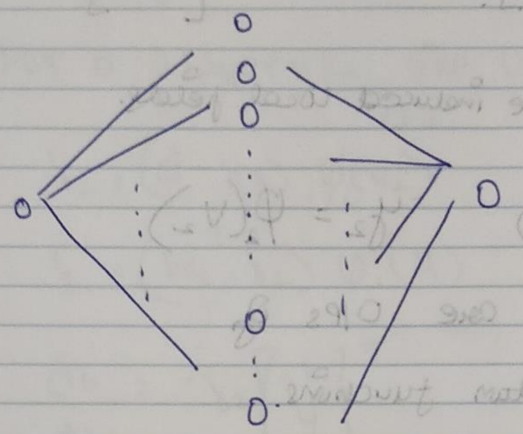
Homework – 4

Network Architecture

HW-4.

* N/w architecture



$M_0 = 1 \quad M_1 = 24 \quad M_2 = 1.$

$M_i \rightarrow$ no. of Nodes in a layer i

$W_1 \Rightarrow$ dim of $\quad$ total weights

$M_1 \times (M_0+1) = \underbrace{24 \times 2}$ $= 48.$ $\quad 24\left\{\begin{bmatrix} \overset{2}{\cdots} \\ \\ \end{bmatrix}\right.$

$W_2 \Rightarrow$ $\quad$ total weights

$\underbrace{1} \times \underbrace{(24+1)} = 25$ $\quad \begin{array}{c c} & 25 \\ 1 & \begin{bmatrix} \vdots \end{bmatrix} \end{array}$

We know,

$$V_l = W_l \begin{bmatrix} 1 \\ y_{l-1} \end{bmatrix}$$

$$V_1 = W_1 \begin{bmatrix} 1 \\ y_0 \end{bmatrix} \quad \& \quad V_2 = W_2 \begin{bmatrix} 1 \\ y_1 \end{bmatrix}$$

these are the induced local fields.

$$y_1 = \phi_1(V_1) \qquad y_2 = \phi_2(V_2)$$

y_1, y_2 are O/Ps of Activation functions.

where $\phi_1 \rightarrow$ tanh i.e $\phi_1(v) = \tanh(v)$

& $\phi_2 \rightarrow$ identity $\phi_2(v) = v$.

writing the backpropagation equations in general form.

$$\delta_L = (d - y_L)\, \phi'(V_L)$$

$$\delta_\ell = W_{\ell+1}^T\, \delta_{\ell+1}\, \phi'(V_\ell)$$

$$\ell = 1, 2 \ldots L-1$$

$$\frac{\partial E}{\partial W_\ell} = -\delta_\ell \begin{bmatrix} 1 \\ y_{\ell-1} \end{bmatrix}^T \qquad \ell = 1, 2 \ldots L$$

$$W_\ell \leftarrow W_\ell + \eta\, \delta_\ell \begin{bmatrix} 1 \\ y_{\ell-1} \end{bmatrix}^T$$

$$E = \frac{1}{2|S|} \sum_{i=1}^{S} \| d_i - y_{2i} \|^2$$

For a 2 layer n/w like ours, $L = 2$

$$\delta_2 = (d - y_2)\, \phi_2'(V_2)$$

$$\delta_1 = (W_2^T \delta_2)\, \phi_1'(V_1)$$

$$\frac{\partial E}{\partial W_2} = -\delta_2 \begin{bmatrix} 1 \\ y_1 \end{bmatrix}^T$$

$$\frac{\partial E}{\partial W_1} = -\delta_1 \begin{bmatrix} 1 \\ y_0 \end{bmatrix}^T$$

$$W_2 \leftarrow W_2 + \eta \delta_2 \begin{bmatrix} 1 \\ y_1 \end{bmatrix}^T$$

$$W_1 \leftarrow W_1 + \eta \delta_1 \begin{bmatrix} 1 \\ y_0 \end{bmatrix}^T$$

* $eta_1 = 0.0005$
$eta_2 = 0.01$

* plot has been attached at the end.

