Assignment 12 - Coder

In [1]:
```
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
import matplotlib.pyplot as plt
%matplotlib inline

(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))
```

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 [2]:
```
def auto_mod(encoding_dim):
    # 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)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

    #history
    history = autoencoder.fit(xtrain, xtrain,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest))

    return history.history['loss'][-1]
```

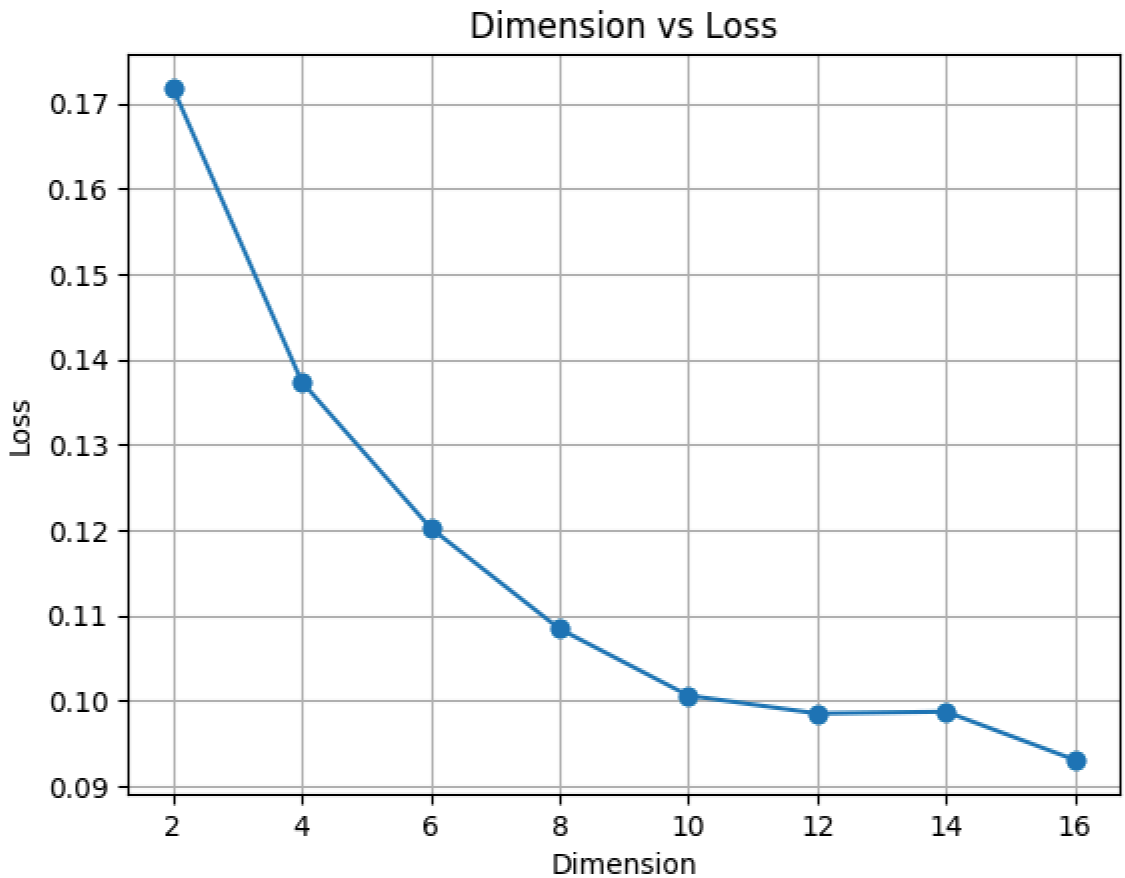
In [3]:
```
dimensions = range(2, 18, 2)
losses = []
for encoding_dim in dimensions:

    loss = auto_mod(encoding_dim)
    losses.append(loss)
```

```
Epoch 23/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0984 - val_loss: 0.0986
Epoch 24/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0979 - val_loss: 0.0980
Epoch 25/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0976 - val_loss: 0.0979
Epoch 26/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0974 - val_loss: 0.0980
Epoch 27/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0969 - val_loss: 0.0973
Epoch 28/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0966 - val_loss: 0.0970
Epoch 29/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0964 - val_loss: 0.0972
Epoch 30/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0961 - val_loss: 0.0967
Epoch 31/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0960 - val_loss: 0.0963
Epoch 32/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0957 - val_loss: 0.0963
Epoch 33/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0954 - val_loss: 0.0961
Epoch 34/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0953 - val_loss: 0.0961
Epoch 35/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0953 - val_loss: 0.0960
Epoch 36/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0950 - val_loss: 0.0959
Epoch 37/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0948 - val_loss: 0.0956
Epoch 38/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0946 - val_loss: 0.0954
Epoch 39/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0942 - val_loss: 0.0954
Epoch 40/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0942 - val_loss: 0.0955
Epoch 41/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0943 - val_loss: 0.0949
Epoch 42/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0939 - val_loss: 0.0951
Epoch 43/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0936 - val_loss: 0.0950
Epoch 44/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0936 - val_loss: 0.0949
Epoch 45/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0936 - val_loss: 0.0949
Epoch 46/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0936 - val_loss: 0.0947
Epoch 47/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0935 - val_loss: 0.0947
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0933 - val_loss: 0.0945
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0930 - val_loss: 0.0945
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0930 - val_loss: 0.0945
```

In [4]:

```
plt.figure()
plt.scatter(dimensions, losses)
plt.plot(dimensions, losses, marker='o')
plt.xlabel('Dimension')
plt.ylabel('Loss')
```

```
plt.title('Dimension vs Loss')
plt.grid(True)
plt.show()
```



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 [5]:
```
encoding_dim = 8

# 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,))
```

```python
# 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))))

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

In [10]:
```python
autoencoder.fit(xtrain, xtrain,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest))
```

```
Epoch 94/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1111 - val_loss: 0.1159
Epoch 95/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1112 - val_loss: 0.1158
Epoch 96/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1112 - val_loss: 0.1158
Epoch 97/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1112 - val_loss: 0.1159
Epoch 98/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1114 - val_loss: 0.1158
Epoch 99/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1112 - val_loss: 0.1160
Epoch 100/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1108 - val_loss: 0.1158
```

Out[10]:
```
<keras.src.callbacks.history.History at 0x192dbcdd750>
```

In [23]:
```
#add noise
np.random.seed(15)
noise = .1
xtest_noise = xtest + (noise*np.random.normal(loc=0.0, scale=1, size=xtest.shape))
xtest_noise = np.clip(xtest_noise, 0., 1.)
```

In [24]:
```
noise_pred = autoencoder.predict(xtest_noise)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [25]:
```
n=10
plt.figure()
for i in range(n):
    #original
    ax = plt.subplot(3, n, i+1)
    plt.imshow(xtest[i].reshape(28, 28))
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    #noisy
    ax = plt.subplot(3, n, i+n+1)
    plt.imshow(xtest_noise[i].reshape(28, 28))
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    #output
    ax = plt.subplot(3, n, i+1+2*n)
    plt.imshow(noise_pred[i].reshape(28, 28))
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```