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 [ ]:
```
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 [ ]:

```
class NeuralNetwork:

    def __init__(self, architecture, alpha):
        '''
            layers: List of integers which represents the architecture of the
            alpha:  Learning rate.
        '''
        # TODO: Initialize the list of weights matrices, then store
        # the network architecture and learning rate

        inputs, layers, neurons, outputs = architecture

        self.alpha = alpha
        self.layers = layers
        self.neurons = neurons
        self.w1 = np.random.randn(inputs, neurons)
        self.w2 = np.zeros((layers - 1, neurons, neurons))
        for i in range(layers - 1):
            self.w2[i] = np.random.randn(neurons, neurons)
        self.w3 = np.random.randn(neurons, outputs)
        self.b1 = np.random.randn(neurons)
        self.b2 = np.random.randn(layers - 1, neurons)
        self.b3 = np.random.randn(outputs)

        self.wT = []

        pass

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

    @staticmethod
    def softmax(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)

    def predict(self, inputs):
        # TODO: Define the predict function
        self.wT = np.zeros((self.layers, inputs.shape[0], self.neurons))
        self.wT[0] = self.sigmoid(np.dot(inputs, self.w1) + self.b1)
        for i in range(self.layers - 1):
            self.wT[i + 1] = self.sigmoid(np.dot(self.wT[i], self.w2[i]) + sel
```

```python

        return self.softmax(np.dot(self.wT[len(self.wT) - 1], self.w3) + self.

    def train(self, inputs, labels, epochs = 1000, displayUpdate = 100):
        # TODO: Define the training step for the network. It should include th
        # steps, the updating of the weights, and it should print the error ev
        # It must return the errors so that they can be displayed with matplot

        e = []
        for i in range(epochs):
            p = self.predict(inputs)
            e1 = labels - p
            e.append(np.average(np.abs(e1)))

            d3 = e1 * self.sigmoid_deriv(p)
            e2 = np.dot(d3, self.w3.T)
            d2 = e2 * self.sigmoid_deriv(self.wT[-1])
            b4 = np.sum(d3)

            self.b3 += b4 * self.alpha
            self.w3 += np.dot(self.wT[-1].T, d3) * self.alpha

            self.w1 += np.dot(inputs.T, d2) * self.alpha
            b4 = np.sum(d2)
            self.b1 += b4 * self.alpha

            for j in range(self.layers - 1):
                temp  = (len(self.w2) - 1) - j
                temp2 = (len(self.wT) - 2) - j

                e2 = np.dot(d2, self.w2[temp])
                self.w2[temp] += np.dot(self.wT[temp2].T, d2) * self.alpha
                b2 = np.sum(d2)
                self.b2[j] += b2 * self.alpha
                d2 = e2 * self.sigmoid_deriv(self.wT[temp2])

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

        return e
```

Test datasets

XOR

In [158]:

```python
# 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
hot_labels = np.zeros((4, 2))
for i in range(4):
    hot_labels[i, XOR_labels[i]] = 1
```

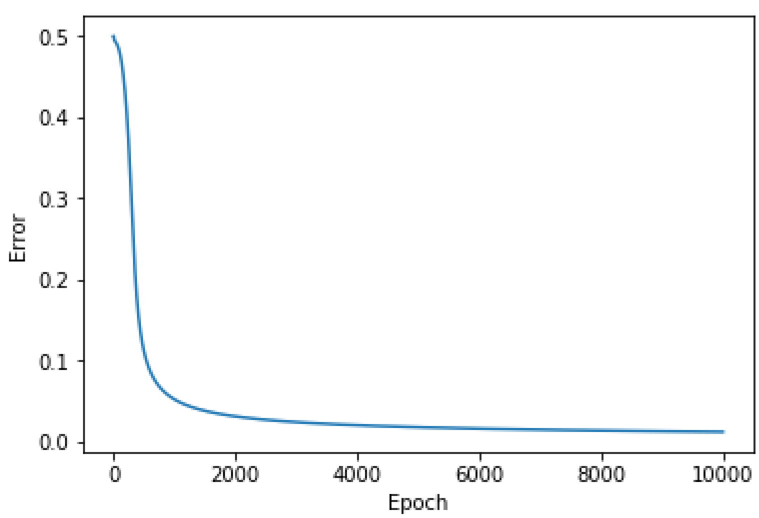
In [168]:

