HW7 due 05/02/2023(Tuesday) in class

1.Regression with tanh activation function (40%)

In this problem, you are asked to replace the sigmodal activation with tanh activation . In the midterm exam, you are asked to show that

$\tanh'(x) = 1 - \tanh^2(x)$

Use this formula to obtain new weight update rule and modify a python code in the appendix for fitting sine function sin(6x) +N (0,0.1) with regression. N(0,0.1) is gaussian noise with zero mean and standard deviation 1. (Or you can download it in eeclass 上課教材 file name regression v3 bp 04-07-23.py)

The neural network structure is specified by

2 input unit (a bias plus one input data)

2 hidden layer units with sigmodal activaion function

1 output layer unit with linear activation function

In the code, I generate 27 sample data points for input data x^t from y=sin(6*x1)+random.gauss(0,0.1)

(a) (20%) Print out all the weight coefficients.

(b) (20%) Also copy input data and its actual output (output y) evaluated on these 27 data points into excel and plot output y versus input data along with .

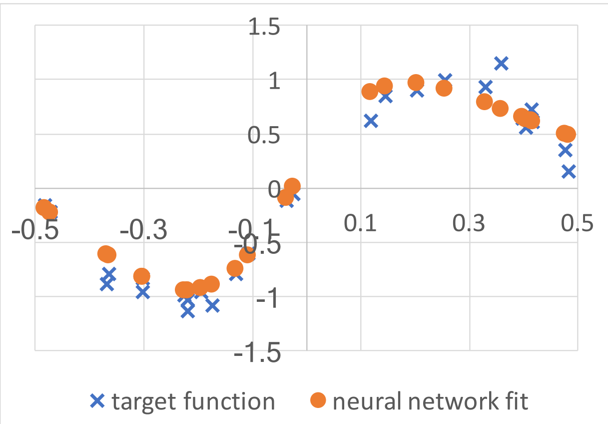
solution:

(a)

v0 0.623 v1 3.175 v2 2.391

wz1_0_bias -0.272 wz1_1 -1.668 wz2_0_bias 0.181 wz2_1 5.591

(b)



python code attached to the end of solution :

2.Bias variance dilemma

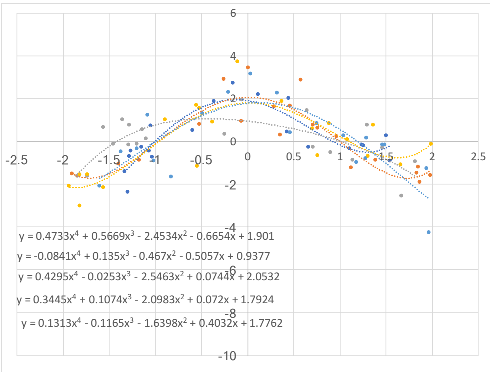
In this problem, you are asked to input a target function Cos(1.5x)+N(0,1). N(0,1) is gaussian noise with zero mean and standard deviation 1. The input $\{X^t\}$ is from a uniform distribution in the interval [-2,2].For example, you can use python code as follows

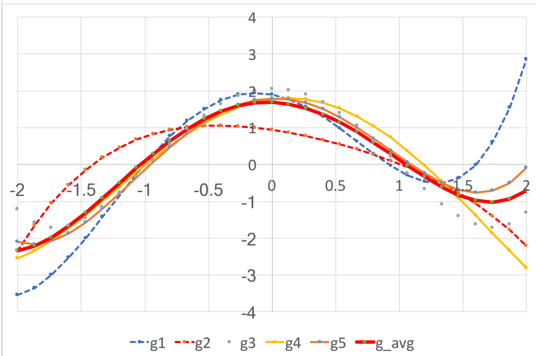
```
x= random.uniform(-2,2)
y=cos(1.5*x)+random.gauss(0,1)
```

Use the python code to generate 5 data sets. Each dataset consists of 20 datapoints $\{x^t,y^t\}$ t=1,2…..20. (t is sample index.) Copy these data points into excel.

