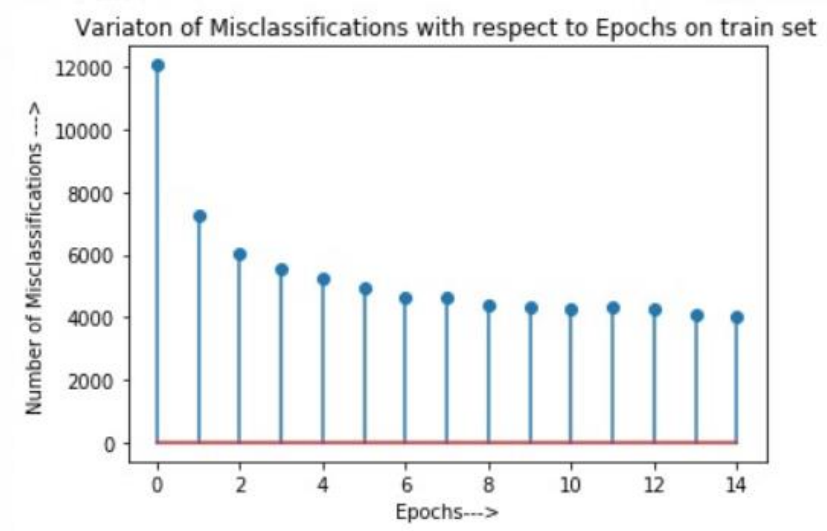
<u>Question 2:</u>

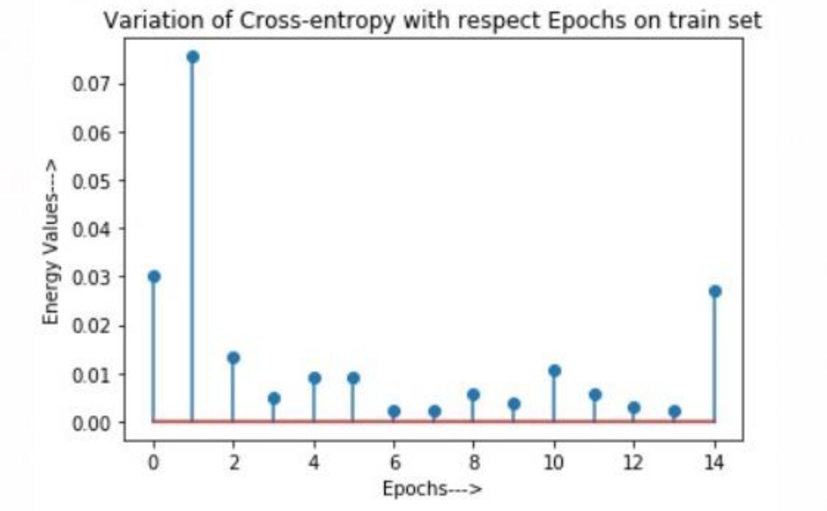
1. **Design of the Neural Network**
   - Network Topology: No of hidden layers used is 1. Number of neurons in the hidden layer is 50 and number of neurons in the output layer is 10.
   - Representation of digits: At the output layer, the labels were one-hot encoded. That is the desired labels were represented as a 10*10 identity matrix. For error and accuracy calculations, the class labels were used as they are, i.e. 0 for class 0 and so on.
   - Network Activation Functions and Learning rates: Network Activation function used in the hidden layer for all the neurons was hyperbolic tangent and the activation function used for the output layer was Softmax. The learning rate used is 0.05. Dynamic update of learning rate after every epoch was not used.
   - Energy/Distance Function: Cross-Entropy cost Function.