```python
#TODO: Test the class with the XOR data
arch = [2, 1, 4, 2]

NN = NeuralNetwork(arch, 0.5)

test = NN.train(XOR_inputs, hot_labels, 10000, 1000)

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

```
Error:  0.5001492298722212
Error:  0.052145621170102026
Error:  0.03114567950809449
Error:  0.024044805453280443
Error:  0.020225780817571516
Error:  0.017763879864099844
Error:  0.016012486500600713
Error:  0.01468651413573158
Error:  0.01363854610299885
Error:  0.012783816630627696
```

Out[168]:

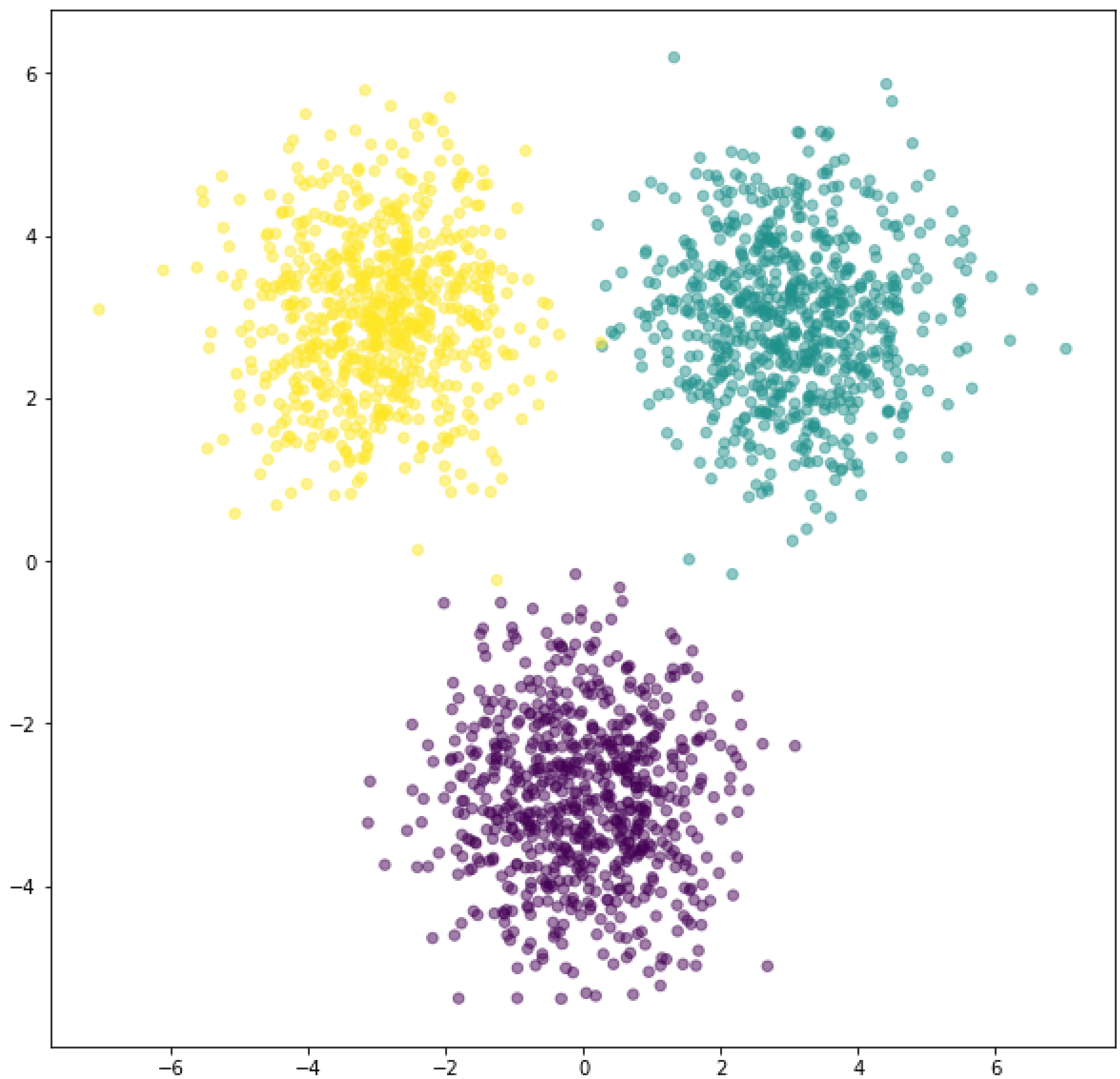
```
[<matplotlib.lines.Line2D at 0x1cdf0af8a90>]
```



