Module 12 - Autoencoders

Assignment

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

In [24]:

```
from keras.callbacks import TensorBoard
from keras.callbacks import EarlyStopping
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import pandas as pd
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[24]:

```
((60000, 784), (10000, 784))
```

In [25]:

```
import tensorflow as tf
from tensorflow import keras
```

In [30]:
```
loss = {}

for i in range(2, 18, 2):

    encoding_dim = i

    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)

    encoder = Model(input_img, encoded)

    encoded_input = Input(shape=(encoding_dim,))

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

    decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))

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

    autoencoder.fit(xtrain, xtrain,
                    epochs=50,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(xtest, xtest),
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=2

    loss[i] = autoencoder.evaluate(xtrain, xtrain, verbose = 0)
```

```
Epoch 1/50
235/235 [==============================] - 3s 8ms/step - loss: 0.2778
- val_loss: 0.2503
Epoch 2/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2336
- val_loss: 0.2156
Epoch 3/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2104
- val_loss: 0.2052
Epoch 4/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2018
- val_loss: 0.1988
Epoch 5/50
235/235 [==============================] - 2s 7ms/step - loss: 0.1961
- val_loss: 0.1942
Epoch 6/50
235/235 [==============================] - 2s 8ms/step - loss: 0.1924
- val_loss: 0.1911
```

```
Epoch 7/50
```

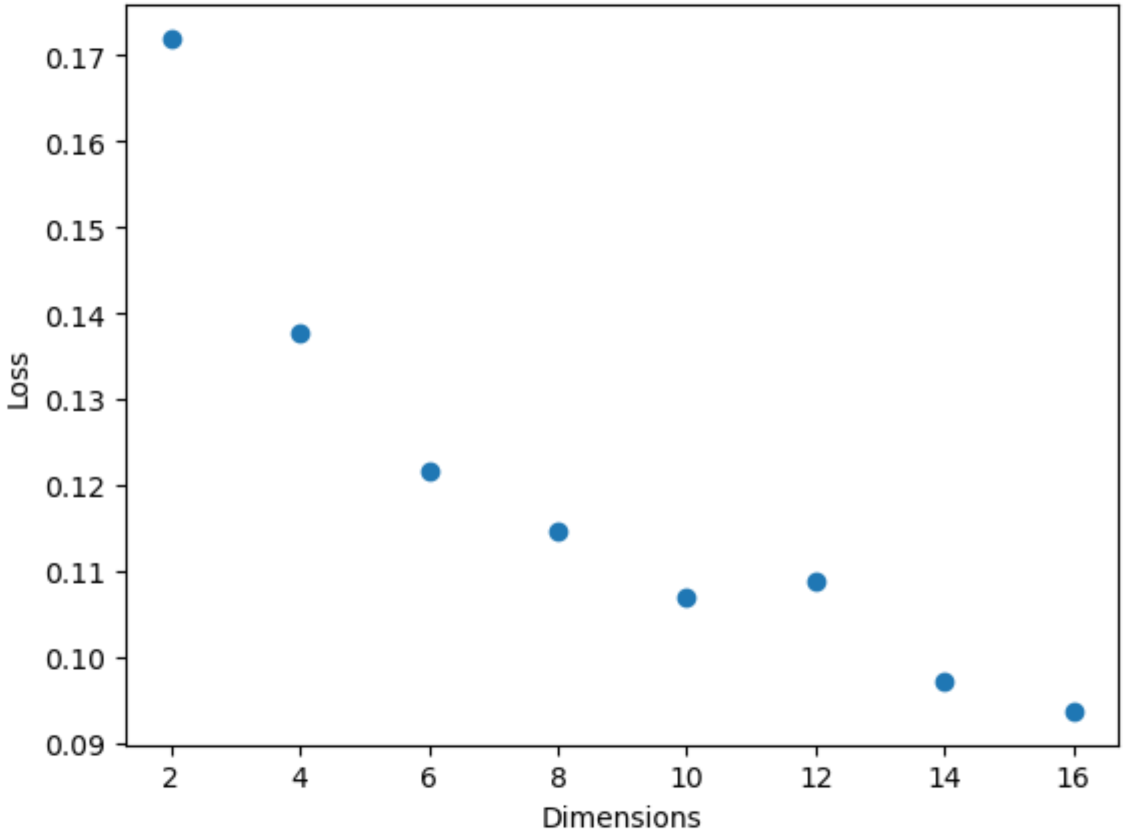
```
In [31]: loss
```

