Assignment is below at the bottom

Video 13.1 https://www.youtube.com/watch?v=kIGHE7Cfe1s

Video 13.2 https://www.youtube.com/watch?v=Rm9bJcDd1KU

Video 13.3 https://youtu.be/6HjZk-3LsjE

In [ ]:
```python
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np

(xtrain, ytrain), (xtest, ytest) = mnist.load_data()

xtrain = xtrain.astype('float32') / 255.
xtest = xtest.astype('float32') / 255.
xtrain = xtrain.reshape((len(xtrain), np.prod(xtrain.shape[1:])))
xtest = xtest.reshape((len(xtest), np.prod(xtest.shape[1:])))
xtrain.shape, xtest.shape
```

In [ ]:
```python
# this is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
x = input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)


# "decoded" is the lossy reconstruction of the input
x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

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
dcd1 = autoencoder.layers[-1]
dcd2 = autoencoder.layers[-2]
dcd3 = autoencoder.layers[-3]

# create the decoder model
decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))
```

In [ ]:
```python
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

In [ ]:
```python
autoencoder.summary()
```

In [ ]:
```python
autoencoder.fit(xtrain, xtrain,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest),
                #callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
               )
```

In [ ]:
```python
encoded_imgs = encoder.predict(xtest)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt

n = 20  # how many digits we will display
plt.figure(figsize=(40, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(xtest[i].reshape(28, 28))
    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[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

In [ ]:
```python
encoded_imgs.shape
```

In [ ]:
```python
noise = np.random.normal(20,4, (4,4))
noise_preds = decoder.predict(noise)
```

In [ ]:
```python
plt.imshow(noise_preds[1].reshape(28,28))
```

In [ ]:
```python
np.max(encoded_imgs)
```

In [ ]:
```python
encoded_imgs
```

In [ ]:
```python
%matplotlib inline
```

In [ ]:
```python
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,0], s=1, c=ytest, cmap='rainbow')
# plt.show()
```

In [ ]:
```python
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,3], s=1, c=ytest, cmap='rainbow')
# plt.show()
```

In [ ]:
```python
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,2], s=1, c=ytest, cmap='rainbow')
# plt.show()
```

In [ ]:
```python
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(encoded_imgs[:,1], encoded_imgs[:,2], encoded_imgs[:,3], c=ytest, cmap='rainbow', s=1)
```

Assignment

1. change the `encoding_dim` through various values (`range(2,18,2)` and save the loss you can get. Plot the 8 pairs of dimensions vs loss on a scatter plot

In [1]:
```python
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np

(xtrain, ytrain), (xtest, ytest) = mnist.load_data()

xtrain = xtrain.astype('float32') / 255.
xtest = xtest.astype('float32') / 255.
xtrain = xtrain.reshape((len(xtrain), np.prod(xtrain.shape[1:])))
xtest = xtest.reshape((len(xtest), np.prod(xtest.shape[1:])))
xtrain.shape, xtest.shape
```

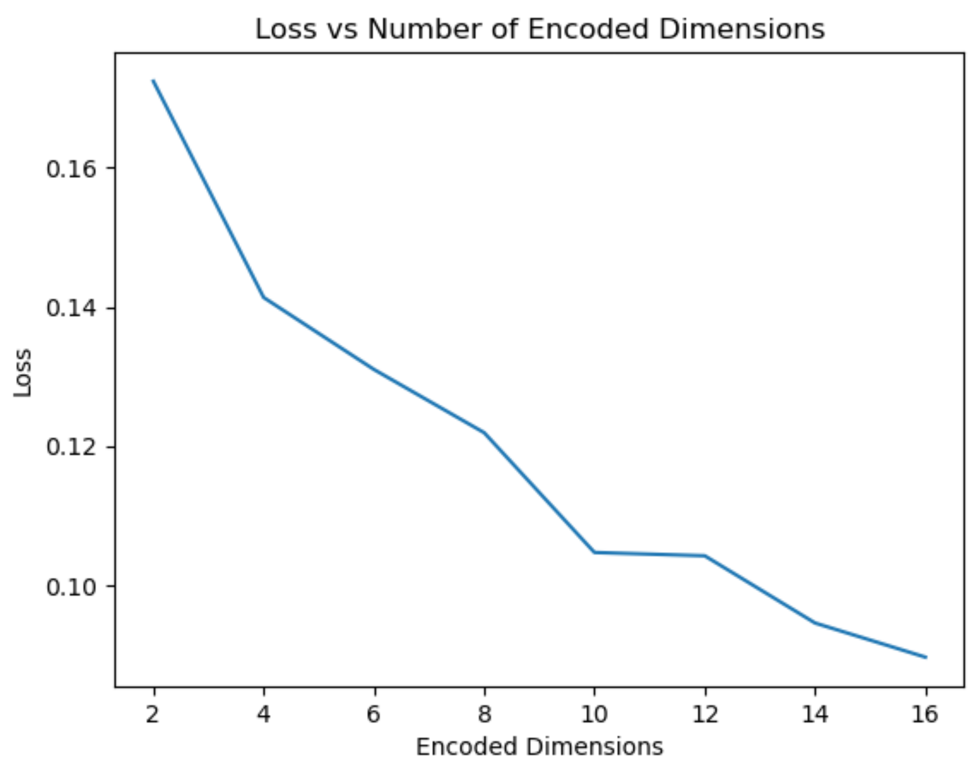
Out[1]:
```
((60000, 784), (10000, 784))
```

