Video 13.1 [https://www.youtube.com/watch?v=kIGHE7Cfe1s (https://www.youtube.com/watch?v=kIGHE7Cfe1s)](https://www.youtube.com/watch?v=kIGHE7Cfe1s)

Video 13.2 [https://www.youtube.com/watch?v=Rm9bJcDd1KU (https://www.youtube.com/watch?v=Rm9bJcDd1KU)](https://www.youtube.com/watch?v=Rm9bJcDd1KU)

Video 13.3 [https://youtu.be/6HjZk-3LsjE (https://youtu.be/6HjZk-3LsjE)](https://youtu.be/6HjZk-3LsjE)

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 [1]:
```
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
```

```
Using TensorFlow backend.
```

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

In [5]:
```
import tensorflow as tf
from tensorflow import keras
```

Documentation for writing callbacks found at: [https://keras.io/guides/writing_your_own_callbacks/ (https://keras.io/guides/writing_your_own_callbacks/)](https://keras.io/guides/writing_your_own_callbacks/)
Documentation for Earlystop at: [https://keras.io/api/callbacks/#earlystopping (https://keras.io/api/callbacks/#earlystopping)](https://keras.io/api/callbacks/#earlystopping)

In [6]:
```
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='adadelta', 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)
```

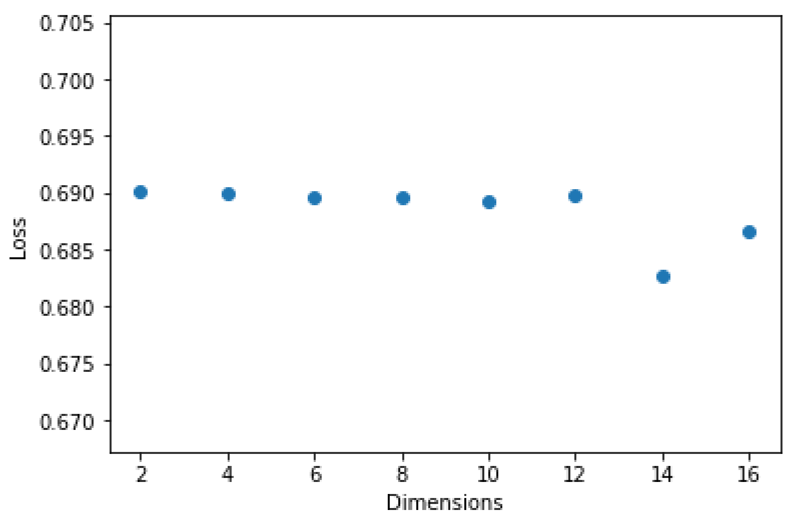
In [20]:
```
loss
```

Out[20]:
```
{2: 0.6900376677513123,
 4: 0.6899483799934387,
 6: 0.6895860433578491,
 8: 0.6895838975906372,
 10: 0.6892221570014954,
 12: 0.6897948384284973,
 14: 0.6827694773674011,
 16: 0.6865726113319397}
```

In [21]:
```
import matplotlib.pyplot as  plt
%matplotlib inline
```

In [27]:
```
plt.scatter(loss.keys(), loss.values())
plt.xlabel("Dimensions")
plt.ylabel('Loss')
```

Out[27]:
```
Text(0, 0.5, 'Loss')
```



2. using the previous assignment's model of detecting images, how does the accuracy change when you run the digit-prediction model on these 'decoded' values?

In [28]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
```

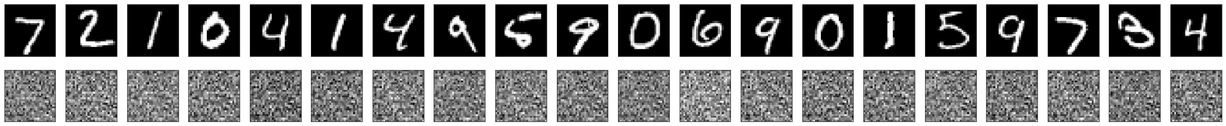
In [29]:
```
(xtrain, ytrain), (xtest, ytest) = mnist.load_data()

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[29]:
```
((60000, 784), (10000, 784))
```

In [31]:
```
batch_size = 128
num_classes = 10
epochs = 20

# convert class vectors to binary class matrices
ytrain = keras.utils.to_categorical(ytrain, num_classes)
ytest = keras.utils.to_categorical(ytest, num_classes)
```