Multiple classes

In [85]:

```
# 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()
```



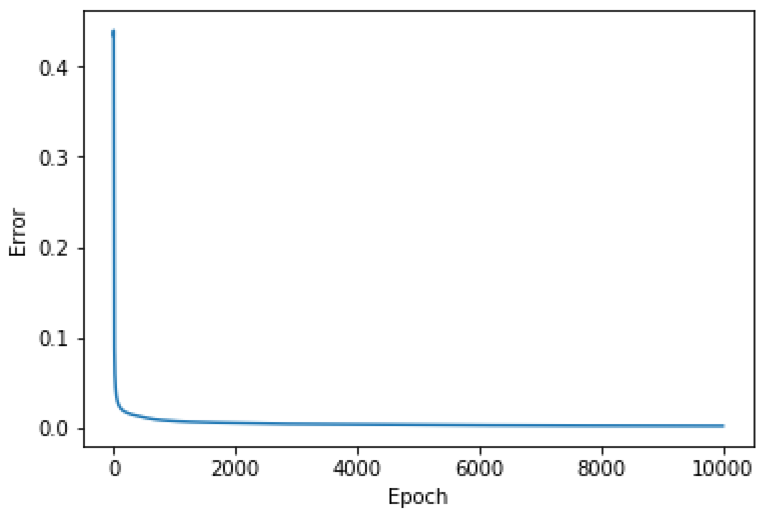
In [171]:

```python
#TODO: Test the class with the multiple classes data
arch2 = [2, 2, 5, 3]

NN2 = NeuralNetwork(arch2, 0.01)

test2 = NN2.train(feature_set, one_hot_labels, 10000, 1000)

f, p2 = plt.subplots(1,1)
p2.set_xlabel('Epoch')
p2.set_ylabel('Error')
p2.plot(test2)
```

```
Error:  0.4340991251649719
Error:  0.007899980940811869
Error:  0.00579617488997552
Error:  0.004692680417311391
Error:  0.004072665950250021
Error:  0.0037220548248621905
Error:  0.003318030023827824
Error:  0.00296658733493185
Error:  0.0027582023960030923
Error:  0.0026158649904001138
```

Out[171]:

```
[<matplotlib.lines.Line2D at 0x1cdf0b68128>]
```



On the mnist data set

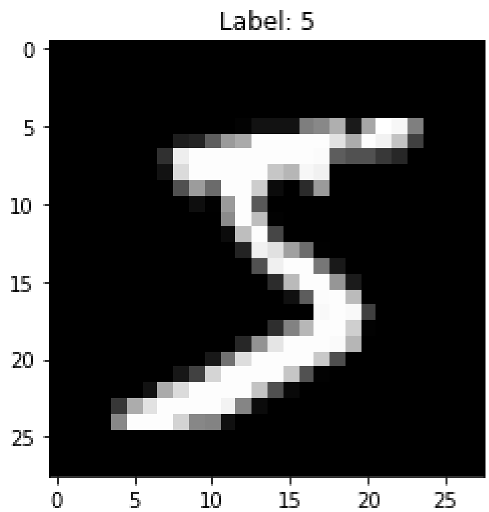
Train the network to classify hand drawn digits.