```
Out[31]: {2: 0.17197497189044952,
          4: 0.13771149516105652,
          6: 0.12169531732797623,
          8: 0.1146538108587265,
          10: 0.1068718358874321,
          12: 0.1089300885796547,
          14: 0.09724261611700058,
          16: 0.09369336813688278}
```

```
In [32]: import matplotlib.pyplot as  plt
         %matplotlib inline
```

```
In [33]: plt.scatter(loss.keys(), loss.values())
         plt.xlabel("Dimensions")
         plt.ylabel('Loss')
```

```
Out[33]: Text(0, 0.5, 'Loss')
```



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 [34]:

```
loss = {}

for i in range(8):

    encoding_dim = i

    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)

    encoder = Model(input_img, encoded)

    encoded_input = Input(shape=(encoding_dim,))

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

    decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))

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

    autoencoder.fit(xtrain, xtrain,
                    epochs=50,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(xtest, xtest),
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=2

    loss[i] = autoencoder.evaluate(xtrain, xtrain, verbose = 0)
```

```
Epoch 1/50
235/235 [==============================] - 3s 9ms/step - loss: 0.6530
- val_loss: 0.6155
Epoch 2/50
235/235 [==============================] - 2s 7ms/step - loss: 0.5832
- val_loss: 0.5535
Epoch 3/50
235/235 [==============================] - 1s 6ms/step - loss: 0.5275
- val_loss: 0.5038
Epoch 4/50
235/235 [==============================] - 2s 6ms/step - loss: 0.4827
- val_loss: 0.4638
Epoch 5/50
235/235 [==============================] - 2s 6ms/step - loss: 0.4466
- val_loss: 0.4314
Epoch 6/50
235/235 [==============================] - 2s 6ms/step - loss: 0.4174
- val_loss: 0.4050
```

```
Epoch 7/50
```

No Noise

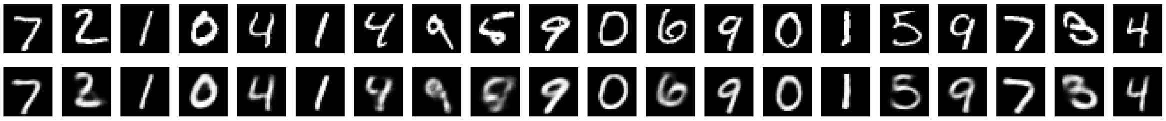
In [35]:

```
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()
```

```
313/313 [==============================] - 0s 773us/step
313/313 [==============================] - 0s 894us/step
```



In [36]:

```
loss
```

Out[36]:

```
{0: 0.26338109374046326,
 1: 0.2629716992378235,
 2: 0.17571580410003662,
 3: 0.15329106152057648,
 4: 0.14185652136802673,
 5: 0.13647115230560303,
 6: 0.12825734913349152,
 7: 0.11857785284519196}
```

In [38]:
```
x_train_noise10 = xtrain + np.random.normal(0, 255*.10, xtrain.shape)
x_test_noise10 = xtest + np.random.normal(0, 255*.10, xtest.shape)

x_train_noise50 = xtrain + np.random.normal(0, 255*.50, xtrain.shape)
x_test_noise50 = xtest + np.random.normal(0, 255*.50, xtest.shape)

x_train_noise1 = xtrain + np.random.normal(0, 255*1, xtrain.shape)
x_test_noise1 = xtest + np.random.normal(0, 255*1, xtest.shape)

x_train_noise2 = xtrain + np.random.normal(0, 255*2, xtrain.shape)
x_test_noise2 = xtest + np.random.normal(0, 255*2, xtest.shape)

x_train_noise4 = xtrain + np.random.normal(0, 255*4, xtrain.shape)
x_test_noise4 = xtest + np.random.normal(0, 255*4, xtest.shape)
```

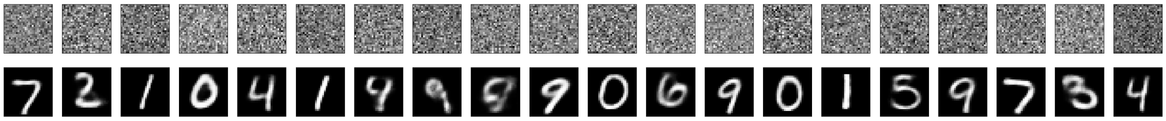
With Noise

In [40]:
```
# Noise .10
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(x_test_noise10[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()
```

```
313/313 [==============================] - 0s 710us/step
313/313 [==============================] - 0s 817us/step
```



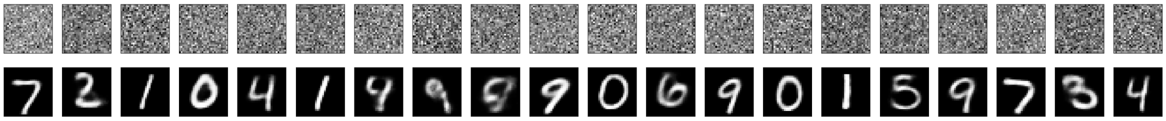
In [41]:

```
# Noise .50
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(x_test_noise50[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()
```

```
313/313 [==============================] - 0s 742us/step
313/313 [==============================] - 0s 880us/step
```



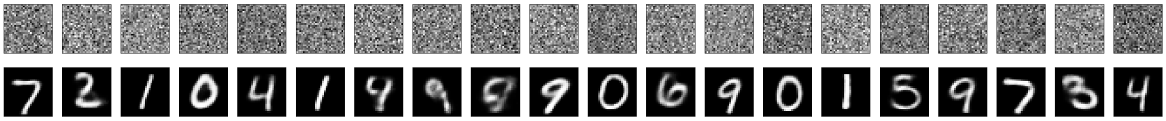
In [42]:

```
# Noise 1
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(x_test_noise1[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()
```

```
313/313 [==============================] - 0s 702us/step
313/313 [==============================] - 0s 826us/step
```



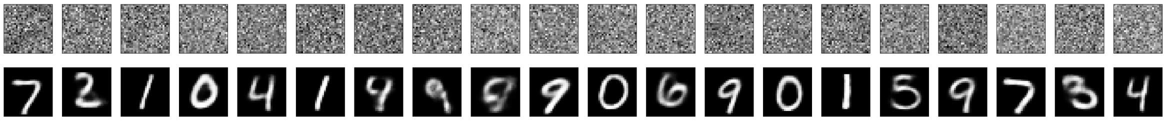
In [43]:

```
# Noise 2
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(x_test_noise2[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()
```

```
313/313 [==============================] - 0s 683us/step
313/313 [==============================] - 0s 780us/step
```



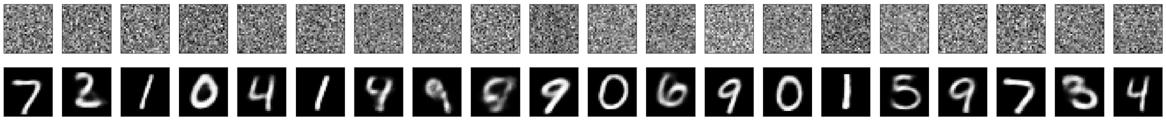
In [44]:

```
# Noise 4
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(x_test_noise4[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()
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 918us/step
```



In [ ]: