```
'''This script demonstrates how to build a variational autoencoder with Keras.

 #Reference

 - Auto-Encoding Variational Bayes
   https://arxiv.org/abs/1312.6114
'''
from __future__ import print_function

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

from keras.layers import Input, Dense, Lambda
from keras.models import Model
from keras import backend as K
from keras import metrics
from keras.datasets import mnist

batch_size = 100
original_dim = 784
latent_dim = 2
intermediate_dim = 256
epochs = 50
epsilon_std = 1.0


x = Input(shape=(original_dim,))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_var = Dense(latent_dim)(h)


def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim), mean=0.,
                              stddev=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon

z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])

# we instantiate these layers separately so as to reuse them later
decoder_h = Dense(intermediate_dim, activation='relu')
decoder_mean = Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)

# instantiate VAE model
vae = Model(x, x_decoded_mean)

# Compute VAE loss
xent_loss = original_dim * metrics.binary_crossentropy(x, x_decoded_mean)
kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
vae_loss = K.mean(xent_loss + kl_loss)
```

```
vae_loss = K.mean(xent_loss + kl_loss)

vae.add_loss(vae_loss)
vae.compile(optimizer='rmsprop')
vae.summary()


# train the VAE on MNIST digits
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

vae.fit(x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

Model: "model_1"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)            (None, 784)          0
__________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          200960      input_1[0][0]
__________________________________________________________________________________________
dense_2 (Dense)                 (None, 2)            514         dense_1[0][0]
__________________________________________________________________________________________
dense_3 (Dense)                 (None, 2)            514         dense_1[0][0]
__________________________________________________________________________________________
lambda_1 (Lambda)               (None, 2)            0           dense_2[0][0]
                                                                 dense_3[0][0]
__________________________________________________________________________________________
dense_4 (Dense)                 (None, 256)          768         lambda_1[0][0]
__________________________________________________________________________________________
dense_5 (Dense)                 (None, 784)          201488      dense_4[0][0]
==========================================================================================
Total params: 404,244
Trainable params: 404,244
Non-trainable params: 0
__________________________________________________________________________________________
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Train on 60000 samples, validate on 10000 samples
Epoch 1/50
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

60000/60000 [==============================] - 8s 133us/step - loss: 189.9023 - val_l
Epoch 2/50
60000/60000 [==============================] - 3s 58us/step - loss: 169.6537 - val_lo
Epoch 3/50
60000/60000 [==============================] - 4s 59us/step - loss: 166.4691 - val_lo
Epoch 4/50
60000/60000 [==============================] - 4s 59us/step - loss: 164.4477 - val_lo
Epoch 5/50
60000/60000 [==============================] - 4s 59us/step - loss: 162.9615 - val_lo
Epoch 6/50
60000/60000 [==============================] - 4s 59us/step - loss: 161.7468 - val_lo
Epoch 7/50
60000/60000 [==============================] - 3s 57us/step - loss: 160.6697 - val_lo
Epoch 8/50
60000/60000 [==============================] - 3s 58us/step - loss: 159.6620 - val_lo
Epoch 9/50
60000/60000 [==============================] - 3s 58us/step - loss: 158.7464 - val_lo
Epoch 10/50
60000/60000 [==============================] - 4s 58us/step - loss: 157.9217 - val_lo
Epoch 11/50
60000/60000 [==============================] - 4s 58us/step - loss: 157.2431 - val_lo
Epoch 12/50
60000/60000 [==============================] - 3s 58us/step - loss: 156.6072 - val_lo
Epoch 13/50
60000/60000 [==============================] - 3s 57us/step - loss: 156.0451 - val_lo
Epoch 14/50
60000/60000 [==============================] - 3s 58us/step - loss: 155.5604 - val_lo
Epoch 15/50
60000/60000 [==============================] - 3s 57us/step - loss: 155.1444 - val_lo
Epoch 16/50
60000/60000 [==============================] - 3s 56us/step - loss: 154.7395 - val_lo
Epoch 17/50
60000/60000 [==============================] - 3s 56us/step - loss: 154.3811 - val_lo
Epoch 18/50
60000/60000 [==============================] - 3s 57us/step - loss: 154.0666 - val_lo
Epoch 19/50
60000/60000 [==============================] - 3s 56us/step - loss: 153.7754 - val_lo
Epoch 20/50
60000/60000 [==============================] - 3s 56us/step - loss: 153.4941 - val_lo
Epoch 21/50
60000/60000 [==============================] - 3s 56us/step - loss: 153.2324 - val_lo
Epoch 22/50
60000/60000 [==============================] - 3s 56us/step - loss: 153.0272 - val_lo
Epoch 23/50
60000/60000 [==============================] - 3s 56us/step - loss: 152.7859 - val_lo
Epoch 24/50
60000/60000 [==============================] - 3s 56us/step - loss: 152.5727 - val_lo
Epoch 25/50
60000/60000 [==============================] - 3s 56us/step - loss: 152.3900 - val_lo
Epoch 26/50
60000/60000 [==============================] - 3s 57us/step - loss: 152.1909 - val_lo
Epoch 27/50
60000/60000 [==============================] - 3s 57us/step - loss: 152.0527 - val_lo
Epoch 28/50
60000/60000 [==============================] - 3s 58us/step - loss: 151.8713 - val_lo
Epoch 29/50
60000/60000 [==============================] - 3s 57us/step - loss: 151.7050 - val_lo
Epoch 30/50
60000/60000 [==============================] - 3s 57us/step - loss: 151.5530 - val_lo
Epoch 31/50
```

