Stevens Institute of Technology

Department of Electrical Engineering

EE628: Data Acquisition/Proc II

Homework Assignment 7

Submitted by:
Hadia Hameed
CWID: 10440803

Submitted to:
Prof. Rensheng Wang

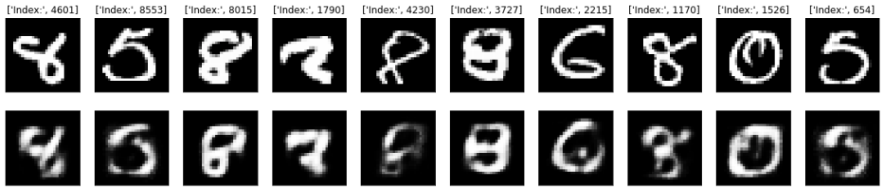
April 15, 2019

1 Problem 1:

Calculate the difference between the decoded images and the original images. Sort the difference values and find the top-10 images with the most autoencoder errors. Show the top-10 images and comment on your findings (see if the images are really different from the most of the rest)

```
diff = decoded_imgs - x_test
mag = np.zeros(diff.shape[0])
for i in range(diff.shape[0]):
    mag[i] = np.linalg.norm(diff[i])

new_mag = np.argsort(-mag, axis=0)
new_mag = new_mag.astype(int)
```

```
# use Matplotlib (don't ask)
import matplotlib.pyplot as plt

n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original

    ax = plt.subplot(2, n, i + 1)
    ii = new_mag[i]

    plt.imshow(x_test[ii].reshape(28, 28))
    plt.title(['Index:', ii])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[ii].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



The original images were not that clear in the first place which is why their reconstructions were inaccurate as well.

2 Problem 2:

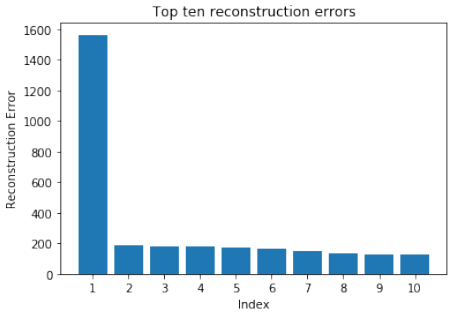
2.1 Part (a):

Use only the first 28 columns $V1 - V28$ to test vanilla auto-encoder with 1-layer for encoder and decoder (hidden layer nodes = 2). Plot the histogram plot of reconstruction errors.

```
import numpy as np
from sklearn.model_selection import train_test_split
import pandas as pd

x_train = pd.read_csv(‘‘data1.csv’’)
x_train = x_train.astype(’float32’)
x_train = x_train.dropna()
x_train.drop([’Class’], inplace=True, axis=1)


x_train, x_test = train_test_split(x_train, test_size=0.1)
x_train = x_train.values
x_test = x_test.values

from keras.layers import Input, Dense
from keras.models import Model

# this is the size of our encoded representations
encoding_dim = 12

# this is our input placeholder
input_img = Input(shape=(28,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation=’relu’)(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(28, activation=’sigmoid’)(encoded)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
```



```
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adadelta', loss='mse')
autoencoder.fit(x_train, x_train,
                epochs=30,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
diff = decoded_imgs - x_test

