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 [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

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

```
Using TensorFlow backend.
```

Define the class

In [2]:
```
class NeuralNetwork:
```

```
    def __init__(self, architecture, alpha):
        '''
            layers: List of integers which represents the architecture
of the network.
            alpha:  Learning rate.
        '''
        # TODO: Initialize the list of weights matrices, then store
        # the network architecture and learning rate
        self.alpha = alpha
        self.layers = architecture
        self.weights = [0 for i in range(len(architecture)-1)]
        self.bias = [0 for i in range(len(architecture)-1)]
        self.output = []
        if len(architecture) < 2:
            print("Error: you need at least two layers in your network
")
        for i in range(len(architecture)-1):
            self.weights[i] = (np.random.randn(architecture[i], archit
ecture[i+1]))
            self.bias[i] = (np.random.randn(architecture[i+1]))
        print("Layer architecture: ")
        for i in range(len(self.weights)):
            print("Layer", i, ": ",self.weights[i].shape)
        print("Bias: ")
        for j in range(len(self.bias)):
            print("Layer", j, ": ",self.bias[j].shape)


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

    def predict(self, inputs):
```

```
        # TODO: Define the predict function
        prediction = [0 for i in range(len(self.weights)+1)]
        prediction[0] = inputs
        for i in range(1, len(prediction)-1):
            prediction[i] = self.sigmoid(np.dot(prediction[i-1], self.
weights[i-1]) + self.bias[i-1])
        prediction[-1] = self.softmax(np.dot(prediction[-2], self.weig
hts[-1]) + self.bias[-1])
        return prediction[-1]

    def train(self, inputs, labels, epochs = 1000, displayUpdate = 100
):
        # TODO: Define the training step for the network. It should in
clude the forward and back propagation
        # steps, the updating of the weights, and it should print the
error every 'displayUpdate' epochs
        # It must return the errors so that they can be displayed with
matplotlib
#         np.seterr(divide='ignore', invalid='ignore')
        errors = []
        for epoch in range(epochs):
            error = [0 for i in range(len(self.weights))]
            delta = [0 for i in range(len(self.weights))]
            biasDelta = [0 for i in range(len(self.bias))]
            prediction = [0 for i in range(len(self.weights)+1)]
            #forward propagation
            prediction[0] = inputs
            for i in range(1, len(prediction)-1):
                prediction[i] = self.sigmoid(np.dot(prediction[i-1], s
elf.weights[i-1]) + self.bias[i-1])
            prediction[-1] = self.softmax(np.dot(prediction[-2], self.
weights[-1]) + self.bias[-1])
            #back propagation
            error[-1] = labels - prediction[-1]
            delta[-1] = error[-1] * self.sigmoid_deriv(prediction[-1])
            errors.append(np.mean(np.abs(error[-1])))
            if epoch%displayUpdate == 0:
                print("Error in epoch(",epoch,"):", np.mean(np.abs(err
or[-1])))
            i = len(self.weights)-2
            while i > -1:
                error[i] = np.dot(delta[i+1], self.weights[i+1].T)
                delta[i] = error[i] * self.sigmoid_deriv(prediction[i+
1])
                i-=1
            for i in range(len(self.weights)):
                biasDelta[i] = np.sum(delta[i])
                self.weights[i] += np.dot(prediction[i].T, delta[i]) *
self.alpha
                self.bias[i] += biasDelta[i] * self.alpha
```

```
        return errors
```

Test datasets

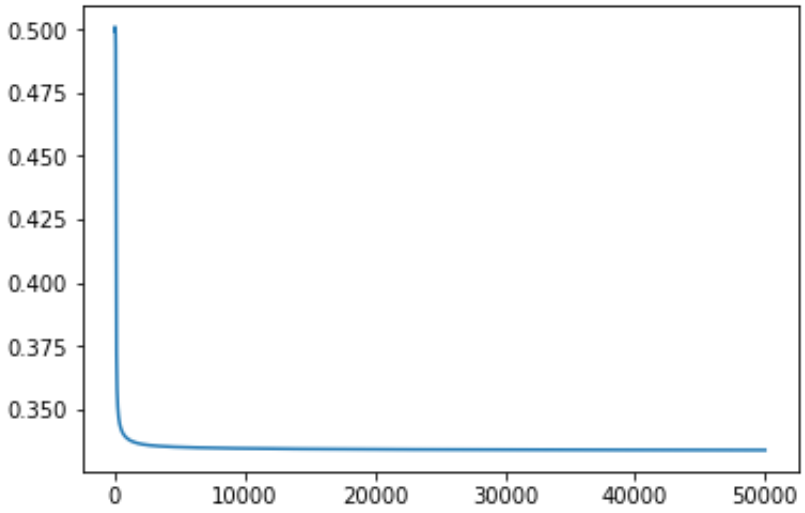
XOR

In [3]:
```
# 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

XOR_labels = np.array([
    [1,0],
    [0,1],
    [0,1],
    [1,0]
])
```

In [4]:
```
#TODO: Test the class with the XOR data
network = NeuralNetwork([2,2,2],4)
print(network)
errors = network.train(XOR_inputs, XOR_labels,50000, 10000)
plt.plot(errors)
# a.evaluate(XOR_inputs, XOR_labels)
```

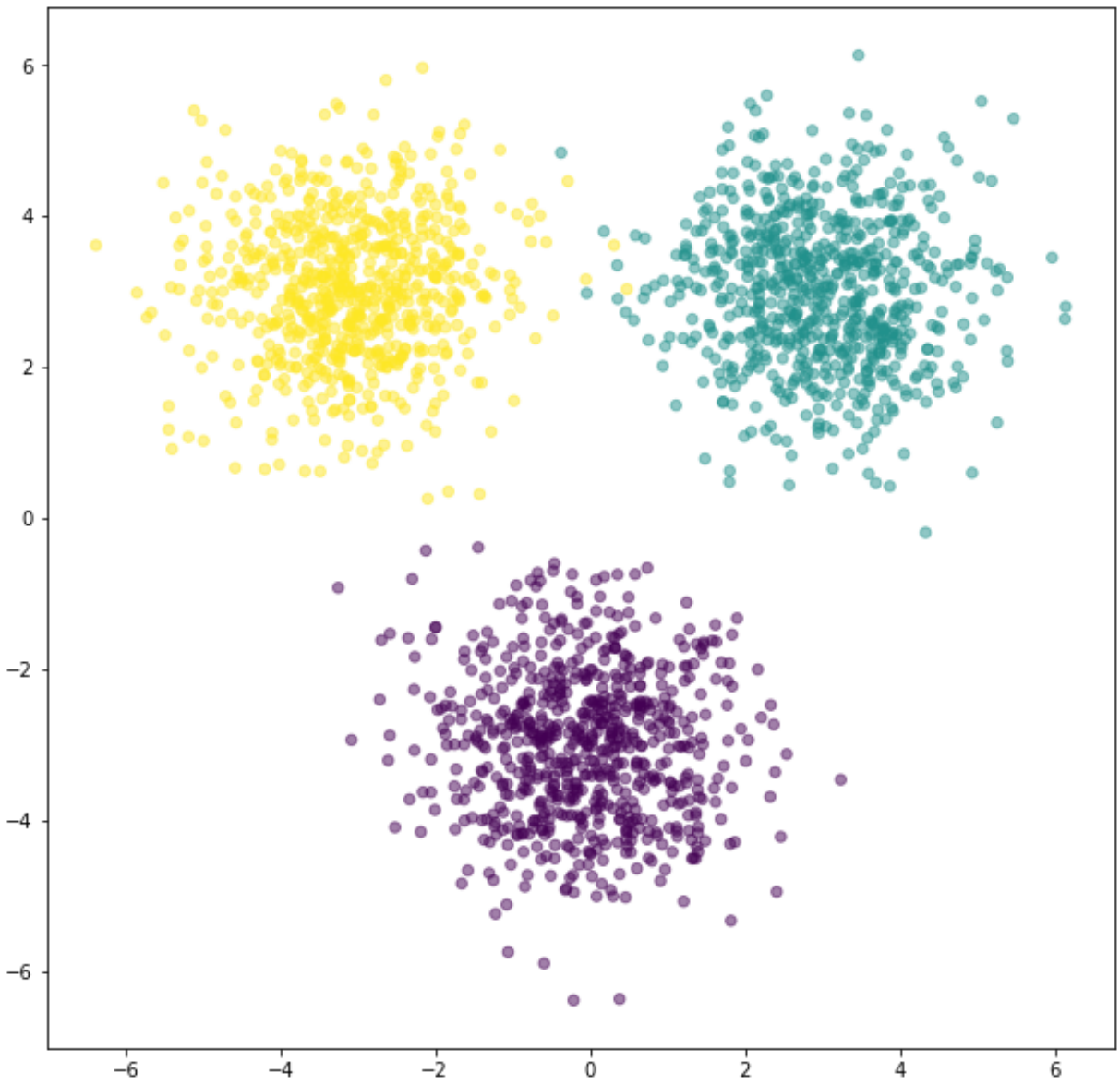
```
Layer architecture:
Layer 0 :  (2, 2)
Layer 1 :  (2, 2)
Bias:
Layer 0 :  (2,)
Layer 1 :  (2,)
NeuralNetwork: 2-2-2
Error in epoch( 0 ): 0.4994570862179424
Error in epoch( 10000 ): 0.33445747889020844
Error in epoch( 20000 ): 0.3341030908555261
Error in epoch( 30000 ): 0.33394937758600113
Error in epoch( 40000 ): 0.33385939535956605
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x1277367f0>]
```



Multiple classes

In [10]:
```
# 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()
```

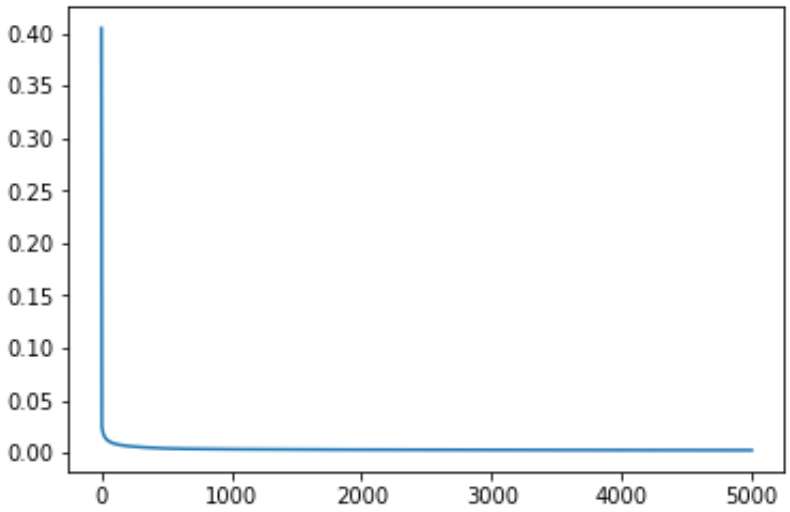


In [12]:
```
#TODO: Test the class with the multiple classes data
network = NeuralNetwork([2,3,3],0.05)
errors = network.train(feature_set, one_hot_labels, 5000)
plt.plot(errors)
```

```
Layer architecture:
Layer 0 :  (2, 3)
Layer 1 :  (3, 3)
Bias:
Layer 0 :  (3,)
Layer 1 :  (3,)
Error in epoch( 0 ): 0.40472832279818005
Error in epoch( 100 ): 0.008778699123642963
Error in epoch( 200 ): 0.006535315225232546
Error in epoch( 300 ): 0.005472657873496668
```

```
Error in epoch( 400 ): 0.004736241555992148
Error in epoch( 500 ): 0.004219074292545964
Error in epoch( 600 ): 0.003930966181480627
Error in epoch( 700 ): 0.003750354576137169
Error in epoch( 800 ): 0.0036191224621429695
Error in epoch( 900 ): 0.003515921223239426
Error in epoch( 1000 ): 0.0034310468960179912
Error in epoch( 1100 ): 0.003359056762595427
Error in epoch( 1200 ): 0.003296558213473803
Error in epoch( 1300 ): 0.003241322307832789
Error in epoch( 1400 ): 0.003191826246081118
Error in epoch( 1500 ): 0.003146990820434514
Error in epoch( 1600 ): 0.003106024718249806
Error in epoch( 1700 ): 0.0030683300657507106
Error in epoch( 1800 ): 0.0030334433791033132
Error in epoch( 1900 ): 0.003000997353652271
Error in epoch( 2000 ): 0.0029706953033294663
Error in epoch( 2100 ): 0.00294229356327766
Error in epoch( 2200 ): 0.0029155890782884336
Error in epoch( 2300 ): 0.002890410465106877
Error in epoch( 2400 ): 0.0028666114552681468
Error in epoch( 2500 ): 0.0028440660001853185
Error in epoch( 2600 ): 0.002822664556284247
Error in epoch( 2700 ): 0.002802311220898441
Error in epoch( 2800 ): 0.0027829214907812605
Error in epoch( 2900 ): 0.0027644204830459674
Error in epoch( 3000 ): 0.002746741504579352
Error in epoch( 3100 ): 0.002729824887788383
Error in epoch( 3200 ): 0.0027136170326692278
Error in epoch( 3300 ): 0.002698069610751399
Error in epoch( 3400 ): 0.002683138897540587
Error in epoch( 3500 ): 0.002668785208052602
Error in epoch( 3600 ): 0.0026549724158380027
Error in epoch( 3700 ): 0.002641667540184285
Error in epoch( 3800 ): 0.0026288403893919843
Error in epoch( 3900 ): 0.0026164632504609804
Error in epoch( 4000 ): 0.002604510617413323
Error in epoch( 4100 ): 0.0025929589519769537
Error in epoch( 4200 ): 0.0025817864715782617
Error in epoch( 4300 ): 0.002570972960628587
Error in epoch( 4400 ): 0.0025604996020056484
Error in epoch( 4500 ): 0.002550348826470097
Error in epoch( 4600 ): 0.0025405041785419255
Error in epoch( 4700 ): 0.0025309501980860702
Error in epoch( 4800 ): 0.0025216723174839
Error in epoch( 4900 ): 0.002512656774726754
```

