Project: Create a neural network class

Based on previous code examples, develop a neural network class that is able to classify any dataset provided. The class should create objects based on the desired network architecture:

1. Number of inputs
2. Number of hidden layers
3. Number of neurons per layer
4. Number of outputs
5. Learning rate

The class must have the train, and predict functions.

Test the neural network class on the datasets provided below: Use the input data to train the network, and then pass new inputs to predict on. Print the expected label and the predicted label for the input you used. Print the accuracy of the training after predicting on different inputs.

Use matplotlib to plot the error that the train method generates.

Don't forget to install Keras and tensorflow in your environment!

Import the needed Packages

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

# Needed for the mnist data
from keras.datasets import mnist
from keras.utils import to_categorical
```

Define the class

In [20]:
```
class NeuralNetwork:

    def __init__(self, architecture, alpha):


        '''
            layers: List of integers which represents the architecture of the networ
            alpha:  Learning rate.
        '''

        # TODO: Initialize the list of weights matrices, then store
        # the network architecture and learning rate

        self.layers = architecture
        self.alpha = alpha
        np.random.seed(13)
        self.FW = np.random.randn(architecture[0], architecture[2])
        self.MW = np.empty((architecture[0] - 1, architecture[2], architecture[2]))

        for x in range(architecture[0] - 2):
            self.MW[x] = np.random.randn(architecture[2], architecture[2])

        self.LW = np.random.randn(architecture[2], architecture[3])
        self.FB = np.random.randn(architecture[2])
        self.MB = np.random.randn(architecture[1] - 1, architecture[2])
        self.LB = np.random.randn(architecture[3])

        pass


    def __repr__(self):

        return "NeuralNetwork: {}".format( "-".join(str(l) for l in self.layers))

    def softmax(self, X):

        expX = np.exp(X)
        return expX / expX.sum(axis=1, keepdims=True)

    def sigmoid(self, x):
        # the sigmoid for a given input value

        return 1.0 / (1.0 + np.exp(-x))

    def sigmoid_deriv(self, x):
        # the derivative of the sigmoid

        return x * (1 - x)

    def predict(self, inputs):
        # TODO: Define the predict function

        self.newWeights = np.empty((self.layers[1], inputs.shape[0], self.layers[2]))
        self.newWeights[0] = self.sigmoid(np.dot(inputs, self.FW) + self.FB)
        for x in range(self.layers[0] - 2):
            self.newWeights[x+1] = self.sigmoid(np.dot(self.newWeights[x], self.MW[x
        finalLevel = self.softmax( np.dot(self.newWeights[len(self.newWeights)-1], s

        return finalLevel

    def train(self, inputs, labels, epochs = 1000, displayUpdate = 100):

