**HW5 - Final Report**

Submitted by Nikita Thakur

Steps 1-5 have been completed as part of the python implementation below, delineated by comments.

import numpy as np

import matplotlib.pyplot as plt

#initializations of x and y

x = np.random.uniform(0.0,1.0,300)

v = np.random.uniform(-0.1,0.1,300)

#calculation of di = sin(20xi) + 3xi + νi

d = np.sin(20\*x) + 3\*x + v

#plotting function d against x

fig, ax = plt.subplots(figsize=(10,10))

plt.scatter(x=x,y=d, c = '#d62728', alpha = 0.5)

plt.xlabel('xi',fontsize=13)

plt.ylabel('di = sin(20xi) + 3xi + νi',fontsize=13)

plt.title('Data Set Visualization')

plt.show()

#activation functions

#feed-forward

activation\_inp = lambda x : np.tanh(x)

activation\_out = lambda x : x

#feed-back

d\_activation\_inp = lambda x : 1 - np.tanh(x)\*\*2

d\_activation\_out = lambda x : 1

#Mean Square Error

mse = lambda d,y : (d-y)\*\*2

#weight and bias initializations

w\_inp = np.random.uniform(-5,5,size = 24)

w\_out = np.random.uniform(-5,5,size = 24)

b\_inp = np.random.uniform(-1,1,size = 24)

w\_fin = np.random.uniform(-1,1,1)

n = 300

N = 24

eta = 8

#calculation of MSE while BP learning

MSE\_records = []

itr = 0

while(True):

U,Y,Alphas,Betas = [],[],[],[]

#calculation of Alpha and Beta parameters

for i in range(n):

V = []

hold = []

for j in range(N):

alpha = (x[i]\*w\_inp[j]) + b\_inp[j]

hold.append(alpha)

V.append(activation\_inp(alpha))

Alphas.append(hold)

U.append(V)

beta = np.matmul(np.array(U[i]),w\_out) + w\_fin

Betas.append(beta[0])

Y.append(activation\_out(beta[0]))

e = -2\*eta\*((d[i]-Y[i]))/n

w\_out\_grad,w\_inp\_grad,w\_inp\_bias\_grad,w\_fin\_grad = [],[],[],[]

w\_fin\_grad.append(-e)

#Calculation of delta weights

for j in range(N):

change\_U = e\*U[i][j]

change\_w = e\*x[i]\*w\_out[j]\*d\_activation\_inp(Alphas[i][j])

change\_bias = e\*w\_out[j]\*d\_activation\_inp(Alphas[i][j])

w\_out\_grad.append(change\_U)

w\_inp\_grad.append(change\_w)

w\_inp\_bias\_grad.append(change\_bias)

#performing weight updates

w\_inp = np.subtract(w\_inp,np.array(w\_inp\_grad))

w\_out = np.subtract(w\_out,np.array(w\_out\_grad))

b\_inp = np.subtract(b\_inp,np.array(w\_inp\_bias\_grad))

w\_fin = np.subtract(w\_fin,np.array(w\_fin\_grad))

#Calculation of Mean Square Error for each epoch

mean\_square\_error = 0

for i in range(n):

mean\_square\_error += mse(d[i],Y[i])

mean\_square\_error /= n

MSE\_records.append(mean\_square\_error)

print("eta: ",eta," MSE: ",mean\_square\_error," iteration: ",itr)

#updating Eta if MSE increases in any consecutive epoch

if MSE\_records[itr] > MSE\_records[itr-1]:

eta = eta\*0.9

#Stop learning when MSE is less than 0.01

if MSE\_records[-1] < 0.01:

break

itr+=1

#plotting the MSE vs Epoch graph

fig, ax = plt.subplots(figsize=(10,10))

plt.xlabel('Number of Epochs')

plt.ylabel('Mean Square Error (MSE)')

plt.legend(loc = 'upper right')

plt.scatter(range(len(MSE\_records)), MSE\_records, c = '#d62728', label = 'MSE')

plt.show()

#calculating the fit

u = []

y = []

A = []

B = []

for l in range(n):

v = []

hold = []

for m in range(N):

a = x[l]\*w\_inp[m] + b\_inp[m]

hold.append(a)

v.append(activation\_inp(a))

A.append(hold)

u.append(v)

b = np.matmul(np.array(u[l]),w\_out) + w\_fin

B.append(b)

y.append(activation\_out(b[0]))

#plotting the dataset against the fit generated by Backpropagation Online Learning

fig, ax = plt.subplots(figsize=(10,10))

plt.ylabel('d')

plt.xlabel('x')

plt.scatter(x=x,y=d, c = 'magenta', alpha = 0.5, label = 'Actual Data')

plt.scatter(x=x,y=y, c = 'purple', label = 'fit',alpha = 0.5)

plt.legend(loc = 'upper right')

plt.show()

Pseudocode of the Algorithm:

Below Pseudocode is for a 1×N ×1 neural network.

