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

X_train = read_files('train-images-idx3-ubyte/train-images.idx3-ubyte') #REading the files
y_train = read_files('train-labels-idx1-ubyte/train-labels.idx1-ubyte')
X_test = read_files('t10k-images-idx3-ubyte/t10k-images.idx3-ubyte')
y_test = read_files('t10k-labels-idx1-ubyte/t10k-labels.idx1-ubyte')

y_train_original_form = y_train
y_test_original_form = y_test

def read_files(file): #Function to read files
    with open(file, 'rb') as file:
        zero, data_type, dims = st.unpack('>HBB', file.read(4))
        shape = tuple(st.unpack('>I', file.read(4))[0] for d in range(dims))
        return np.frombuffer(file.read(), dtype=np.uint8).reshape(shape)

y_train = convert_onehot_vectors(y_train)
y_test = convert_onehot_vectors(y_test)

def convert_onehot_vectors(labels): #Function to convert to one hot notation
    vct = np.zeros((labels.size, labels.max()+1))
    vct[np.arange(labels.size), labels] = 1
    return vct
nodes0 = 784            #DEfines the network architecture
nodes1 = 128
nodes2 = 128
nodes3 = 10
def sigmoid_act(val):   # function to implement activation function
    return 1 / (1 + np.exp(-val))
W1 = np.random.normal(0, 1 , (nodes1, nodes0 + 1))# Randomly initilizing the weights
W2 = np.random.normal(0, 1 , (nodes2, nodes1 + 1))
W3 = np.random.normal(0, 1 , (nodes3, nodes2 + 1))
V1 = np.zeros(W1.shape) #Randomly initializing the local fields
V2 = np.zeros(W2.shape)
V3 = np.zeros(W3.shape)
learning_rate = 0.1
train_errors = []   #Containers for training and tesing stats
test_errors = []
energy_train = []
energy_test = []
MSE = []
MSE_test = []
epoch = 0
n = 60000
n_test = 10000
unit_vector = np.array([1]).reshape(1, 1)
while (True):       #Loop for implimenting the training of neural network
    y_3s = []
    y_3s_test = []
    train_accuracy = 0
    Energy_train_total = 0
    for i in range(n):
        y0 = np.array(X_train[i]).reshape(784, 1)
        int_y0 = np.vstack((unit_vector, y0))
        v_1 = W1 @ int_y0
```

```
    y_1 = sigmoid_act(v_1)
    int_y1 = np.vstack((unit_vector, y_1))
    v_2 = W2 @ int_y1
    y_2 = sigmoid_act(v_2)
    int_y2 = np.vstack((unit_vector, y_2))
    v_3 = W3 @ int_y2
    y_3 = sigmoid_act(v_3)
    y_3s.append(y_3)
    check = (np.argmax(y_3) == y_train_original_form[i])
    if check == True:
        train_accuracy += 1
    Energy_train_total += np.sum((y_train[i].reshape(y_3.shape) - y_3) ** 2) / (2 * n)
    d_3 = np.multiply((y_train[i].reshape(y_3.shape) - y_3), (sigmoid_act(v_3) * (1 - sigmoid_a
    d_2 = np.multiply((np.transpose(W3) @ d_3)[1:, :], (sigmoid_act(v_2) * (1 - sigmoid_act(v_2
    d_1 = np.multiply((np.transpose(W2) @ d_2)[1:, :], (sigmoid_act(v_1) * (1 - sigmoid_act(v_2
    de_dW1 = -d_1 @ int_y0.transpose()
    de_dW2 = -d_2 @ int_y1.transpose()
    de_dW3 = -d_3 @ int_y2.transpose()
    V1 = np.subtract(np.multiply(0.15, V1), np.multiply(learning_rate, de_dW1))
    V2 = np.subtract(np.multiply(0.15, V2), np.multiply(learning_rate, de_dW2))
    V3 = np.subtract(np.multiply(0.15, V3), np.multiply(learning_rate, de_dW3))
    W1 = W1 + V1
    W2 = W2 + V2
    W3 = W3 + V3
energy_train.append(Energy_train_total)
train_errors.append(n - train_accuracy)
Energy_test_total = 0
test_accuracy = 0
for i in range(n_test):
    y0 = np.array(X_test[i]).reshape(784, 1)
    int_y0 = np.vstack((unit_vector, y0))
    v_1 = W1 @ int_y0
    y_1 = sigmoid_act(v_1)
    int_y1 = np.vstack((unit_vector, y_1))
    v_2 = W2 @ int_y1
    y_2 = sigmoid_act(v_2)
    int_y2 = np.vstack((unit_vector, y_2))
    v_3 = W3 @ int_y2
    y_3 = sigmoid_act(v_3)
    y_3s_test.append(y_3)
    check = (np.argmax(y_3) == y_test_original_form[i])
    if check == True:
        test_accuracy += 1
    Energy_test_total += np.sum((y_test[i].reshape(y_3.shape) - y_3) ** 2) / (2 * n)
energy_test.append(Energy_test_total)
test_errors.append(n_test - test_accuracy)
sum = 0
for i in range(n):
    sum += np.square(np.subtract(y_train[i].reshape(y_3.shape),y_3s[i])).mean()
mse = (sum / (n))
MSE.append(mse)
sum = 0
for i in range(n_test):
    sum += np.square(np.subtract(y_test[i].reshape(y_3.shape),y_3s_test[i])).mean()
mse_test = (sum / (n_test))
MSE_test.append(mse_test)
if epoch != 0:
```

```
        if MSE[epoch] > MSE[epoch - 1]:
            learning_rate = learning_rate * 0.9
        if (test_accuracy / n_test) >= 0.95 or (train_accuracy/n) >0.96:
            break
    print ("Test accuracy ",test_accuracy/n_test," at epoch", epoch)
    epoch += 1
range_epoch = [i for i in range(0,epoch+1)] #plots the required graph for the evaluation of model
print("Following is a graph showing the relation between epochs and train errors")
plt.plot(range_epoch,train_errors)
plt.show()
print("Following is a graph showing the relation between epochs and test errors")
plt.plot(range_epoch,test_errors)
plt.show()
print("Following is a graph showing the relation between epochs and train energy")
plt.plot(range_epoch,energy_train)
plt.show()
print("Following is a graph showing the relation between epochs and train energy")
plt.plot(range_epoch,energy_test)
plt.show()
# Printing model stats
print('Accuracy on training set ', train_accuracy / n)
print('Accuracy on testing set ', test_accuracy / n_test)
print('Mean Squared error of training dataset is ', MSE[-1])
print('Mean Squared error of testing dataset is ', MSE_test[-1])
print('learning_rate used is', learning_rate)
print('Total number if epochs ', epoch)
```