        fail = []
```

Test datasets

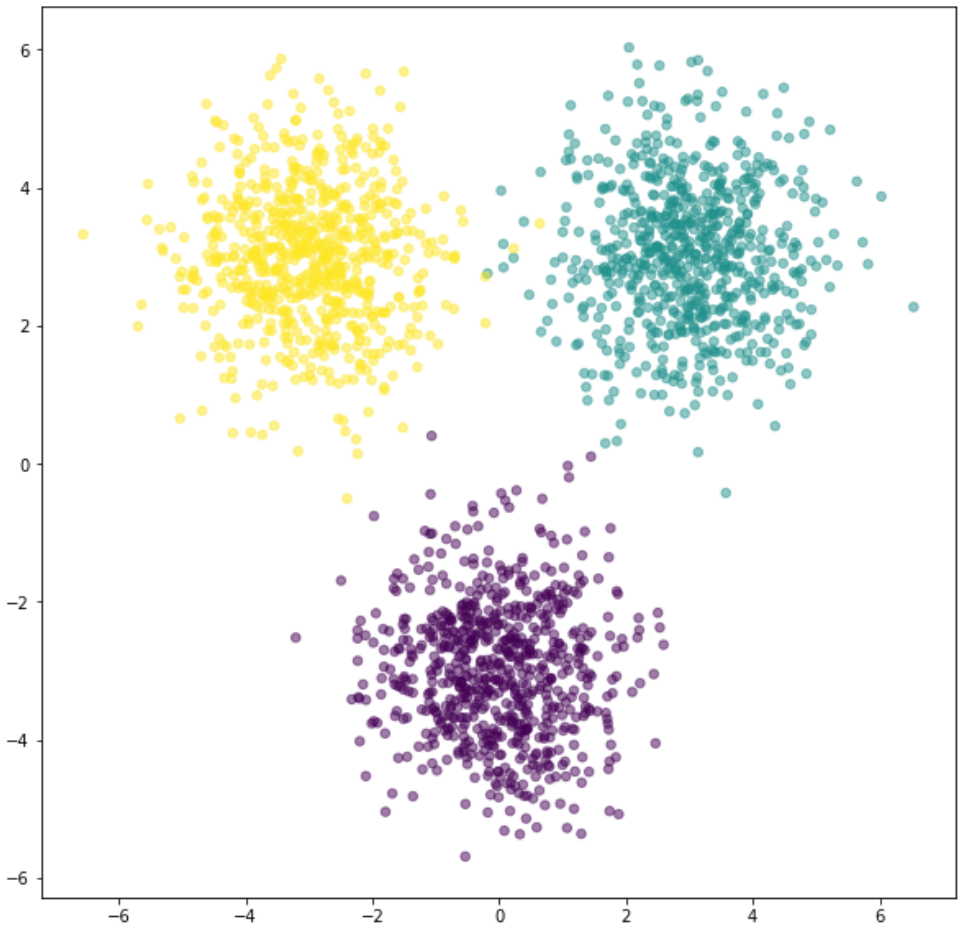
XOR

In [21]:
```
# input dataset
XOR_inputs = np.array([
                [0,0],
                [0,1],
                [1,0],
                [1,1]
            ])

# labels dataset
XOR_labels = np.array([[0,1,1,0]]).T
```

In [28]:
```
#TODO: Test the class with the XOR data
```

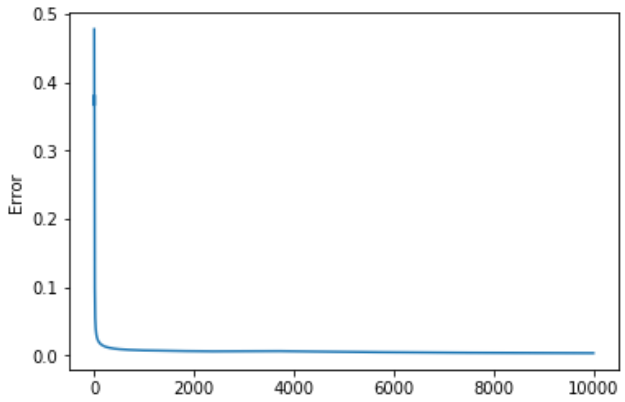
Multiple classes

In [23]:
```
# Creates the data points for each class
class_1 = np.random.randn(700, 2) + np.array([0, -3])
class_2 = np.random.randn(700, 2) + np.array([3, 3])
class_3 = np.random.randn(700, 2) + np.array([-3, 3])

feature_set = np.vstack([class_1, class_2, class_3])

labels = np.array([0]*700 + [1]*700 + [2]*700)

one_hot_labels = np.zeros((2100, 3))

for i in range(2100):
    one_hot_labels[i, labels[i]] = 1

plt.figure(figsize=(10,10))
plt.scatter(feature_set[:,0], feature_set[:,1], c=labels, s=30, alpha=0.5)
plt.show()
```



In [29]:
```
#TODO: Test the class with the multiple classes data

r = NeuralNetwork([2,2,5,3], 0.01)
fails = r.train(feature_set, one_hot_labels, 10000, 1000)
fig, ax = plt.subplots(1,1)
ax.set_ylabel('Error')
ax.plot(fails)
test = np.array([[0,-4]])
print(str(test) + str(network.predict(test)))
```

```
Error: 0.47806009636665425
Error: 0.007121601211763323
Error: 0.005938405234795827
Error: 0.005920131593441376
Error: 0.00585185558757003
Error: 0.0049490985751735606
Error: 0.004301948969147726
Error: 0.0038221899933325782
Error: 0.003507891190406313
Error: 0.003280260509683804

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-d491052e3341> in <module>
      7 ax.plot(fails)
      8 test = np.array([[0,-4]])
----> 9 print(str(test) + str(network.predict(test)))

