The 6 cases generally have exactly the same structure, except the parameter episodes (N) and K.

2. Neural Network's simulation of mean value function

(a) From question, we know the initial state has been modified as (10, 20, 30, 40, 50, 80). In this report, I first used Monte Carlo Method to generate the estimator and 95% confidence interval for K=20, K=200, K=2000. And generate 3 different data set as training set & test set for $\widetilde{v(x)}$. In order to make the result for estimator $\widetilde{v(x)}$ more accurate. I used episodes=1000 for this problem, which generally makes the result more representative. (Because with more repeated times, the unbiased estimator will be generally closer to its true value.

Note: For all dataset, I post it in the reference material, because for K=2000, the dataset is too large which is difficult to show that here.

For this question, I set the training set is 80% randomly generated from the dataset, the test set is the rest 20%.

Case 1

K=20;

count	1000
d	16
episode	1000
epsilon	0.3014
j	20
k	20
m	0
M	[ ]
profit	-1.1000
T	20000
v	113.6126
V	1x1000 double
vmean	109.7557

Fig 2-1

From above, $\widetilde{v(x)}$=109.7557, the 95% confidence interval is (109.4543, 110.0571)

The data points of K=20 is:

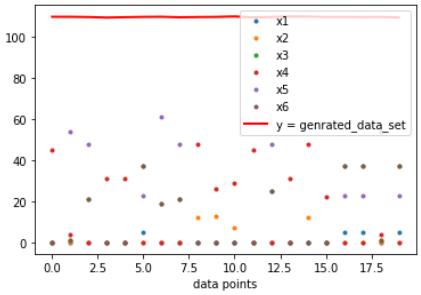

Fig2-2

Use Neural Network, the approximated mean squared error compared with the original set is approximately 0.01473. Thus, $\widetilde{v(x)}$= 109.6343

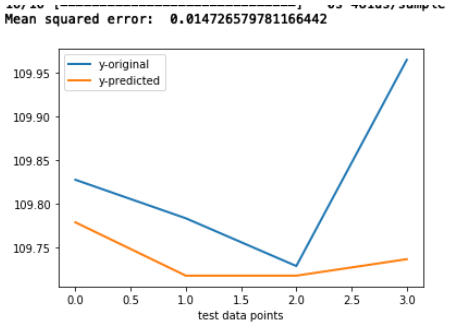

Fig2-3 (For detailed information, check "K=20.csv")

Case 2

For K=200, Use monte Carlo Method, I got:

d	15
episode	1000
epsilon	0.3065
j	200
k	200
m	0
M	[ ]
profit	−0.9000
T	200000
v	110.1772
V	1x1000 double
vmean	110.0982

Fig 2-4

For the case K=200, $\widehat{v(x)}$=110.0982, the 95% confidence interval is (109.7917,110.4047). Data points of K=200 is:

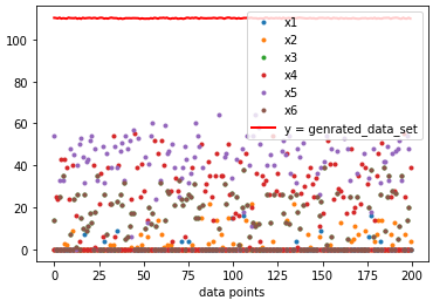

Fig2-5 (For detailed information, check "K=200.csv")

After training 80% of them & using 20% to make test, the approximated mean squared error is: 0.03084. Thus, $\widehat{v(x)}$=110.806.

Fig2-6

Case 3

For K=2000, Use Monte Carlo Method, I got:

count	1000
d	24
episode	1000
epsilon	0.3033
j	2000
k	2000
m	0
M	[ ]
profit	5.1000
T	2000000
v	110.2371
V	1x1000 double
vmean	110.2133

Fig 2-7

From above, for K=2000, $\widehat{v(x)}$=110.2133. The 95% confidence interval is (109.91, 110.5166). Then generate the data set, I got:

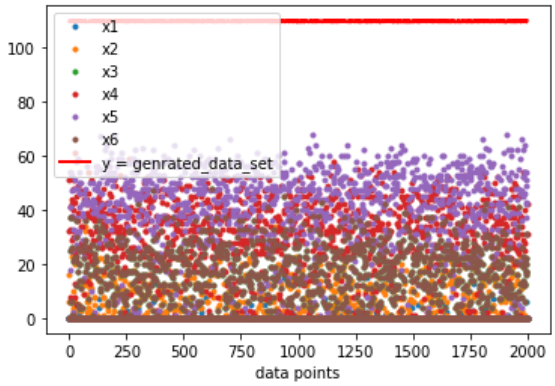

Fig 2-6 (For detailed information, check reference "K=2000.csv")

Randomly choose 1600 set as the training set to generate the parameter θ, then use 400 set as the test set to make prediction, I got:

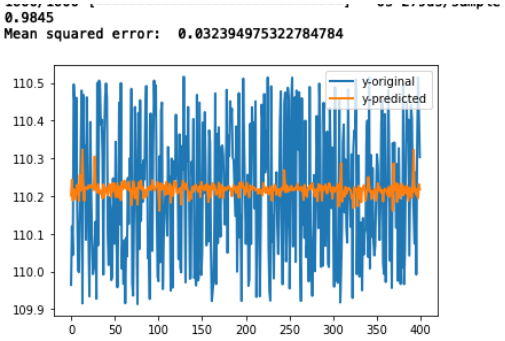

Fig 2-7

From the figure, the mean squared error is approximately 0.02944. Thus, $\widetilde{v(x)}$=110.2453.

(b) For question b, I first generated the test set with size 50. Then use different data set from question (a) (That is K=20; K=200; K=2000) to train the neural network. Then put the test set in to get the estimated value and mean squared error.

Note: The code for this problem is in reference, its name is "STA4001_auxiliary.py".

50test

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	21	0	0	11	12	5	0	0	0	0	0	0	0	0	0	0	0	12	0	0	9	0	0	0	0	0	0	0	0	17	0	0	5	0	2	5	0	0	0	6	0	0	0	0	0
0	23	15	5	0	30	7	0	0	0	0	18	0	26	7	9	0	36	7	0	23	27	0	21	8	0	46	17	0	0	0	0	0	20	0	0	0	0	0	0	21	0	15	0	20	0	26	15	16
40	0	0	0	44	0	0	32	33	52	54	0	0	0	0	0	32	0	0	21	0	0	47	0	0	22	0	0	45	35	27	6	30	0	42	43	29	12	29	28	44	0	29	0	53	0	0	0	0
0	34	46	57	0	35	62	0	0	0	0	45	50	42	48	62	0	39	42	0	51	25	0	43	34	0	22	61	0	0	0	34	0	44	0	0	0	13	0	0	0	50	0	41	0	53	17	47	56	47
40	0	0	0	25	0	0	47	45	33	36	0	0	0	0	58	0	0	69	0	13	31	0	3	59	0	0	45	55	63	16	60	0	31	47	61	47	61	60	41	0	61	0	31	0	42	0	0	0

Fig2-8 (for detailed information, check the reference material "50test.csv")

We first run this Markov Chain using Monte Carlo Method to get the real value. In this case, I also use episodes=1000 to make the result more accurate. The result is:

count	1000
d	13
episode	1000
epsilon	0.2949
j	50
k	50
m	0
M	[ ]
profit	2.5000
T	50000
v	109.5492
V	1x1000 double
vmean	110.1061

Fig 2-9

From above, $\widehat{v(x)}$=110.1061, the 95% confidence interval is (109.8112, 110.401).

Case 1, Training set is "K=20.csv":

```
Mean squared error:  0.2713588880879868
```

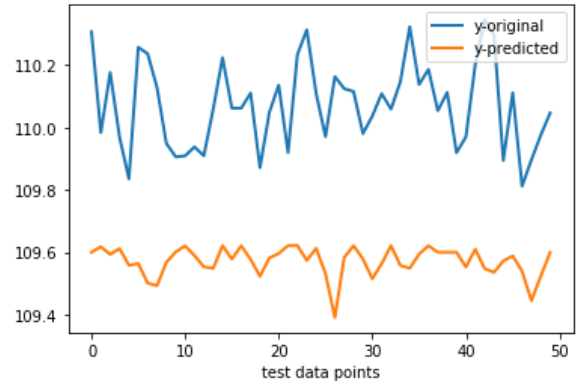

Fig 2-10

Use command "np.mean(y_from_nn)", I got $\widetilde{v(x)}$= 109.5717, the approximated mean value error is 0.27136.

