Question statements can be found in separate pdf

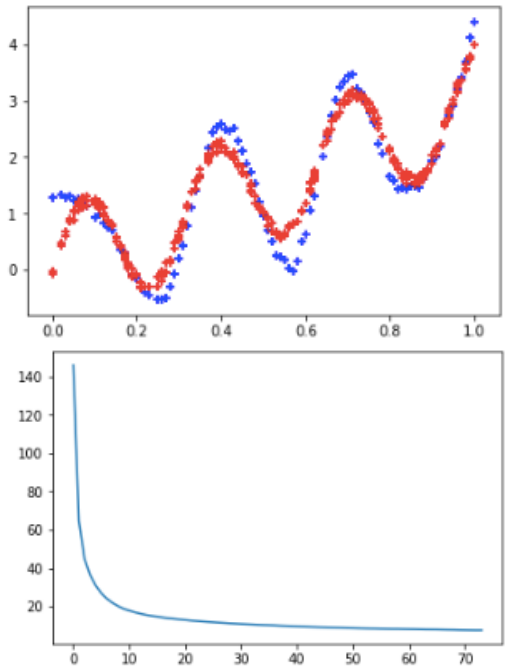
Q1) – Curve Fitting

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from math import *

n=300
def get_random_in_range(start_range, end_range):
    return round(np.random.uniform(start_range, end_range), 2)

def get_real_numbers(n):
    x = list()
    v = list()
    for i in range(n):
        x.append(get_random_in_range(0, 1))
        v.append(get_random_in_range(-0.1, 0.1))
    return x, v


x, v = get_real_numbers(n)

d = list()

for i in range(n):
    d.append(sin(20*x[i]) + 3*x[i] + v[i])

plt.scatter(x, d, color='b', marker='+')
plt.xlabel("Xi")
plt.ylabel("di")
# plt.show()
#initialise Weight vector for 3N+1 neurons
s = -12
e = 12
w_l1 = np.array([[get_random_in_range(s, e), get_random_in_range(s, e)]])
w_l2 = np.array([get_random_in_range(s, e)])
N=24
for i in range(N-1):
    temp = np.array([[get_random_in_range(s, e), get_random_in_range(s, e)]])
    w_l1 = np.concatenate((w_l1, temp))
    w_l2 = np.append([w_l2],[[get_random_in_range(s, e)]])
print(w_l2.shape)
w_l2 = np.append(w_l2,[get_random_in_range(s, e)])
# w_l1 = np.random.rand(24,2)*np.sqrt(1/(24))

def find_differen_induced_local_l1(v):
    #we dont need weight_l2 in this example because phi'(V2)=1 always
    return 1.0 - np.tanh(v)**2
```

```
eta = 0.0001

mse_list = []
while True:
    mse = 0
    y=[]
    for i in range(300):
        w_l2=w_l2.reshape(25,1)
        v1 = np.matmul(w_l1, np.array([[1,x[i]]]).T)

        y1 = np.round(np.tanh(v1),2)
        #print(y1.shape)

        y2=np.matmul(w_l2.T, np.append(1,y1.T).reshape(25,1))
        #print(y2.shape)
        y2 = y2[0][0]
        y.append(y2)
        diff_v1 = np.vectorize(find_differen_induced_local_l1)
        diff_v1= diff_v1(v1)
#         grad_l1 = np.array([])
#         grad_l2 = np.array([-(d[i]-y2)])
        grad_l1 = np.array([[-(d[i]-y2)*1*w_l2[1][0]*diff_v1[0][0], -x[i]*(d[i]-y2)*1*w_l2[1][0]*diff_v1[0][0]]])
        grad_l2 = np.array([-(d[i]-y2), -y1[0]*(d[0]-y2)])
        for j in range(1,24):
            temp = np.array([[-x[i]*(d[i]-y2)*1*w_l2[j+1][0]*diff_v1[j][0], -(d[i]-
y2)*1*w_l2[j+1][0]*diff_v1[j][0]]])
            grad_l1 = np.concatenate((grad_l1, temp))
            grad_l2 = np.append(grad_l2, -y1[j]*(d[i]-y2))
        grad_l2 = grad_l2.reshape(25,1)
    #     print(w_l2.shape)
#         print(grad_l1)
        w_l1 = w_l1-eta*grad_l1
        w_l2 = w_l2-eta*grad_l2
        mse+=(d[i]-y2)**2
    #     print(w_l2.shape)
    mse = mse/300
    mse_list.append(mse)
    if len(mse_list)>1:
        print(mse_list)
        if mse_list[-1]>mse_list[-2]:
            eta = eta*0.8
        if mse_list[-2]-mse_list[-1]<0.00000000001:

            break


final_y = []
for i in range(300):
    w_l2=w_l2.reshape(25,1)
    v1 = np.matmul(w_l1, np.array([[1,x[i]]]).T)
    y1 = np.round(np.tanh(v1),2)
```

```
    #print(y1.shape)
    y2=np.matmul(w_l2.T, np.append(1,y1.T).reshape(25,1))
    #print(y2.shape)
    y2 = y2[0][0]
    final_y.append(y2)

plt.scatter(x, final_y, color='b', marker='+')
plt.scatter(x, d, color='r', marker='+')
```



Red points are Output after training
Blue points are desired output
X axis is X_i
Y axis is $f(x, W_0)$

Pseudocode for Q1:

1. Number of inputs 'n' = 300
2. Take n random numbers on [0,1] represent by X : (x1, x2, ….xn)
3. Take n random numbers on [0,1] represent by u : (u1, u2,…un)
4. $d_i = \sin(20x_i)+3x_i+u_i$, i = 1 to n
5. Plot points (x_i, d_i). this is the curve to match
6. We are taking 1 input neuron, 24 hidden layer neurons and 1 output layer neuron
7. Initialize weights for 2 layers (W1 and W2)
8. W1 $\in R^{24 X 2}$ (With Biases for 24 neurons)
9. W2 $\in R^{25X1}$ (Including Bias of output neuron)
10. Take learning parameter eta = 0.001
11. Initialize 'mse_list' as empty list
12. While True:
    a. Initialize mse = 0

b. For i = 0 to 299
    i. Check shapes of matrices and convert to required shapes
    ii. Reshape W2 to 25 X 1
    iii. Find Induced local field for layer 1 (v1)= W1 * $[[1, x_i]]^T$ *(W1 is 24X2, $[[1,x_i]]$ is 2X1)*
    iv. Activation function: tanh() such that ϕ(v1) = tanh(v1)
    v. Find Output of first layer (Y1) = tanh(v1)
    vi. Second layer activation function is ϕ(v2)=v2
    vii. Second Layer output Y2 = v2 * 1= W2.T * np.append(1,$y1^T$)
    viii. Reshape to 25X1 (if required)
    ix. Find Φ'(v1) = 1.0 - np.tanh(v1)**2
    x. Find gradient
    xi. Gradient for layer1 g1
    xii. For j in range(24)
        1. Append to g1 values of differentiation of energy with respect to each weight and bias of layer 1
           Bias Update formula: [-(d[i]-y2)*1*w_l2[j+1][0]*diff_v1[j][0]]
           Weights update formula: [-x[i]*(d[i]-y2)*1*w_l2[j+1][0]*diff_v1[j][0]
           Append to g1(Bias Update formula, Weights update formula)
        2. Similarly, append to g2 values of differentiation of energy with respect to each weight and bias of layer 2
    xiii. Update weights for each layer
    xiv. W1 = W1-eta*g1
    xv. W2 = W2-eta*g2
    xvi. Mse+= (d[i]-y2)**2
c. Mse=mse/n
d. Append mse to mse_list
e. If length of mse_list > 1:
    i. if mse_list[-1]>mse_list[-2]:
        1. eta = eta*0.8 //This will not occur mostly
    ii. if mse_list[-2]-mse_list[-1]<0.00000000001:
        1. break

13. Plot graph between input (x) and final value of y2 for updated weights

Q2) - Neural network for digit classification using the backpropagation algorithm

2. In this computer project, we will design a neural network for digit classification using the backpropagation algorithm (see the notes above). You should use the MNIST data set (see Homework 2 for details) that consists of 60000 training images and 10000 test images. **The training set should only be used for training, and the test set should only be used for testing.** Your final report should include the following:

   - The details of your architecture/design, including
     - Your network topology, i.e. how many layers, how many neurons in each layer, etc. (Obviously you will have 784 neurons in the input layer).
     - The way you represented digits $0, \ldots, 9$ in the output layer. For example, one may use the same setup as in Homework 2, 10 output neurons, with $[1\ 0\ \cdots 0]$ representing a 0, $[0\ 1\ 0 \cdots 0]$ representing a 1 and so on. Another person may have just one output neuron with an output of 0 representing a 0, an output of 1 representing a 1, and so on.
     - Neuron activation functions, learning rates for each neuron, and any dynamic update of learning rates (as explained in the question above) if it exists.
     - The energy/distance functions of your choice.

Code:

```
import gzip
import numpy as np
from math import *
import matplotlib.pyplot as plt

# accepts the mnist image input in zipped format and return np array
def get_data(filename, data_size, type='img'):
    if type == 'img':
        F_SIZE = 16
    elif type == 'label':
        F_SIZE = 8
    with gzip.open(filename) as f:
        f.read(F_SIZE)
        data_buffer = f.read(data_size[0] * data_size[1] * data_size[2])
        if type == 'img':
            data = np.frombuffer(data_buffer, dtype=np.uint8).astype(np.float32)
            data = data.reshape(data_size[2], data_size[1], data_size[0])
        elif type == 'label':
            data = np.frombuffer(data_buffer, dtype=np.uint8).astype(np.float64)
            # print(data)
        return data


def create_desired_output_vector(index):
    # print(index)
    zero_list = np.zeros(10)
    zero_list[index] = 1
    return zero_list.reshape(10, 1)
#Activation function to find sigmoid function
def get_sigmoid_out(alpha, v):
    return 1.0/(1.0 + exp(-alpha*v))

# train the neural network model for 60000 training images of size 28*28
train_data = get_data('/Users/kislaya/Documents/UIC/NN/train-images-idx3-ubyte.gz', [28, 28, 60000], 'img')
train_labels = get_data('/Users/kislaya/Documents/UIC/NN/train-labels-idx1-ubyte.gz', [1, 1, 60000], 'label')
epoch_number = 0
# n = 400
n = 60000
# n = 1000
param = 10
epoch_errors = dict()
```

```
epoch_test_errors = dict()


thresh = 0.085
epoch_errors[epoch_number] = 0
epoch_test_errors[epoch_number] = 0

w_l1 = np.random.uniform(low=-1, high=1, size=(400,785))
w_l2 = np.random.uniform(low=-1, high=1, size=(401,10))
alpha = 0.01
test_data = get_data('/Users/kislaya/Documents/UIC/NN/t10k-images-idx3-ubyte.gz', [28, 28, 10000], 'img')
test_labels = get_data('/Users/kislaya/Documents/UIC/NN/t10k-labels-idx1-ubyte.gz', [1, 1, 10000], 'label')



def find_differen_induced_local_l1(alpha, v):
    return alpha*exp(-alpha*v)/((1+exp(-alpha*v))**2)

N=400
alpha = 0.01
eta = 1

energy_list = []
y=[]
while True:
    epoch_errors[epoch_number] = 0
    epoch_test_errors[epoch_number] = 0
    print("EPOCH")
    print(epoch_errors)
    n_errors = 0

    energy = 0
    for i in range(n):
#         print(i)
        x = train_data[i].reshape(784, 1)
        des = create_desired_output_vector(int(train_labels[i]))

        w_l2=w_l2.reshape(N+1,10)
        v1 = np.matmul(w_l1, np.append(1,x))
        temp = np.vectorize(get_sigmoid_out)
        y1 = temp(alpha, v1)
        v2=np.matmul(w_l2.T, np.append(1,y1.T).reshape(N+1,1))
        y2 = temp(alpha, v2)

        my_out = np.argmax(y2)
        y.append(my_out)
        if int(train_labels[i])!=my_out:
            epoch_errors[epoch_number]+=1
        diff_v = np.vectorize(find_differen_induced_local_l1)
        diff_v1= diff_v(alpha, v1)
        diff_v2 = diff_v(alpha, v2)


        delta2 = np.multiply((des-y2), diff_v2)

        diff_v1 = diff_v1.reshape(N,1)

        delta1 = np.multiply(np.matmul(w_l2, delta2)[1:], diff_v1)

        biased_y1 = np.append(1,y1).reshape(N+1,1).T

        biased_y0 = np.append(1,x).reshape(1, 785)



        g2 = np.matmul(-delta2, biased_y1)

        g1 = np.matmul(-delta1,biased_y0 )
        w_l1 = w_l1-eta*g1
        w_l2 = w_l2-eta*g2.T
#         print(g2.shape)
```

```
#       print(np.linalg.norm((des-y2))**2)
        energy+= np.linalg.norm((des-y2))**2

    energy = energy/n
    print("**********************************")
    print(energy)
    energy_list.append(energy)
#    if len(energy_list)>1:
#        print(energy_list[-2]-energy_list[-1])
#        if energy_list[-1]<0.1:
#            print("**********")
#            break

    errors = 0

    for i in range(10000):
        x_test = test_data[i].reshape(784, 1)
        des_test = create_desired_output_vector(int(test_labels[i]))
        w_l2=w_l2.reshape(N+1,10)
        v1_test = np.matmul(w_l1, np.append(1,x_test))
        temp = np.vectorize(get_sigmoid_out)
        y1_test = temp(alpha, v1_test)
        v2_test=np.matmul(w_l2.T, np.append(1,y1_test.T).reshape(N+1,1))
        y2_test = temp(alpha, v2_test)
        my_out = np.argmax(y2_test)
        if int(test_labels[i])!=my_out:
            epoch_test_errors[epoch_number]+=1
            errors+=1
    epoch_number+=1
    print("TEST ERRORS")
    print(errors)
    if errors<499:
        break
```

The reasons as to why you chose a particular hyperparameter the way it is (e.g. why did you choose 100 hidden neurons, but not 50, why do you have 1 hidden layer but not 2?)

I tried with 800 neurons, 400 neurons and 50 neurons. All had their advantages and disadvantages.

When I took 800 neurons, it was very slow in processing and I had to shift to 400 neurons.

Comparision between 400 and 50 Neurons in hidden layer:

In case of 400 neurons, its execution of code was slower than 50 neurons but it had faster rate of learning. By rate of learning, I mean it takes less number of epochs as compared to 50 neurons.

I also chose 1 hidden layer in order to avoid any type of overfitting.

Comments about my design process:

I tuned following parameters:

- Alpha (Parameter of Sigmoid activation function)
- eta (Learning Parameter for weight update)
- Weights initialisation

I was facing some issues while taking bigger weights. One of the issue was Bigger values of induced local fields which gives out of range issue with Sigmoid function.

Setting alpha as 0.01 solved this issue of out of range error to some extent.

Most important parameter was eta. I tried bigger values of eta and it was no where taking me to the right direction (diverging). I also tried very small values but it was taking lot of time. I finally took eta=1 which worked perfectly for me.

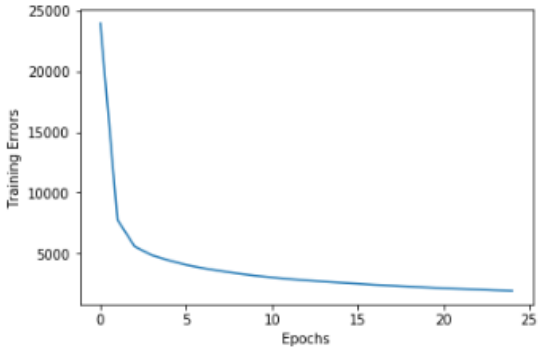
Pseudocode Q2:

1. Read MNIST data from stored .gz input files
   Train_data -> 60000*28*28
   train_labels – 1*1*60000
   test_data – 28*28*10000
   test_labels = 1*1*10000
2. Initialize a dictionary : epoch_errors with keys as epoch_number and values as count of errors
3. Initialize weights for 2 layers (W1 and W2)
4. W1 $\in R^{400X785}$ (With Biases for 24 neurons)
5. W2 $\in R^{401X10}$ (Including Bias of output neuron)
6. Take learning parameter eta = 0.001
7. y = []
8. sigmoid parameter alpha = 0.001
9. while True:
   a. Train the model here
   b. for i in 1 to n
      i. x = train_data[i]
      ii. des = Binary vector form of train_labels[i]
      iii. Induced local field v1 = W1 * $[1,x]^T$
      iv. Vector y1 = [for all v 1.0/(1.0 + exp(-alpha*v))]
      v. Similarly, v2 = $W2^T$ * [1,y1]
      vi. y2 = [for all v 1.0/(1.0 + exp(-alpha*v))]
      vii. My_out = np.argmax(y2)
      viii. Append my_out in y
      ix. If train_labels[i]!=my_out:
         1. epoch_errors[epoch_number]+=1
      x. find Φ'(v1) for all v in v1 find alpha*exp(-alpha*v)/((1+exp(-alpha*v))**2)
      xi. similarly find Φ'(v2)
      xii. delta2 = hadamard multiplication (des-y2), Φ'(v2))
      xiii. delta1 = (W2 * delta2)[1 to end] * Φ'(v1)
      xiv. g2 = -delta2 * $[1\ y1]^T$
      xv. g1 = -delta1 * $[1\ y0]^T$
      xvi. Make sure shapes are same
      xvii. Update weights for each layer:
         W1 = W1-eta*g1
         W2 = W2-eta*g2.T
   c. Epoch_number+=1
   d. Energy=energy/n
   e. Start Testing
   f. Errors=0
   g. For i in 0 to 9999

i. X_test = test_data -> 784X1 reshaped
ii. Des_test = Binary vector form of test_labels[i]
iii. v1_test = W1 * [1,x_test]^T
iv. Induced local field v1_test = W1 * [1,x_test]^T
v. Vector y1_test = [for all v 1.0/(1.0 + exp(-alpha*v))]
vi. Similarly, v2_test = W2^T * [1,y1_test]
vii. y2_test = [for all v 1.0/(1.0 + exp(-alpha*v))]
viii. my_out = np.argmax(y2_test)
ix. if int(test_labels[i])!=my_out:
    1. errors+=1

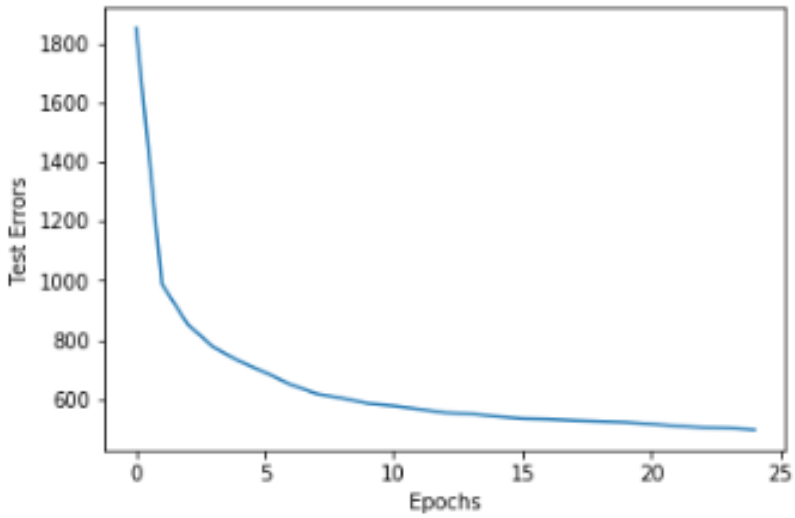
h. if errors<499:
    i. break