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 [193]:

```python
# 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[193]: <matplotlib.image.AxesImage at 0x1cd8079f438>



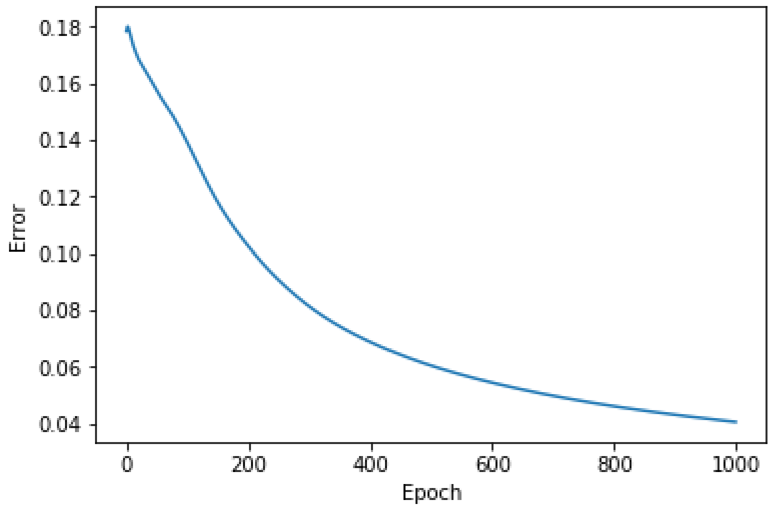
In [194]:

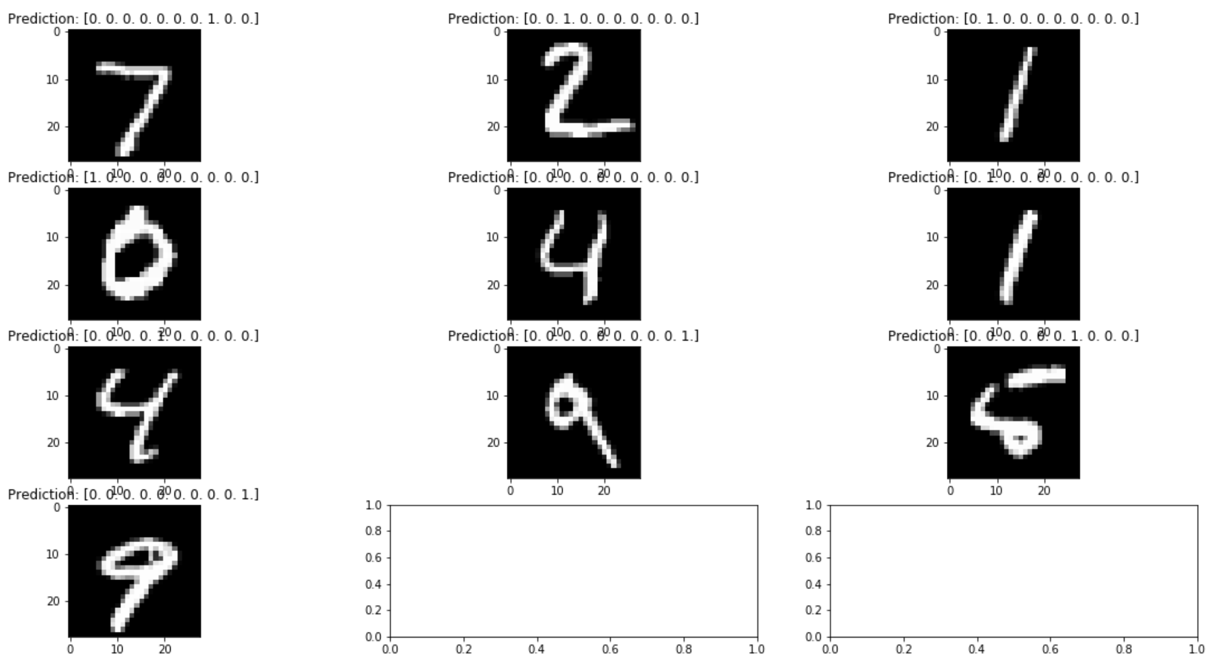
```python
# Standardize the data

# 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 [200]:

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

arch3 = [train_images.shape[1], 1, 64, 10]

NN3 = NeuralNetwork(arch3, 0.0005)

test3 = NN3.train(train_images[0:5000], train_labels[0:5000], 1000, 100)

f, p3 = plt.subplots(1,1)
p3.set_xlabel('Epoch')
p3.set_ylabel('Error')
p3.plot(test3)

test4 = NN3.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 = 50)

# turn predictions to one hot encoding labels
predictions = np.copy(test4)
predictions[predictions > 0.5] = 1
predictions[predictions < 0.5] = 0

error_predictions = []
for index, (prediction, label) in enumerate(zip(predictions[0:10], one_hot_te
    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:  0.17837547071470752
Error:  0.13908448888998953
Error:  0.10255748558218525
Error:  0.08142391919435729
Error:  0.06884309268063253
Error:  0.060457891182728374
Error:  0.0544508429341307
Error:  0.049825324135395824
Error:  0.04613892005206881
Error:  0.04313388000309443
```





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?** The inconsistency of the numbers in the images made the model fail in some results.
2. **What could you try doing to improve the classification accuracy?** Adding more training data, and tweaking the parameters.

In [ ]:
```
1
```