Out[12]: [<matplotlib.lines.Line2D at 0x11106da20>]



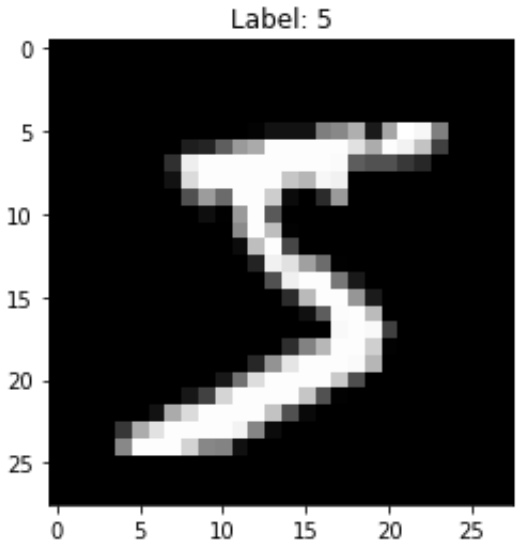
On the mnist data set

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 [13]:
```
# 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[13]: <matplotlib.image.AxesImage at 0x12b404908>

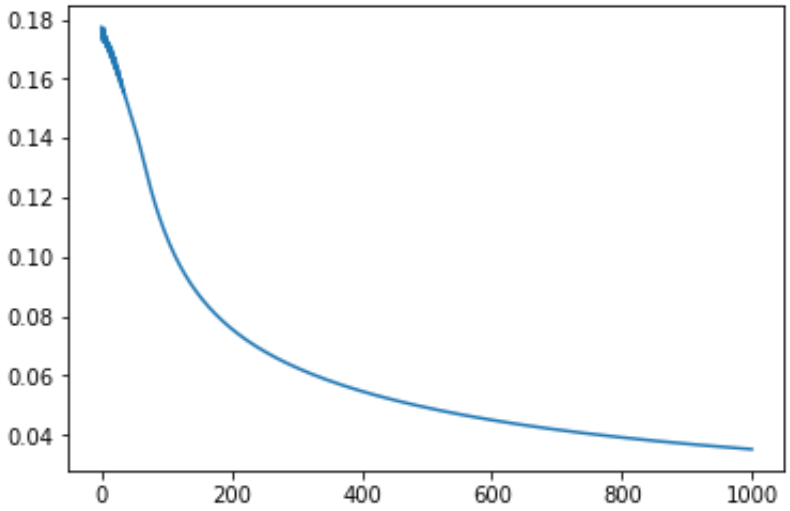


In [14]:
```
# Standardize the data

# 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 [15]:
```
# TODO: Test the class with the mnist data. Test the training of the n
etwork with the test_images data, and
# record the accuracy of the classification.
network = NeuralNetwork([784,64,32,10], 0.0001)
errors = network.train(train_images, train_labels, 1000,100)
# print(error)
# print("weights",network.weights)
plt.plot(errors)
```

```
Layer architecture:
Layer 0 :  (784, 64)
Layer 1 :  (64, 32)
Layer 2 :  (32, 10)
Bias:
Layer 0 :  (64,)
Layer 1 :  (32,)
Layer 2 :  (10,)
Error in epoch( 0 ): 0.17730999701827327
Error in epoch( 100 ): 0.10684674641706027
Error in epoch( 200 ): 0.07574056425577304
Error in epoch( 300 ): 0.06261616216925256
Error in epoch( 400 ): 0.05468607316964369
Error in epoch( 500 ): 0.04914531520610385
Error in epoch( 600 ): 0.04497854069241818
Error in epoch( 700 ): 0.04172167643978114
Error in epoch( 800 ): 0.0391061877692127
Error in epoch( 900 ): 0.03694916009069103
```