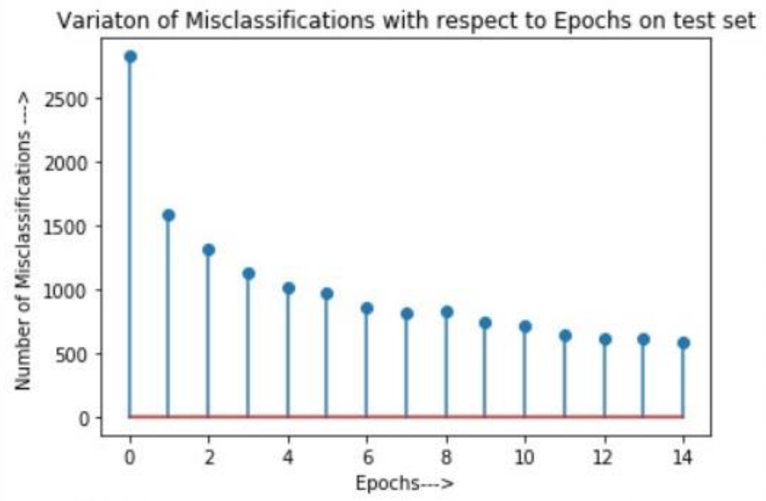
$$-\sum_{i} y_i * log y_i$$

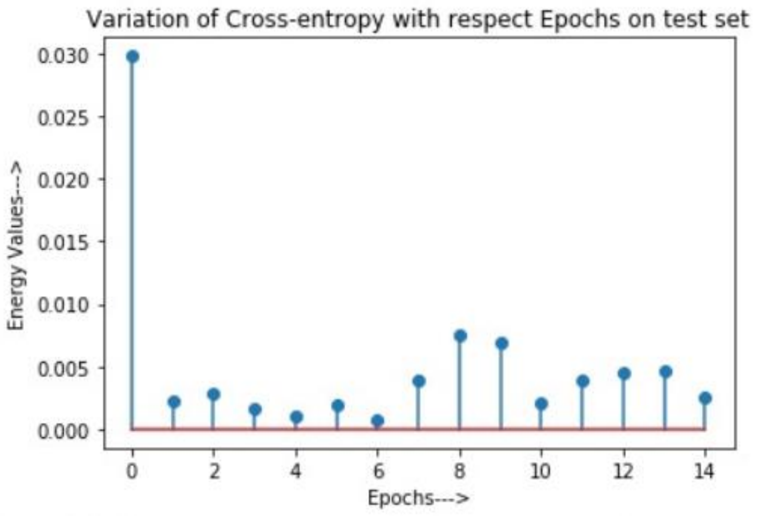
2. **Choice of hyperparameters:**
   - **Choice for number of hidden layers and number of neurons:** The number of hidden layers considered was 1. To test whether larger number of neurons gave a better accuracy, 2 cases were considered. The first was with 200 neurons. Train accuracy was found to be 94.04% and test accuracy was found to be 93.57%. In the second case, we considered 50 neurons. Train accuracy was found to be 95.4% and test accuracy was found to 94.7% keeping other parameters like eta and number of epochs at 0.05 and 15 respectively. One can observe that in both cases we obtain similar accuracy results but the choice of 50 neurons was more favorable since the time taken for training and testing was approximately 5 minutes as opposed to the choice of 200 neurons as it took approximately 30 mins. This also made the case of using 2 hidden layers a bad idea since that would increase the complexity for no significant improvement in accuracy.
   - **Choice of Learning rate:** When the learning rate was decreased from 0.05 to 0.001 the learning process slowed down, i.e. in 15 epochs the accuracy in the test set reduced from 94.7% to 89.05% which means that larger number of epochs would have to be considered to obtain comparable accuracy. For a large learning rate of 1, the weights in subsequent updates became unstable given us an oscillatory behavior.
   - **Choice of Number of epochs:** 5 epochs gave an accuracy of 91% on the training set but increasing to 15 epochs gave rise 94.04% accuracy. The only trade off was the time taken to run. 5 epochs took much lesser time as compared to 15 epochs.
   - **Choice of Weight initialization:** Although the weight initialization was randomized, taking only positive random weights led to the learning rate becoming unstable and slow. When the random function included both negative and positive weights a better result was observed.

Variations of Misclassifications/Cross-entropy loss with respect to epochs on the train and test dataset:









Pseudocode: Back Propagation on MNIST dataset.