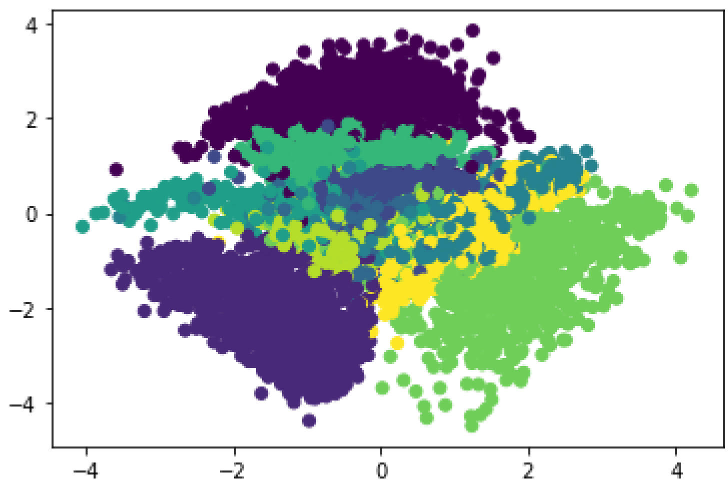
```
60000/60000 [==============================] - 3s 56us/step - loss: 151.3902 - val_lo
Epoch 32/50
60000/60000 [==============================] - 3s 56us/step - loss: 151.2661 - val_lo
Epoch 33/50
60000/60000 [==============================] - 3s 56us/step - loss: 151.1015 - val_lo
Epoch 34/50
60000/60000 [==============================] - 3s 57us/step - loss: 150.9959 - val_lo
Epoch 35/50
60000/60000 [==============================] - 3s 57us/step - loss: 150.8663 - val_lo
Epoch 36/50
60000/60000 [==============================] - 3s 57us/step - loss: 150.7106 - val_lo
Epoch 37/50
60000/60000 [==============================] - 4s 60us/step - loss: 150.6327 - val_lo
Epoch 38/50
60000/60000 [==============================] - 4s 59us/step - loss: 150.5002 - val_lo
Epoch 39/50
60000/60000 [==============================] - 4s 58us/step - loss: 150.4055 - val_lo
Epoch 40/50
60000/60000 [==============================] - 3s 57us/step - loss: 150.2593 - val_lo
Epoch 41/50
60000/60000 [==============================] - 3s 57us/step - loss: 150.1737 - val_lo
Epoch 42/50
60000/60000 [==============================] - 3s 58us/step - loss: 150.0865 - val_lo
Epoch 43/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.9949 - val_lo
Epoch 44/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.8776 - val_lo
Epoch 45/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.7981 - val_lo
Epoch 46/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.7199 - val_lo
Epoch 47/50
60000/60000 [==============================] - 3s 57us/step - loss: 149.6193 - val_lo
Epoch 48/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.5234 - val_lo
Epoch 49/50
60000/60000 [==============================] - 3s 57us/step - loss: 149.4342 - val_lo
Epoch 50/50
60000/60000 [==============================] - 3s 58us/step - loss: 149.3737 - val_lo
<keras.callbacks.History at 0x7f4079473978>
```

Here we start writing our code

Section C - Add an encoder which maps MNIST digits to the latent space. Using this encoder, visu

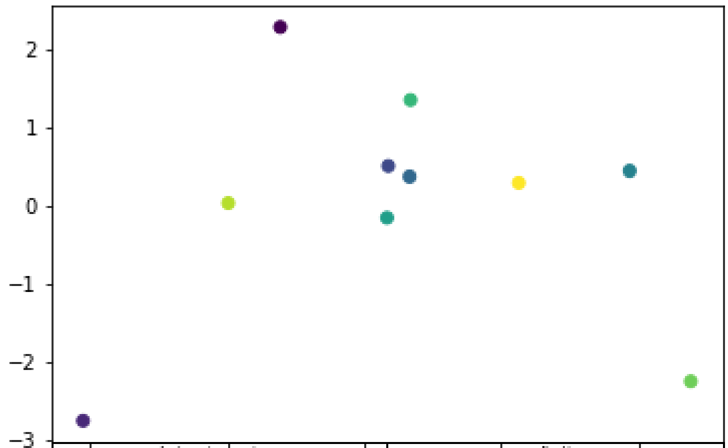
```
# encoder which maps MNIST digits to the latent space
encoder = Model(x, z_mean)

# visualize the test set in the latent space
x_test_encoded = encoder.predict(x_test, batch_size=batch_size)
```

```
plt.scatter(x_test_encoded[:, 0], x_test_encoded[:, 1], c=y_test)
plt.show()
```



Section C - Take one image per digit and print its corresponding mapping coordinates in the latent

```
# Take one image per digit and print its corresponding mapping coordinates
# in the latent space, present the answer as a table
NUM_OF_DIGITS = 10
digits_to_latent = {}
for i, digit in enumerate(y_test):
    if len(digits_to_latent) == NUM_OF_DIGITS:
        break
    if digit in digits_to_latent:
        continue
    digits_to_latent[digit] = (x_test_encoded[i, 0], x_test_encoded[i, 1])
assert len(digits_to_latent) == NUM_OF_DIGITS
plt.scatter([latent[0] for latent in digits_to_latent.values()], [latent[1] for latent in
plt.table(colLabels=('latent vector', 'digit'),
                cellText=[[latent, digit] for digit, latent in digits_to_latent.items()])
plt.show()
```

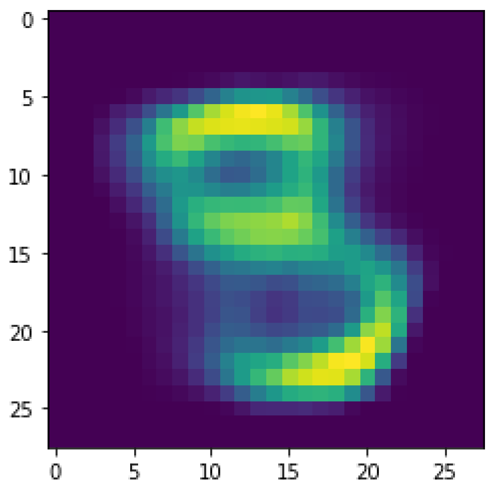


latent vector	digit
(2.3814528, -2.2591352)	7
(0.1742696, 0.49609005)	2
(-2.0521808, -2.764799)	1
(-0.61504954, 2.27745)	0
(1.9348451, 0.435468)	4
(1.1276757, 0.2855484)	9
(0.16512802, -0.1614288)	5
(0.33668974, 1.3422341)	6
(0.33013064, 0.36274582)	3
(-0.99512625, 0.026479289)	8

Section D - Use the following code to define a generator that based on a sample from the latent sp

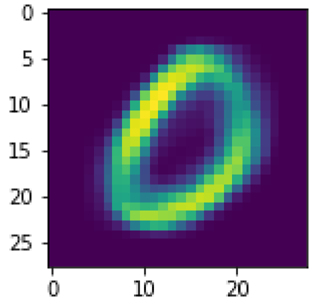
```
# Use the following code to define a generator that based on a sample from
# the latent space, generates a digits.
decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
generator = Model(inputs=decoder_input, outputs=_x_decoded_mean)
z_sample = np.array([[0.5, 0.2]])
x_decoded = generator.predict(z_sample)

plt.imshow(x_decoded.reshape(28,28))
```

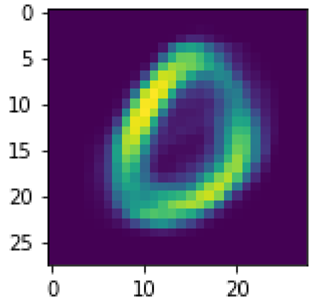
```
<matplotlib.image.AxesImage at 0x7f4016b92160>
```

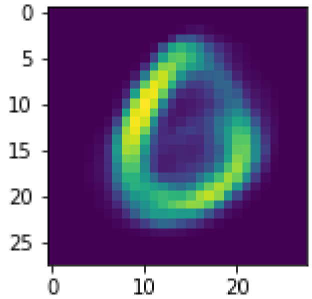


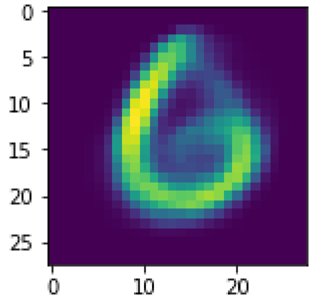
Section E - Take two original images from MNIST of different digits. Sample 10 points from the lin
latent space and generate their images

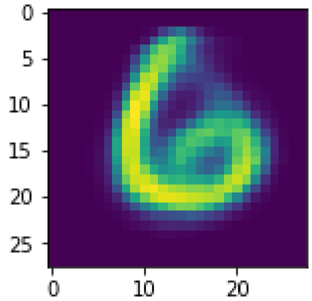
```
# Take two original images from MNIST of different digits
FIRST_DIGIT = 0
SECOND_DIGIT = 9
first_z = digits_to_latent[FIRST_DIGIT]
second_z = digits_to_latent[SECOND_DIGIT]

# Sample 10 points from the line connecting the two representations
# in the latent space and generate their images
SAMPLE_AMOUNT = 10
sampled = list(zip(np.linspace(first_z[0], second_z[0], SAMPLE_AMOUNT), np.linspace(first_
fig=plt.figure(figsize=(28, 28))
columns = 1
rows = 10
for i, z_sample in enumerate(sampled):
  x_sample = generator.predict(np.array([list(z_sample)]))
  fig.add_subplot(rows, columns, i+1)
  plt.imshow(x_sample.reshape(28,28))
plt.show()
```

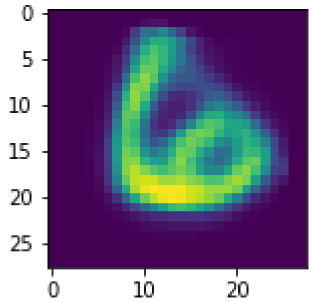

0
5
10
15
20
25
0 10 20

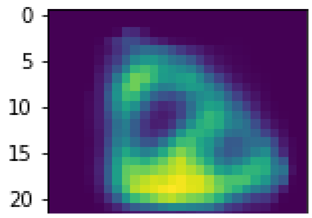

0
5
10
15
20
25
0 10 20

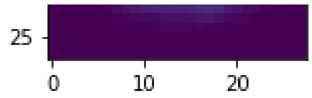

0
5
10
15
20
25
0 10 20

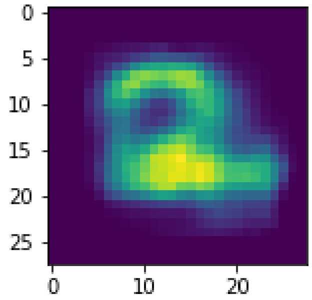

0
5
10
15
20
25
0 10 20

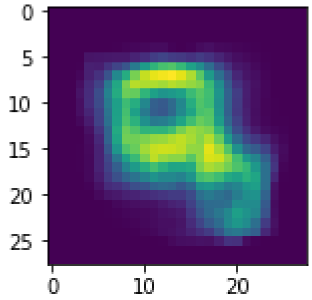

0
5
10
15
20
25
0 10 20

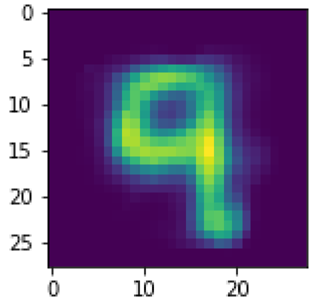

0
5
10
15
20
25
0 10 20


0
5
10
15
20


25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20