Neural Network Assignment

February 11, 2019

0.0.1 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!

0.0.2 Import the needed Packages

```
In [3]: 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
```

```
Using TensorFlow backend.
```

0.0.3 Define the class

In [80]:
```
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 = []
        self.bs = []

        for i in range(len(architecture['nodes'])):
            self.bs.append(np.random.randn(architecture['nodes'][i]))
            if i == 0:
                self.weights.append(np.random.randn(architecture['inputs'], architectu
            else:
                self.weights.append(np.random.randn(architecture['nodes'][i-1], archi

        self.weights.append(np.random.randn(architecture['nodes'][len(architecture['no
        self.bs.append(np.random.randn(architecture['outputs']))
        self.outputs = architecture['outputs']

    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)
```

```
def predict(self, inputs):
    # TODO: Define the predict function
    self.digits = [inputs]

    for i in range(len(self.weights)):
        if i == len(self.weights) - 1 and self.outputs > 1:
            self.digits.append(self.softmax(np.dot(self.digits[i], self.weights[i]
        else:
            self.digits.append(self.sigmoid(np.dot(self.digits[i], self.weights[i]

    return self.digits[len(self.digits)-1]

def train(self, inputs, labels, epochs = 1000, displayUpdate = 100):
    # TODO: Define the training step for the network. It should include the forwa
    # steps, the updating of the weights, and it should print the error every 'di
    # It must return the errors so that they can be displayed with matplotlib
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('Epoch')
    ax.set_ylabel('Error')

    errors = []
    for i in range(epochs):
        prediction = self.predict(inputs)
        error = labels - prediction
        errors.append(np.mean(np.abs(error)))

        if i%displayUpdate == 0:
            print("Error:", np.mean(np.abs(error)))

        deltas = []
        j = len(self.digits) - 1

        while j > 0:
            if j != len(self.digits) - 1:
                error = np.dot(delta, self.weights[j].T)
            delta = error * self.sigmoid_deriv(self.digits[j])
            deltas.append(delta)

            j-=1

        deltas = deltas[::-1]

        bs_deltas = []
        for d in deltas:
            bs_deltas.append(np.sum(d))

        for j in range(len(deltas)):
            self.weights[j] += np.dot(self.digits[j].T, deltas[j]) * self.alpha
```

```
                    for j in range(len(bs_deltas)):
                        self.bs[j] += bs_deltas[j] * self.alpha

                ax.plot(errors)
```

