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 [142]:

```
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 [143]:

```
np.random.seed(200)


class NeuralNetwork:
```

```python
    def __init__(self, architecture, alpha):
        '''
            layers: List of integers which represents the architecture of the networ
            alpha:  Learning rate.
        '''

        _inputs, _layer, _neurons, _output = architecture

        self.alpha = alpha
        self.layers = _layer
        self.neurons = _neurons

        # Weights

        self.initialWeight = np.random.randn(_inputs, _neurons)
        self.middleWeight = np.zeros((_layer-1, _neurons, _neurons))
        self.lastWeight = np.random.randn(_neurons, _output)
        self.calcWeight = []

        # Bias

        self.initialBias = np.random.randn(_neurons)
        self.middleBias = np.random.randn(_layer - 1, _neurons)
        self.lastBias = np.random.randn(_output)
        for i in range(_layer - 1):
            self.middleWeight[i] = np.random.randn(_neurons, _neurons)

        pass


    def __repr__(self):
        # construct and return a string that represents the network
        # architecture
        return "NeuralNetwork: {}".format( "-".join(str(l) for l in self.layers))


    def softmax(self, X):
        # applies the softmax function to a set of values

        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)
```

```python
    def predict(self, inputs):
        self.calcWeight = np.zeros( (self.layers, inputs.shape[0], self.neurons) )

        self.calcWeight[0] = self.sigmoid( np.dot(inputs, self.initialWeight) + self

        for i in range(self.layers - 1):
            self.calcWeight[i + 1] = self.sigmoid( np.dot(self.calcWeight[i], self.m

        return self.softmax( np.dot(self.calcWeight[len(self.calcWeight)-1], self.la

    def train(self, inputs, labels, epochs = 1000, displayUpdate = 100):
        # TODO: Define the training step for the network. It should include the forw
        # steps, the updating of the weights, and it should print the error every 'd
        # It must return the errors so that they can be displayed with matplotlib
        error = []
        for i in range(epochs):

            # Forward propagation

            prediction = self.predict(inputs)
            level_error = labels - prediction
            error.append( np.average(np.abs(level_error)) )

            # Back propagation

            level_delta_last = level_error * self.sigmoid_deriv(prediction)
            level_error_middle = np.dot(level_delta_last, self.lastWeight.T)
            level_delta_middle = level_error_middle * self.sigmoid_deriv(self.calcWe

            b_delta_last = np.sum(level_delta_last)

            self.lastBias += b_delta_last * self.alpha
            self.lastWeight += np.dot(self.calcWeight[-1].T, level_delta_last) * sel


            self.initialWeight += np.dot(inputs.T, level_delta_middle) * self.alpha
            b_delta_last = np.sum(level_delta_middle)
            self.initialBias += b_delta_last * self.alpha

            for j in range(self.layers - 1):
                tmp = (len(self.middleWeight) - 1) - j
                tmp2 = (len(self.calcWeight) - 2) - j

                level_error_middle = np.dot(level_delta_middle, self.middleWeight[tm
                self.middleWeight[tmp] += np.dot(self.calcWeight[tmp2].T, level_delt
                b_delta_middle = np.sum(level_delta_middle)
                self.middleBias[j] += b_delta_middle * self.alpha
                level_delta_middle = level_error_middle * self.sigmoid_deriv(self.ca

            if i % displayUpdate == 0:
                print("Error: ", error[-1])

        return error
```

Test datasets

XOR

In [144]:

```
# 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
labels = np.array([0]*2 + [1]*2)
one_hot_labels = np.zeros((4, 2))
for i in range(4):
    one_hot_labels[i, XOR_labels[i]] = 1
```

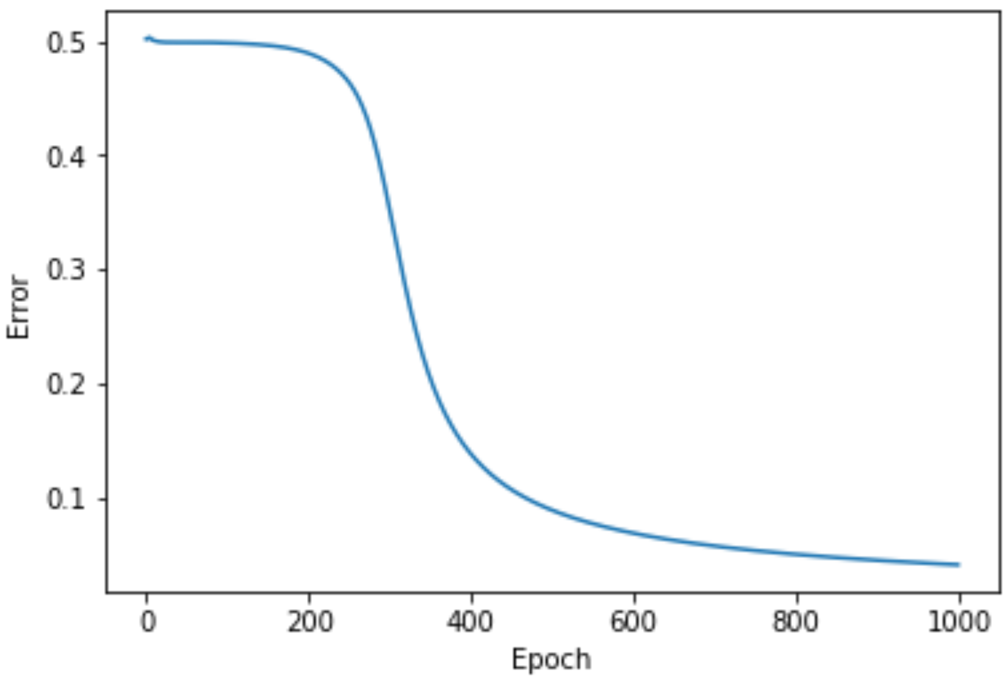
In [145]:

```
# TODO: Test the class with the XOR data
# inputs, hlayers, neurons, outputs
architecture_1 = [2, 1, 4, 2]

neural_network_1 = NeuralNetwork(architecture_1, 1)
error_1 = neural_network_1.train(XOR_inputs, one_hot_labels)

f, p = plt.subplots(1,1)
p.set_xlabel('Epoch')
p.set_ylabel('Error')

p.plot(error_1)
```

```
Error:  0.5017493293680011
Error:  0.4983728217824976
Error:  0.48930687453185734
Error:  0.3512074681728592
Error:  0.13792298984674883
Error:  0.088764066749876
Error:  0.06865791409948196
Error:  0.05744263493277516
Error:  0.05014917394690713
Error:  0.04495732435452973
```

Out[145]:

```
[<matplotlib.lines.Line2D at 0x10eda6780>]
```

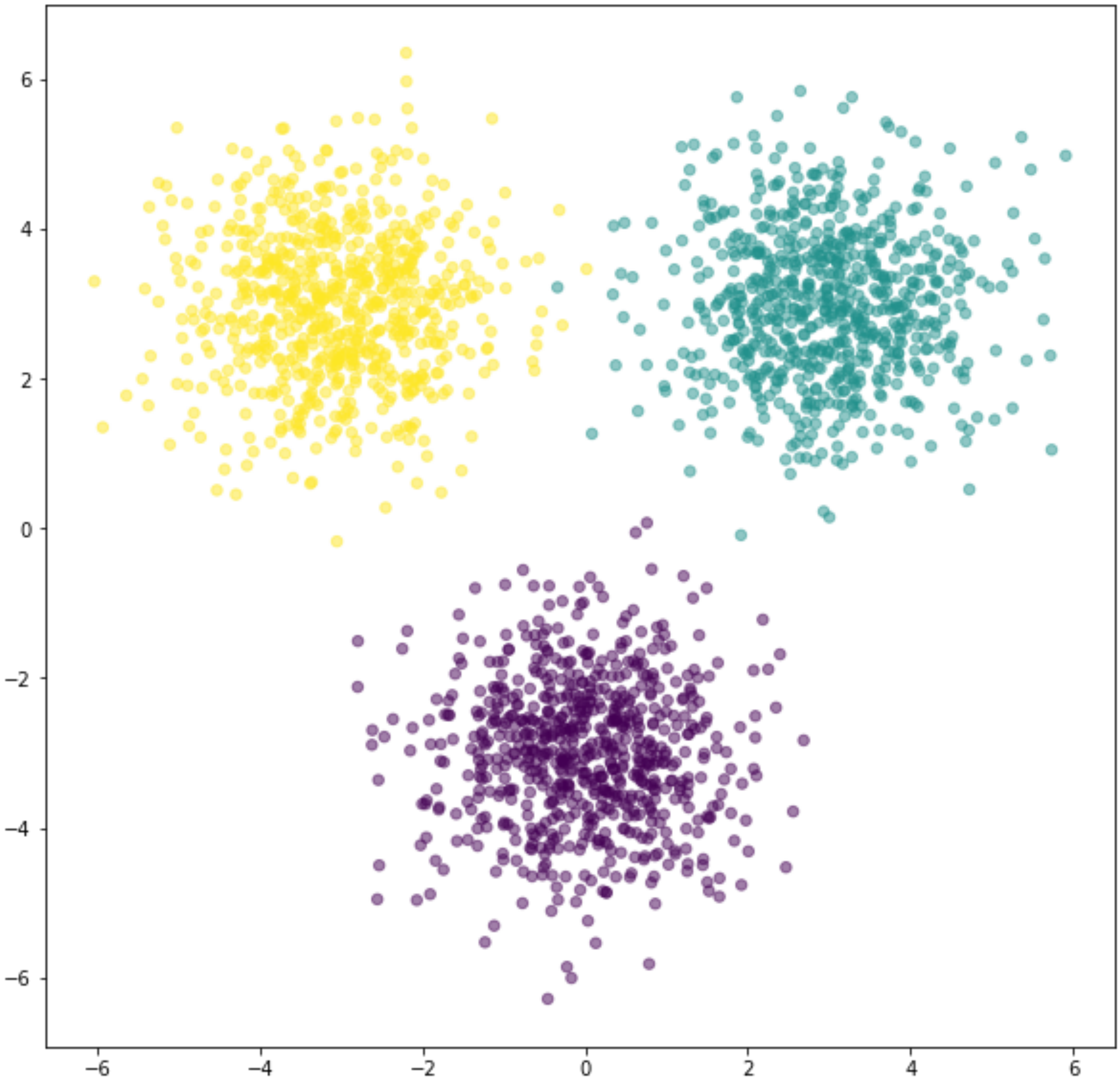


Multiple classes

In [146]:

```
# 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_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()
```



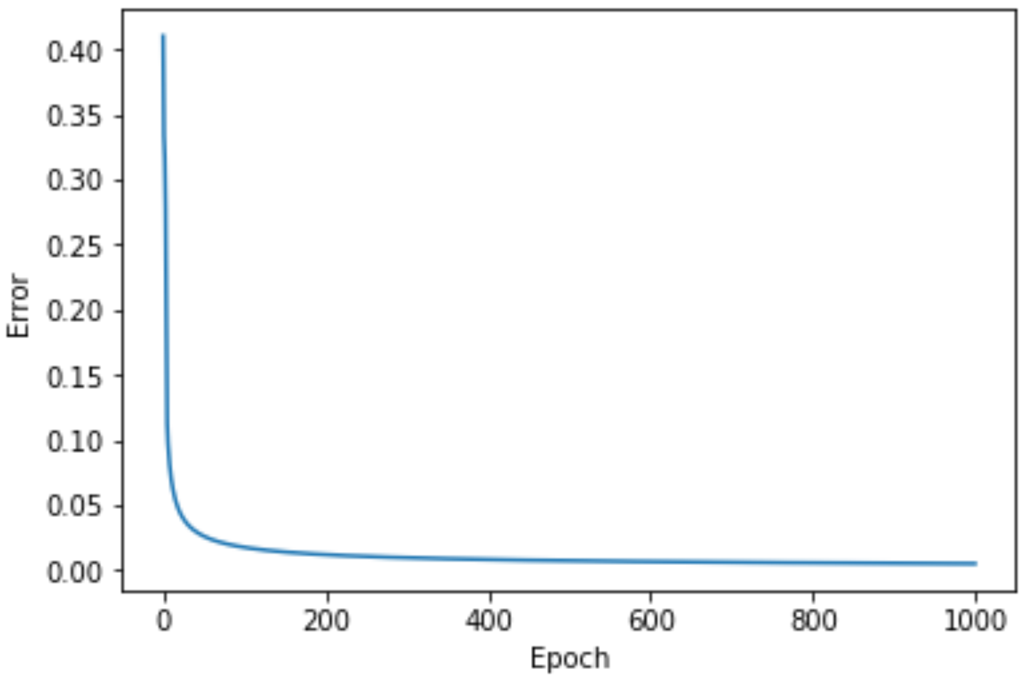
In [147]:

```
#TODO: Test the class with the multiple classes data
architecture_2 = [2, 1, 4, 3]
neural_network_2 = NeuralNetwork(architecture_2, 0.01)
error_2 = neural_network_2.train(feature_set, one_hot_labels)

f, p2 = plt.subplots(1,1)
p2.set_xlabel('Epoch')
p2.set_ylabel('Error')

p2.plot(error_2)
```

```
Error:  0.41045926139047656
Error:  0.01760283280870611
Error:  0.011973600644582638
Error:  0.009606340638471253
Error:  0.008235905099066517
Error:  0.007316719946206627
Error:  0.006645224628868974
Error:  0.006126936415419389
Error:  0.005711373561470597
Error:  0.0053687459825060065
```

Out[147]:

```
[<matplotlib.lines.Line2D at 0xb4191af60>]
```



On the mnist data set

Train the network to classify hand drawn digits.

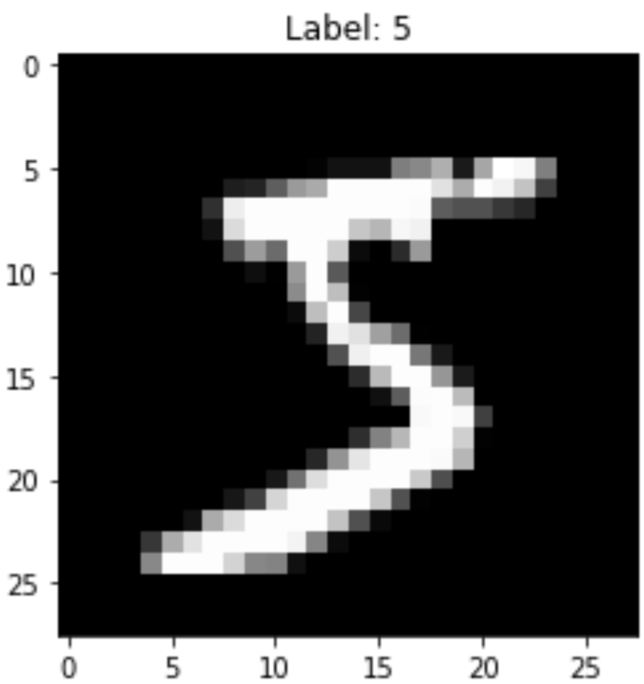
For this data set, if the training step is taking too long, you can try to adjust the architecture of the network to have fewer layers, or you could try to train it with fewer input. The data has already been loaded and preprocesed so that it can be used with the network.

In [148]:

```
# 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[148]:

```
<matplotlib.image.AxesImage at 0xb437c5c18>
```



In [149]:

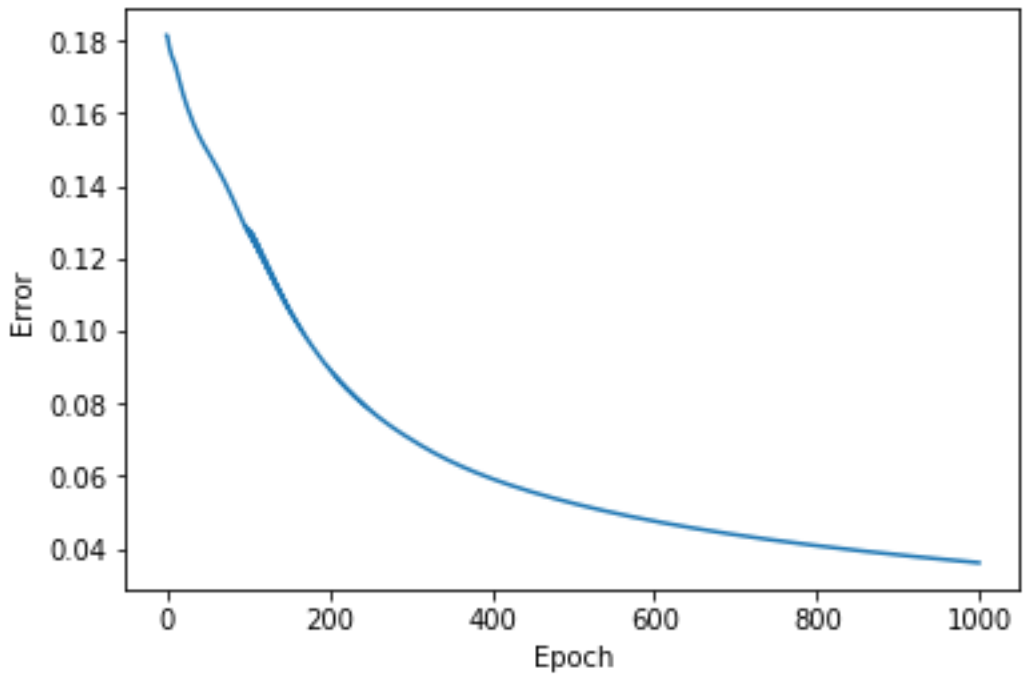
```
# Standardize the data

# 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)
```