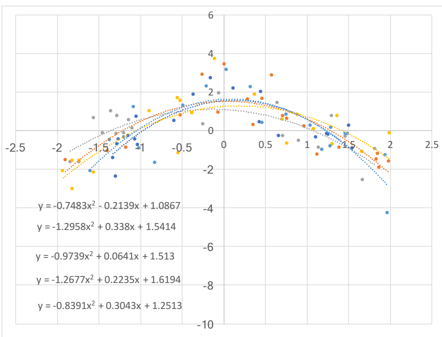
(a) (20%)For each data set, fit the target function with trend line of 4^{th} order polynomial and display the equation in the graph. In total, you will get five trend line, each correspond to a function

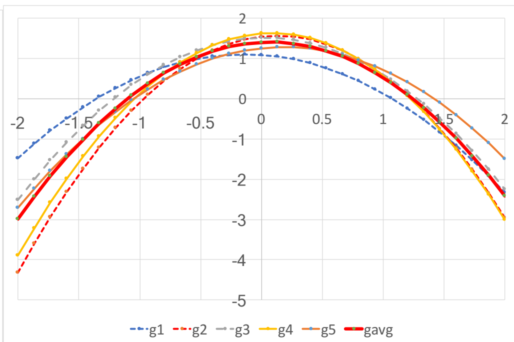
$g_i(x)=w_0+w_1x+w_2x^2+w_3x^3+w_4x^4$. Record the coefficients w_i (i=0,…4)to find average of these five function $\bar{g}=\left(\frac{1}{N}\right)\sum_{i=1}^{N} g_i(x)$ (N=5)





(b) (20%) Repeat (a) with 2^{nd} order polynomial and hence $g_i(x)=w_0+w_1x+w_2x^2$. find average of these five function $\bar{g}=\left(\frac{1}{N}\right)\sum_{i=1}^{N} g_i(x)$(N=5)





3. (20%, 10% each)For a bin with two kind of marbles (red and green), we denote the actual probability μ of getting red marble and ν is the fraction of red marbles in a

sample of size N. (This means we make drawing N times and after each drawing of a marble, we replenish the bin with the marble with the same color to keep μ constant.) Use bionomial distribution for N= 10 to evaluate.

(a) $\mathrm{P}[\,|\nu - \mu| < \epsilon]$ $\mu = 0.75$ $\epsilon = 0.06$

We use $\mathrm{P}[..]$ to denote the probability.

(b) $\mathrm{P}[\,\nu < 1]$ $\mu = 0.9$

solution:

(a)N=10 $\mathrm{P}[\,|\nu - 0.75| < 0.06] = P[0.69 < v < 0.81]$

the only possible v= 7 or 8

$$P[0.69 < v < 0.81] = P[v = 7] + P[v = 8]$$

P= (10!/7!3!)0.9^7 0.1^3 + (10!/8!2!)0.9^8 0.1^2 = (0.0004783*120+0.0043*45)=0.2511

(b) N=10, the only possible v= 0

P[v=0]= $(1\text{-}0.9)^{10}$=10^{-10}

Appendix: solution to tanh activation function

```
import random
from random import randrange
from math import *
import matplotlib.pyplot as plt
def tanh(x):
        return (exp(x)-exp(-x))/(exp(x)+exp(-x))
w_z1=[0,0]
w_z2=[0,0]
#w_z3=[0,0]

#initialize weight coefficients
for i in range(2):
        w_z1[i]=random.uniform(-0.1,0.1)
        w_z2[i]=random.uniform(-0.1,0.1)
      # w_z3[i]=random.uniform(-0.1,0.1)
v_1=random.uniform(-0.1,0.1)
v_2=random.uniform(-0.1,0.1)
#v_3=random.uniform(-0.1,0.1)
v_0=random.uniform(-0.1,0.1) # adding bias term for v

eta=0.05 #define learning rate

# repurpose the input vector
# the first element is bias unit
# the second is the input x
# target function f(x)= sin 6x from Alpaydin'book


# set the rest of element to be zero
x = [ [1,0.0], [1,0.0], [1,0.0], \
           [1,0.0], [1,0.0], [1,0.0], \
           [1,0.0], [1,0.0], [1,0.0],
           [1,0.0], [1,0.0], [1,0.0], \
           [1,0.0], [1,0.0], [1,0.0], \
           [1,0.0], [1,0.0], [1,0.0],
           [1,0.0], [1,0.0], [1,0.0], \
```

```
        [1,0.0], [1,0.0], [1,0.0], \
        [1,0.0], [1,0.0], [1,0.0]

        ]

# desired output array
r= [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, \
    0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, \
    0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
    ]

for i in range(27):
    x1= random.uniform(-0.5,0.5)
    x[i][1]=x1
    y=sin(6*x1)+random.gauss(0,0.1)
    r[i]=y

for i in range(27000):
    j= randrange(27) #randomly pick sample vector of out of 8
    desiredoutput=r[j]
    sum_w_z1=0
    sum_w_z2=0
    #sum_w_z3=0
    sum_v=0
    for k in range(2):
        sum_w_z1=sum_w_z1+ w_z1[k]*x[j][k]
        sum_w_z2=sum_w_z2+ w_z2[k]*x[j][k]
        #sum_w_z3=sum_w_z3+ w_z3[k]*x[j][k]
    z1_h=tanh(sum_w_z1)
    z2_h=tanh(sum_w_z2)
    #z3_h=tanh(sum_w_z3)
    sum_v=v_1*z1_h+v_2*z2_h+v_0 #keep only two hidden unit

    output_y= sum_v    #use linear unit as output



#delta rule
    # weight update Ethm Alpaydin' pseudo code
```

```
        # update= learning rate*(Desired output - Actualouput)*input
        v_1=v_1-eta*(output_y-desiredoutput)*z1_h
        v_2=v_2-eta*(output_y-desiredoutput)*z2_h
        #v_3=v_3-eta*(output_y-desiredoutput)*z3_h
        v_0=v_0-eta*(output_y-desiredoutput)*1

        for m in range(2):
                # weight update Ethm Alpaydin' pseudo code
                # update= learning rate *v*z(1-z)*(Desired output - Actualouput)*input

                w_z1[m]=w_z1[m]-eta*v_1*(1-z1_h*z1_h)*(output_y-
desiredoutput)*x[j][m]
                w_z2[m]=w_z2[m]-eta*v_2*(1-z2_h*z2_h)*(output_y-
desiredoutput)*x[j][m]
                #w_z3[m]=w_z3[m]-eta*v_3*(1-z3_h*z3_h)*(output_y-
desiredoutput)*x[j][m]

for j in range(27):
        desiredoutput = r[j]
        sum_w_z1=0
        sum_w_z2=0
        #sum_w_z3=0
        sum_v=0
        for k in range(2):
                sum_w_z1=sum_w_z1+ w_z1[k]*x[j][k]
                sum_w_z2=sum_w_z2+ w_z2[k]*x[j][k]
            #    sum_w_z3=sum_w_z3+ w_z3[k]*x[j][k]
        z1_h=tanh(sum_w_z1)
        z2_h=tanh(sum_w_z2)
        #z3_h=tanh(sum_w_z3)

        sum_v=v_1*z1_h+v_2*z2_h+v_0
        output_y= sum_v

        print('input x',round(x[j][1],3), 'desiredoutput',
round(desiredoutput,3),'actualoutput', round(output_y,3))
# in total, this newtork has 7 coefficient, namely 3 v coefficents and 4 w cofficients
print('v0', round(v_0,3), 'v1', round(v_1,3), 'v2', round(v_2,3))
```

```
print('wz1_0_bias', round(w_z1[0],3), 'wz1_1', round(w_z1[1],3), 'wz2_0_bias',
round(w_z2[0],3),'wz2_1', round(w_z2[1],3))
```