Pseudo-Code

Pseudo Code

1) 1.1 $X \leftarrow$ initialise with 300 random values b/w $(0,1)$

such that $X = [\cancel{x_0}\ x_1 \ \ldots \ x_n]$ $\quad n = 300.$

1.2 $V \leftarrow$ initialise with 300 random values b/w $(-1/10, 1/10)$

$V = [\cancel{v_0}\ v_1 \ \ldots \ v_n]$ $\quad n = 300$

1.3 $D \Leftarrow \sin(30 x_i) + 3(x_i) + v_i$

#D contains 300 values of the above func.

2) Define activation functions & their derivative (later we call them)

2.1 def $\phi_2(v)$:
return(v)

2.2 def $\phi_1(v)$
return $(\tanh(v))$

$\alpha \rightarrow$ can be assigned any value.
0.01.

2.3 def derivative $\phi_1(v)$:
return $(1 - \tanh^2(v))$

2.4 def derivative $\phi_2(v)$:
return (1)

3) 3.1 Initialise eta
$\downarrow$
eta_1 & eta_2

3.2 Initialise weights (Mean 0 preferably
$\sigma = \sqrt{\frac{c}{M}}$
$c \rightarrow$ Any constant

4) Compute: V_1, V_2, Y_1, Y_2
4.1.
$$V_1 = W_1 \begin{bmatrix} 1 \\ y_0 \end{bmatrix}$$
$$V_2 = W_2 \begin{bmatrix} 1 \\ y_1 \end{bmatrix}$$
$$Y_1 = \phi_1(V_1)$$
$$Y_2 = \phi_2(V_2)$$

4.2.

$\begin{pmatrix} d \rightarrow \text{Actual} \\ y_2 \rightarrow \text{Predicted} \end{pmatrix}$

Compute $\delta_2 = (d - y_2)\ \phi'(v_2)$

$$\delta_1 = (W_2^T \delta_2) \circ \phi'(v_1)$$

4.3. update weights W_1, W_2

$$W_2 \leftarrow W_2 + \eta_2 \delta_2 \begin{bmatrix} 1 \\ y_1 \end{bmatrix}^T$$

$$W_1 \leftarrow W_1 + \eta_1 \delta_1 \begin{bmatrix} 1 \\ y_0 \end{bmatrix}^T$$

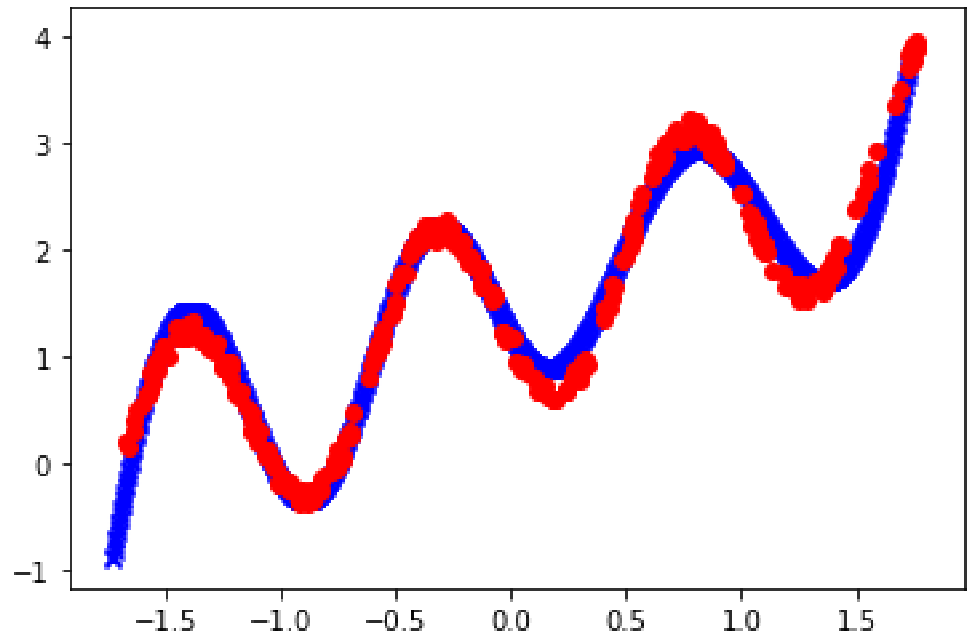
5) Loop over (4)^n untill convergence
(i.e MSE < 0.02 for my case)

return MSE, no. of epochs

Note : eta_2, eta_1 η has different values in different layers

η has been updated as $\eta \leftarrow 0.48$ if MSE increases

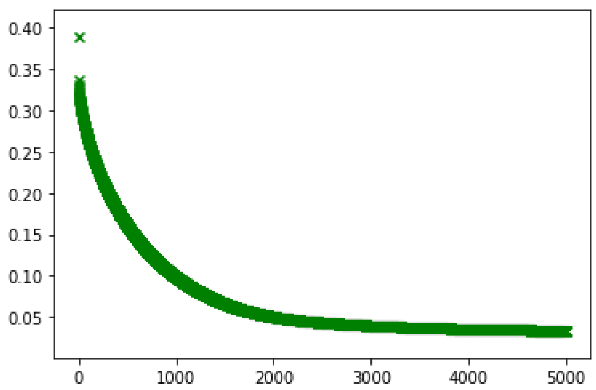
Results



F(x,W)-> Blue Line is the Predicted Function by the Backpropagation Algorithm

Di - >Red Line is the Original Data Points Initialised.

```
In [94]: print("eta value for layer 2", eta2)
    ...: print("eta value for layer 1", eta1)
    ...: Epochs=np.linspace(0,len(MSElist), num=len(MSElist))
    ...: plt.scatter(Epochs[:5000],MSElist[:5000],color='green',
marker='x');
eta value for layer 2 0.05
eta value for layer 1 0.01
```



The Graph Shows as the number of epochs increase, The MSE is reduced to almost 0. (0.02 ~ Threshold set for the algorithm to exit)

Note the learning rate is different for both the layers and it is adjusted everytime the MSE overshoot (increases in the following epoch)

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  7 13:23:12 2020

@author: Varun
"""
###########import libraries#############
import os
from os import listdir
from os.path import isfile, join
import struct
import numpy as np
import random
import operator
import matplotlib.pyplot as plt
import gzip
from numpy.linalg import inv
import math
#############################################
#1. Draw n = 300 real numbers uniformly at random on [0, 1], call them x1, . . . , xn.
X=[]
for i in range(0,300):
   X.append((random.uniform(0, 1)))

#Draw n real numbers uniformly at random on [-1/10,1/10] call them v1, . . . , vn.
V=[]
for i in range(0,300):
   V.append((random.uniform((-1/10), (1/10))))

#3. Let di = sin(20xi) + 3xi + vi, i = 1, . . . , n. Plot the points (xi, di), i = 1, . . . , n.
D=[]
for i in range(0,300):
   D.append(math.sin(20*X[i])+(3*X[i])+ (V[i]))

plt.scatter(X,D,color='red', marker='o');

"""
###########################################
def activationphi2(v):
   return(v)
def derivateactivationphi2(v):
   return(1)

def activationphi(v):
   return(np.tanh(v))
def derivativeactivationphi(v):
   return(1-(np.tanh(v)*np.tanh(v)))

########"Input Layer"####################
"""
def activationphi2(v):
   return(v)
def derivateactivationphi2(v):
   return(1)
```

```
def activationphi1(v):
   return(0.8*np.tanh(v))
def derivateactivationphi1(v):
   return(.8*(1-(np.tanh(v)*np.tanh(v))))