1. Read the contents of the webpage http://yann.lecun.com/exdb/mnist/
2. There are 4 files listed in the beginning of the page as training set images, training set labels, test set images, and test set labels, download them. The function which defines the download has been included in the code.
3. Each image is 28x28 pixels, so that we will have a neural network 28x28 = 784 nodes in the input layer, 50 neurons in the hidden layer with an activation function ***$\varphi(v) = \tanh(v)$.*** and 10 nodes in the output layer where the softmax activation is used.
4. Including the bias for 50 neurons, We wish to find 784x50 +50 weights(considering both weights and biases), such that the network outputs $[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$ if the input image corresponds to a 0, $[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$ if the input image corresponds to a 1, and so on.

5. Initialize the weights w Є R^{50x784} and biases b Є R^{50x1} in such a way that it follows a random normal distribution; the learning rate eta is taken to be 0.05.
6. Calculate the induced local fields with the current training sample and weights: v = $\boldsymbol{w*x_i + b}$, where x_i Є R^{784x1} is the ith training sample (the vectorized version of the 28x28 image from the training set images).
7. The given input image may result in different values other than 0 and 1 at output neurons. For such scenarios, only for the purpose of calculating the misclassification errors, we choose the output neuron with the largest induced local field. In other words, we find the largest component of v = $[v_0\ v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8\ v_9]T$. Now, suppose that the largest component of v is vj , where j Є {0,...,9}. Correspondingly, our network decides that the input image xi corresponds to the digit j.
8. If j is not the same as the input label (which is obtained from the training set labels), then err(epoch) = err(epoch) + 1. This gives us the total number of misclassifications which will thereby give us the accuracy of the training same which in this case was 95.4%.
9. We use the backpropagation algorithm to update weights. The gradients are calculated with respect to the weights and biases.
10. Similarly perform the same operation for all the test samples with the updated weights and measure the accuracy of the training set. For the given network, the accuracy was 94.7%.

11. The induced local field is calculated using the equation given below

$$z_k^l = b_k^l + \sum_j W_{kj}^l a_j^{l-1}$$

Where L indicates the last layer. *l* indicates a specific layer. It could be equal to L, i.e., *l*−*1*=L−1 *l*−*1*=L−1, but not always the case. The subscript k usually denotes neuron indices in the output. layer (layer *l*). The subscript j usually denotes neuron indices in layer *l*−*1*. The subscript i usually denotes neuron indices in layer *l*−*2*. z is the weighted sum of activations from the previous layer

$$a_k^L = softmax(z_k^L) = \frac{e^{z_k^L}}{\sum_c e^{z_c^L}}$$

The cross entropy energy function is given as

$$E = -\sum_d t_d \log a_k^L = -\sum_d t_d (z_d^L - \log \sum_c e^{z_c^L})$$

Weight and bias updates for the last layer is given as:

$$\frac{\partial E}{\partial W_{kj}^L} = \frac{\partial E}{\partial z_k^L} \frac{\partial z_k^L}{\partial W_{kj}^L} = \delta_k^L a_j^{L-1}$$

$$\frac{\partial E}{\partial b_k^L} = \frac{\partial E}{\partial z_k^L} \frac{\partial z_k^L}{\partial b_k^L} = \delta_k^L(1) = \delta_k^L$$

Where

$$\delta_k^L = \frac{\partial E}{\partial z_k^L} = a_k^L - t_k$$

Weight and bias updates for the hidden layer:

$$\frac{\partial E}{\partial W_{ji}^{l-1}} = \delta_j^{l-1} a_i^{l-2}$$

$$\frac{\partial E}{\partial b_j^{l-1}} = \frac{\partial E}{\partial a_j^{l-1}} \frac{\partial a_j^{l-1}}{\partial z_j^{l-1}} \frac{\partial z_j^{l-1}}{\partial b_j^{l-1}} = \delta_j^{l-1}(1) = \delta_j^{l-1}$$

Where

$$\frac{\partial E}{\partial W_{ji}^{l-1}} = \delta_j^{l-1} a_i^{l-2}$$

Source Code:

```
import numpy as np
import sys, os
import matplotlib.pyplot as mlp
def load_dataset():
    # We first define a download function, supporting both Python 2 and 3.
    if sys.version_info[0] == 2:
        from urllib import urlretrieve
    else:
        from urllib.request import urlretrieve

    def download(filename, source='http://yann.lecun.com/exdb/mnist/'):
        print("Downloading %s" % filename)
        urlretrieve(source + filename, filename)

    # We then define functions for loading MNIST images and labels.
    # For convenience, they also download the requested files if needed.
    import gzip

    def load_mnist_images(filename):
        if not os.path.exists(filename):
            download(filename)
        # Read the inputs in Yann LeCun's binary format.
        with gzip.open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=16)
        # The inputs are vectors now, we reshape them to monochrome 2D images,
        # following the shape convention: (examples, channels, rows, columns)
        # data = data.reshape(-1, 1, 28, 28)
        # The inputs come as bytes, we convert them to float32 in range [0,1].
        # (Actually to range [0, 255/256], for compatibility to the version
        # provided at http://deeplearning.net/data/mnist/mnist.pkl.gz.)
        return data / np.float32(256)

    def load_mnist_labels(filename):
        if not os.path.exists(filename):
            download(filename)
        # Read the labels in Yann LeCun's binary format.
        with gzip.open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=8)
        # The labels are vectors of integers now, that's exactly what we want.
        return data

    # We can now download and read the training and test set images and labels.
    X_train = load_mnist_images('train-images-idx3-ubyte.gz')
    y_train = load_mnist_labels('train-labels-idx1-ubyte.gz')
    X_test = load_mnist_images('t10k-images-idx3-ubyte.gz')
    y_test = load_mnist_labels('t10k-labels-idx1-ubyte.gz')

    # We reserve the last 10000 training examples for validation.
    # X_train, X_val = X_train[:-10000], X_train[-10000:]
    # y_train, y_val = y_train[:-10000], y_train[-10000:]

    # We just return all the arrays in order, as expected in main().
    # (It doesn't matter how we do this as long as we can read them again.)
    return X_train, y_train, X_test, y_test

d= np.eye(10,10)
def softmax(inp):
    e=np.zeros([1,10])
    for j in range (10):
        e[0,j]=np.exp(inp[0,j])/(np.sum(np.exp(inp)))
    return e
print("Loading data...")
X_train, y_train, X_test, y_test = load_dataset()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
```

```
eta=0.05
W1=np.random.normal(size=[784,50])
b1=np.random.normal(size=[1,50])
W2=np.random.normal(size=[50,10])
b2=np.random.normal(size=[1,50])
err_list=[]
energy_list=[]
for z in range (15):
    errors=0
    for i in range(60000):
        y1=np.tanh(np.matmul(X_train[i].reshape(1,784),W1)+b1)
        v=np.matmul(y1,W2)+b2
        y2=softmax(v)
        y=np.argmax(y2,axis=1)
        cost=-1*(np.matmul(d[y_train[i]],np.log(y2).transpose()))
        des=d[y_train[i]]
        err=des-y2
        gradW2=np.matmul(y1.transpose(),err)
        W2=W2+(eta*gradW2)
        gradb2=err
        b2=b2+(eta*gradb2)
        delj=(np.ones([1,50])-y1**2)*(np.matmul(err,W2.transpose()))
        gradW1=np.matmul(X_train[i].reshape(784,1),delj)
        W1=W1+(eta*gradW1)
        b1=b1+(eta*delj)
        if(y_train[i]!=y):
            errors+=1
    err_list.append(errors)
    energy_list.append(cost)
mlp.stem(np.arange(15),err_list)
mlp.title('Variaton of Misclassifications with respect to Epochs on traning set')
mlp.xlabel('Epochs--->')
mlp.ylabel('Number of Misclassifications --->')
mlp.figure()
mlp.stem(np.arange(15),energy_list)
mlp.title('Variation of Cross-entropy with respect Epochs on training set')
mlp.xlabel('Epochs--->')
mlp.ylabel('Energy Values--->')
acc=0
for i in range(10000):

    y1=np.tanh(np.matmul(X_test[i].reshape(1,784),W1)+b1)
    v=np.matmul(y1,W2)+b2
    y2=softmax(v)
    y=np.argmax(y2,axis=1)
    if(y_test[i]==y):
        acc+=1
print(acc/100)
```