<ipython-input-20-bb1c4eba42bc> in predict(self, inputs)
     59
     60         #first level
---> 61         self.newWeights[0] = self.sigmoid(np.dot(inputs, self.FW) + self.F
B)
     62
     63         #middle levels

ValueError: shapes (1,2) and (784,64) not aligned: 2 (dim 1) != 784 (dim 0)
```



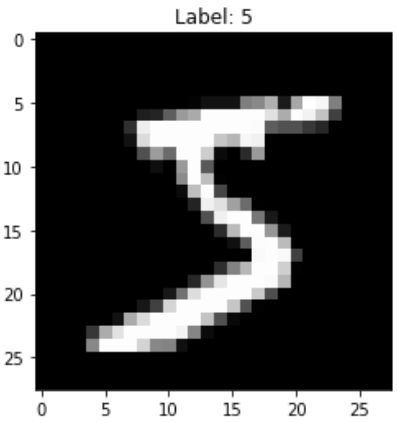
On the mnist data set

Train the network to classify hand drawn digits.

For this data set, if the training step is taking too long, you can try to adjust the architecture of the network to have fewer layers, or you could try to train it with fewer input. The data has already been loaded and preprocesed so that it can be used with the network.

In [25]:
```
# Load the train and test data from the mnist data set
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Plot a sample data point
plt.title("Label: " + str(train_labels[0]))
plt.imshow(train_images[0], cmap="gray")
```

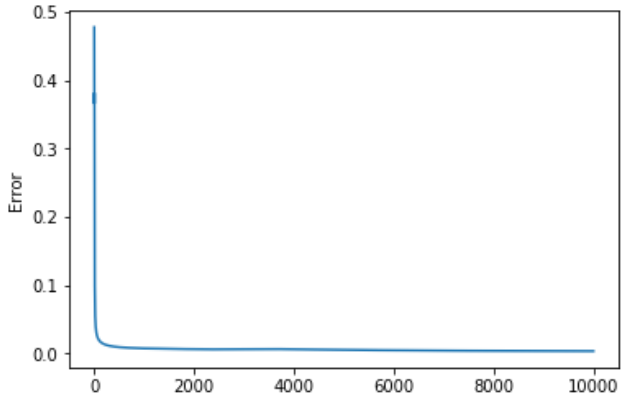
Out[25]: <matplotlib.image.AxesImage at 0xb40a2cb38>



In [26]:
```
# Standardize the data

# Flatten the images
train_images = train_images.reshape((60000, 28 * 28))
# turn values from 0-255 to 0-1
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

# Create one hot encoding for the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

In [42]:
```
# TODO: Test the class with the mnist data. Test the training of the network with th
# record the accuracy of the classification.

r = NeuralNetwork([2,2,5,3], 0.01)
fails = r.train(feature_set, one_hot_labels, 10000, 1000)
fig, ax = plt.subplots(1,1)
ax.set_ylabel('Error')
ax.plot(fails)
test = np.array([[0,-3]])
prf = network.predict(test_images[0:1000])
one_hot_test_labels = to_categorical(test_labels[0:1000])
np.set_printoptions(precision=10, suppress= True, linewidth=75)
guess = np.copy(prf)
guess[guess > 0.5] = 1
guess[guess < 0.5] = 0
fails = []
for index, (guess, label) in enumerate(zip(predictions[0:10], one_hot_test_labels[0:
    if not np.array_equal(prediction,label):
        fails.append((index, prediction, label))

for img, plot in zip(fails, plots):
    plot.imshow(test_images[img[0]].reshape(28,28), cmap = "gray")
    plot.set_title(str(img[1]))
```

```
Error: 0.47806009636665425
Error: 0.007121601211763323
Error: 0.005938405234795827
Error: 0.005920131593441376
Error: 0.00585185558757003
Error: 0.0049490985751735606
Error: 0.004301948969147726
Error: 0.0038221899933325782
Error: 0.003507891190406313
Error: 0.003280260509683804
```



In [ ]:

After predicting on the *test_images*, use matplotlib to display some of the images that were not correctly classified. Then, answer the following questions:

1. **Why do you think those were incorrectly classified?**
2. **What could you try doing to improve the classification accuracy?**

In [ ]: