Autoencoders

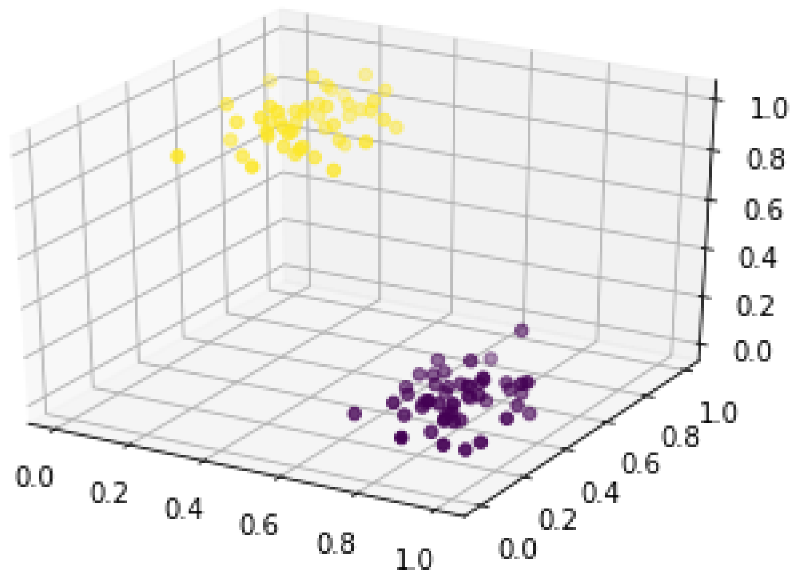
December 21, 2017

1 Simple Autoencoder for Principle Component Analysis

1.1 Create some data and scale it

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D

data = make_blobs(n_samples=100, n_features=3,centers=2,random_state=101)
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data[0])
# data[0]
data_x = scaled_data[:,0]
data_y = scaled_data[:,1]
data_z = scaled_data[:,2]

fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.scatter(data_x,data_y,data_z,c=data[1])
```

Out[7]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x22c5803abe0>
```



1.2 The Linear Autoencoder

```
In [8]: import tensorflow as tf
        from tensorflow.contrib.layers import fully_connected
        num_inputs = 3  # 3 dimensional input
        num_hidden = 2  # 2 dimensional representation
        num_outputs = num_inputs # Must be true for an autoencoder!

        learning_rate = 0.01
```

1.2.1 Placeholder

Notice there is no real label here, just X.

```
In [9]: X = tf.placeholder(tf.float32, shape=[None, num_inputs])
```

1.2.2 Layers

Using the fully_connected layers API, we do not provide an activation function!

```
In [10]: hidden = fully_connected(X, num_hidden, activation_fn=None)
         outputs = fully_connected(hidden, num_outputs, activation_fn=None)
```

1.2.3 Loss function

```
In [11]: loss = tf.reduce_mean(tf.square(outputs - X))  # MSE
```

1.2.4 Optimizer

```
In [12]: optimizer = tf.train.AdamOptimizer(learning_rate)
         train   = optimizer.minimize(loss)
```

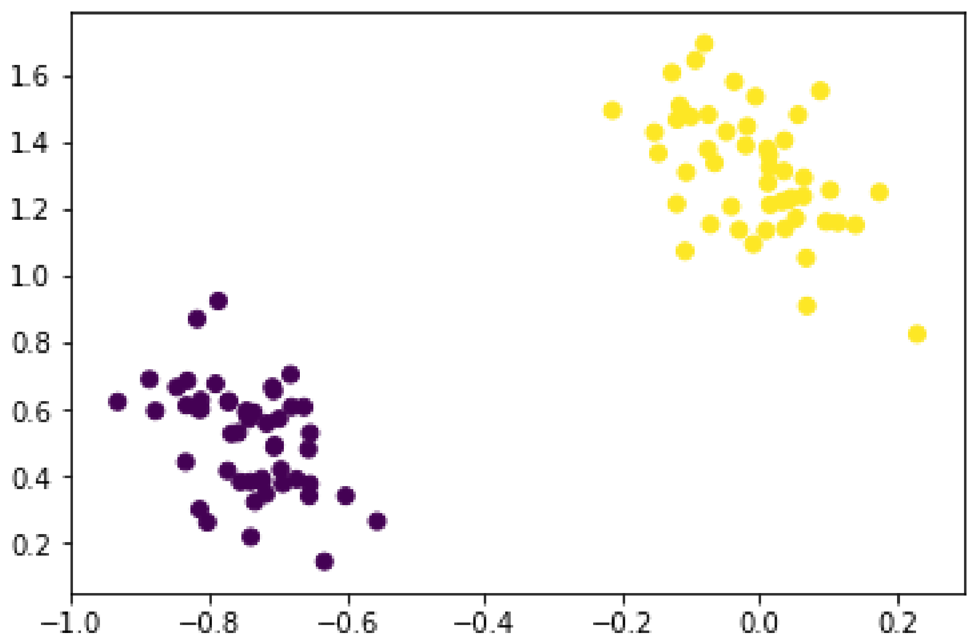
1.2.5 Init

```
In [13]: init = tf.global_variables_initializer()
```

1.2.6 Running the Session

```
In [14]: num_steps = 1000

         with tf.Session() as sess:
             sess.run(init)

             for iteration in range(num_steps):
                 sess.run(train,feed_dict={X: scaled_data})


             # Now ask for the hidden layer output (the 2 dimensional output)
             output_2d = hidden.eval(feed_dict={X: scaled_data})

         plt.scatter(output_2d[:,0],output_2d[:,1],c=data[1])
```

```
Out[14]: <matplotlib.collections.PathCollection at 0x22c64266208>
```



2 Linear Autoencoder Exercise

2.1 The Data

1. Import numpy, matplotlib, and pandas.
2. Use pandas to read in the csv file called anonymized_data.csv . It contains 500 rows and 30 columns of anonymized data along with 1 last column with a classification label, where the columns have been renamed to 4 letter codes.
3. Take a look at the head.
4. Take a look at the info().

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        df = pd.read_csv('anonymized_data.csv')
        df.head()
```

```
Out[1]:       EJWY      VALM       EGXO      HTGR       SKRF      NNSZ      NYLC  \
        0 -2.032145  1.019576  -9.658715 -6.210495   3.156823  7.457850 -5.313357
        1  8.306217  6.649376  -0.960333 -4.094799   8.738965 -3.458797  7.016800
        2  6.570842  6.985462  -1.842621 -1.569599  10.039339 -3.623026  8.957619
        3 -1.139972  0.579422  -9.526530 -5.744928   4.834355  5.907235 -4.804137
        4 -1.738104  0.234729 -11.558768 -7.181332   4.189626  7.765274 -2.189083

              GWID      TVUT      CJHI  ...       LKKS       UOBF      VBHE  \
        0  8.508296  3.959194 -5.246654  ...  -2.209663 -10.340123 -7.697555
        1  6.692765  0.898264  9.337643  ...   0.851793  -9.678324 -6.071795
        2  7.577283  1.541255  7.161509  ...   1.376085  -8.971164 -5.302191
        3  6.798810  5.403670 -7.642857  ...   0.270571  -8.640988 -8.105419
        4  7.239925  3.135602 -6.211390  ...  -0.013973  -9.437110 -6.475267

              FRWU       NDYZ      QSBO      JDUB      TEVK      EZTM  Label
        0 -5.932752  10.872688  0.081321  1.276316  5.281225 -0.516447    0.0
        1  1.428194  -8.082792 -0.557089 -7.817282 -8.686722 -6.953100    1.0
        2  2.898965  -8.746597 -0.520888 -7.350999 -8.925501 -7.051179    1.0
        3 -5.079015   9.351282  0.641759  1.898083  3.904671  1.453499    0.0
        4 -5.708377   9.623080  1.802899  1.903705  4.188442  1.522362    0.0

        [5 rows x 31 columns]
```

```
In [2]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 31 columns):
EJWY     500 non-null float64
VALM     500 non-null float64
EGXO     500 non-null float64
```

```
HTGR     500 non-null float64
SKRF     500 non-null float64
NNSZ     500 non-null float64
NYLC     500 non-null float64
GWID     500 non-null float64
TVUT     500 non-null float64
CJHI     500 non-null float64
NVFW     500 non-null float64
VLBG     500 non-null float64
IDIX     500 non-null float64
UVHN     500 non-null float64
IWOT     500 non-null float64
LEMB     500 non-null float64
QMYY     500 non-null float64
XDGR     500 non-null float64
ODZS     500 non-null float64
LNJS     500 non-null float64
WDRT     500 non-null float64
LKKS     500 non-null float64
UOBF     500 non-null float64
VBHE     500 non-null float64
FRWU     500 non-null float64
NDYZ     500 non-null float64
QSBO     500 non-null float64
JDUB     500 non-null float64
TEVK     500 non-null float64
EZTM     500 non-null float64
Label    500 non-null float64
dtypes: float64(31)
memory usage: 121.2 KB
```

2.2 Scale the data

5. Use scikit learn to scale the data with a MinMaxScaler. Remember not to scale the Label column, just the data. Save this scaled data as a new variable called scaled_data.

```
In [3]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        scaled_data = scaler.fit_transform(df.drop('Label',axis=1))
```

2.3 The Linear Autoencoder

6. Import tensorflow and import fully_connected layers from tensorflow.contrib.layers.

7. Fill out the number of inputs to fit the dimensions of the data set and set the hidden number of units to be 2. Also set the number of outputs to match the number of inputs. Also choose a learning_rate value.

8. Create a placeholder fot the data called X.

9. Create the hidden layer and the output layers using the fully_connected function. Remember that to perform PCA there is no activation function.

10. Create a Mean Squared Error loss function.

11. Create an AdamOptimizer designed to minimize the previous loss function.

12. Create an instance of a global variable intializer.

```
In [4]: import tensorflow as tf
        from tensorflow.contrib.layers import fully_connected

        num_inputs = 30  # 3 dimensional input
        num_hidden = 2  # 2 dimensional representation
        num_outputs = num_inputs # Must be true for an autoencoder!

        learning_rate = 0.01

        X = tf.placeholder(tf.float32, shape=[None, num_inputs])

        hidden = fully_connected(X, num_hidden, activation_fn=None)
        outputs = fully_connected(hidden, num_outputs, activation_fn=None)

        loss = tf.reduce_mean(tf.square(outputs - X))  # MSE

        optimizer = tf.train.AdamOptimizer(learning_rate)
        train  = optimizer.minimize( loss)

        init = tf.global_variables_initializer()
```

2.4 Running the Session

13. Now create a Tensorflow session that runs the optimizer for at least 1000 steps. (You can also use epochs if you prefer, where 1 epoch is defined by one single run through the entire dataset.

14. Now create a session that runs the scaled data through the hidden layer. (You could have also done this in the last step after all the training steps.

15. Confirm that your output is now 2 dimensional along the previous axis of 30 features.

```
In [5]: num_steps = 1000

        with tf.Session() as sess:
            sess.run(init)

            for iteration in range(num_steps):
                sess.run(train,feed_dict={X: scaled_data})

        with tf.Session() as sess:
            sess.run(init)

            # Now ask for the hidden layer output (the 2 dimensional output)
            output_2d = hidden.eval(feed_dict={X: scaled_data})
```

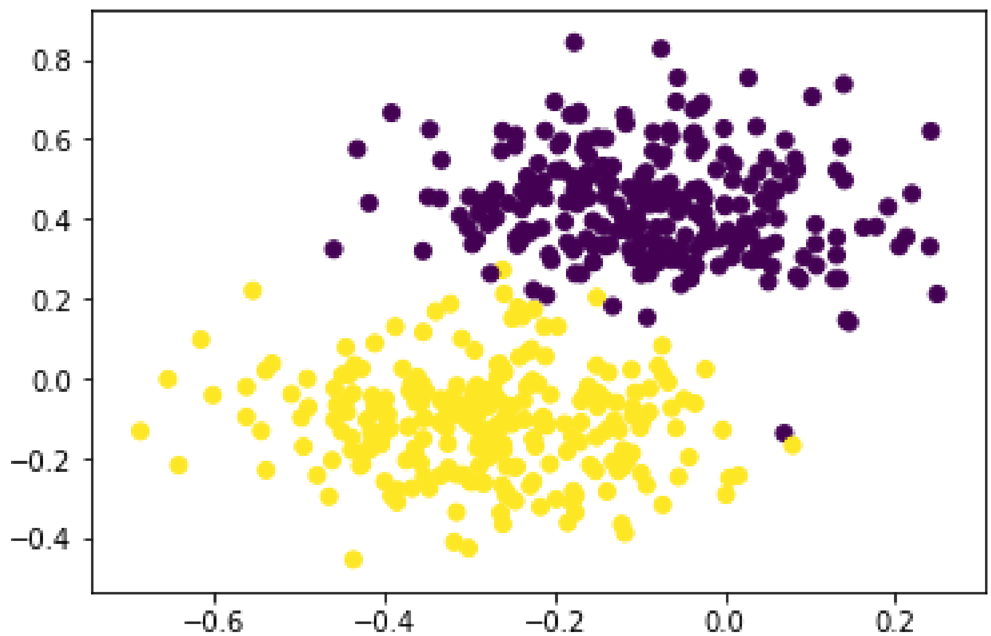
```
output_2d.shape
```

Out[5]: (500, 2)

16. Now plot out the reduced dimensional representation of the data. Do you still have clear separation of classes even with the reduction in dimensions? Hint: You definitely should, the classes should still be clearly seperable, even when reduced to 2 dimensions.

In [6]:
```
plt.scatter(output_2d[:,0],output_2d[:,1],c=df['Label'])
```

Out[6]: <matplotlib.collections.PathCollection at 0x22ad8789588>



3 Stacked Autoencoder

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets(
    "../03-Convolutional-Neural-Networks/MNIST_data/",one_hot=True)
tf.reset_default_graph()
```

```
Extracting ../03-Convolutional-Neural-Networks/MNIST_data/train-images-idx3-ubyte.gz
Extracting ../03-Convolutional-Neural-Networks/MNIST_data/train-labels-idx1-ubyte.gz
Extracting ../03-Convolutional-Neural-Networks/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting ../03-Convolutional-Neural-Networks/MNIST_data/t10k-labels-idx1-ubyte.gz
```

3.1 Parameters

```
In [17]: num_inputs = 784 # 28*28
         neurons_hid1 = 392
         neurons_hid2 = 196
         neurons_hid3 = neurons_hid1 # Decoder Begins
         num_outputs = num_inputs

         learning_rate = 0.01
```

3.2 Activation function

```
In [18]: actf = tf.nn.relu
```

3.3 Placeholder

```
In [19]: X = tf.placeholder(tf.float32, shape=[None, num_inputs])
```

3.4 Weights

Initializer capable of adapting its scale to the shape of weights tensors.

With `distribution="normal"`, samples are drawn from a truncated normal distribution centered on zero, with `stddev = sqrt(scale / n)` where n is: - number of input units in the weight tensor, if mode = "fan_in" - number of output units, if mode = "fan_out" - average of the numbers of input and output units, if mode = "fan_avg"

With `distribution="uniform"`, samples are drawn from a uniform distribution within [-limit, limit], with `limit = sqrt(3 * scale / n)`.

```
In [20]: initializer = tf.variance_scaling_initializer()

         w1 = tf.Variable(initializer([num_inputs, neurons_hid1]), dtype=tf.float32)
         w2 = tf.Variable(initializer([neurons_hid1, neurons_hid2]), dtype=tf.float32)
         w3 = tf.Variable(initializer([neurons_hid2, neurons_hid3]), dtype=tf.float32)
         w4 = tf.Variable(initializer([neurons_hid3, num_outputs]), dtype=tf.float32)
```

3.5 Biases

```
In [21]: b1 = tf.Variable(tf.zeros(neurons_hid1))
         b2 = tf.Variable(tf.zeros(neurons_hid2))
         b3 = tf.Variable(tf.zeros(neurons_hid3))
         b4 = tf.Variable(tf.zeros(num_outputs))
```

3.6 Activation Function and Layers

In [22]:
```
act_func = tf.nn.relu

hid_layer1 = act_func(tf.matmul(X, w1) + b1)
hid_layer2 = act_func(tf.matmul(hid_layer1, w2) + b2)
hid_layer3 = act_func(tf.matmul(hid_layer2, w3) + b3)
output_layer = tf.matmul(hid_layer3, w4) + b4
```

3.7 Loss Function

In [23]:
```
loss = tf.reduce_mean(tf.square(output_layer - X))
```

3.8 Optimizer

In [24]:
```
#tf.train.RMSPropOptimizer
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

3.9 Initialize Variables

In [25]:
```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
num_epochs = 5
batch_size = 150

with tf.Session() as sess:
    sess.run(init)

    # Epoch == Entire Training Set
    for epoch in range(num_epochs):

        num_batches = mnist.train.num_examples // batch_size

        # 150 batch size
        for iteration in range(num_batches):

            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(train, feed_dict={X: X_batch})

        training_loss = loss.eval(feed_dict={X: X_batch})

        print("Epoch {} Complete. Training Loss: {}".format(epoch,training_loss))

    saver.save(sess, "./stacked_autoencoder.ckpt")
```

```
Epoch 0 Complete. Training Loss: 0.03219767287373543
Epoch 1 Complete. Training Loss: 0.0320863351225853
Epoch 2 Complete. Training Loss: 0.028678443282842636
```

```
Epoch 3 Complete. Training Loss: 0.02804933674633503
Epoch 4 Complete. Training Loss: 0.027236798778176308
```

3.10 Test Autoencoder outpout on test data

In [26]:
```python
num_test_images = 10

with tf.Session() as sess:

    saver.restore(sess,"./stacked_autoencoder.ckpt")

    results = output_layer.eval(feed_dict={X:mnist.test.images[:num_test_images]})
```

```
INFO:tensorflow:Restoring parameters from ./stacked_autoencoder.ckpt
```

In [27]:
```python
# Compare original images with their reconstructions
f, a = plt.subplots(2, 10, figsize=(20, 4))
for i in range(num_test_images):
    a[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    a[1][i].imshow(np.reshape(results[i], (28, 28)))
```