0.0.4 Test datasets

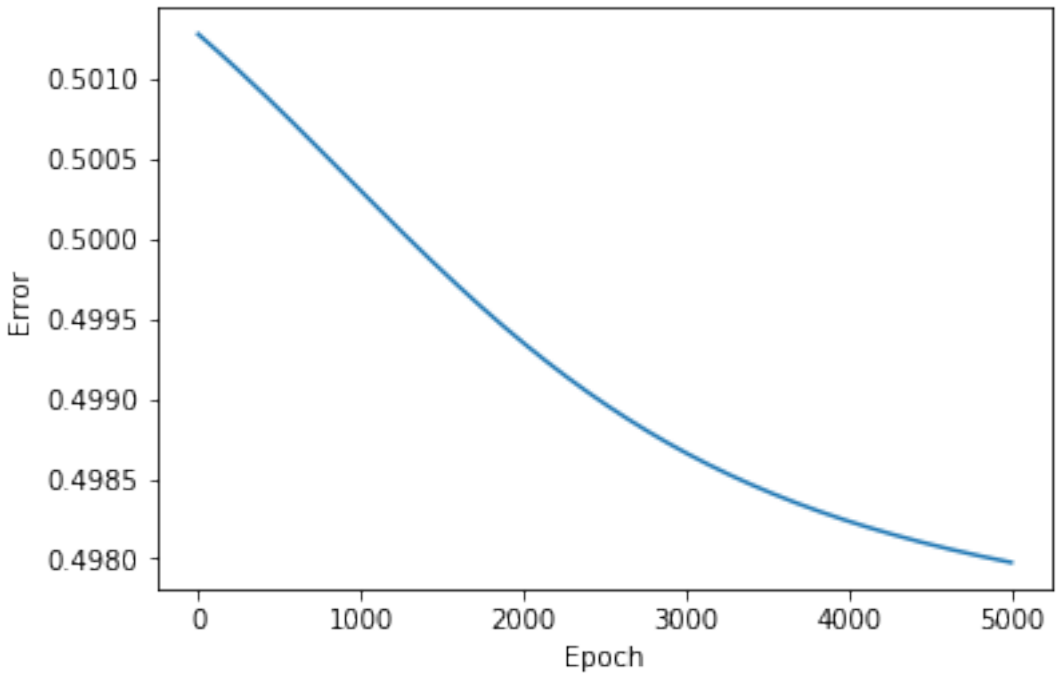
XOR

In [81]:
```
# 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 [82]:
```
#TODO: Test the class with the XOR data
arch = {
    'inputs': XOR_inputs.shape[1],
    'nodes': [4],
    'outputs': 1
}

nn = NeuralNetwork(arch, 1)

nn.train(XOR_inputs, XOR_labels, 5000, 100)
```

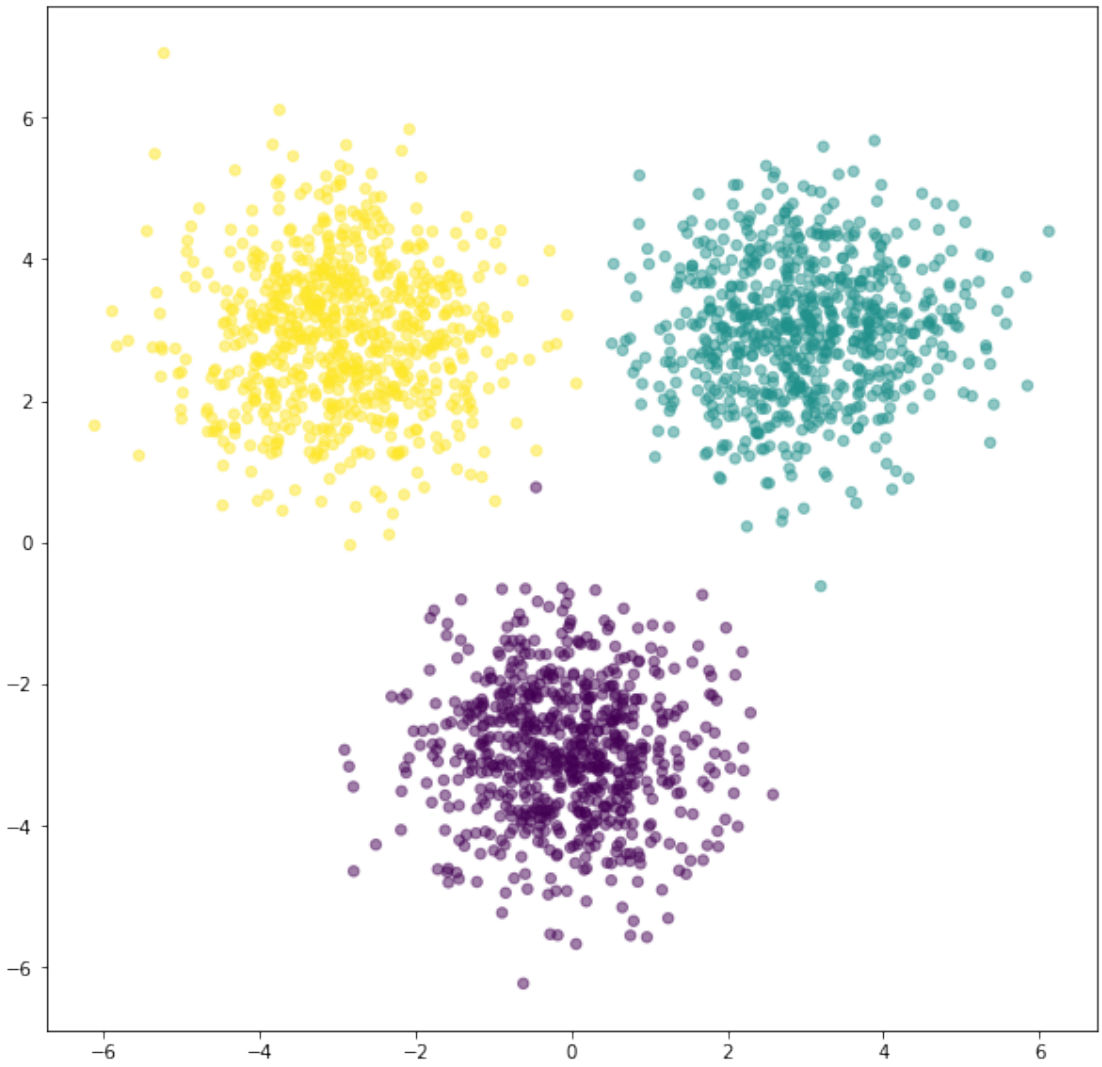
```
Error: 0.5012811266177135
Error: 0.5011920426693214
Error: 0.5011000146109018
Error: 0.501005357817945
Error: 0.5009084138125153
Error: 0.5008095459109614
Error: 0.5007091343117572
Error: 0.5006075707596347
Error: 0.5005052529421856
Error: 0.50040257878743
Error: 0.5002999408345117
Error: 0.5001977208444851
Error: 0.500096284804508
Error: 0.4999959784577853
Error: 0.499897123464903
Error: 0.49980001427175236
Error: 0.4997049157271717
```

```
Error: 0.49961206146182174
Error: 0.49952165301050183
Error: 0.49943385963459663
Error: 0.49934881878066983
Error: 0.4992666370959449
Error: 0.49918739191167394
Error: 0.49911113310089783
Error: 0.4990378852172998
Error: 0.4989676498259493
Error: 0.4989004079438556
Error: 0.49883612251749887
Error: 0.4987747408750184
Error: 0.4987161971017773
Error: 0.498660414298952
Error: 0.49860730669513964
Error: 0.4985567815903978
Error: 0.49850874112040494
Error: 0.4984630838354638
Error: 0.49841970609483677
Error: 0.49837850328146077
Error: 0.49833937084553054
Error: 0.498302205187886
Error: 0.4982669043957433
Error: 0.4982333688441918
Error: 0.498201501677197
Error: 0.4981712091817126
Error: 0.4981424010680383
Error: 0.4981149906688449
Error: 0.4980888950684218
Error: 0.4980640351727331
Error: 0.4980403357298663
Error: 0.4980177253094428
Error: 0.49799613624858213
```



Multiple classes

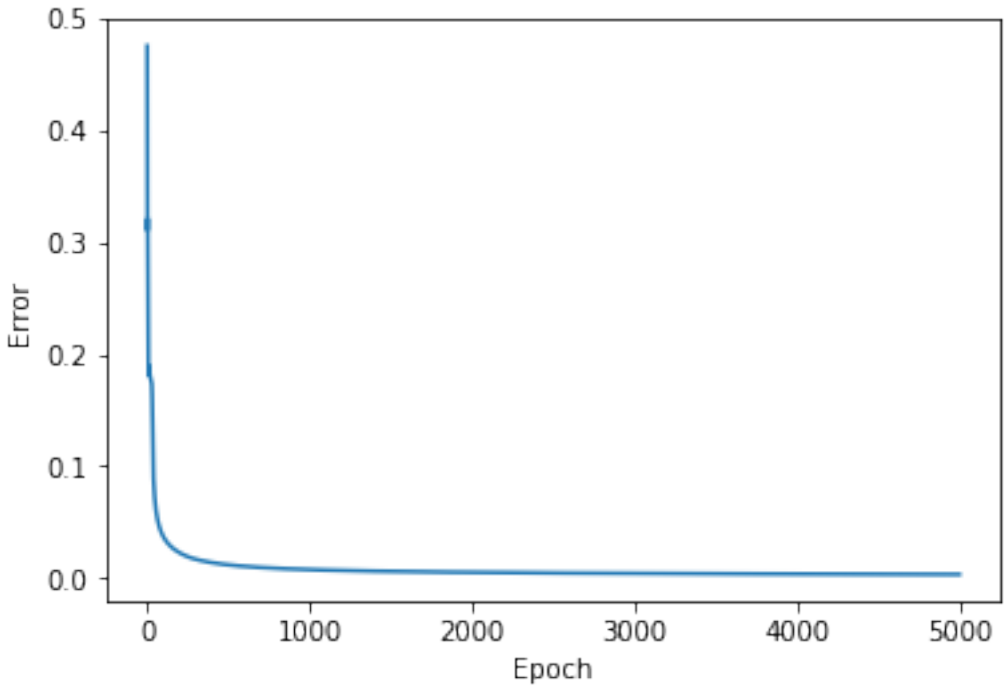
```
In [83]: # Creates the data points for each class
         class_1 = np.random.randn(700, 2) + np.array([0, -3])
         class_2 = np.random.randn(700, 2) + np.array([3, 3])
         class_3 = np.random.randn(700, 2) + np.array([-3, 3])

         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 [84]:
```
#TODO: Test the class with the multiple classes data
arch = {
    'inputs': feature_set.shape[1],
    'nodes': [3],
    'outputs': 3
}

nn = NeuralNetwork(arch, 0.01)

nn.train(feature_set, one_hot_labels, 5000, 100)
```

```
Error: 0.47549317999251584
Error: 0.03681351120203427
Error: 0.022589960287088123
```

```
Error: 0.016789057129925834
Error: 0.013803733105924941
Error: 0.011851277174585327
Error: 0.010494161929326454
Error: 0.00949306412712513
Error: 0.008721541279537438
Error: 0.008106966335250114
Error: 0.007604608898340339
Error: 0.0071854017579216015
Error: 0.006829628520499966
Error: 0.006523433063301123
Error: 0.006256781902762244
Error: 0.006022220918530529
Error: 0.005814087892753817
Error: 0.005627997423245
Error: 0.005460494381123594
Error: 0.005308814909306444
Error: 0.005170717929117233
Error: 0.005044364011800633
Error: 0.004928226770278374
Error: 0.004821027022656666
Error: 0.004721683187112074
Error: 0.00462927343329715
Error: 0.004543006472863375
Error: 0.0044621987808141205
Error: 0.004386256658993399
Error: 0.0043146619822667966
Error: 0.004246960769998274
Error: 0.0041827539411765985
Error: 0.004121689767921857
Error: 0.004063457657077018
Error: 0.004007782975305603
Error: 0.003954422697867422
Error: 0.0039031617107473555
Error: 0.0038538096340449374
Error: 0.00380619806428805 5
Error: 0.0037601781565990756
Error: 0.003715618485850357 7
Error: 0.0036724031401580252
Error: 0.0036304300110800683
Error: 0.0035896092533339212
Error: 0.0035498618932333255
Error: 0.003511118569779091
Error: 0.00347331839577009
Error: 0.003436407928722081
Error: 0.0034003402430381182
Error: 0.003365074095968890 4
```



On the mnist data set

```
In [85]: # 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")
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
```

```
---------------------------------------------------------------------------

SSLError                                  Traceback (most recent call last)

/usr/lib/python3.6/urllib/request.py in do_open(self, http_class, req, **http_conn_args
   1317                 h.request(req.get_method(), req.selector, req.data, headers,
-> 1318                           encode_chunked=req.has_header('Transfer-encoding'))
   1319             except OSError as err: # timeout error


/usr/lib/python3.6/http/client.py in request(self, method, url, body, headers, encode_
   1238         """Send a complete request to the server."""
```

```
-> 1239         self._send_request(method, url, body, headers, encode_chunked)
   1240

    /usr/lib/python3.6/http/client.py in _send_request(self, method, url, body, headers, en
   1284             body = _encode(body, 'body')
-> 1285         self.endheaders(body, encode_chunked=encode_chunked)
   1286

    /usr/lib/python3.6/http/client.py in endheaders(self, message_body, encode_chunked)
   1233             raise CannotSendHeader()
-> 1234         self._send_output(message_body, encode_chunked=encode_chunked)
   1235

    /usr/lib/python3.6/http/client.py in _send_output(self, message_body, encode_chunked)
   1025         del self._buffer[:]
-> 1026         self.send(msg)
   1027

    /usr/lib/python3.6/http/client.py in send(self, data)
    963             if self.auto_open:
--> 964                 self.connect()
    965             else:

    /usr/lib/python3.6/http/client.py in connect(self)
   1399             self.sock = self._context.wrap_socket(self.sock,
-> 1400                                                   server_hostname=server_hostname)
   1401             if not self._context.check_hostname and self._check_hostname:

    /usr/lib/python3.6/ssl.py in wrap_socket(self, sock, server_side, do_handshake_on_conne
    406                         server_hostname=server_hostname,
--> 407                         _context=self, _session=session)
    408

    /usr/lib/python3.6/ssl.py in __init__(self, sock, keyfile, certfile, server_side, cert_
    816                         raise ValueError("do_handshake_on_connect should not be spe
--> 817                     self.do_handshake()
    818

    /usr/lib/python3.6/ssl.py in do_handshake(self, block)
   1076                 self.settimeout(None)
```

```
-> 1077             self._sslobj.do_handshake()
   1078         finally:


    /usr/lib/python3.6/ssl.py in do_handshake(self)
    688         """Start the SSL/TLS handshake."""
--> 689         self._sslobj.do_handshake()
    690         if self.context.check_hostname:


    SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:847)


During handling of the above exception, another exception occurred:


    URLError                                  Traceback (most recent call last)

    ~/Documents/tec/vision/DLV-Course-Material/Notebooks/venv/lib/python3.6/site-packages/
    221             try:
--> 222                 urlretrieve(origin, fpath, dl_progress)
    223             except HTTPError as e:


    /usr/lib/python3.6/urllib/request.py in urlretrieve(url, filename, reporthook, data)
    247
--> 248     with contextlib.closing(urlopen(url, data)) as fp:
    249         headers = fp.info()


    /usr/lib/python3.6/urllib/request.py in urlopen(url, data, timeout, cafile, capath, ca
    222         opener = _opener
--> 223     return opener.open(url, data, timeout)
    224


    /usr/lib/python3.6/urllib/request.py in open(self, fullurl, data, timeout)
    525
--> 526         response = self._open(req, data)
    527


    /usr/lib/python3.6/urllib/request.py in _open(self, req, data)
    543         result = self._call_chain(self.handle_open, protocol, protocol +
--> 544                                   '_open', req)
    545         if result:
```

```
    /usr/lib/python3.6/urllib/request.py in _call_chain(self, chain, kind, meth_name, *args
    503             func = getattr(handler, meth_name)
--> 504             result = func(*args)
    505             if result is not None:


    /usr/lib/python3.6/urllib/request.py in https_open(self, req)
   1360             return self.do_open(http.client.HTTPSConnection, req,
-> 1361                 context=self._context, check_hostname=self._check_hostname)
   1362


    /usr/lib/python3.6/urllib/request.py in do_open(self, http_class, req, **http_conn_args
   1319             except OSError as err: # timeout error
-> 1320                 raise URLError(err)
   1321             r = h.getresponse()


    URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_s


During handling of the above exception, another exception occurred:


    Exception                                 Traceback (most recent call last)

    <ipython-input-85-adcd44275059> in <module>
      1 # Load the train and test data from the mnist data set
----> 2 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
      3
      4 # Plot a sample data point
      5 plt.title("Label: " + str(train_labels[0]))


    ~/Documents/tec/vision/DLV-Course-Material/Notebooks/venv/lib/python3.6/site-packages/k
     21     path = get_file(path,
     22                     origin='https://s3.amazonaws.com/img-datasets/mnist.npz',
---> 23                     file_hash='8a61469f7ea1b51cbae51d4f78837e45')
     24     f = np.load(path)
     25     x_train, y_train = f['x_train'], f['y_train']


    ~/Documents/tec/vision/DLV-Course-Material/Notebooks/venv/lib/python3.6/site-packages/k
    224                 raise Exception(error_msg.format(origin, e.code, e.msg))
    225             except URLError as e:
--> 226                 raise Exception(error_msg.format(origin, e.errno, e.reason))
    227         except (Exception, KeyboardInterrupt):
    228             if os.path.exists(fpath):
```

```
Exception: URL fetch failure on https://s3.amazonaws.com/img-datasets/mnist.npz: None -
```

```
In [8]: # 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 [9]: # TODO: Test the class with the mnist data. Test the training of the network with the
        # record the accuracy of the classification.
        arch = {
            'inputs': train_images.shape[1],
            'nodes': [64, 32],
            'outputs': 10
        }

        nn = NeuralNetwork(arch, 0.001)

        nn.train(train_images[0:1000], train_labels[0:1000], 1000, 100)
```

```
In [ ]: f, plots = plt.subplots((12+3-1)//3, 3, figsize=(15,15))
        plots = [plot for sublist in plots for plot in sublist]

        res = nn.predict(train_images[0:1000])
        res[res > 0.5] = 1
        res[res < 0.5] = 0

        for image, im_data, plot, r in zip(test_images[0:12], train_images[0:12], plots, res):
            plot.set_title(r)
            plot.imshow(image.reshape((28,28)), cmap="gray")
```

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?** Not enough training or different method is needed
2. **What could you try doing to improve the classification accuracy?** Excecute more iterations