#calculating magnitude of reconstruction error for each test data
mag = np.zeros(10)
for i in range(10):
    mag[i] = np.linalg.norm(diff[i])
```



(a) Top ten reconstruction errors for reconstruction of the test data set.

	reconstruction_error
count	56962.000000
mean	1.003171
std	7.461340
min	0.097851
25%	0.333289
50%	0.553362
75%	0.874585
max	1569.892334

(b) Summary of reconstruction errors

Figure 1: Reconstruction error in case of auto-encode with 1 layer for encoder.

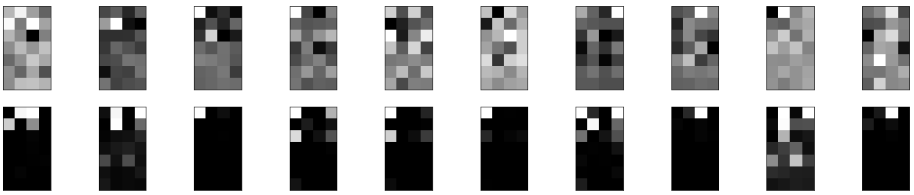
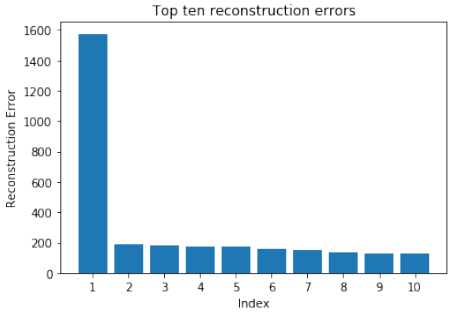


Figure 2: The top row shows first ten original test images and the bottom row shows their reconstructions. This was achieved using an autoencoder with a single layer encoder

2.2 Part (b):

Use only the first 28 columns $V1-V28$ to test Deep auto-encoder with AT LEAST 2-layers for encoder and decoder (final hidden layer nodes = 2). Plot the histogram plot of reconstruction errors. Compare this with step 1 results.

```
encoding_dim = 2
input_img = Input(shape=(x_train.shape[1],))
encoded = Dense(56, activation='relu')(input_img)
encoded = Dense(28, activation='relu')(encoded)
encoded = Dense(14, activation='relu')(encoded)
encoded = Dense(encoding_dim, activation='relu')(encoded)


decoded = Dense(14, activation='relu')(encoded)
decoded = Dense(28, activation='relu')(decoded)
decoded = Dense(56, activation='relu')(decoded)
decoded = Dense(28, activation='sigmoid')(decoded)

autoencoder = Model(input_img, decoded)

encoder = Model(input_img, encoded)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer1 = autoencoder.layers[-4]
decoder_layer2 = autoencoder.layers[-3]
decoder_layer3 = autoencoder.layers[-2]
decoder_layer4 = autoencoder.layers[-1]

decoder = Model(encoded_input,  ( decoder_layer4(decoder_layer3(
          decoder_layer2(decoder_layer1(encoded_input)) ))  ) )
```

(a) Top ten reconstruction errors for reconstruction of the test data set.

	reconstruction_error
count	56962.000000
mean	0.937164
std	7.439058
min	0.053067
25%	0.307290
50%	0.503173
75%	0.782126
max	1567.421265

(b) Summary of reconstruction errors

Figure 3: Reconstruction error in case of auto-encode with 4 layers for encoder. We can see there is a decrease of 7% in mean reconstruction error.

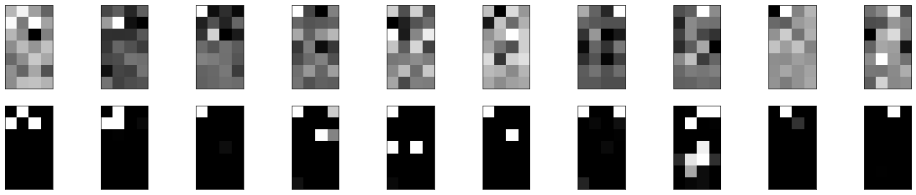


Figure 4: The top row shows first ten original test images and the bottom row shows their reconstructions. This was achieved using an auto-encoder with four layers of encoder

2.3 Part (c), (d):

For both step 1 & 2, use hidden layer nodes = 2. Plot the scatter plots of the hidden layer vector samples. Use the column 29 "class" to put colors of the scatter plots in Step3. "0"-class with green and "1" class with red.

```
encoded_imgs = encoder.predict(x_data)
i = np.where(y_data == 1)
x_1 = encoded_imgs[i,0]
y_1 = encoded_imgs[i,1]
```



```
i = np.where(y_data == 0)
x_0 = encoded_imgs[i,0]
y_0 = encoded_imgs[i,1]

ax1 = plt.subplot(121)
plt.scatter(x_0, y_0,color='blue', alpha=0.1)
plt.title('Class 0')
ax3 = plt.subplot(122, sharex=ax1, sharey=ax1)
plt.scatter(x_1, y_1,color='red', alpha=0.1)
plt.title('Class 1')
plt.show()
```

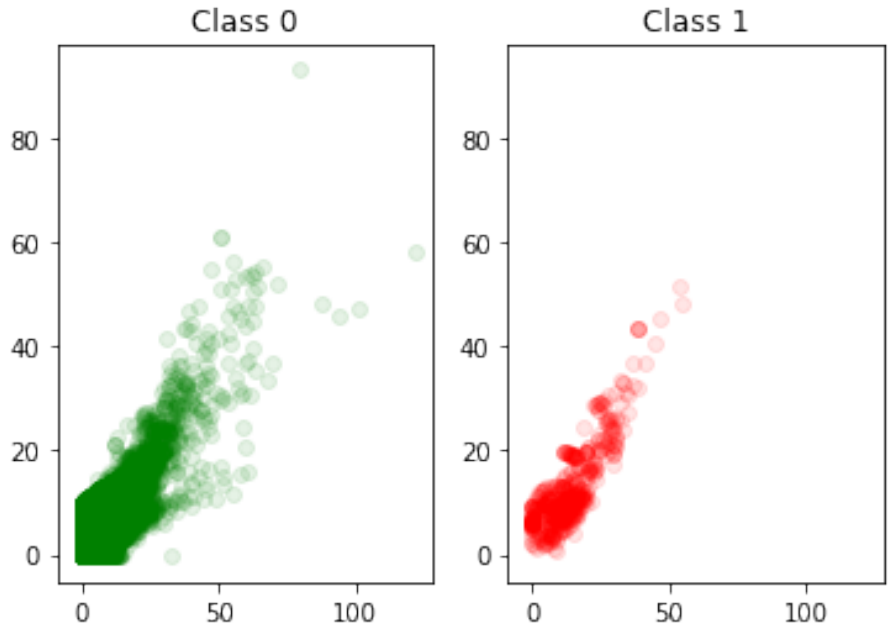


Figure 5: Two-dimensional representation for test images belonging to each class using 1-layer encoder. For class 1 the two-dimensional vector has smaller magnitude as compared to class 0 encoded vectors

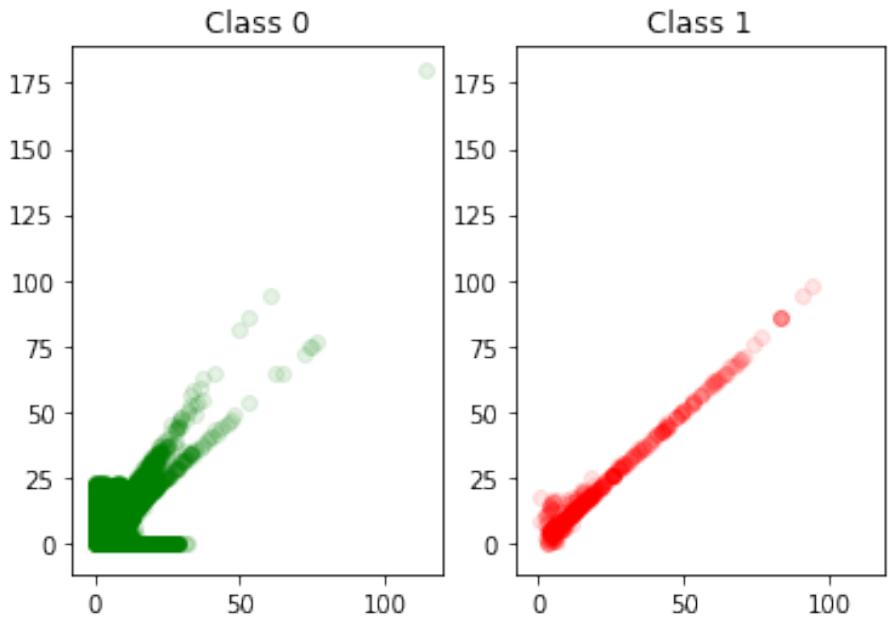


Figure 6: Two-dimensional representation for test images belonging to each class using 4-layer encoder.