In [2]:
```python
%%capture
losses = []
dimensions = range(2,18,2)

for encoding_dim in dimensions:
    x = input_img = Input(shape=(784,))
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    encoded = Dense(encoding_dim, activation='relu')(x)

    x = Dense(128, activation='relu')(encoded)
    x = Dense(256, activation='relu')(x)
    decoded = Dense(784, activation='sigmoid')(x)

    autoencoder = Model(input_img, decoded)

    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

    autoencoder.fit(xtrain, xtrain,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest)
                #callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
               )

    loss = autoencoder.evaluate(xtest, xtest, verbose=0)
    losses.append(loss)
```

In [3]:
```python
import matplotlib.pyplot as  plt
%matplotlib inline
```

In [4]:
```python
plt.figure()
plt.xlabel('Encoded Dimensions')
plt.ylabel('Loss')
plt.plot(dimensions, losses)
plt.title("Loss vs Number of Encoded Dimensions")
```

Out[4]:
```
Text(0.5, 1.0, 'Loss vs Number of Encoded Dimensions')
```



2. ***After*** training an autoencoder with `encoding_dim=8`, apply noise (like the previous assignment) to *only* the input of the trained autoencoder (not the output). The output images should be without noise.

Print a few noisy images along with the output images to show they don't have noise.

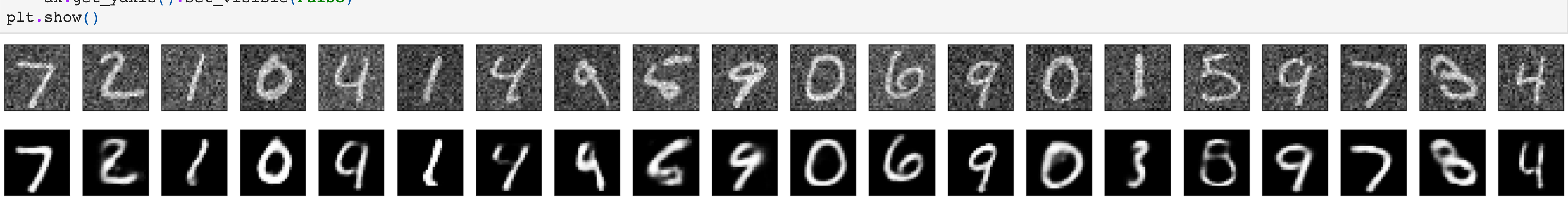
In [6]:
```python
%%capture

encoding_dim = 8

x = input_img = Input(shape=(784,))
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)

x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(xtrain, xtrain,
        epochs=100,
        batch_size=256,
        shuffle=True,
        validation_data=(xtest, xtest)
        #callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
        )
```

In [7]:
```python
noise_train = np.random.normal(scale=0.2, size=[60000,784])
noise_test = np.random.normal(scale=0.2, size=[10000,784])

xtrain_noisy = xtrain + noise_train
xtest_noisy = xtest + noise_test
```

In [8]:
```python
noise_predictions = autoencoder.predict(xtest_noisy)
```

```
313/313 [==============================] - 0s 663us/step
```

In [9]:
```python
import matplotlib.pyplot as plt

n = 20  # how many digits we will display
plt.figure(figsize=(40, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(xtest_noisy[i].reshape(28, 28))
    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(noise_predictions[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



In [ ]: