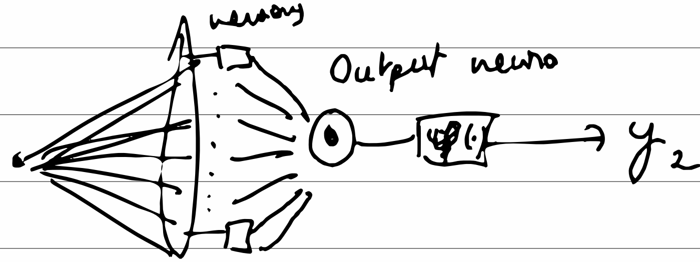
HW 4

Chinmay Nautiyal
677285320

Update eq^{ns} and weight matrices

Given NN,



W_L1 => weights for Layer 1
W_L2 => weights for Layer 2

v_1 => intermediate activation
y_1 = intermediate output

v_2 => final activation
y_2 => final output

Energy function

$$\frac{1}{n}\sum_{i=1}^{n}(d_i - y_i)^2$$

$$\boxed{\partial_2 = 2(\underbrace{d_i}_{\text{desired output}} - \underbrace{y_i}_{\text{final output}})}$$

$$\boxed{\partial_1 = \underline{W_{L2} \cdot \partial_2} \; \phi'(v_2)}$$

where $\phi'(v_2) = 1 - \tanh^2(v_2)$

$$\frac{\partial E}{\partial W\_L2} = -\partial_2 \begin{bmatrix} 1 \\ \underbrace{y_1}_{\text{intermediate output}} \end{bmatrix}^T$$

$$\frac{\partial E}{\partial W\_L1} = -\partial_1 \begin{bmatrix} 1 \\ \underbrace{y^0}_{\text{input}} \end{bmatrix}$$

Weight update equations

$$W\_L2 \leftarrow W\_L2 - \eta \frac{\partial E}{\partial W\_L2}$$

$$W\_L1 \leftarrow W\_L1 - \eta \frac{\partial E}{\partial W\_L1}$$

Dimension of weight matrices

$$W\_L1 = \begin{bmatrix} b_1 & w_1 \\ \vdots & w_2 \\ \vdots & \vdots \\ b_{24} & w_{24} \end{bmatrix}_{24 \times 2}$$

$$W\_L2 = \begin{bmatrix} 1 & w_1 & \cdots & w_{24} \end{bmatrix}_{1 \times 25}$$

hw4

October 30, 2019

[453]:
```
#line fitting with backpropagation
#import cells
import numpy as np
import math
import import matplotlib.pyplot as plt
```

[5]:
```
#randomly choosing inputs
n = 300
x = np.random.uniform(low=0.0, high=1.0, size=n)
v = np.random.uniform(low=-1/10, high=1/10, size=n)
```

[30]:
```
#desired function
d = np.zeros(n)
for i in range(len(d)):
    d[i] = math.sin(20*x[i]) + 3*x[i] + v[i]

t = []
for j in range(0,i+1):
        t.append(j)
drawGraph(x,d, 'Desired function')
```

```
    intermediate_activation = np.matmul(w_L1, X_input)



    #apply the hyperbolic tangent function
    intermediate_output = np.tanh(intermediate_activation)
    #print('intermediate_output shape', intermediate_output.shape)
    #intermediate output goes to second layer

    #adding bias in intermediate output
    intermediate_output_bias = np.ones([25,1])
    k = 1
    for j in intermediate_output:
        intermediate_output_bias[k][0] = j
        k = k + 1


    #print('dimensions of w_l2', w_L2.shape)
    #print('dimensions of intermediate_output_bias', intermediate_output_bias.
 ↪shape )
    final_out = np.matmul(w_L2, intermediate_output_bias)
    return intermediate_activation, intermediate_output_bias, final_out
```

