Neural Nets

HW-II

1. In order to get an output of 1 in the shaded region, we will have to use ~~tt~~ multiple layers of neurons, out of ~~wit~~ which the first layer of neurons will corrospond to the lines that form the enclosed area.

for line 1:

equation of line => $x=1 \Rightarrow x-1=0$ or $-x+1=0$

comparing it with equation of the perceptron $w_0+w_1X_1+w_2X_2$

we get, $w_0=1$, $w_1=-1$ and $w_2=0$

These weights will give an output of 1 at the left of the line.

for line 2:

equation of line => $-1+x+y=0$

Compare it with $w_0+w_1x_1+w_2x_2$

we get $w_0=-1, w_1=1, w_2=1$

These weights will give an output of 1 ~~below~~ above the line.

for line 3: equation of line : $1+x-y=0$

compare it with $w_0+w_1x_1+w_2x_2$

we get, $w_0=1$, $w_1=1$, $w_2=-1$

These weights will give an output of 1 below the line.

for line 4: equation of line => $-2 \mp x=0$

comparing it with $w_0+w_1x_1+w_2x_2$

we get, $w_0=-2$, $w_1=$ ~~1~~ -1, $w_2=0$

These weights will give an output of 1 to the ~~right~~ left of the line.

for line 5: equation of line => $y=0$

comparing it with $w_0+w_1X_1+w_2x_2$

we get, $w_0=0$, $w_1=0$, $w_2=1$

These weight will give an output of 1 above the line.

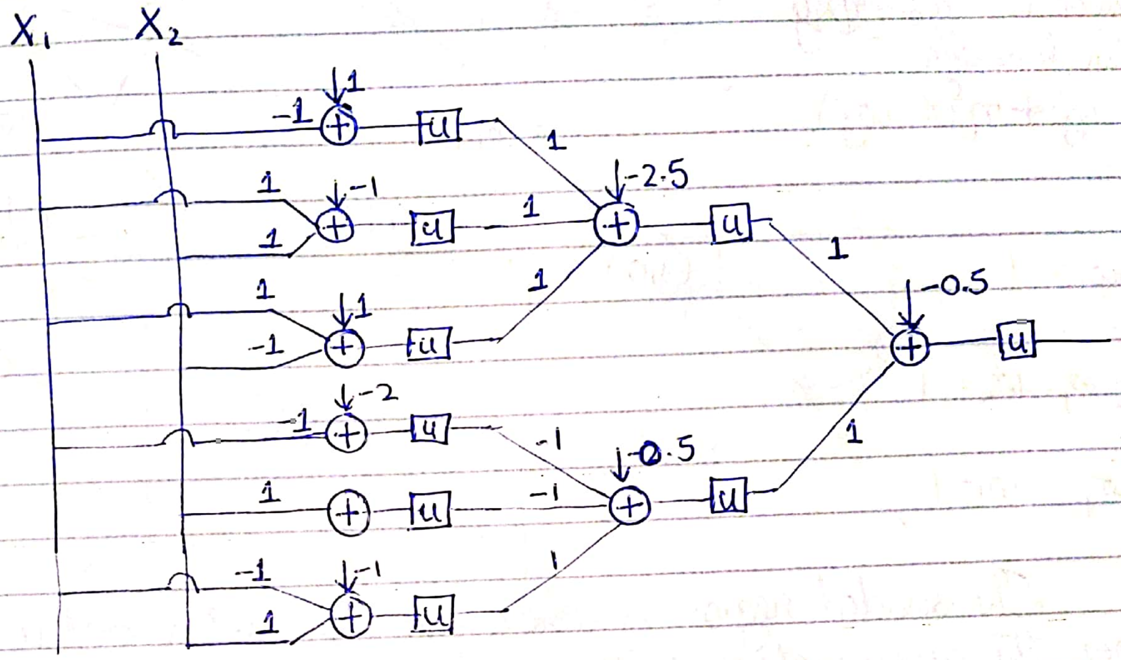
for line 6: equation of line = -1(equation of line 3) => $-1-x+y=0$

Compare it with $w_0+w_1X_1+w_2X_2$

we get, $w_0=-1, w_1=-1, w_2=1$

These weights will get an output of 1 above the line.

The circuit will be as follows.



We have put AND gate on line 1, 2 and 3 with bias $-n+\frac{1}{2}$ and weights $w_1, w_2 = 1$ where n is the number of inputs.

OR gate has been implemented using bias of -0.5 and weights of 1 for each input.

Scanned by CamScanner

In [55]:
```
import os
import path
import struct
import gzip
import numpy as np
import matplotlib.pyplot as plt
```

In [56]:
```
# This function is created to read the input idx files as a numpy array
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)
```

In [57]:
```
def step_fn(x):
    i = 0
    y = np.empty([10,1])
    for each in x:
        if each >= 0:
            y[i] = 1.0
        else:
            y[i] = 0.0
        i += 1
    return y
```

In [58]:
```
#This function is used to find out the number of errors when training the mode
l.
def training_errors(epoch, errors):
    for i in range(n):
        xi = train_data[i]
        xi.resize(784, 1)
        v = np.matmul(w,xi)
        prediction = v.argmax(axis=0)
        actual = train_labels[i]
        if prediction != actual:
            errors[epoch] += 1
    return errors[epoch]
```

In [59]:
```
#This is a function to update the weights in order to correctly classify the i
mages.
def update_weight(w):
    for i in range(n):
        xi = train_data[i]
        xi.resize(784, 1)
        y = np.array(step_fn(np.matmul(w,xi)))
        label = np.zeros((1,10)).T
        label[train_labels[i]] = 1
        difference = label - y
        xit = np.transpose(xi)
        update = learning_rate * np.matmul(difference, xit)
        w += update
```

In [60]:
```
# This function is created to read the input idx files as a numpy array
def read_idx_gz(filename):
    with gzip.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)
```

In [61]:
```
# This function is used to calculate the number of errors when classifying the
test dataset.
def find_errors():
    test_errors = 0
    for i in range(len(test_data)):
        xi = test_data[i]
        xi.resize(784, 1)
        v = np.matmul(w ,xi)
        prediction = v.argmax(axis=0)
        actual = test_labels[i]
        if prediction != actual:
            test_errors += 1
    print("Errors in test data: ", test_errors)
    print("Percentage of test errors: ", test_errors*100/len(test_data))
```

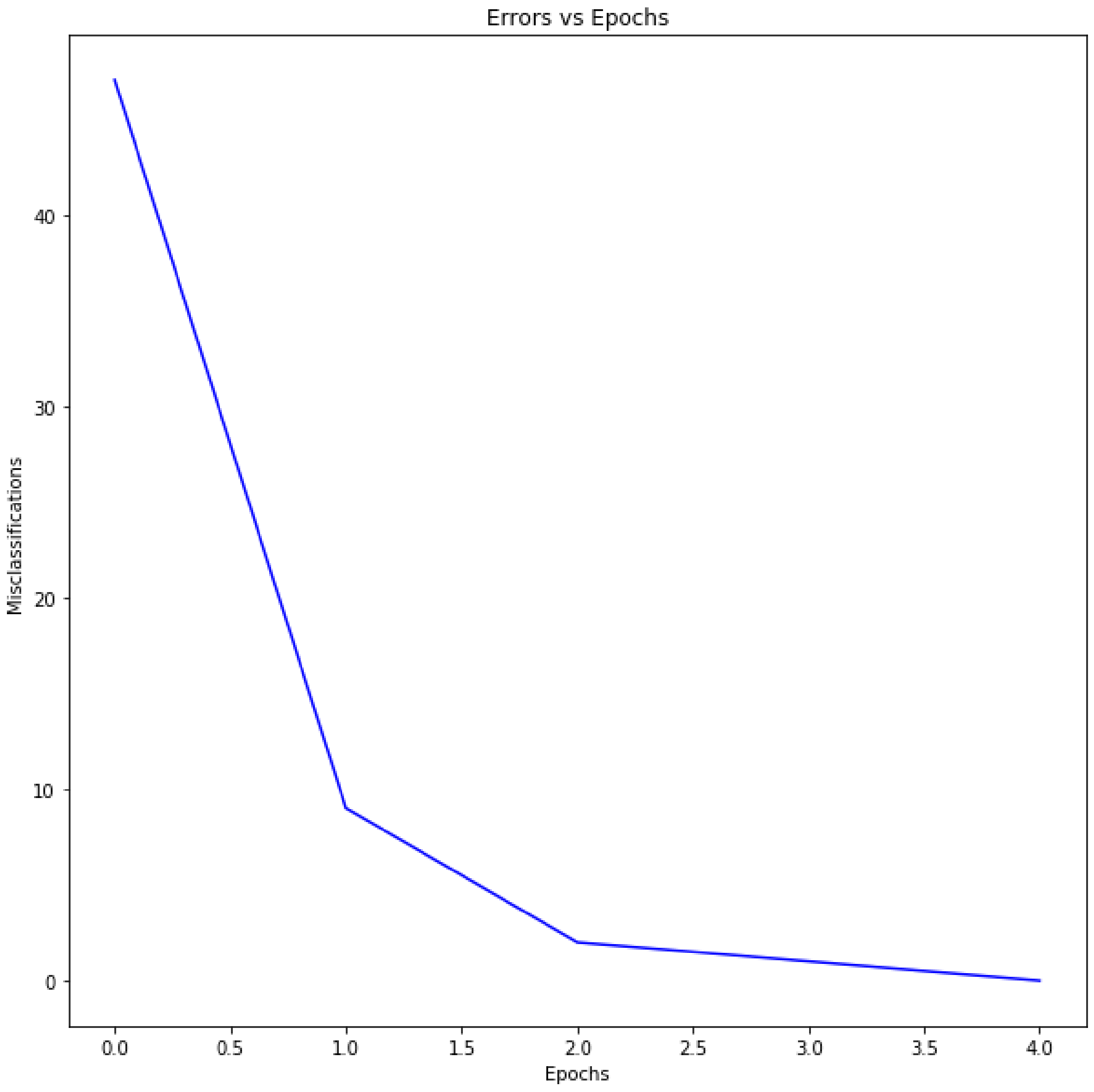
In [62]:
```
# This Function is used to plot the graphs.
def plot_graph(errors):
    fig, ax = plt.subplots(figsize=(10,10))
    plt.plot(range(len(errors)), errors, c = 'blue')
    plt.ylabel('Misclassifications')
    plt.xlabel('Epochs')
    plt.title('Errors vs Epochs')
    plt.show()
```