In [33]:
```
import matplotlib.pyplot as plt
```

In [34]:
```
encoded_imgs = encoder.predict(xtest)
decoded_imgs = decoder.predict(encoded_imgs)

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 [35]:
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(xtrain, ytrain,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(decoded_imgs, ytest))
score_nn = model.evaluate(decoded_imgs, ytest, verbose=0)
print('Test loss:', score_nn[0])
print('Test accuracy:', score_nn[1])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_81 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_82 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_83 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 8s 18ms/step - loss: 0.2450 - accura
cy: 0.9237 - val_loss: 5.5860 - val_accuracy: 0.0892
Epoch 2/20
469/469 [==============================] - 8s 16ms/step - loss: 0.1016 - accura
cy: 0.9689 - val_loss: 10.6933 - val_accuracy: 0.0892
Epoch 3/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0756 - accura
cy: 0.9779 - val_loss: 15.5065 - val_accuracy: 0.0958
Epoch 4/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0608 - accura
cy: 0.9817 - val_loss: 17.0012 - val_accuracy: 0.0958
Epoch 5/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0499 - accura
cy: 0.9846 - val_loss: 27.6631 - val_accuracy: 0.0958
Epoch 6/20
```

```
469/469 [==============================] - 10s 20ms/step - loss: 0.0433 - accur
acy: 0.9874 - val_loss: 29.8694 - val_accuracy: 0.1032
Epoch 7/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0371 - accura
cy: 0.9886 - val_loss: 43.2751 - val_accuracy: 0.1032
Epoch 8/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0338 - accura
cy: 0.9896 - val_loss: 46.0997 - val_accuracy: 0.0958
Epoch 9/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0311 - accura
cy: 0.9911 - val_loss: 69.4659 - val_accuracy: 0.1032
Epoch 10/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0281 - accura
cy: 0.9913 - val_loss: 61.7929 - val_accuracy: 0.1032
Epoch 11/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0265 - accura
cy: 0.9923 - val_loss: 54.4139 - val_accuracy: 0.0958
Epoch 12/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0244 - accura
cy: 0.9926 - val_loss: 63.5142 - val_accuracy: 0.1032
Epoch 13/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0228 - accura
cy: 0.9935 - val_loss: 50.2890 - val_accuracy: 0.1032
Epoch 14/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0202 - accura
cy: 0.9937 - val_loss: 75.2488 - val_accuracy: 0.0958
Epoch 15/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0198 - accura
cy: 0.9944 - val_loss: 79.1382 - val_accuracy: 0.0958
Epoch 16/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0210 - accura
cy: 0.9941 - val_loss: 85.6847 - val_accuracy: 0.0961
Epoch 17/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0193 - accura
cy: 0.9949 - val_loss: 97.1585 - val_accuracy: 0.0958
Epoch 18/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0186 - accura
cy: 0.9947 - val_loss: 109.5125 - val_accuracy: 0.1032
Epoch 19/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0176 - accura
cy: 0.9956 - val_loss: 93.1350 - val_accuracy: 0.1032
Epoch 20/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0166 - accura
cy: 0.9956 - val_loss: 90.6213 - val_accuracy: 0.0982
Test loss: 90.62125396728516
Test accuracy: 0.0982000008225441
```

3. apply noise to *only* the input of the autoencoder (not the output). demonstrate that your autoencoder can strip out noise.

In [37]:
```
xtrain_noise10 = xtrain + np.random.normal(0, 255*.10, xtrain.shape)
xtest_noise10 = xtest + np.random.normal(0, 255*.10, xtest.shape)
```

In [39]:
```python
autoencoder.fit(xtrain_noise10, xtrain,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest_noise10, xtest),
                callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])
```

```
Epoch 1/20
235/235 [==============================] - 7s 28ms/step - loss: 0.8259 - val_lo
ss: 0.7846
Epoch 2/20
235/235 [==============================] - 6s 26ms/step - loss: 0.7538 - val_lo
ss: 0.7271
Epoch 3/20
235/235 [==============================] - 6s 26ms/step - loss: 0.7066 - val_lo
ss: 0.6887
Epoch 4/20
235/235 [==============================] - 7s 30ms/step - loss: 0.6740 - val_lo
ss: 0.6608
Epoch 5/20
235/235 [==============================] - 8s 32ms/step - loss: 0.6488 - val_lo
ss: 0.6375
Epoch 6/20
235/235 [==============================] - 7s 32ms/step - loss: 0.6259 - val_lo
ss: 0.6144
Epoch 7/20
235/235 [==============================] - 8s 34ms/step - loss: 0.6015 - val_lo
ss: 0.5884
Epoch 8/20
235/235 [==============================] - 7s 30ms/step - loss: 0.5734 - val_lo
ss: 0.5579
Epoch 9/20
235/235 [==============================] - 7s 28ms/step - loss: 0.5406 - val_lo
ss: 0.5227
Epoch 10/20
235/235 [==============================] - 8s 32ms/step - loss: 0.5036 - val_lo
ss: 0.4841
Epoch 11/20
235/235 [==============================] - 7s 30ms/step - loss: 0.4645 - val_lo
ss: 0.4450
Epoch 12/20
235/235 [==============================] - 7s 30ms/step - loss: 0.4266 - val_lo
ss: 0.4088
Epoch 13/20
235/235 [==============================] - 7s 28ms/step - loss: 0.3931 - val_lo
ss: 0.3784
Epoch 14/20
235/235 [==============================] - 6s 27ms/step - loss: 0.3660 - val_lo
ss: 0.3547
Epoch 15/20
235/235 [==============================] - 7s 30ms/step - loss: 0.3455 - val_lo
ss: 0.3373
Epoch 16/20
235/235 [==============================] - 8s 32ms/step - loss: 0.3306 - val_lo
ss: 0.3246
Epoch 17/20
```

```
235/235 [==============================] - 7s 32ms/step - loss: 0.3197 - val_lo
ss: 0.3153
Epoch 18/20
235/235 [==============================] - 7s 29ms/step - loss: 0.3118 - val_lo
ss: 0.3085
Epoch 19/20
235/235 [==============================] - 7s 30ms/step - loss: 0.3059 - val_lo
ss: 0.3033
Epoch 20/20
235/235 [==============================] - 7s 29ms/step - loss: 0.3014 - val_lo
ss: 0.2993
```

Out[39]:

```
<tensorflow.python.keras.callbacks.History at 0x19b02bf8e08>
```

In [40]:

```python
encoded_imgs = encoder.predict(xtest_noise10)
decoded_imgs = decoder.predict(encoded_imgs)

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()
```