In [1]:

```
'''
This file trains Neural Network for Digit Classifications of MNIST Using BackP
ropagation
Neural Architecture:
1 - Three Layers - 1: Input Layer 2: Hidden Layer 3: Output Layer
Input Layer - 784 Inputs
Hidden Layer - 40 Neurons Activation Function tanh
Output Layer - 10 Neurons Activation Function softmax
2 - Weight: w(i, j) indicates weight fed to ith neuron from jth input
Weights in Layer 2 (Hidden Layer) W_Layer_2 [ w(1,0).....w(1,784)
                                              ...................
                                              w(40,0).....w(40,784)] 40x785
Weights in Layer 3 (Output Layer) W_Layer_3 [ w(1,0).....w(1,40)
                                              ...................
                                              w(10,0).....w(10,40)] 10x41
3 - Energy or Loss function: Cross Entropy plus Momentum
'''
```

Out[1]:

```
'\nThis file trains Neural Network for Digit Classifications of MNIST Using B
ackPropagation\nNeural Architecture:\n1 - Three Layers - 1: Input Layer 2: Hi
dden Layer 3: Output Layer\nInput Layer - 784 Inputs\nHidden Layer - 40 Neuro
ns Activation Function tanh\nOutput Layer - 10 Neurons Activation Function so
ftmax\n2 - Weight: w(i, j) indicates weight fed to ith neuron from jth input
\nWeights in Layer 2 (Hidden Layer) W_Layer_2 [ w(1,0).....w(1,784)\n
                                      ...................\n
                            w(40,0).....w(40,784)] 40x785\nWeights in Layer 3
(Output Layer) W_Layer_3 [ w(1,0).....w(1,40)\n
                ...................\n
      w(10,0).....w(10,40)] 10x41\n3 - Energy or Loss function: Cross Entropy
plus Momentum\n'
```

In [2]:

```
# Import Required Files
import numpy
from sklearn import preprocessing
import matplotlib.pyplot as plt
from LoadData import load_training_labels, load_training_images
from LoadData import load_test_images, load_test_labels
from LoadData import InitialWeights
```

In [3]:

```
# Load Data
scaler = preprocessing.StandardScaler().fit(load_training_images('train-images-idx3-ubyte.gz'))
Images = scaler.transform(load_training_images('train-images-idx3-ubyte.gz'))
Labels_Train = load_training_labels('train-labels-idx1-ubyte.gz')
Test = scaler.transform(load_test_images('t10k-images-idx3-ubyte.gz'))
Labels_Test = load_test_labels('t10k-labels-idx1-ubyte.gz')
if numpy.DataSource().exists('InitialWeightsLayer2.txt') and numpy.DataSource().exists('InitialWeightsLayer3.txt'):
    W_Layer_2_Guess = numpy.loadtxt('InitialWeightsLayer2.txt')  # Load the Initial Weights
    W_Layer_3_Guess = numpy.loadtxt('InitialWeightsLayer3.txt')   # Load the Initial Weights
else:
    W_Layer_2_Guess, W_Layer_3_Guess = InitialWeights()
    numpy.savetxt('InitialWeightsLayer2.txt', W_Layer_2_Guess)  # Generate the Weights and save them
    numpy.savetxt('InitialWeightsLayer3.txt', W_Layer_3_Guess)  # Generate the Weights and save them
```

```
C:\Users\BV SAMEER KUMAR\PyCharmProjects\untitled\venv\lib\site-packages\skle
arn\utils\validation.py:590: DataConversionWarning: Data with input dtype uin
t8 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\BV SAMEER KUMAR\PyCharmProjects\untitled\venv\lib\site-packages\skle
arn\utils\validation.py:590: DataConversionWarning: Data with input dtype uin
t8 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\BV SAMEER KUMAR\PyCharmProjects\untitled\venv\lib\site-packages\skle
arn\utils\validation.py:590: DataConversionWarning: Data with input dtype uin
t8 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [4]:

```
# Hyperparameters
alpha = 0  # Regularization Parameter
beta = 0.9  # Momentum Parameter
eta = 5  # Learning Rate

# Parameters
iterations = 0  # No. of training epochs
epsilon = 0.047  # Error Ratio
M2 = 0  # Momentum Vector for Layer 2
M3 = 0  # Momentum Vector for Layer 3
Epoch = numpy.array([])  # Array for storing No. of Training Iterations
CE_Train = numpy.array([])  # Mean Squared Error on Training Set
Error_Train = numpy.array([])  # No. of Misclassfications on Training Set
CE_Test = numpy.array([])  # Mean Squared Error on Test Set
Error_Test = numpy.array([])  # No. of Misclassfications on Test Set
max_iter = 500  # Maximum allowed Iterations for convergence
row, col = Images.shape  # Shape of Input 60000x784
row1, col1 = Test.shape  # Shape of Input 60000x784
D_Train = numpy.zeros((row, 10))
D_Train[numpy.arange(row), Labels_Train] = 1
D_Test = numpy.zeros((row1, 10))
D_Test[numpy.arange(row1), Labels_Test] = 1
```

In [5]:

```
# Required Functions
# SoftMax
def softmax(v):
    e = numpy.exp(v - numpy.max(v))
    return e/numpy.sum(e)


# Forward Pass
def forward_pass(image, w_layer_2, w_layer_3):
    # V2 and V3 are Locally Induced Fields at Layer 2 and 3
    # Y2 and Y3 are Locally Induced Fields at Layer 2 and 3
    temp_x = numpy.concatenate(([1], image), axis=0)
    v2 = numpy.dot(w_layer_2, temp_x.T)
    y2 = numpy.tanh(v2)
    temp_y = numpy.concatenate(([1], y2), axis=0)
    v3 = numpy.dot(w_layer_3, temp_y)
    y3 = softmax(v3)
    return v2, v3, y2, y3


# Backward Pass
def backward_pass(image, d, w_layer_3, v2, y2, y3):
    delta3 = (1/row) * (d - y3)
    derivative_layer2 = numpy.array([1 - numpy.square(numpy.tanh(i)) for i in v2])
    delta2 = numpy.multiply(numpy.dot(w_layer_3[:, 1:].T, delta3), derivative_layer2)
    gradient_layer_2 = numpy.matmul(-delta2[:, numpy.newaxis], numpy.concatenate(([1], image), axis=0)[:, numpy.newaxis].T)
    gradient_layer_3 = numpy.matmul(-delta3[:, numpy.newaxis], numpy.concatenate(([1], y2), axis=0)[:, numpy.newaxis].T)
    return gradient_layer_2, gradient_layer_3


# Update Weights
def update_weights(m2, m3, gradient_layer_2, gradient_layer_3, w_layer_2, w_layer_3):
    m2 = (beta * m2) - (eta * (gradient_layer_2 + alpha * w_layer_2))  # Momentum plus Regularization
    m3 = (beta * m3) - (eta * (gradient_layer_3 + alpha * w_layer_3))  # Momentum plus Regularization
    w_layer_2 = w_layer_2 + m2
    w_layer_3 = w_layer_3 + m3
    return w_layer_2, w_layer_3, m2, m3


# Calculate Cross Entropy Error
# This done for all 60k training set at once
def calculate_ce(image, label, w_layer_2, w_layer_3):
    rows, cols = image.shape
    ce = 0
    error = 0
    for i in range(0, rows):
        v2, v3, y2, y3 = forward_pass(image[i], w_layer_2, w_layer_3)
        d = numpy.zeros(10)
        d[label[i]] = 1
        ce = ce + numpy.sum(numpy.dot(-d, numpy.log(y3.T)))/rows
        if label[i] != numpy.argmax(y3):
            error += 1
    return ce, error

# Calculate Misclassifications
# This done for all 60k training set at once
def calculate_error(image, label, w_layer_2, w_layer_3):
    v2, v3, y2, y3 = forward_pass(image, w_layer_2, w_layer_3)
    y = numpy.argmax(y3, axis=0)
    return numpy.count_nonzero(label - y)


# Learning Rate Decay
def check_learning_rate(eta_prime, ce):
    if ce[-1] >= ce[-2]:
        eta_prime = 0.4 * eta_prime
    return eta_prime
```

In [6]:

```
# Main Loop
# Iteration 0
# Backpropagation
L = numpy.arange(60000)
numpy.random.shuffle(L)
temp_w2 = W_Layer_2_Guess
temp_w3 = W_Layer_3_Guess
for i in L:
    V2, V3, Y2, Y3 = forward_pass(Images[i], temp_w2, temp_w3)
    Gradient_Layer_2, Gradient_Layer_3 = backward_pass(Images[i], D_Train[i],
temp_w3, V2, Y2, Y3)
    temp_w2, temp_w3, M2, M3 = update_weights(M2, M3, Gradient_Layer_2, Gradie
nt_Layer_3, temp_w2, temp_w3)
# Book Keeping
# W2 = numpy.concatenate(([W_Layer_2_Guess], [temp_w2]), axis=0)
# W3 = numpy.concatenate(([W_Layer_3_Guess], [temp_w3]), axis=0)
Epoch = numpy.concatenate((Epoch, [iterations]), axis=0)
ce_train, e_train = calculate_ce(Images, Labels_Train, temp_w2, temp_w3)
CE_Train = numpy.concatenate((CE_Train, [ce_train]), axis=0)
# e_train = calculate_error(Images, Labels_Train, temp_w2, temp_w3)
Error_Train = numpy.concatenate((Error_Train, [e_train]), axis=0)
ce_test, e_test = calculate_ce(Test, Labels_Test, temp_w2, temp_w3)
CE_Test = numpy.concatenate((CE_Test, [ce_test]), axis=0)
# e_Test = calculate_error(Test, Labels_Test, W2[-1], W3[-1])
Error_Test = numpy.concatenate((Error_Test, [e_test]), axis=0)
# Print
print('Epoch: ', iterations, ' MSE: ', ce_train, ' M1: ', e_train, 'M2: ', e_t
est, '\n')
# Next...
iterations += 1
# Remaining Epochs
while iterations <= max_iter:
    # Backpropagation
    L = numpy.arange(60000)
    numpy.random.shuffle(L)
    for i in L:
        V2, V3, Y2, Y3 = forward_pass(Images[i], temp_w2, temp_w3)
        Gradient_Layer_2, Gradient_Layer_3 = backward_pass(Images[i], D_Train[
i], temp_w3, V2, Y2, Y3)
        temp_w2, temp_w3, M2, M3 = update_weights(M2, M3, Gradient_Layer_2, Gr
adient_Layer_3, temp_w2, temp_w3)
    # Book Keeping
    # W2 = numpy.concatenate((W2, [temp_w2]), axis=0)
    # W3 = numpy.concatenate((W3, [temp_w3]), axis=0)
    Epoch = numpy.concatenate((Epoch, [iterations]), axis=0)
    ce_train, e_train = calculate_ce(Images, Labels_Train, temp_w2, temp_w3)
    CE_Train = numpy.concatenate((CE_Train, [ce_train]), axis=0)
#     e_train = calculate_error(Images, Labels_Train, temp_w2, temp_w3)
    Error_Train = numpy.concatenate((Error_Train, [e_train]), axis=0)
    ce_test, e_test = calculate_ce(Test, Labels_Test, temp_w2, temp_w3)
    CE_Test = numpy.concatenate((CE_Test, [ce_test]), axis=0)
    # e_Test = calculate_error(Test, Labels_Test, W2[-1], W3[-1])
    Error_Test = numpy.concatenate((Error_Test, [e_test]), axis=0)
    # Print
    print('Epoch: ', iterations, ' MSE: ', ce_train, ' M1: ', e_train, 'M2: ',
 e_test, '\n')
```

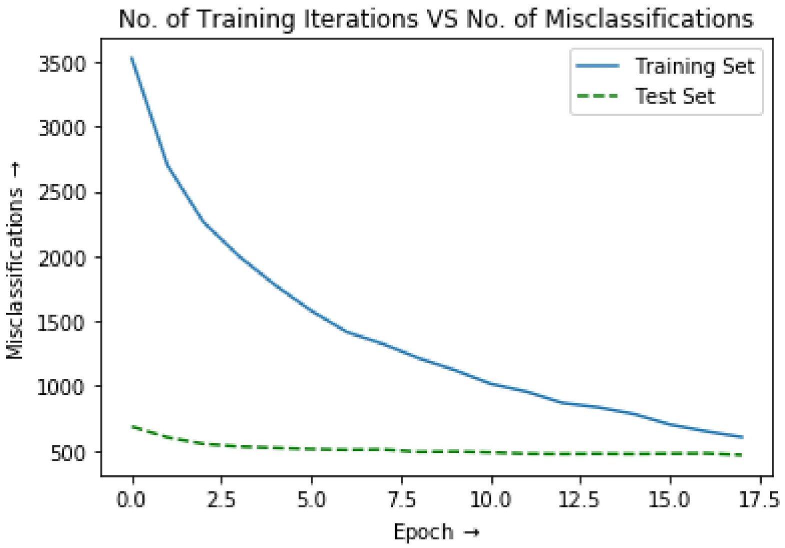
```
    # Check Termination
    if (Error_Test[-1]/10000) < epsilon:
        # Save Final Weights
        numpy.savetxt('FinalOptimalWeights2.txt', temp_w2)
        numpy.savetxt('FinalOptimalWeights3.txt', temp_w3)
        print('Optimal Weights Reached!!!!!')
        break
    else:
        # Check Learning Rate
        eta = check_learning_rate(eta, CE_Train)
        # Next...
        iterations += 1
```

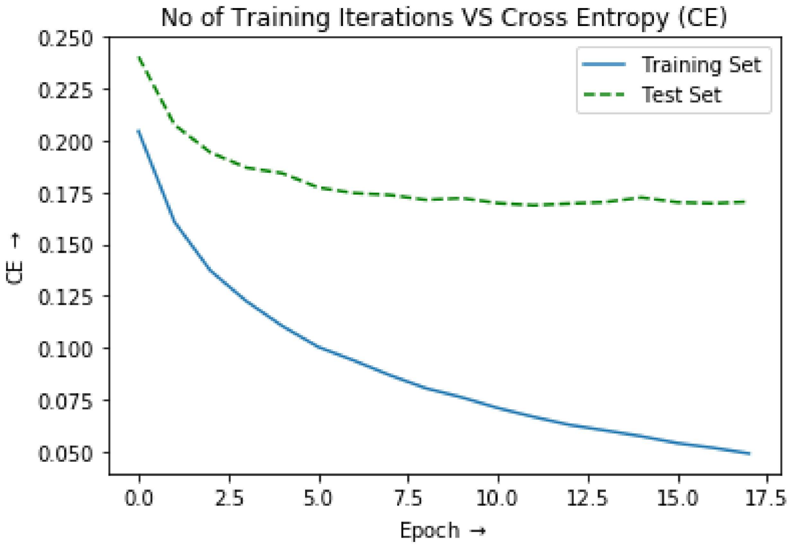
```
Epoch:  0  MSE:  0.20427775605149243  M1:  3525 M2:  687

Epoch:  1  MSE:  0.16063631298815761  M1:  2696 M2:  603

Epoch:  2  MSE:  0.13715030976726428  M1:  2261 M2:  553

Epoch:  3  MSE:  0.12249140184003654  M1:  1996 M2:  532

Epoch:  4  MSE:  0.11049130948100949  M1:  1776 M2:  522

Epoch:  5  MSE:  0.10040489195551645  M1:  1580 M2:  513

Epoch:  6  MSE:  0.09381163795052323  M1:  1415 M2:  507

Epoch:  7  MSE:  0.08675491795452275  M1:  1324 M2:  510

Epoch:  8  MSE:  0.08048602458259761  M1:  1213 M2:  491

Epoch:  9  MSE:  0.07610823101089928  M1:  1122 M2:  494

Epoch:  10  MSE:  0.07096985211915251  M1:  1017 M2:  487

Epoch:  11  MSE:  0.06668285003913973  M1:  956 M2:  477

Epoch:  12  MSE:  0.06281660362805329  M1:  869 M2:  475

Epoch:  13  MSE:  0.06013256053090702  M1:  835 M2:  477

Epoch:  14  MSE:  0.05734392736220835  M1:  783 M2:  475

Epoch:  15  MSE:  0.05404073869700681  M1:  702 M2:  478

Epoch:  16  MSE:  0.051764609770691666  M1:  650 M2:  480

Epoch:  17  MSE:  0.0490674637281017  M1:  605 M2:  467

Optimal Weights Reached!!!!!
```

```
In [ ]:
```

In [7]:

```
# Plot
# Plot 1
fig1, ax1 = plt.subplots()
ax1.plot(Epoch, Error_Train, label='Training Set')
ax1.plot(Epoch, Error_Test, 'g--', label='Test Set')
plt.title(r'No. of Training Iterations VS No. of Misclassifications')
plt.xlabel(r'Epoch $\rightarrow$')
plt.ylabel(r'Misclassifications $\rightarrow$')
plt.legend()
plt.savefig('1.pdf')
# Plot 2
fig2, ax2 = plt.subplots()
ax2.plot(Epoch, CE_Train, label='Training Set')
ax2.plot(Epoch, CE_Test, 'g--', label='Test Set')
plt.title('No of Training Iterations VS Cross Entropy (CE)')
plt.xlabel(r'Epoch $\rightarrow$')
plt.ylabel(r'CE $\rightarrow$')
plt.legend()
plt.savefig('2.pdf')
plt.show()
```


No. of Training Iterations VS No. of Misclassifications
Training Set
Test Set
Misclassifications →
Epoch →


No of Training Iterations VS Cross Entropy (CE)
Training Set
Test Set
CE →
Epoch →