In [63]:
```
#This is the funtion to learn weigths from the dataset
def learning_weights(w, epoch, threshold, learning_rate):
    while epoch<100:
        errors.append(0)
        errors[epoch] = training_errors(epoch, errors)
        update_weight(w)
        epoch += 1
        if errors[epoch-1]/n <= threshold:
            break
```

In [64]:
```
# This data has been collected from the link mentioned in the homeowork assign
ment
train_data = read_idx_gz('train-images-idx3-ubyte.gz')
train_labels = read_idx_gz('train-labels-idx1-ubyte.gz')
test_data = read_idx_gz('t10k-images-idx3-ubyte.gz')
test_labels = read_idx_gz('t10k-labels-idx1-ubyte.gz')
```

In [65]:
```
# Initializing the weights to a ramdom value along with learning rate, epoch a
nd threshold
w = np.random.uniform(-1, 1, size=(10,784))
n = 50
epoch = 0
threshold = 0.0
learning_rate = 1.0
errors = []
```

In [66]:
```
# This will plot a graph of Misclassifications against Epochs
learning_weights(w, epoch, threshold, learning_rate)
plot_graph(errors)
find_errors()
```



```
Errors in test data:  4696
Percentage of test errors:  46.96
```

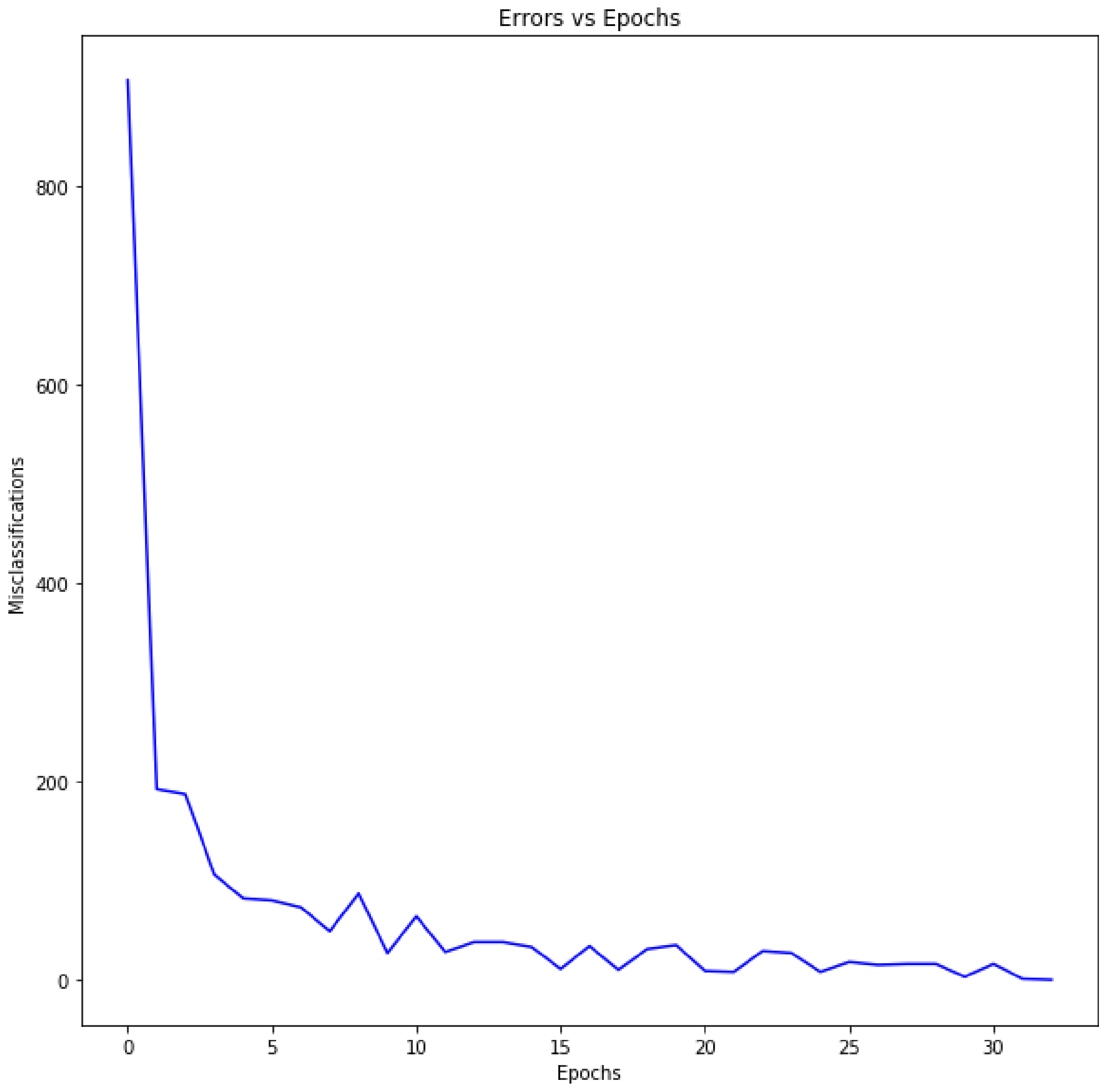