#learning rate
eta2=0.05
eta1=0.01
X=np.array(X)
X=(X-np.mean(X))/(np.std(X))
#X.shape
#weights initialisation
W1 =[[ round(random.uniform(-1,1),5) for y in range(2)] for x in range(24) ]
W1=W1-np.mean(W1)
W1=W1/np.std(W1)
W2=[[ round(random.uniform(-1,1),3) for y in range(25)] for x in range(1) ]
W2=W2-np.mean(W2)
W2=W2/np.std(W2)

W1=np.array(W1)
W2=np.array(W2)

MSElist=[]
c=0

while True:
   ListofDs=[]
   ListofYs=[]
   for i in range(0,len(D)):
      ##############################FORWARD PROPOGATION############################
      ##############################FORWARD PROPOGATION############################
      ##############################FORWARD PROPOGATION############################
      ##############################FORWARD PROPOGATION############################
      ####################LAYER 1 ###########################
      ####################LAYER 1 ###########################
      ########"W1  Input Layer-Mid Layer#####################
      #W1=np.array(W1)
      W1.shape
      ########"V1" Induced Local Field ######################
      V1=np.matmul(W1,[1,X[i]])
      V1.shape
      ########"Y1" After Activation Func of Local Field #####
      Y1=np.array([activationphi1(x) for x in V1])
      Y1.shape
      ####################LAYER 2 ###########################
      ####################LAYER 2 ###########################
      ########"W2  Mid Layer-Output Layer####################
      #W2=np.array(W2)
      W2.shape
      ########"V2" Induced Local Field #####################
      V2=np.matmul(W2,np.array([1]+list(Y1)))
      V2.shape
      ########"Y2" After Activation Func of Local Field #####
      Y2=np.array([activationphi2(x) for x in V2])
      Y2.shape
      ##############################Backward PROPOGATION############################
      ##############################Backward PROPOGATION############################
```

```
        ################################Backward PROPOGATION##############################
        lambda2=(D[i]-Y2)*derivateactivationphi2(V2)
        ToUnderline=np.matmul(W2.transpose(),lambda2)
        lambda1=np.matmul(np.array(ToUnderline[1:]),derivateactivationphi1(V1))
        W2=W2+eta2*(lambda2*np.array([1]+list(Y1)).transpose())
        W1=W1+eta1*(lambda1*np.array([1]+[X[i]]).transpose())
        ##################################################################################
        ListofYs.append(Y2[0])
        ListofDs.append(D[i])
        #print("Y2 ",Y2[0])
        #print("D ",D[i])
    print("Epoch  ",c )
    ##calculate the total mean square error i.e (d-y)^2
    MSE=sum([(x[0]-x[1])**2 for x in list(zip(ListofYs,ListofDs))])/(2*len(D))
    print("MSE IS",MSE)
    MSElist.append(MSE)
    c=c+1
    if MSE>MSElist[-1]:
        eta2=0.9*eta2
        eta1=0.9*eta1
    if MSE<0.02:
        break

"""##############after we get optimal weights#####################
XNew=[i for i in range(0,1)]
XNew=np.linspace(0,1, num=1000)
XNew=(XNew-np.mean(XNew))/np.std(XNew)
Outp=[]
i=0
for i in range(0,len(XNew)):
    V1=np.matmul(W1,[1,XNew[i]])
    Y1=np.array([activationphi1(x) for x in V1])
    V2=np.matmul(W2,np.array([1]+list(Y1)))
    Y2=np.array([activationphi2(x) for x in V2])
    Outp.append(Y2)
plt.scatter(XNew,Outp,color='blue', marker='x');
plt.scatter(X,D,color='red', marker='o');
"""
print("eta value for layer 2", eta2)
print("eta value for layer 1", eta1)
Epochs=np.linspace(0,len(MSElist), num=len(MSElist))
plt.scatter(Epochs[:5000],MSElist[:5000],color='green', marker='x');

W1
```