Case 2, Training set is "K=200.csv":

```
Mean squared error:  0.025281892627452677
```

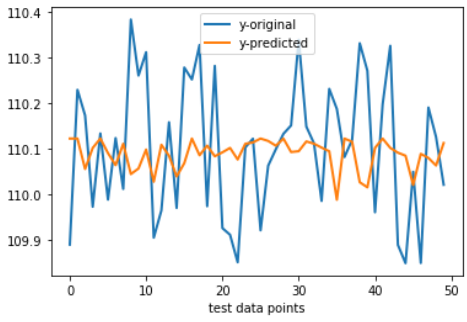

Fig 2-11

Use the same way from case 1, $\widetilde{v(x)}$=110.0883, the approximated mean squared error is 0.02528.

Case 3, Training set is "K=2000.csv":

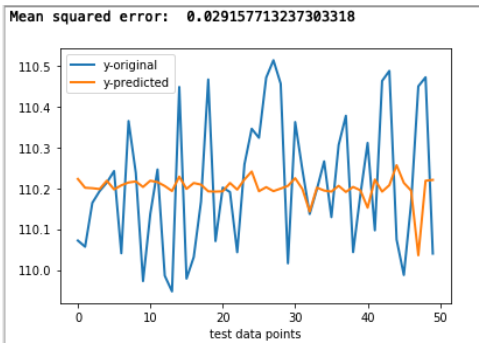

Fig 2-12

Use the command "np.mean(y_from_nn_)", we can get $\widetilde{v(x)}$=110.2012, the approximated mean square error is 0.02916.

3.

From the above experiment, I got the minimized mean value square error directly

For "K=20", $\widehat{mse}$ = 0.2173588880879868

For "K=200", $\widehat{mse}$ = 0.030840726854449402

For "K=2000", $\widehat{mse}$ = 0.029157713237303318

Conclusion

1.Compared with two different ways of simulation, meanly Monte Carlo simulation and Neural Network estimate. I concluded from the graph that Neural Network's estimate is generally more accurate than using Monte Carlo only. During the experiment, I also found when simulate long-run Markov Chains. The time used by Monte Carlo Method is longer than using Neural Network based on TensorFlow.

2. Starting with different initial conditions, the estimate is different. From part 1. I found starting with initial condition (0, 0, 0, 0, 0, 0). The long-run average profit is negative, meaning the manager cannot earn money in this way. But if we start with (10, 20, 30, 40, 50, 80), the average profit is approximately 110, which means the manager can earn through this strategy.

Reference Material

Proj.m: This document is used to simulate the value function's estimate with different K using Monte Carlo Method.

Proj_2.m: This document is used to generate training set and test set (K=20, 200, 2000, 50).

STA4001_main.py: Used to simulate $\widetilde{v(x)}$ in 2(a). In this set, 80% of the set is used to train Neural Network, 20% of them is used to make prediction, batch size=8.

STA4001_aulixary.py: Used for 2(b) and 2(c). In these 2 questions, the test set is "50test.csv". The training set is "K=20.csv", "K=200.csv", "K=2000.csv". We use these 3 set and the result got from (a) to train Neural Network. And then induce the test set to make prediction. And compared with the real value generated from Monte Carlo Method.

"K=20.csv"; "K=200.csv"; "K=2000.csv"; "K=50.csv": The K length Markov Chain picked from long run Discrete Markov Chains.

(I submitted all the codes, dataset in the .zip file, the following pdf are the codes in PDF file.)

Appendix I : proj.m

```
%%This set was used to make Monte Carlo estimate.
clear all
%set initial conditions
x01=10;
x02=20;
x03=30;
x04=40;
x05=50;
x06=80;
beta=0.8;
k=50;%K can also be 20, 200, 2000.
T=10^3*k;
%V is used to estimate the value function
V=[];
%M is used to generate the set we want.
M=[];
%for k=1:20
%     x1=x01;
%     x2=x02;
%     x3=x03;
%     x4=x04;
%     x5=x05;
%     x6=x06;
%     x=[x1;x2;x3;x4;x5;x6];
%     v=0;
%     for j=0:T
%         d=random('Poisson',17);
%         profit=min(90,d)-0.4*max(90-x6-x5-x4-x3-x2-x1,0)-0.1*max(x1-d,0)-0.1*(x2+x3+x4+x5+x6);
%         v=v+(beta^j)*profit;
%         x1=max(x2-max(d-x1,0),0);
%         x2=max(x3-max(d-x1-x2,0),0);
%         x3=max(x4-max(d-x1-x2-x3,0),0);
%         x4=max(x5-max(d-x1-x2-x3-x4,0),0);
%         x5=max(x6-max(d-x1-x2-x3-x4-x5,0),0);
%         x6=max(90-max(d,x1+x2+x3+x4+x5+x6),0);
%     end
%     M=[M,x];
%     V=[V,v];
% end

%%Simulation based on Monte Carlo Method.
vmean=mean(V);
episode=1000;
count=0
epsilon=1.96*std(V)/sqrt(episode);
for count=0:(episode-1)
    count=count+1;
    x1=x01;
    x2=x02;
    x3=x03;
    x4=x04;
    x5=x05;
    x6=x06;
    v=0;
    m=0
    for j=1:k
        d=random('Poisson',17);
        %generate profit function
        profit=min(90,d)-0.4*(90-x6-x5-x4-x3-x2-x1)-0.1*max(x1-d,0)-0.1*(x1+x2+x3+x4+x5+x6);
        v=v+beta^j*profit;
        x1=max(x2-max(d-x1,0),0);
        x2=max(x3-max(d-x1-x2,0),0);
        x3=max(x4-max(d-x1-x2-x3,0),0);
        x4=max(x5-max(d-x1-x2-x3-x4,0),0);
        x5=max(x6-max(d-x1-x2-x3-x4-x5,0),0);
        x6=max(90-max(d,x1+x2+x3+x4+x5+x6),0);
    end
    V=[V,v];
    vmean=mean(V);
    epsilon=1.96*std(V)/sqrt(episode);
end
```

Appendix II: proj_2.m

```
%%This document was used to generate the data set.
clear all
%set initial conditions
x01=10;
x02=20;
x03=30;
x04=40;
x05=50;
x06=80;
%discount rate
beta=0.8;
%the size we need generate
k=50;
T=50000;
%V is used for calculating the value function
V=[];
%M is used for storing the data we generate.
M=[];
x1=x01;
x2=x02;
x3=x03;
x4=x04;
x5=x05;
x6=x06;
v=0;
for j=1:50000
    %decide the initial size of Markov Chain. Here is the size for Q2 (b)
    d=random('Poisson',17);
    %generate the profit function
    profit=min(90,d)-0.4*max(90-x6-x5-x4-x3-x2-x1,0)-0.1*max(x1-d,0)-0.1*(x2+x3+x4+x5+x6);
    v=v+(beta^j)*profit;
    x1=max(x2-max(d-x1,0),0);
    x2=max(x3-max(d-x1-x2,0),0);
    x3=max(x4-max(d-x1-x2-x3,0),0);
    x4=max(x5-max(d-x1-x2-x3-x4,0),0);
    x5=max(x6-max(d-x1-x2-x3-x4-x5,0),0);
    x6=max(90-max(d,x1+x2+x3+x4+x5+x6),0);
    m=[x1;x2;x3;x4;x5;x6];
    %collect the data set we need.
    if mod(j,1000)==0
        M=[M,m]
    end
end
%After generating, use the command "xlswrite(filename, variable) to collect
%the data set.
```

Appendix III: STA4001_main.py

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scipy import stats
import math

my_matrix1=np.array(np.loadtxt(open("K=2000.csv","rb"),delimiter=",",skiprows=0))
#Read Data from data set "K=2000.csv". And transform the datatype.
N=2000
#N represents the No. of samples in the data set.
v0=110.2133
#Monte Carlo estimate generated in previous problem.
epsilon=0.3033
#the parameter for 95% confidence interval

class MyNeuralNetwork:
    def __init__(self, input_dimension):
        # define the keras model
        self.model = Sequential()
        self.model.add(Dense(5, input_dim=input_dimension,
                             activation='relu'))
        self.model.add(Dense(5, activation='relu'))
        self.model.add(Dense(1, activation='linear'))
        # compile the keras model
        self.model.compile(loss='mean_squared_error', optimizer='adam')

class Scaler:
    # save mean and variance of x, y sets
    def __init__(self, x, y):
        self.x_mean = np.mean(x, axis=0)
        self.y_mean = np.mean(y, axis=0)
        self.x_std = np.std(x, axis=0)
        self.y_std = np.std(y, axis=0)



    def get_x(self):
        # return saved mean and variance of x
        return self.x_std, self.x_mean

    def get_y(self):
        # return saved mean and variance of y
        return self.y_std, self.y_mean
```

```
def CreateDataset(N):
    # dataset is regenerated based on 6 features, which is the column in the data set
    #Thus, The feauters are[x0;x1;x2;x3;x4;x5]
    dex=np.random.randint(0,N-1,N)
    x1=my_matrix1[0,dex]
    x2=my_matrix1[1,dex]
    x3=my_matrix1[2,dex]
    x4=my_matrix1[3,dex]
    x5=my_matrix1[4,dex]
    x6=my_matrix1[5,dex]
    #y is the estimate we produced from the Monte Carlo Method.
    y=np.random.uniform(v0-epsilon,v0+epsilon,N)
    return np.vstack([x1,x2,x3,x4,x5,x6]).T, y[:, np.newaxis]
```

```
if __name__ == "__main__":
    N = 2000 # set size
    x, y = CreateDataset(N)


    ######## dataset visualization ############################
    plt.plot(range(N), x[:,0], 'o', label="x1", markersize=3)
    plt.plot(range(N), x[:,1], 'o', label="x2", markersize=3)
    plt.plot(range(N), x[:,2], 'o', label="x3", markersize=3)
    plt.plot(range(N), x[:,3], 'o', label="x4", markersize=3)
    plt.plot(range(N), x[:,4], 'o', label="x5", markersize=3)
    plt.plot(range(N), x[:,2], 'o', label="x6", markersize=3)
    plt.plot(range(N), y, lw=2, color="red", label="y = genrated_data_set")
    plt.xlabel('data points')
    plt.legend()
    plt.show()
    ###########################################################

    ########  divide data on training set and test set; here  80% of data is used for training and 20% for testing
    idx_train = np.random.choice(np.arange(len(x)), int(N * 0.8), replace=False)  # indexes included in training set
    idx_test = np.ones((N,),bool)
    idx_test[idx_train] = False  # indexes included in the test set
    x_train = x[idx_train]
    x_test = x[idx_test]
    y_train = y[idx_train]
    y_test = y[idx_test]
    ########################################

    neural_network = MyNeuralNetwork(input_dimension=6)  # create a neural network object; 6 is a number of features

    ####### normilize data #####################
    normalizer = Scaler(x_train, y_train)
    std_x, mean_x = normalizer.get_x()
    x_train_norm = (x_train - mean_x) / std_x
    x_test_norm = (x_test - mean_x)/ std_x
    std_y, mean_y = normalizer.get_y()
    y_train_norm = (y_train - mean_y) / std_y
    ############################################

    neural_network.model.fit(x_train_norm, y_train_norm, epochs=100, batch_size=8)  # train neural network

    y_from_nn_norm = neural_network.model.predict(x_test_norm)  # predict values for x_test states
    y_from_nn = y_from_nn_norm * std_y + mean_y  # tranform the results into original scaling

    mse = mean_squared_error(y_test, y_from_nn) # compute mean squared error
    print('Mean squared error: ', mse)



    ### compare true and predicted values of y from the test set###
    plt.plot(y_test, label="y-original", lw=2)
    plt.plot(y_from_nn, label="y-predicted", lw=2)
    plt.xlabel('test data points')
    plt.legend()
    plt.show()
    ################################################################
```

Appendix IV: STA4001_auxilary.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 17:10:04 2019

@author: yangyufeng
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scipy import stats
import math

my_matrix1=np.array(np.loadtxt(open("K=2000.csv","rb"),delimiter=",",skiprows=0))
#my_matrix2=np.array(np.loadtxt(open("Data_y.csv","rb"),delimiter=",",skiprows=0))
N=2000
v0=110.2133
epsilon=0.3033
#N,v0 & epsilon are same as the previous question.
sigama=0.2928
v1=110.2278
#v1 & sigama are Monte Carlo result for test set, which will be used for generating real value.


class MyNeuralNetwork:
    def __init__(self, input_dimension):
        # define the keras model
        self.model = Sequential()
        self.model.add(Dense(5, input_dim=input_dimension,
                             activation='relu'))
        self.model.add(Dense(5, activation='relu'))
        self.model.add(Dense(1, activation='linear'))
        # compile the keras model
        self.model.compile(loss='mean_squared_error', optimizer='adam')

class Scaler:
    # save mean and variance of x, y z sets
    def __init__(self, x, y, z):
        self.x_mean = np.mean(x, axis=0)
        self.y_mean = np.mean(y, axis=0)
        self.z_mean = np.mean(x,axis=0)
        self.x_std = np.std(x, axis=0)
        self.y_std = np.std(y, axis=0)
        self.z_std = np.std(z,axis=0)
```

```
    def get_x(self):
        # return saved mean and variance of x
        return self.x_std, self.x_mean

    def get_y(self):
        # return saved mean and variance of y
        return self.y_std, self.y_mean

    def get_z(self):
        return self.z_std, self.z_mean


def CreateDataset(N):
    # dataset is regenerated based on 6 features x = [x0; x1; x2; x3; x4; x5] and one output
    dex=np.random.randint(0,N-1,N)
    x1=my_matrix1[0,dex]
    x2=my_matrix1[1,dex]
    x3=my_matrix1[2,dex]
    x4=my_matrix1[3,dex]
    x5=my_matrix1[4,dex]
    x6=my_matrix1[5,dex]
    y=np.random.uniform(v0-epsilon,v0+epsilon,N)
    return np.vstack([x1,x2,x3,x4,x5,x6]).T, y[:, np.newaxis]




if __name__ == "__main__":
    N = 2000 # set size
    x, y = CreateDataset(N)

    ######## dataset visualization ############################
    plt.plot(range(N), x[:,0], 'o', label="x1", markersize=3)
    plt.plot(range(N), x[:,1], 'o', label="x2", markersize=3)
    plt.plot(range(N), x[:,2], 'o', label="x3", markersize=3)
    plt.plot(range(N), x[:,3], 'o', label="x4", markersize=3)
    plt.plot(range(N), x[:,4], 'o', label="x5", markersize=3)
    plt.plot(range(N), x[:,2], 'o', label="x6", markersize=3)
    plt.plot(range(N), y, lw=2, color="red", label="y = genrated_data_set")
    plt.xlabel('data points')
    plt.legend()
    plt.show()
    ###########################################################
```

```
    ########  Now, all the set was used for training set.
    idx_train = np.random.choice(np.arange(len(x)), int(N * 1), replace=False)  # indexes included in training set
#    idx_test = np.ones((N,),bool)
#    idx_test[idx_train] = False  # indexes included in the test set
    x_train = x[idx_train]
#   induce the test set.
    test_set = np.transpose(np.array(np.loadtxt(open("50test.csv","rb"),delimiter=",",skiprows=0)))
#   Set the test idx
    test_idx= np.random.choice(np.arange(len(test_set)),50,replace=False)
#   generate the test input
    z=test_set[test_idx]
    y_train = y[idx_train]
    y_test = np.random.uniform(v1-sigama,v1+sigama,50)
    ########################################


    neural_network = MyNeuralNetwork(input_dimension=6)  # create a neural network object; 6 is a number of features

    ####### normilize data ####################
    normalizer = Scaler(x_train, y_train, z)
    std_x, mean_x = normalizer.get_x()
    x_train_norm = (x_train - mean_x) / std_x
    std_y, mean_y = normalizer.get_y()
    y_train_norm = (y_train - mean_y) / std_y
    std_z, mean_z = normalizer.get_z()
    x_test_norm = (z - mean_z)/ std_z
    ########################################


    neural_network.model.fit(x_train_norm, y_train_norm, epochs=100, batch_size=9)  # train neural network

    y_from_nn_norm = neural_network.model.predict(x_test_norm)  # predict values for x_test states
    y_from_nn = y_from_nn_norm * std_y + mean_y  # tranform the results into original scaling

    mse = mean_squared_error(y_test, y_from_nn) # compute mean squared error
    print('Mean squared error: ', mse)


    ### compare true and predicted values of y from the test set###
    plt.plot(y_test, label="y-original", lw=2)
    plt.plot(y_from_nn, label="y-predicted", lw=2)
    plt.xlabel('test data points')
    plt.legend()
    plt.show()
    ##################################################################
```