[403]:
```
#identical to forward pass; only returns final_out

def NNfval(w_L1, w_L2, x):
    X_input = np.ones(2)

    #adding bias term in input
    X_input[1] = x
    X_input = np.reshape(X_input, (2,1))
    print('x_input shape is :', X_input.shape)
    print('x_input ',X_input )
    intermediate_activation = np.matmul(w_L1, X_input)


    #apply the hyperbolic tangent function
    intermediate_output = np.tanh(intermediate_activation)
    #print('intermediate_output shape', intermediate_output.shape)
    #intermediate output goes to second layer

    #adding bias in intermediate output
    intermediate_output_bias = np.ones([25,1])
    k = 1
    for j in intermediate_output:
        intermediate_output_bias[k][0] = j
```

```
        k = k + 1


    #print('dimensions of w_l2', w_L2.shape)
    #print('dimensions of intermediate_output_bias', intermediate_output_bias.
 ↪shape )
    final_out = np.matmul(w_L2, intermediate_output_bias)
    return final_out
```

[420]:
```
#testing forward pass - test2
test_w_L1 = np.ones((24,2))
test_w_L2 = np.ones((1,25))
test_di = np.ones(300) * 10

x_test = np.ones(300)
print(energy(test_w_L1, test_w_L2, x_test, test_di))
```

```
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
```

```
x_input  [[1.]
 [1.]]
x_input shape is : (2, 1)
x_input  [[1.]
 [1.]]
199.845210291824
```

[419]:
```python
#Energy calculation function
# params: weight_layer1, weights_layer2

#calls forward pass and finds the output of the neural network with given␣
 ↪weights

def energy(weights_L1, weights_L2):
    sum = 0.0
    for i in range(n):
        diff_term = d[i] - NNfval(weights_L1, weights_L2, x[i])[0][0]
        #print (diff_term)
        #print(type(diff_term))
        #print('Shape of diff term',diff_term.shape)
        #print(diff_term)
        intermediate_sum = math.pow(diff_term,2)
        sum = sum + intermediate_sum
    return sum/n
```

[384]:
```python
#other energy function
def f(w1,w2,x,y):
    #print("w0:",w[0])
    #print(w2)
    b=w2[:,0]
    w2  = np.delete(w2, 0,1)
    #print(w2.shape)
    #w2 = np.reshape(w2,(24,1))
    #print(w2)
    funcn = 0
    v= np.zeros([24,300])
    z=np.zeros([300,1])
    for j in range(0,24):
        #print((math.pow((y[ti]-(w[0]+(w[1]*x[i]))),2)))
        for i in range(0,300):
            v[j][i] = math.tanh(w1[j][1]*x[i]+w1[j][0])
    #print("v=",v)
    for i in range(0,300):
        #print("w2=",w2[:,j])
        #print(np.dot(w2,v[:,i])+b)
        z[i]=(np.dot(w2,v[:,i])+b)
        funcn = funcn+(math.pow((y[i]-z[i]),2))
    #print("z=",z)
```

```
    #print(funcn)
    return funcn/300
```

