HW5

Q1.

Network Topology

Architecture Details:

For the digit classification of MNIST dataset, I have used 2-layer neural network, 784 * 24 * 10. So, the output vector represents a vector for the digits Eg ([1 0 0 ..]) for 0. Activation function is hyperbolic tangent activation function as represents the value within the range of 0-1 and in our case, we require the output vector to be between 0-1. Furthermore, hyperbolic tangent gives a probabilistic value for the input within (0 1), hence making it easier to classify digits. The no of nodes for the hidden layer has been based on the previous setup. I have also normalized the input vector as tanh(x) was giving math error with raw input as pixel value range from 0-255.

Design Process:

So initially I started with un-normalized data that led to math error in case of tanh(x) function. Also, initially I took step function but that had zero gradient so the back-propagation algorithm could not work in that scenario. So, I adjusted the learning rate to 0.1 – 0.001 as initially the learning rate was too high that led to overflow of values and the it was converging. Also, it took a lot of time for one epoch to run. So, the value of eta chosen was less than 10. I chose 8 to start the program.

PSEUDOCODE

0) import all the dependency classes.

1) function to read the idx files for training & testing data.

   1.1) Using tuple unpack to unpack the data.

   1.2) Using numpy frombuffer to read the data.

2) Normal distribution of weight initialization.

3) Feed forward activation function. - hyperbolic tangent defined.

   3.1) feedback activation function - hyperbolic tangent.

4). Run training loop while (True).

5) for loop for training & testing

   for ~~x~~ $i < len(data)$.

Normalize data → feedforward update → layer1 append.

feedforward update → layer 2 append.

6) Check correct training sample.

$$x = norm\ (trainlabel - New\ output)^2$$

7) Find weight updates for layer 1 & layer 2.

~~8~~ 8) Repeat 3) to 7) for testing data.

9). Exit while loop if ~~accuracy~~ accuracy > 0.95.

10) plot graph of Number of epochs vs No of misclassification.