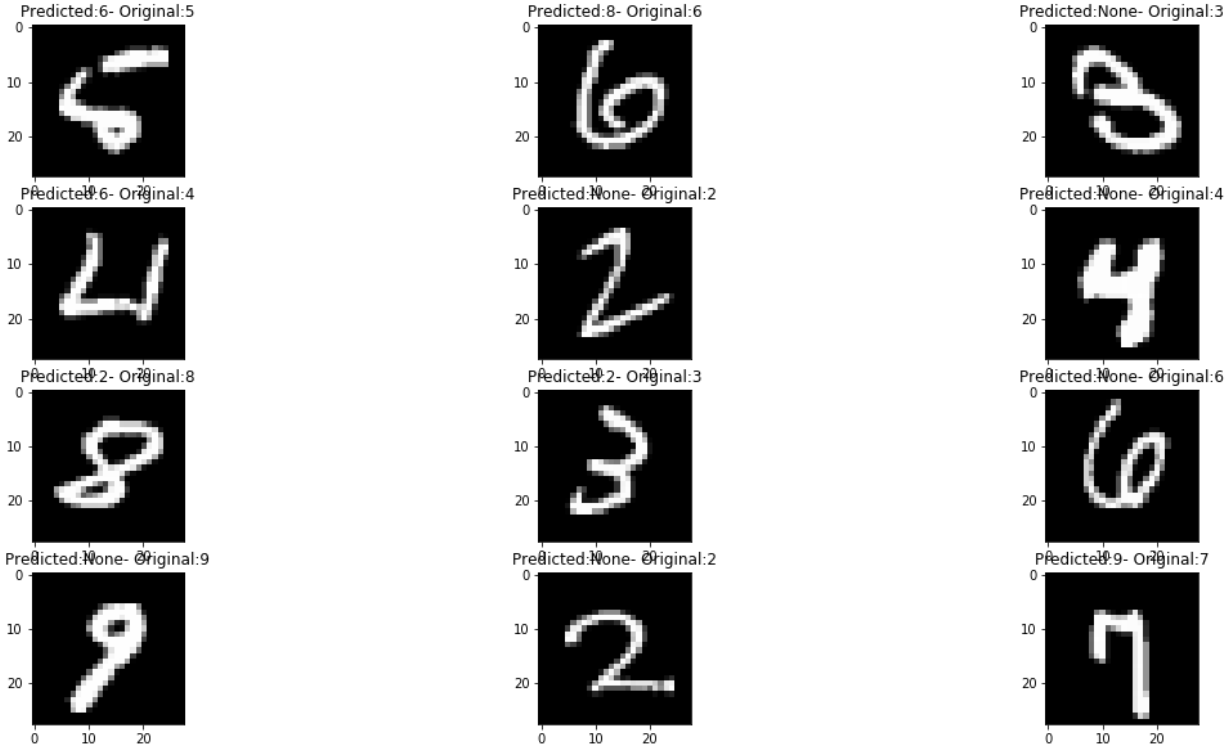
Out[15]: [<matplotlib.lines.Line2D at 0x127de12e8>]



In [16]:
```
def getLabel(label):
    for i in range(len(label)):
        if label[i] == 1:
            return i
```

In [17]:
```
predictedLabels = network.predict(test_images)
misslabeled = []

predictedLabels[predictedLabels > 0.5] = 1
predictedLabels[predictedLabels < 0.5] = 0
for i in range(len(test_labels)):
#     print(test_labels[i])
#     print(predictedLabels[i])
    if np.array_equal(test_labels[i], predictedLabels[i])== False:
        misslabeled.append([test_images[i], test_labels[i], predictedL
abels[i]])

print("Misslabeled: ", len(misslabeled))


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

for label, plot in zip(misslabeled, plots):
    plot.set_title("Predicted:" + str(getLabel(label[2])) + "- Origina
l:" + str(getLabel(label[1])))
    plot.imshow(label[0].reshape(28,28), cmap="gray")
```

```
Misslabeled:  1286
```



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?** It would need more training to understand the type of writing of some people. There are some numbers that even people might not understand. This network had 1000 epochs of training but it may be necesary to add more epochs.
2. **What could you try doing to improve the classification accuracy?** Adding other variables or processing the images in another way