(f) The training part of the network does terminate with 0 errors and misclassifications decrease exponentially as number of epochs rise. When we test the model on training dataset we get 0% error and 100% accuracy as the model has been trained using the same images. The learned weights cannot generalize better to new unseen examples and therefore the test accuracy is lower than the training accuracy. Moreover, we observe that number of misclassifications in the training set comes down to 0% error after about 3 epochs.

The network got only 50 examples to train on and after that it was evaluated on 10,000 test samples. Hence, the network was biased towards those 50 training samples. Therefore, the test error rate is high and not 0%.

However at 0th epoch, the percentage of errors obtained in test data is 42.27% and far lower than the initial error obtained in the training data because we have updated the weights accordingly and the network better understands how to process the inputs after being trained for 5 epochs and not just take randomly initialized values for its weights.

In [67]:
```
# Initializing the weights to a ramdom value along with learning rate, epoch a
nd threshold
w = np.random.uniform(-1, 1, size=(10,784))
n = 1000
epoch = 0
threshold = 0.0
learning_rate = 1.0
errors = []
```

In [68]:
```
# This will plot a graph of Misclassifications against Epochs
learning_weights(w, epoch, threshold, learning_rate)
plot_graph(errors)
find_errors()
```



```
Errors in test data:  1846
Percentage of test errors:  18.46
```