```
In [649]: plt.plot(list(epoch_errors.keys()), epoch_errors.values())
          plt.xlabel('Epochs')
          plt.ylabel('Training Errors')
Out[649]: Text(0,0.5,'Training Errors')
```



Number of Epochs Vs Training Errors

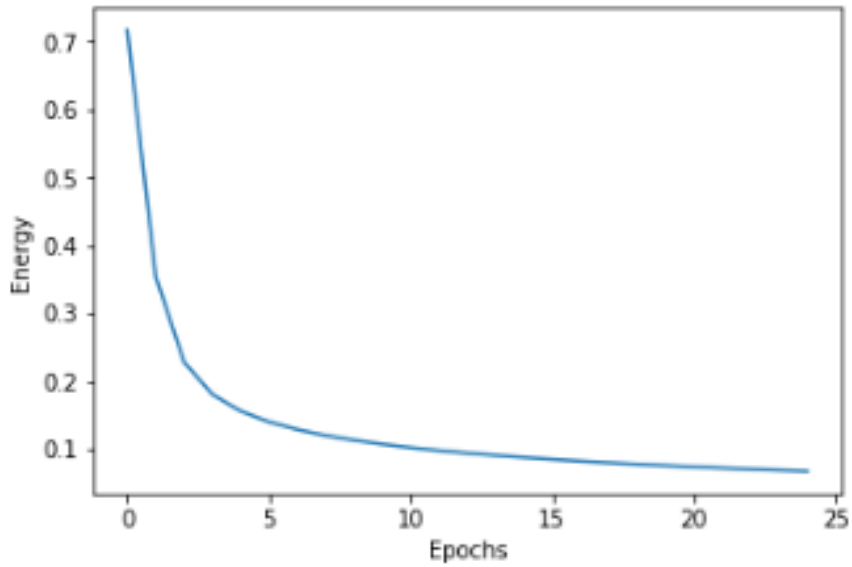
```
: plt.plot(list(epoch_test_errors.keys()), epoch_test_errors.values())
  plt.xlabel('Epochs')
  plt.ylabel('Test Errors')

: Text(0,0.5,'Test Errors')
```



Number of Epochs Vs Test Errors

```
]: plt.plot(list(range(len(energy_list))), energy_list)
   plt.xlabel('Epochs')
   plt.ylabel('Energy')
```

```
]: Text(0,0.5,'Energy')
```



Number of Epochs Vs Training Energy