**Assumptions:**

X →List Input data points

n → number of data points in X

V → List Constants

F(X) → Function of X

D → Set of values calculated by calculation F(X) for each data point in X

Y → Values of F(X) predicted by the learning algorithm

N → Number of Neurons in the hidden layer

A\_inp() → Activation function for the input layer; A\_inp(v) = tanh(v)

A\_out() → Activation function of the hidden layer; A\_out(v) = v

delta\_A\_inp() → Activation function during feedback from hidden layer to input layer;

delta\_A\_inp(v) = 1 - tanh(v)^2

delta\_A\_out() → Activation function during feedback from output layer to hidden layer;

delta\_A\_out(v) = 1

Backpropagation():

Initialize N Weights randomly for input and hidden layers as W\_inp and W\_out

Initialize N biases for input layer as B\_inp and 1 bias of output layer as B\_out

Initialize the Learning Rate as eta randomly

Initialize empty lists to store alphas,betas,U and Y

While condition → True, perform:

For i = 1...n and j in range 1...N, perform:

Calculate alpha as X(i)\*W\_in(j) + b\_inp(j)

V → A\_inp(alpha)

Alphas → append alpha

U→ append V

beta = U(i)\*W\_out + B\_out

Betas → append beta

Y → append A\_out(beta)

Calculate e as -2 \* eta \*( d(i) - y(i) ) / n

Initialize W\_out\_update,W\_inp\_update,Bias\_inp\_update,Bias\_out\_update as

Empty lists

For range 1...N

Bias\_out\_update → append -e

W\_out\_update → append e\*U(i)(j)

W\_inp\_update → append e\*x(i)\*W\_out(j)\*delta\_A\_inp(Alphas(i)(j))

Bias\_inp\_update → append e\*W\_out(j)\*delta\_A\_inp(Alphas(i)(j))

W\_inp = W\_inp - W\_inp\_update

W\_out = W\_out - W\_out\_update

B\_inp = B\_inp - Bias\_inp\_update

B\_out = B\_out - Bias\_out\_update

For all inputs(i) in range 1..n:

Calculate MSE(i) → (d(i) - y(i)^2

Calculate Mean Square Error as MSE/n

Maintain a list of MSE for each epoch in MSE\_LIST

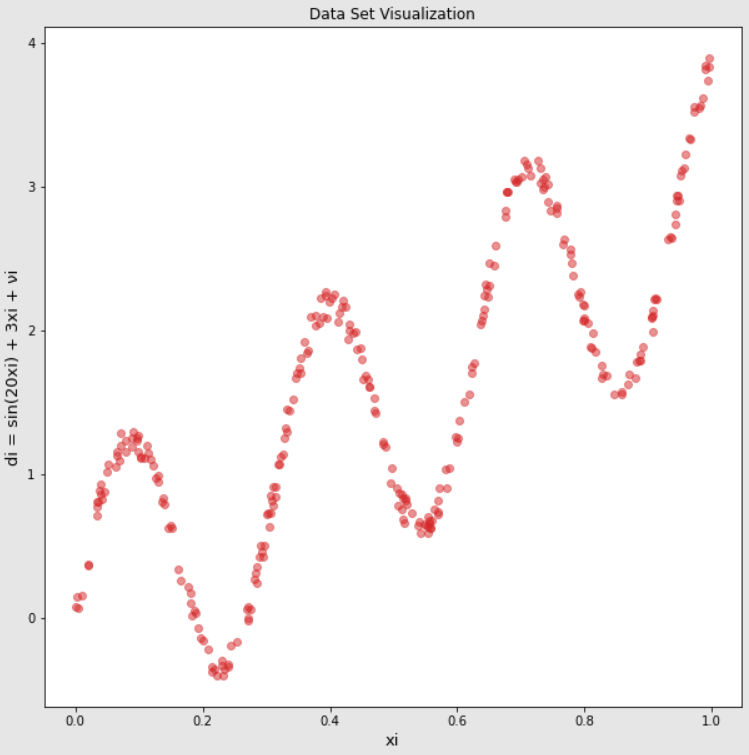
if MSE(iteration+1) > MSE(iteration) → update eta as eta\*0.9

If MSE of last update < 0.01 → break the outer loop

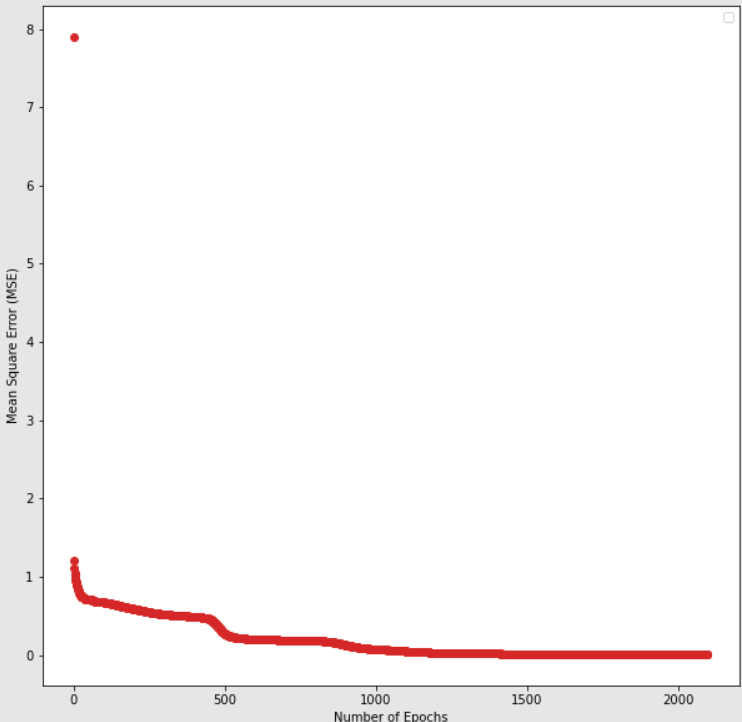
Iteration = iteration + 1

Graphs:

Step 3 : data set visualization



Step 4: MSE vs EPOCH



Step 5: The good fit:

