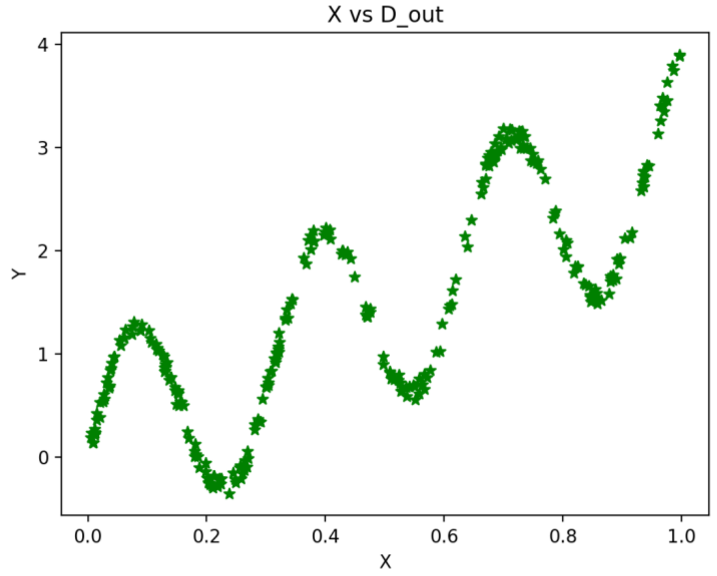
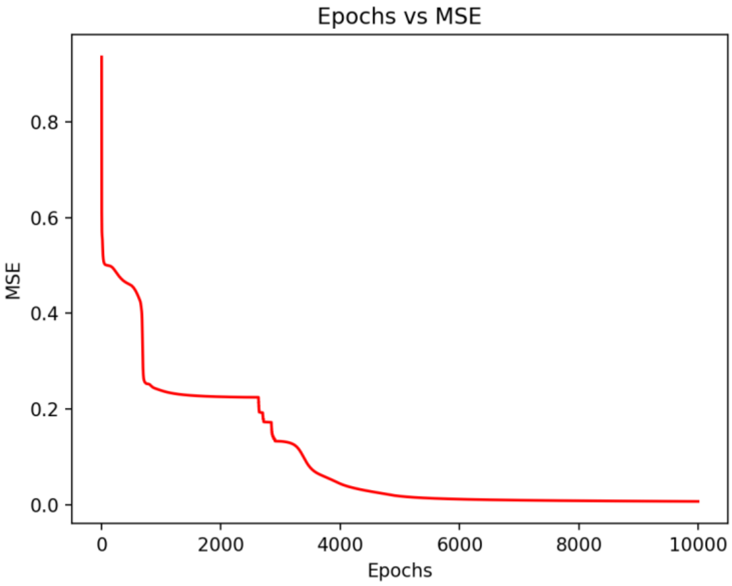
Name: Advait Pai
Email: apai21@uic.edu
UIN: 677368201

Homework 4

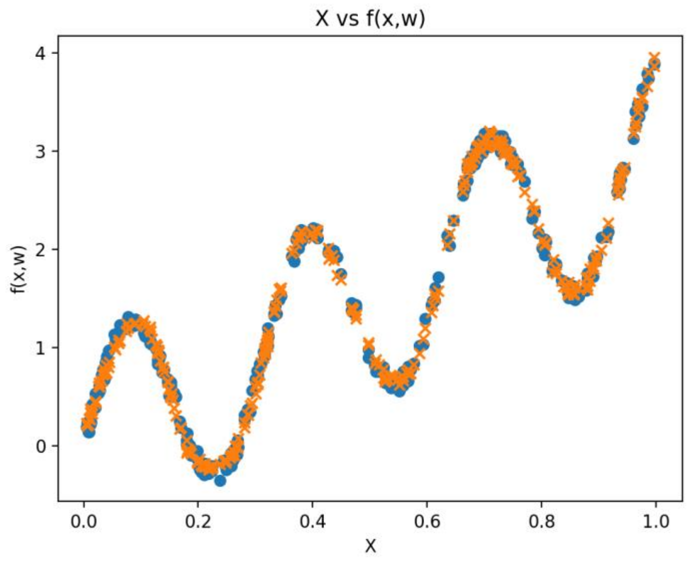
1. Graph of (xi, di)