[444]:
```
#main loop

#issues are definitely here
def gdBackprop(w_L1, w_L2, eta):
    print("eta= ", eta)

    #set epoch to 0
    epoch = 0
    w1_old = w_L1-10*5
    w2_old = w_L2-10*5
    mse = f(w_L1, w_L2, x, d)
    print('Mse at start of backprop is : ',mse)
    mse_list = []
    mse_list.append(mse)
    epoch_list = [epoch]
    threshold = math.pow(10,-6)

    print(norm(f(w1_old,w2_old,x,d)-f(w_L1,w_L2, x, d))> threshold)
    while (norm(f(w1_old,w2_old,x,d)-f(w_L1,w_L2, x, d))> threshold):
        epoch = epoch + 1
        w1_old = w_L1
        w2_old = w_L2
        for i in range(n):
            #going forward
            intermediate_activation, intermediate_output_bias, final_output =␣
↪forward_pass(w_L1, w_L2, x[i])

            #going backward
            delta2 = 2*(d[i] - final_output[0][0])

            tanh_activation_derivative = 1 - np.square(np.
↪tanh(intermediate_activation))
            tanh_activation_derivative = np.reshape(tanh_activation_derivative,␣
↪(24,1))


            product_w_L2delta2 = np.multiply(w_L2[:,1:25].T,delta2)

            delta1 = np.multiply(product_w_L2delta2, tanh_activation_derivative)
            #print("delta 1 is : ", delta1)

            dEdW2 = -np.dot(delta2, intermediate_output_bias.T)
            #print('Shape of dedW2', dEdW2.shape)
            input_matrix = np.ones([2,1])
```

```
            input_matrix[1][0] = x[i]

            #print('Shape of input matrix', input_matrix.shape)

            dEdW1 = -np.dot(delta1, input_matrix.T)
            #print('Shape of dedW2', dEdW1.shape)







            #TO-DO: try different etas for different layers
            product_intermediate = (eta*delta2) * intermediate_output_bias.
 ↪transpose()
            #print('Shape of product_intermediate', product_intermediate.shape)


            #update weights
            w_L2 = np.subtract(w_L2, eta*dEdW2)
            #print("new W_L2: ", w_L2)


            input_matrix_transpose = input_matrix.transpose()
            w_L1 = w_L1 - (eta*dEdW1)
            #print("new W_L1: ", w_L1)

            #check mse with new weights to check if we're overshooting
            #correction: don't check mse here
        mse = f(w_L1, w_L2, x, d)
        print("MSE after epoch: ",epoch, "  " , mse)
        mse_list.append(mse)
        if(mse_list[epoch-1]<mse_list[epoch]):
            eta = (0.9*eta)
            w_L1,w_L2,mse_list,epoch=gdBackprop(w1_old,w2_old,eta)
            return w1,w2,mse_list,epoch

    return w_L1, w_L2, mse_list, epoch
```

[445]:
```
# driver code

#set initial weights
w_L1 = np.random.normal(loc=0.0, scale=1, size=[24,2])
```

```python
w_L2 = np.random.normal(loc=0.0, scale=1, size=[1,25])
w_L2 = w_L2 * np.sqrt(1/(24))
#set eta
eta = float(input("Enter the learning rate:"))
final_w_L1, final_w_L2, energy_list, final_epochs = gdBackprop(w_L1, w_L2, eta)
```

```
Enter the learning rate:0.01
eta=  0.01
Mse at start of backprop is :  4.203640318253139
True
MSE after epoch:  1    0.4808846821261425
MSE after epoch:  2    0.4765606392578042
MSE after epoch:  3    0.4747810281503644
MSE after epoch:  4    0.47321383547554896
MSE after epoch:  5    0.47181524376716727
MSE after epoch:  6    0.47056072934479287
MSE after epoch:  7    0.469430135238671
MSE after epoch:  8    0.46840655421570676
MSE after epoch:  9    0.4674757660587711
MSE after epoch:  10    0.4666257827746635
MSE after epoch:  11    0.465846473215178
MSE after epoch:  12    0.4651292533355078
MSE after epoch:  13    0.4644668306417004
MSE after epoch:  14    0.4638529932331255
MSE after epoch:  15    0.46328243547443143
MSE after epoch:  16    0.4627506137227391
MSE after epoch:  17    0.462253626701408
MSE after epoch:  18    0.46178811607667464
MSE after epoch:  19    0.4613511835868496
MSE after epoch:  20    0.4609403217244358
MSE after epoch:  21    0.46055335550576726
MSE after epoch:  22    0.4601883933035095
MSE after epoch:  23    0.45984378508370716
MSE after epoch:  24    0.4595180866967354
MSE after epoch:  25    0.45921002913240555
MSE after epoch:  26    0.45891849187253475
MSE after epoch:  27    0.45864247966531424
MSE after epoch:  28    0.4583811022082794
MSE after epoch:  29    0.45813355636212855
MSE after epoch:  30    0.4578991106263784
MSE after epoch:  31    0.45767709168994725
MSE after epoch:  32    0.45746687292546784
MSE after epoch:  33    0.45726786472692793
MSE after epoch:  34    0.4570795065989685
MSE after epoch:  35    0.45690126089712907
MSE after epoch:  36    0.4567326080972585
MSE after epoch:  37    0.45657304344532607
```

```
MSE after epoch:  3782    0.007640239090841078
MSE after epoch:  3783    0.0076392107565715775
MSE after epoch:  3784    0.007638183111473399
MSE after epoch:  3785    0.007637156154759572
MSE after epoch:  3786    0.007636129885644641
MSE after epoch:  3787    0.00763510430334274
MSE after epoch:  3788    0.007634079407069811
MSE after epoch:  3789    0.007633055196045391
MSE after epoch:  3790    0.007632031669487588
MSE after epoch:  3791    0.007631008826617216
MSE after epoch:  3792    0.007629986666655933
MSE after epoch:  3793    0.007628965188827503
MSE after epoch:  3794    0.0076279443923564305
MSE after epoch:  3795    0.007626924276469481
MSE after epoch:  3796    0.007625904840392585
MSE after epoch:  3797    0.007624886083354252
MSE after epoch:  3798    0.007623868004586491
MSE after epoch:  3799    0.007622850603319251
MSE after epoch:  3800    0.007621833878784951
MSE after epoch:  3801    0.007620817830218153
MSE after epoch:  3802    0.0076198024568544164
MSE after epoch:  3803    0.007618787757929981
MSE after epoch:  3804    0.0076177737326830865
MSE after epoch:  3805    0.007616760380352729
MSE after epoch:  3806    0.007615747700178888
MSE after epoch:  3807    0.00761473569140534
MSE after epoch:  3808    0.007613724353273124
MSE after epoch:  3809    0.007612713685028044
MSE after epoch:  3810    0.00761170368591536
MSE after epoch:  3811    0.007610694355182665
MSE after epoch:  3812    0.007609685692077338
MSE after epoch:  3813    0.0076086776958501
MSE after epoch:  3814    0.007607670365751242
MSE after epoch:  3815    0.007606663701033224
MSE after epoch:  3816    0.0076056577009499145
MSE after epoch:  3817    0.0076046523647547845
MSE after epoch:  3818    0.007603647691705574
MSE after epoch:  3819    0.007602643681057662
MSE after epoch:  3820    0.00760164033207148
MSE after epoch:  3821    0.007600637644006157
MSE after epoch:  3822    0.0075996356161223
MSE after epoch:  3823    0.007598634247683068
MSE after epoch:  3824    0.007597633537951235
MSE after epoch:  3825    0.007596633486192473
MSE after epoch:  3826    0.00759563409167159
```