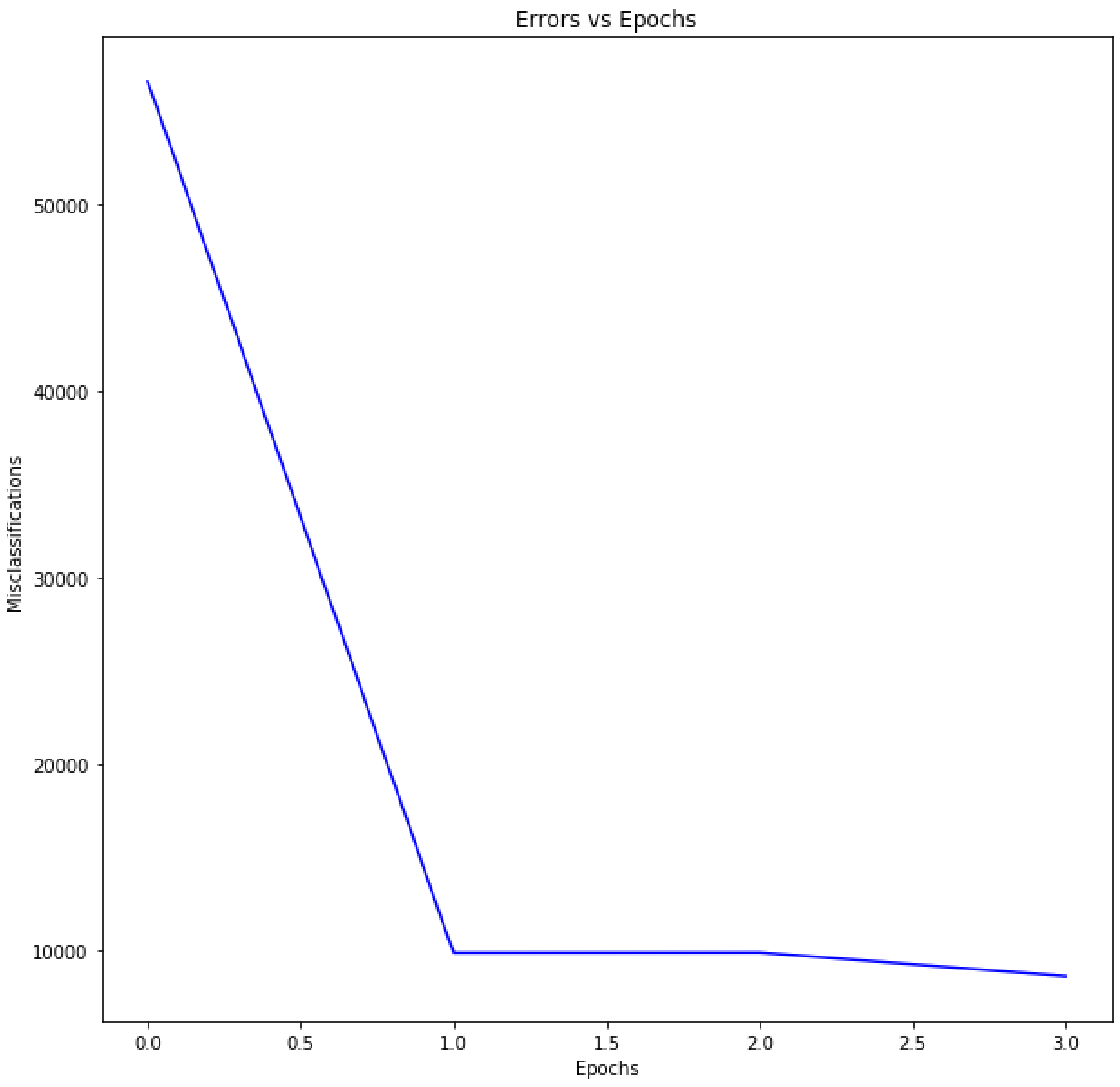
(g) It is observed that number of misclassifications in the training set comes down to 0% error after about 33 epochs. The number of epochs the network took to converge increased because there are 1000 training samples now.

The network got only 1000 examples to train on and after that it was evaluated on 10,000 test samples. Hence, the model is biased towards those 1000 training samples. The percentage of errors obtained in test data is 16.73% which is lower than the initial error obtained in the training data but not 0% because we have updated the weights accordingly.

The error rate approaches 0 but it can never be 0 because we want a more generalized model which is simply not overfitted on training samples and which is capable to handling unseen test samples and does not do a bad job in classifying the test samples.

In [69]:
```
# Initializing the weights to a ramdom value along with learning rate, epoch a
nd threshold
w = np.random.uniform(-1, 1, size=(10,784))
n = 60000
epoch = 0
threshold = 0.15
learning_rate = 1.0
errors = []
```

In [70]:
```
# This will plot a graph of Misclassifications against Epochs
learning_weights(w, epoch, threshold, learning_rate)
plot_graph(errors)
find_errors()
```



```
Errors in test data:  1457
Percentage of test errors:  14.57
```

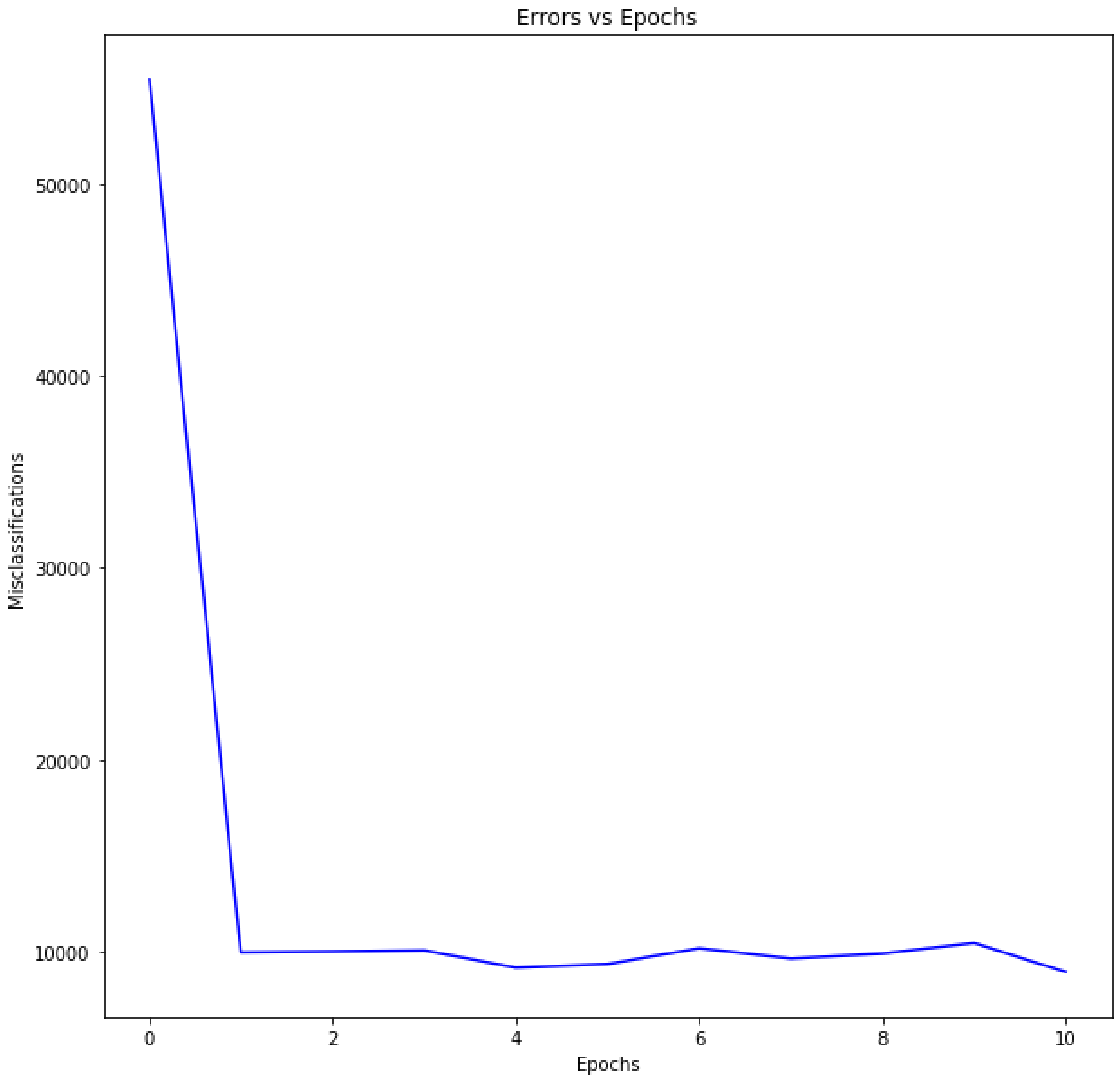
(h) The number of epochs have been limited to 100. After the first few epochs, we observe that the error rate does not fall anymore and we just keeps oscillating between 7,000 to 12,000 for the training data. For the testing data, the percentage of misclassified data is 15.19% that is less similar to n = 1000, which means that the network has learned a sufficient amount of knowledge from just 1000 samples and performs only a little bit better when trained over all 60,000 training samples.

(i) Based on the above results the threshold value I have chosen is 0.15 because it converges quickly and the results are very much comparable to a lower threshold value

First time

In [71]:
```
# Initializing the weights to a ramdom value along with learning rate, epoch a
nd threshold
w = np.random.uniform(-1, 1, size=(10,784))
n = 60000
epoch = 0
threshold = 0.15
learning_rate = 1.0
errors = []
```

In [72]:
```
# This will plot a graph of Misclassifications against Epochs
learning_weights(w, epoch, threshold, learning_rate)
plot_graph(errors)
find_errors()
```

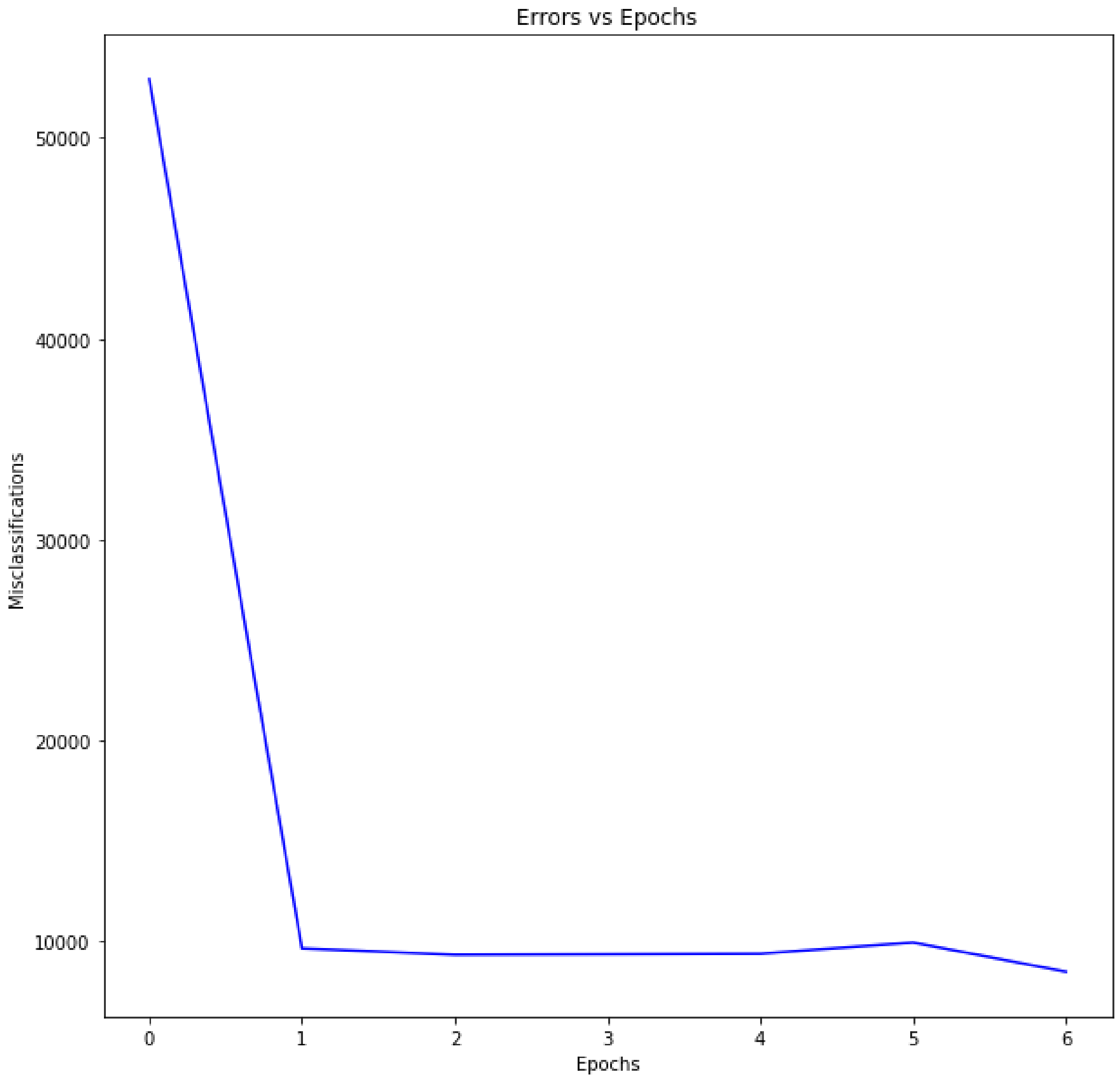


```
Errors in test data:  1537
Percentage of test errors:  15.37
```

Second Time

```
In [73]: # Initializing the weights to a ramdom value along with learning rate, epoch a
         nd threshold
         w = np.random.uniform(-1, 1, size=(10,784))
         n = 60000
         epoch = 0
         threshold = 0.15
         learning_rate = 1.0
         errors = []
```

```
In [74]: # This will plot a graph of Misclassifications against Epochs
         learning_weights(w, epoch, threshold, learning_rate)
         plot_graph(errors)
         find_errors()
```

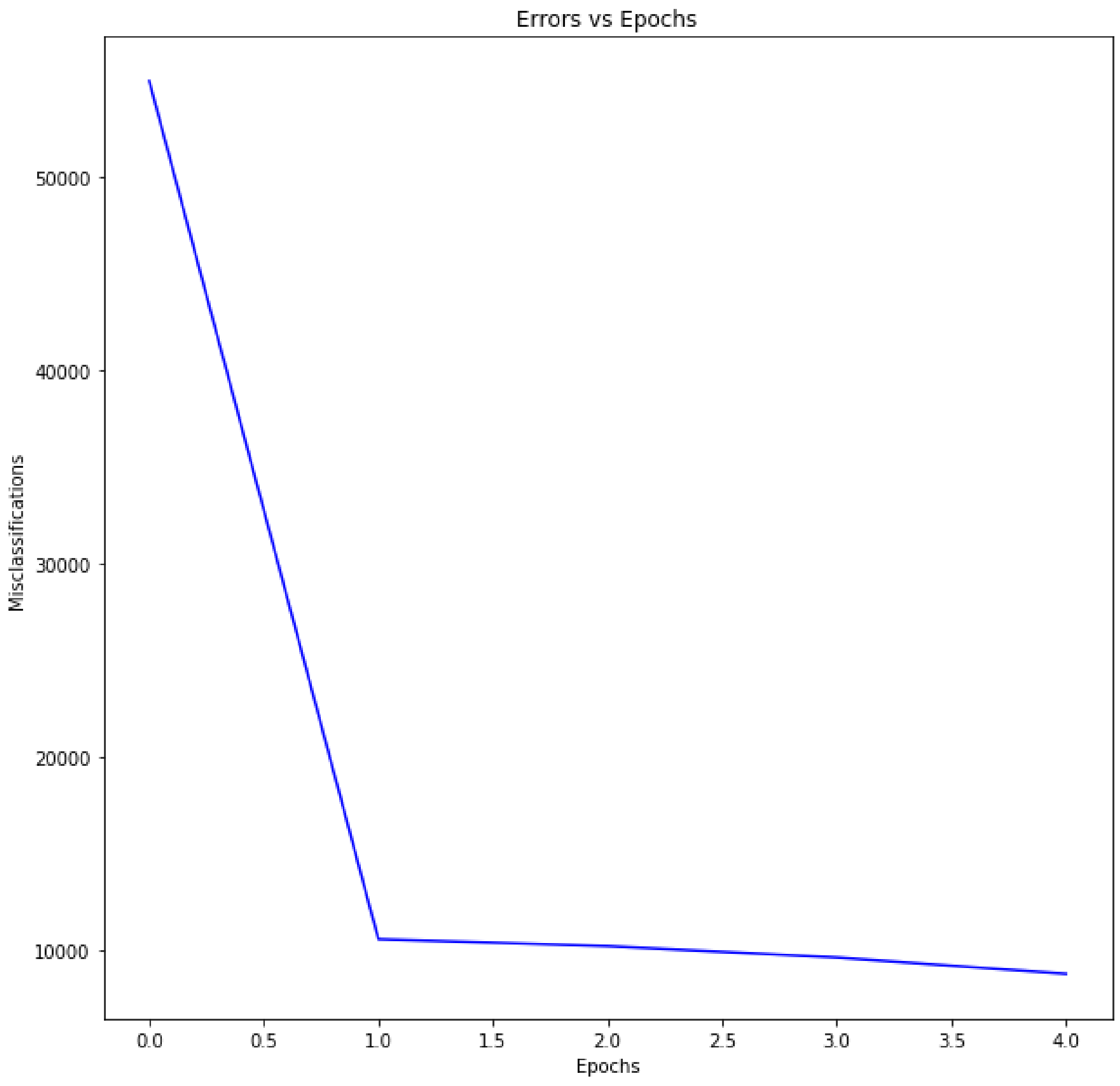


```
Errors in test data:  1840
Percentage of test errors:  18.4
```

Third time

In [75]:
```
# Initializing the weights to a ramdom value along with learning rate, epoch a
nd threshold
w = np.random.uniform(-1, 1, size=(10,784))
n = 60000
epoch = 0
threshold = 0.15
learning_rate = 1.25
errors = []
```

In [76]:
```
# This will plot a graph of Misclassifications against Epochs
learning_weights(w, epoch, threshold, learning_rate)
plot_graph(errors)
find_errors()
```



```
Errors in test data:  1646
Percentage of test errors:  16.46
```

The graphs for all the attempts are not much different because the percentage error varies from 14-16%, that is still comparable to the results where n=1000.

```
In [ ]:
```