2. Epoch vs MSE Graph



3. Fit to the curve



4. Final Results

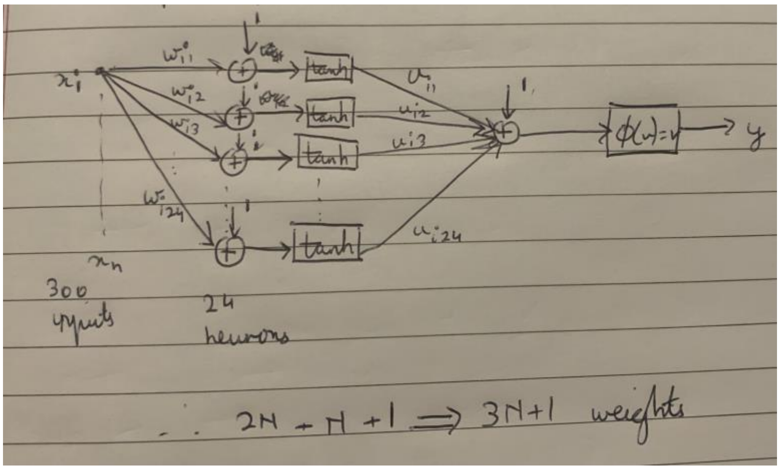
Epoch: 10000
Mean Squared Error: [0.00652244]

Algorithm:

1. Intiate values

X => 300 random numbers between [0,1]
V=> 300 random numbers between [-0.1,0.1]
Di => sin(20xi) + 3xi + vi, i = 1, . . . , 300
W_list => 3N + 1 weights
    Create a list W_list
    Append array of 24 weights (for input layer)
    Append array of 24 weights (for bias of neurons)
    Append array of 24 weights (for output layer neurons
    Append array of 1 weight (for output neuron bias)

2. Forward Propagation



Now we need to store two induced fields. The first induced field is for the inputs. Before the. Tanh boxes. The second induced field is of the output from the tanh functions into the output neuron. Once we have the second induced field, we can calculate the output y. A pseudo code is given below:

ind_field_list = [] # Length of 2
ind_field_list.append(np.dot(W_list[0],1)+np.dot(W_list[1],x)) # Induced Field 1
out_tanh = np.tanh(ind_field_list[0])
ind_field_list.append(out_tanh)# Output Layer 1 == Inputs for Output Layer
 # Hidden Layer -> Output Layer
*output = 1*W_list[3]# Initialising output neuron using the weight of bias since bias == 1*
output=output+(np.dot(out_tanh,W_list[2]))
return ind_field_list,output

Here:
    W[0] => Bias Weights for input layer
    W[1] => Weights for Inputs
    W[2]=> Weights for induced field after applying tanh activation
    W[3]=> Bias Weight for output layer
    x => input

Now we return the induced_field_list and the output. The induced field list will be used to calculate the backward propagation weight update and the output will be used for weight update and MSE calculations.

3. Backward Propagation

The generalised weight update for backpropagation is given as:

W ← W – learning rate * dE/dw

And

dE/dw = - (the signal before multiplication by w in the feedforward network) × (the signal before dw multiplication by w in the feedback network)

Here is the pseudo-code to do the backpropagation step.

#Calculating de/dw
*update_w3 = np.array((-1*1*(d-y))) # Output Layer Bias Weight Update*
update_w2 = -1((d-y)*(ind_fields[1])) # Output Layer Weights Update*
update_w1 = -1(der_tanh(ind_fields[0]))*(x)*(d-y)*W_list[2] #*ind_fields[1] # Input Layer Weights Update*
update_w0 = -1(der_tanh(ind_fields[0]))*(1)*W_list[2]*(d-y)#*ind_fields[1] # Input Layer Bias Weight Update*

#Update Step
W_list[0] = W_list[0] - lr(update_w0) # Input Bias*
W_list[1] = W_list[1] - lr(update_w1) # Input Weights*
W_list[2] = W_list[2] - lr(update_w2) # Hidden Layer Weights*
W_list[3] = W_list[3] - lr(update_w3) # Output Neuron Bias*

Return W_list

Here we return the weights for the next input.

4. Now we keep repeating this step for all values of x
5. We then calculate the MSE for all the outputs
6. With this one epoch is completed, now we keep doing steps 1 to 5 i.e. more epochs till we either reduce the MSE or we complete a set number of epochs.
7. Once 6 is complete, we plot the result we get f(x,y) and plot it against the original f(x,d) to see if the curve is a good fit

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Helper Functions
def calc_di(x,v):
    return np.sin(20*x) + (3*x) + v

def der_tanh(val):
    return 1 - (np.tanh(val)**2)

def der_output(val):
    return 1

def create_weights(N):
    W_temp  = []
    for n in N:
        W_temp.append(np.random.uniform(0,1,n))
    return W_temp

def forward_propagation(x,W_list):
    #Input Layer -> Hidden Layer
    ind_field_list = [] # Length of 2
    ind_field_list.append(np.dot(W_list[0],1)+np.dot(W_list[1],x)) # Induced Field 1
    out_tanh = np.tanh(ind_field_list[0])
    ind_field_list.append(out_tanh )# Output Layer 1 == Inputs for Output Layer
    # Hidden Layer -> Output Layer
    output = 1*W_list[3]# Initialising output neuron using the weight of bias since bias == 1
    output=output+(np.dot(out_tanh,W_list[2]))
    return ind_field_list,output

def backward_propogation(d,y,x,ind_fields,lr,W_list):
    #Calculating de/dw
    update_w3 = np.array((-1*1*(d-y))) # Output Layer Bias Weight Update
    update_w2 = -1*((d-y)*(ind_fields[1])) # Output Layer Weights Update
    update_w1 = -1*(der_tanh(ind_fields[0]))*(x)*(d-y)*W_list[2] #*ind_fields[1] # Input Layer Weights Update
    update_w0 = -1*(der_tanh(ind_fields[0]))*(1)*W_list[2]*(d-y)#*ind_fields[1] # Input Layer Bias Weight Update
    #Update Step
    W_list[0] = W_list[0] - lr*(update_w0) # Input Bias
```

```
    W_list[1] = W_list[1] - lr*(update_w1) # Input Weights
    W_list[2] = W_list[2] - lr*(update_w2) # Hidden Layer Weights
    W_list[3] = W_list[3] - lr*(update_w3) # Output Neuron Bias

    return W_list

def main_algo(X,w,lr):
    w_current = w
    mse = 0
    Y = []
    for i in range(0,len(X)):
        ind_fields,y = forward_propagation(X[i],w_current)
        Y.append(y)
        mse+=((D[i]-y)**2)
        w_update = backward_propogation(D[i],Y[i],X[i],ind_fields,lr,w_current)
        w_current = w_update
    mse = mse/len(X)
    print("Mean Squared Error: ",mse)
    return w_current,mse,Y

# Driver Code Below

np.random.seed(2702)
X = np.random.uniform(0,1,300)
V = np.random.uniform(-0.1,0.1,300)
N = [24,24,24,1] # 3N + 1 weights
W_list = create_weights(N) #List of size 4, containing weights of input bias(24), input layer(24), second layer(24)
and output bias(24)

D = [calc_di(X[i],V[i]) for i in range(len(X))]

plt.title("X vs D_out")
plt.xlabel("X")
plt.ylabel("Y")
plt.scatter(X,D, marker='*',label='X',color ="green")
plt.show()

lr_st = 0.1
init_w,init_mse,Y_final = main_algo(X,W_list,lr_st)
epoch = 0
```

```
mse_list = [init_mse]
while epoch < 10000:
    old_mse = init_mse
    print("Epoch:",(epoch+1))
    init_w,init_mse,Y_final = main_algo(X,init_w,lr_st)
    mse_list.append(init_mse)
    if(init_mse>old_mse):
        lr_st = 0.9*lr_st
    if(init_mse==old_mse):
        break
    if(init_mse<0.001):
        print("MSE Limit Reached")
        break
    epoch+=1

plt.title("X vs f(x,w)")
plt.xlabel("X")
plt.ylabel("f(x,w)")
plt.scatter(X,D,marker='o')
plt.scatter(X, Y_final,marker='x')
plt.show()

# Plotting Epochs vs MSE
plt.title("Epochs vs MSE")
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.plot([i for i in range(epoch+1)],mse_list,color="red")
plt.show()
```