[446]:
```
final_w_L1
```

[446]:
```
array([[ 5.76338341e+00, -7.12943283e+00],
       [-2.55007083e-01,  7.66357460e-01],
       [ 3.63470761e+00, -1.14346581e+01],
       [-8.30586451e-02,  2.38364933e-01],
       [-6.79171136e+00,  1.07622682e+01],
       [ 2.14501969e-02,  8.25079007e+00],
       [-5.39679553e-02,  1.53456868e-01],
       [ 4.02475560e-01, -1.13546955e+00],
       [-9.82542747e-02,  2.83636722e-01],
       [-8.30016429e+00,  9.06081090e+00],
       [-9.92268107e-01,  7.97111617e+00],
       [ 2.90215395e+00, -6.21754717e+00],
       [ 2.23724286e-01, -6.72965734e-01],
       [ 1.28343983e-03, -3.62304986e-03],
       [-9.62977628e-02,  2.77769867e-01],
       [ 2.41017102e-02, -6.81378805e-02],
       [ 2.96908397e-02, -8.40027521e-02],
       [-3.80225588e-02,  1.07727563e-01],
       [ 1.25481688e-01, -3.66431246e-01],
       [ 1.03322888e-01, -2.98888096e-01],
       [ 2.14282022e-01, -6.43984274e-01],
       [-1.25167580e-01,  3.65464394e-01],
       [-1.34409871e-01,  3.94015341e-01],
       [-1.33228791e-02,  3.76263987e-02]])
```

[452]:
```
final_w_L2
```

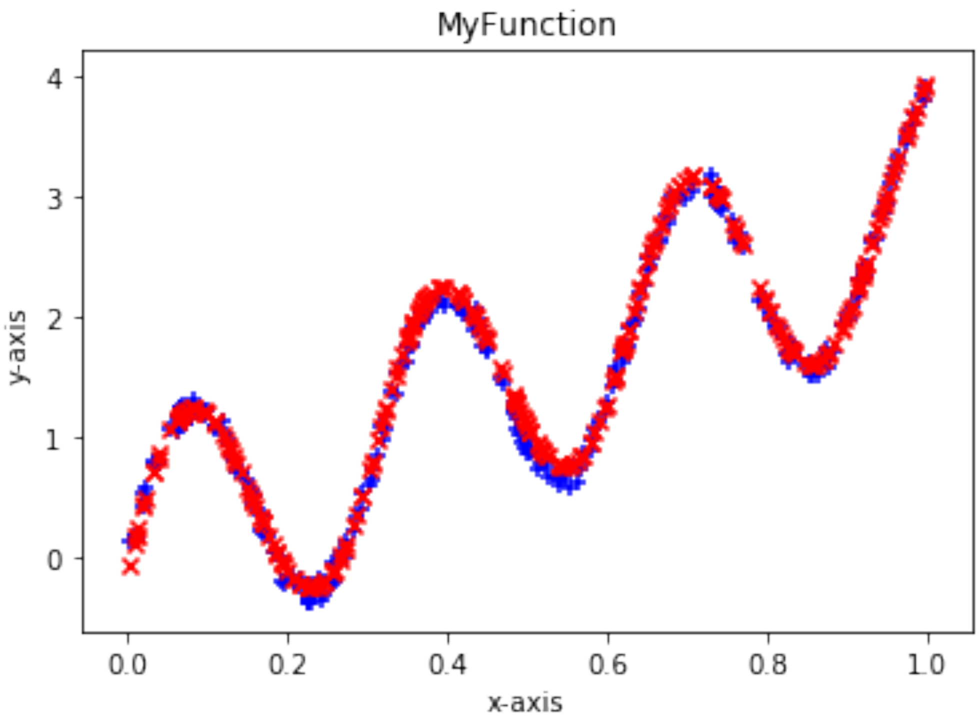
[452]:
```
array([[ 3.02272310e-01,  4.68061986e+00, -1.30965258e-01,
        -3.52413306e+00, -3.46889393e-02,  3.92110212e+00,
         5.18870223e+00, -2.18378436e-02,  2.09565977e-01,
        -4.18692541e-02,  4.38022017e+00, -3.70021420e+00,
         3.62576647e+00,  1.12479338e-01,  5.06505740e-04,
        -4.09250074e-02,  9.56040355e-03,  1.18084090e-02,
        -1.51960181e-02,  5.56354468e-02,  4.43432778e-02,
         1.06822261e-01, -5.54699969e-02, -6.04007503e-02,
        -5.26603949e-03]])
```

[450]:
```
myfunction = np.zeros(n)
for i in range(n):
    myfunction[i] = NNfval(final_w_L1, final_w_L2, x[i])
```

```
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.73885965]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.16605833]]
x_input shape is : (2, 1)
x_input  [[1.       ]
```

```
 [0.63606938]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.10905234]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.07513531]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.98314979]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.34320985]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.89906777]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.44666924]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.54286185]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.57926749]]
x_input shape is : (2, 1)
x_input  [[1.        ]
 [0.62671393]]
```

[451]:
```
drawAll(x, myfunction, 'MyFunction')
```



[259]:
```
#test forward pass again - why getting a straight line?

test_wl1 = np.ones((24,2))
test_wl2 = np.ones((1,25))
test_i = 0.5
test_output = NNfval(test_wl1, test_wl2, test_i)
print(test_output)
print(math.tanh(1.5)*24 + 1)
```

```
[[22.72355809]]
22.723558087476793
```

[252]:
```
print(x[0])
myfunction[0]
```

```
0.7388596462466387
```

[252]: 2.100389621054921

[232]:
```
#draw both Graphs
def drawAll(x, y, title):
    plt.scatter(x, d, color = 'blue', marker = '+')
    plt.scatter(x, y, color = 'red', marker = 'x')
```

```python
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')

    plt.title(title)
    plt.show()
```

[ ]:

```python

```