SOURCE CODE

```
import os
import path
import struct
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize


# Source for reading the idx files as numpy arrays: https://gist.github.com/tylerneylc
def read_idx(filename):
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)

# Data Source: http://yann.lecun.com/exdb/mnist/
train_data = read_idx('train-images.idx3-ubyte')
train_labels = read_idx('train-labels.idx1-ubyte')
test_data = read_idx('t10k-images.idx3-ubyte')
test_labels = read_idx('t10k-labels.idx1-ubyte')

def convert_labels(data):
    new_data = []
    for i in range(len(data)):
        temp = [0]*10
        temp[data[i]] = 1
        new_data.append(temp)
    return new_data

train_labels = convert_labels(train_labels)
test_labels = convert_labels(test_labels)

# Based on Xavier Normal initialization
w_input = np.random.uniform(low=-0, high=1, size=(784, 10))* np.sqrt(6/(784+10))
w_layer1 = np.random.uniform(low=-0, high=1, size=(10, 10))* np.sqrt(6/(10+10))
w_layer1_bias = np.random.uniform(low=-0, high=1, size=(10,1))* np.sqrt(6/(10+10))
w_layer2_bias = np.random.uniform(low=-0, high=1, size=(10,1))* np.sqrt(6/(10+10))

# feed-forward activation functions - hyperbolic tangent
def act_fun(v):
    return np.tanh(v)

# feedback activation function - hyperbolic tangent
def derv_act_fun(v):
    return (1 - np.tanh(v)**2)

def feedforward(input_data, bias, weight):
    local_ind_field = np.dot(weight.T,input_data) + bias
    output = act_fun(local_ind_field)
```

```python
    return local_ind_field, output

eta = 4
training = []
testing = []
energy_training = []
energy_testing = []
while(True):
    train_correct = 0
    test_correct = 0
    layer1_local_field = []
    layer1_output = []
    layer2_local_field = []
    layer2_output = []
    for i in range(len(train_data)):
        xi = train_data[i]
        xi.resize(784, 1)
        xi = normalize(xi)
        local_ind_field, output = feedforward(xi, w_layer1_bias, w_input)
        layer1_local_field.append(local_ind_field)
        layer1_output.append(output)

        local_ind_field, output = feedforward(output, w_layer2_bias, w_layer1)
        layer2_local_field.append(local_ind_field)
        layer2_output.append(output)

        max_index = output.argmax(axis=0)[0]
        new_output = [0]*10
        new_output[max_index] = 1
        x = np.linalg.norm(np.asarray(train_labels[i]) - np.asarray(new_output))**2
        if x==0:
            train_correct += 1
        e = 2 * np.subtract(np.asarray(train_labels[i]).reshape(10,1), output)/len(train_data)

        local_ind_field = local_ind_field.reshape(10,)
        e = e.reshape(10,)
        w_layer2_bias_grad = - eta * np.asarray([e[i]*derv_act_fun(local_ind_field)[i]
        for i in range(10)]).reshape(10,1)

        w_layer1_grad = - eta * np.dot(layer1_output[i],
                                       np.asarray([e[i]*derv_act_fun(local_ind_field)[i]
        for i in range(10)]).reshape(1,10))

        w_layer1_bias_grad = -eta * np.dot(np.dot(layer1_output[i],
                                                  np.asarray([e[i]*derv_act_fun(local_ind_field)[i]
        for i in range(10)]).reshape(1,10)), derv_act_fun(layer1_local_field[i]))
        w_input_grad = - eta * np.dot(xi , np.dot(np.dot(layer1_output[i],
                                                  np.asarray([e[i]*derv_act_fun(local_ind_field)[i]
        for i in range(10)]).reshape(1,10)), derv_act_fun(layer1_local_field[i])).reshape(1,10))

        # update weights
        w_input = np.subtract(w_input, w_input_grad)
        w_layer1_bias = np.subtract(w_layer1_bias, w_layer1_bias_grad)
        w_layer1 = np.subtract(w_layer1, w_layer1_grad)
        w_layer2_bias = np.subtract(w_layer2_bias, w_layer2_bias_grad)

    training_accuracy = train_correct/len(train_data)
```

```
    training.append(len(train_data) - train_correct)
    mse = 0
    for i in range(len(train_data)):
        mse += np.linalg.norm(layer2_output[i] - train_labels[i])**2
    mse = mse/len(train_data)
    energy_training.append(mse)
    print ("Root mean square Error:",mse,"Training accuracy:", training_accuracy,
           "No. of misclassifications:", (len(train_data) - train_correct))
    for i in range(len(test_data)):
        xi = test_data[i]
        xi.resize(784, 1)
        xi = normalize(xi)
        local_ind_field, output = feedforward(xi, w_layer1_bias, w_input)
        layer1_local_field.append(local_ind_field)
        layer1_output.append(output)

        local_ind_field, output = feedforward(output, w_layer2_bias, w_layer1)
        layer2_local_field.append(local_ind_field)
        layer2_output.append(output)

        max_index = output.argmax(axis=0)[0]
        new_output = [0]*10
        new_output[max_index] = 1
        x = np.linalg.norm(np.asarray(test_labels[i]) - np.asarray(new_output))**2
        if x==0:
            test_correct += 1
    testing_accuracy = test_correct/len(test_data)
    testing.append(len(test_data) - test_correct)
    mse = 0
    for i in range(len(test_data)):
        mse += np.linalg.norm(layer2_output[i] - test_labels[i])**2
    mse = mse/len(test_data)
    energy_testing.append(mse)
    print ("Root mean square Error:",mse,"Testing accuracy:", testing_accuracy,
           "No. of misclassifications:", (len(test_data) - test_correct))
    if testing_accuracy>0.95:
        break

fig, ax = plt.subplots(figsize=(10,10))
ax.set_ylim([0,60000])
plt.xlabel('Number of Epochs')
plt.ylabel('Number of Misclassifications')
plt.plot(range(len(training)), training, c = 'green', label='Training misclassifications')
plt.plot(range(len(testing)), testing, c = 'blue', label='Testing misclassifications')
plt.legend(loc = 'best')
plt.show()

fig, ax = plt.subplots(figsize=(10,10))
plt.xlabel('Number of Epochs')
plt.ylabel('Energies')
plt.plot(range(len(energy_training)), energy_training, c = 'green', label='Training energies')
plt.plot(range(len(energy_testing)), energy_testing, c = 'blue', label = 'Testing energies')
plt.legend(loc = 'best')
plt.show()
```

Output while Neural network runs on Testing and training data

```
Root mean square Error: 10.595203968891004 Testing accuracy: 0.4822 No. of
misclassifications: 5178
Root mean square Error: 10.595888714680559 Training accuracy: 0.48696666666666666 No. of
misclassifications: 30782
Root mean square Error: 10.608701316348444 Testing accuracy: 0.4885 No. of
misclassifications: 5115
Root mean square Error: 10.604331540971778 Training accuracy: 0.49035 No. of
misclassifications: 30579
Root mean square Error: 10.616981383383703 Testing accuracy: 0.4922 No. of
misclassifications: 5078
Root mean square Error: 10.602684500603774 Training accuracy: 0.4938666666666667 No. of
misclassifications: 30368
Root mean square Error: 10.617049364868278 Testing accuracy: 0.4986 No. of
misclassifications: 5014
Root mean square Error: 10.593028731413742 Training accuracy: 0.4955833333333333 No. of
misclassifications: 30265
Root mean square Error: 10.608969221866106 Testing accuracy: 0.5048 No. of
misclassifications: 4952
Root mean square Error: 10.588686580151618 Training accuracy: 0.4884 No. of
misclassifications: 30696
Root mean square Error: 10.600266382664287 Testing accuracy: 0.4937 No. of
```