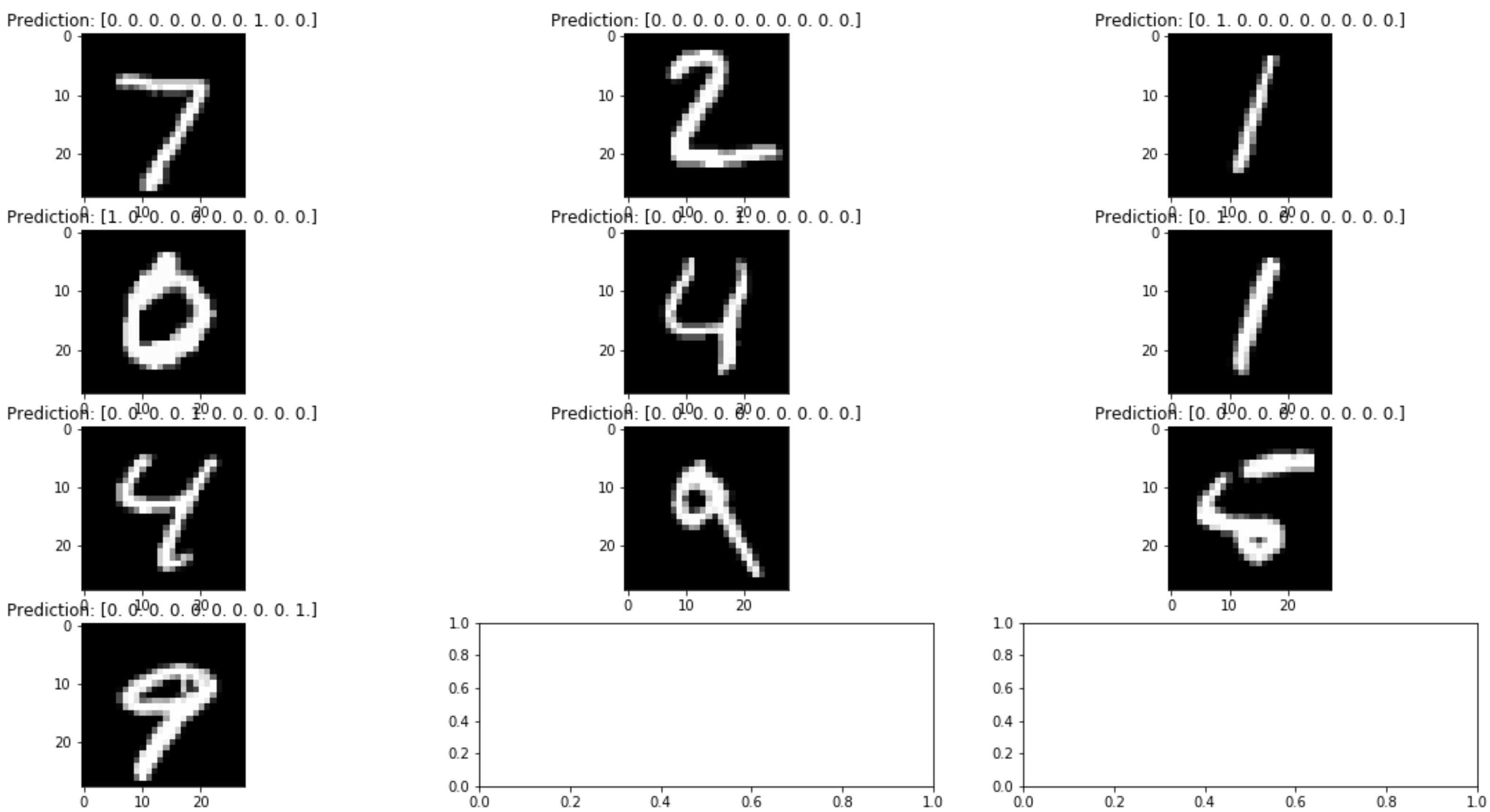
In [150]:

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

architecture_3 = [train_images.shape[1], 1, 64, 10]
neural_network_3 = NeuralNetwork(architecture_3, 0.0007)
error_3 = neural_network_3.train(train_images[0:5000], train_labels[0:5000])

f, p3 = plt.subplots(1,1)
p3.set_xlabel('Epoch')
p3.set_ylabel('Error')

p3.plot(error_3)

tests = neural_network_3.predict(test_images[0:1000])
# create one hot encoding on the test data
one_hot_test_labels = to_categorical(test_labels[0:1000])

np.set_printoptions(precision=3, suppress= True, linewidth=75)
# turn predictions to one hot encoding labels
predictions = np.copy(tests)
predictions[predictions > 0.5] = 1
predictions[predictions < 0.5] = 0

error_predictions = []
for index, (prediction, label) in enumerate(zip(predictions[0:10], one_hot_test_labe
    if not np.array_equal(prediction,label):
        error_predictions.append((index, prediction, label))

f, plots = plt.subplots((len(error_predictions)+3-1)//3, 3, figsize=(20,10))
plots = [plot for sublist in plots for plot in sublist]

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

```
Error:  0.18143985456065 3
```

```
Error:  0.181439854560653
Error:  0.12712597581483073
Error:  0.0896552578058866
Error:  0.07018148892509353
Error:  0.05928412649464351
Error:  0.05249161452742556
Error:  0.047573538274639504
Error:  0.04382075917153189 4
Error:  0.04080964587797075
Error:  0.038286800222143 6
```





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?**

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 [142]:

```
```

Define the class

In [143]:

```
```

Test datasets

XOR

In [144]:

```

```

In [145]:

```

```

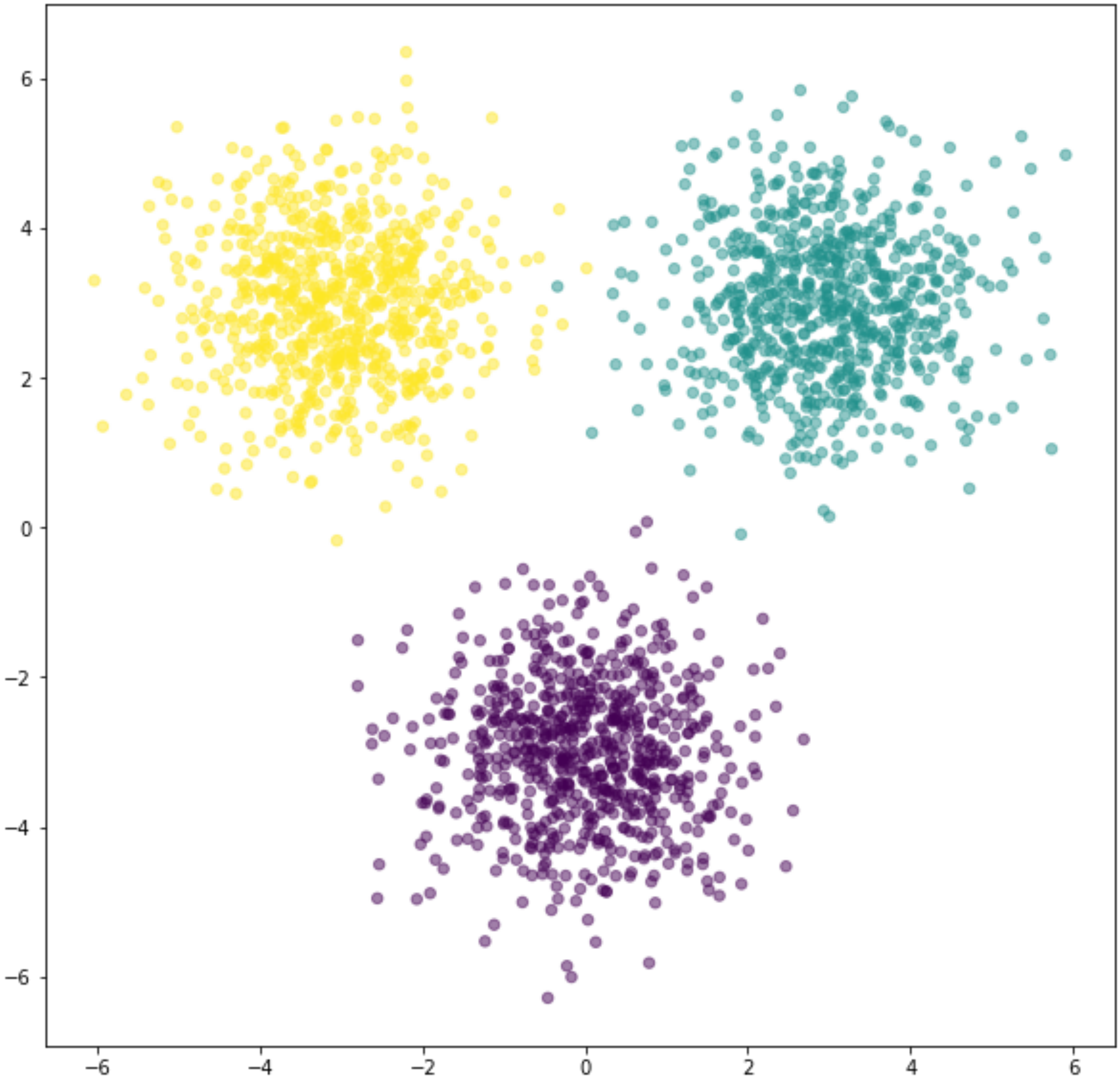
```
Error:  0.5017493293680011
Error:  0.4983728217824976
Error:  0.48930687453185734
Error:  0.3512074681728592
Error:  0.13792298984674883
Error:  0.088764066749876
Error:  0.06865791409948196
Error:  0.05744263493277516
Error:  0.05014917394690713
Error:  0.04495732435452973
```

Out[145]:

```
[<matplotlib.lines.Line2D at 0x10eda6780>]
```

Multiple classes

In [146]:



In [147]:

```
Error:  0.41045926139047656
Error:  0.01760283280870611
Error:  0.011973600644582638
Error:  0.009606340638471253
Error:  0.008235905099066517
Error:  0.007316719946206627
Error:  0.006645224628868974
Error:  0.006126936415419389
Error:  0.005711373561470597
Error:  0.0053687459825060065
```

Out[147]:

```
[<matplotlib.lines.Line2D at 0xb4191af60>]
```

On the mnist data set

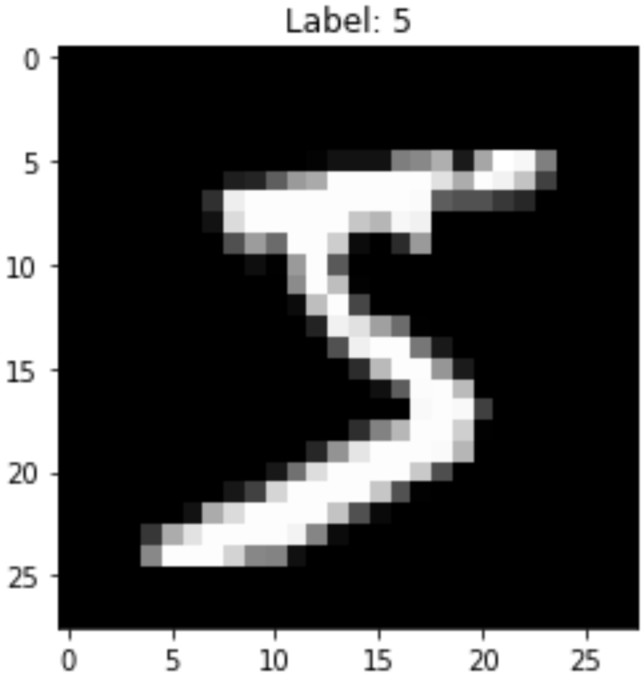
Train the network to classify hand drawn digits.

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 [148]:

Out[148]:

```
<matplotlib.image.AxesImage at 0xb437c5c18>
```



In [149]:

In [150]:

```
Error:  0.181439854560653
Error:  0.12712597581483073
Error:  0.0896552578058866
Error:  0.07018148892509353
Error:  0.05928412649464351
Error:  0.05249161452742556
Error:  0.047573538274639504
Error:  0.043820759171531894
Error:  0.04080964587797075
Error:  0.0382868002221436
```

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?**