Variational Autoencoder and Generate Images in Python

Classical autoencoder simply learns how to encode input and decode the output based on given data using in between randomly generated latent space layer. By using this method, we can not increase the model training ability by updating parameters in learning.

The variational autoencoders, on the other hand, apply some statistical findings by using learned mean and standard deviations to learn the distribution. The latent space mean, and variance are kept to update in each layer and this helps to improve the generator model.

Here we will learn how to build the Variational Autoencoder (VAE) and generate the images with Keras in Python. We will cover:

1. Preparing the data
2. Defining the encoder
3. Defining decoder
4. Defining the VAE model
5. Generating images
6. Source code listing

We'll start by loading the required libraries:

```
from keras.models import Model
from keras.datasets import mnist
from keras.layers import Dense, Input
from keras.layers import Conv2D, Flatten, Lambda
from keras.layers import Reshape, Conv2DTranspose
from keras import backend as K
from keras.losses import binary_crossentropy
from numpy import reshape
import matplotlib.pyplot as plt
```

Preparing the data

We'll use MNIST handwritten digit dataset to train the VAE model. We'll start loading the dataset and check the dimensions.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, x_test.shape)

(60000, 28, 28) (10000, 28, 28)
```

Here, the first element is sample numbers, the second and third elements are the dimension (width and height) of the image. Then, we'll reshape the array again.

```
image_size = x_train.shape[1]
x_train = reshape(x_train, [-1, image_size, image_size, 1])
x_test = reshape(x_test, [-1, image_size, image_size, 1])

print(x_train.shape, x_test.shape)

(60000, 28, 28, 1) (10000, 28, 28, 1)
```

Next, we'll scale the array data.

```
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

Defining the encoder

Encoder is convolutional network model to receive input data and transform it into the latent space array. Here, we need to define sampling function to use in encoding layer. The latent space sampling function helps to sample the distribution by using mean and variance and returns sampled latent vector.

```python
latent_dim = 8

def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon
```

After the first layers, we'll extract the mean and log variance of this layer. We can create a z layer based on those two parameters to generate an input image.

```
input_img = Input(shape=(image_size, image_size, 1),)

h=Conv2D(16,kernel_size=3,activation='relu',padding='same',strides=2)(input_i
mg)
enc_ouput=Conv2D(32,kernel_size=3,activation='relu',padding='same',strides=2)
(h)

shape = K.int_shape(enc_ouput)
x = Flatten()(enc_ouput)
x = Dense(16, activation='relu')(x)
z_mean = Dense(latent_dim, name='z_mean')(x)
z_log_var = Dense(latent_dim, name='z_log_var')(x)
z = Lambda(sampling, output_shape=(latent_dim,), name='z')([z_mean,
z_log_var])

encoder = Model(input_img, [z_mean, z_log_var, z], name='encoder')
encoder.summary()


______________________________________________________________________________
__
Layer (type)                Output Shape         Param #     Connected to
==============================================================================
==
input_2 (InputLaye          (None, 28, 28, 1)    0

______________________________________________________________________________
__
conv2d_5 (Conv2D)           (None, 14, 14, 16)   160         input_2[0][0]

______________________________________________________________________________
__
conv2d_6 (Conv2D            (None, 7, 7, 32)     4640        conv2d_5[0][0]

______________________________________________________________________________
```

```
__
flatten_3 (Flatten)          (None, 1568)         0           conv2d_6[0][0]
__________________________________________________________________________________
__
dense_4 (Dense)              (None, 16)           25104       flatten_3[0][0]
__________________________________________________________________________________
__
z_mean (Dense)               (None, 8)            136         dense_4[0][0]
__________________________________________________________________________________
__
z_log_var (Dense)            (None, 8)            136         dense_4[0][0]
__________________________________________________________________________________
__
z (Lambda)                   (None, 8)            0           z_mean[0][0]
                                                              z_log_var[0][0]
==================================================================================
==
Total params: 30,176
Trainable params: 30,176
Non-trainable params: 0
```

Defining the decoder

Decoder model generates the image from the latent input layer. We can define it as below.

```
latent_inputs = Input(shape=(latent_dim,), name='z_sampling')
x = Dense(shape[1] * shape[2] * shape[3], activation='relu')(latent_inputs)
x = Reshape((shape[1], shape[2], shape[3]))(x)

x=Conv2DTranspose(32,
kernel_size=3,activation='relu',strides=2,padding='same')(x)
x=Conv2DTranspose(16,
kernel_size=3,activation='relu',strides=2,padding='same')(x)
dec_output=Conv2DTranspose(1, kernel_size=3,
activation='relu',padding='same')(x)

decoder = Model(latent_inputs, dec_output, name='decoder')
decoder.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 8)                 0
_________________________________________________________________
```

```
dense_5 (Dense)              (None, 1568)              14112     
_________________________________________________________________
reshape_2 (Reshape)          (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 14, 14, 32)        9248      
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 16)        4624      
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 28, 28, 1)         145       
=================================================================
Total params: 28,129
Trainable params: 28,129
Non-trainable params: 0
_________________________________________________________________
```

Defining the VAE model

Next, we'll define the VAE model. The VAE model combines both encoder and decoder layers. We need to define the loss function and feed into the model.

```
outputs = decoder(encoder(input_img)[2])
vae = Model(input_img, outputs, name='vae')

reconst_loss = binary_crossentropy(K.flatten(input_img), K.flatten(outputs))
reconst_loss *= image_size * image_size
kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconst_loss + kl_loss)

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

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         (None, 28, 28, 1)         0         
_________________________________________________________________
encoder (Model)              [(None, 8), (None, 8), (N 30176     
_________________________________________________________________
decoder (Model)              (None, 28, 28, 1)         28129     
=================================================================
Total params: 58,305
Trainable params: 58,305
```

```
Non-trainable params: 0
_________________________________________________________________
```

Now, we can fit the model on training data.

```
vae.fit(x_train,epochs=20,batch_size=128,shuffle=True,validation_data=(x_test
,None))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20

  128/60000 [..............................] - ETA: 13:05 - loss: 1450.7305
  256/60000 [..............................] - ETA: 6:48 - loss: 991.3974
  512/60000 [..............................] - ETA: 3:33 - loss: 688.9767
  768/60000 [..............................] - ETA: 2:28 - loss: 575.8487
 1024/60000 [..............................] - ETA: 1:55 - loss: 515.392
 1280/60000 [..............................] - ETA: 1:35 - loss: 477.5530
 1536/60000 [..............................] - ETA: 1:22 - loss: 449.4145
 1792/60000 [..............................] - ETA: 1:12 - loss: 426.2855
 2048/60000 [>.............................] - ETA: 1:05 - loss: 408.3294
 2304/60000 [>.............................] - ETA: 1:00 - loss: 392.8099
 2560/60000 [>.............................] - ETA: 55s - loss: 380.6519
 .....
```

Generating the images

To generate images, first we'll encode test data with encoder and extract z_mean value. Then we'll predict it with decoder.

```
z_mean, _, _ = encoder.predict(x_test)
decoded_imgs = decoder.predict(z_mean)
```

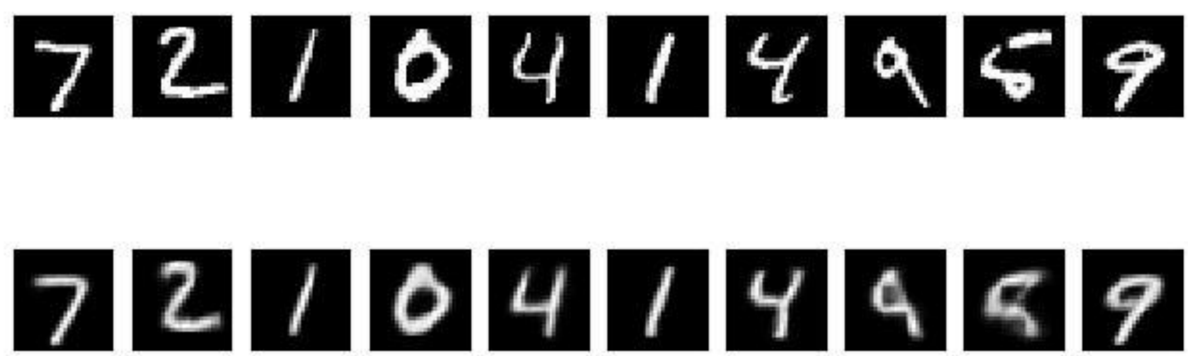
Finally, we'll visualize the first 10 images of both original and predicted data.

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(10):
       plt.gray()
       ax = plt.subplot(2, n, i+1)
       plt.imshow(x_test[i].reshape(28, 28))
```

```
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        ax = plt.subplot(2, n, i +1+n)
        plt.imshow(decoded_imgs[i].reshape(28, 28))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
plt.show()
```

The result looks as below.



We have learned how to build the VAE model and generated the images with Keras in Python. The full source code is listed below.

Source code listing

```
from keras.models import Model
from keras.datasets import mnist
from keras.layers import Dense, Input
from keras.layers import Conv2D, Flatten, Lambda
from keras.layers import Reshape, Conv2DTranspose
from keras import backend as K
from keras.losses import binary_crossentropy
from numpy import reshape
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, x_test.shape)

image_size = x_train.shape[1]
x_train = reshape(x_train, [-1, image_size, image_size, 1])
x_test = reshape(x_test, [-1, image_size, image_size, 1])
print(x_train.shape, x_test.shape)

x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255


latent_dim = 8
input_img = Input(shape=(image_size, image_size, 1),)

def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon


h=Conv2D(16, kernel_size=3, activation='relu',
padding='same',strides=2)(input_img)
enc_ouput=Conv2D(32, kernel_size=3, activation='relu',
padding='same',strides=2)(h)

shape = K.int_shape(enc_ouput)
```

```
x = Flatten()(enc_ouput)
x = Dense(16, activation='relu')(x)
z_mean = Dense(latent_dim, name='z_mean')(x)
z_log_var = Dense(latent_dim, name='z_log_var')(x)
z = Lambda(sampling, output_shape=(latent_dim,), name='z')([z_mean,
z_log_var])

encoder = Model(input_img, [z_mean, z_log_var, z], name='encoder')
encoder.summary()

# decoder
latent_inputs = Input(shape=(latent_dim,), name='z_sampling')
x = Dense(shape[1] * shape[2] * shape[3], activation='relu')(latent_inputs)
x = Reshape((shape[1], shape[2], shape[3]))(x)

x=Conv2DTranspose(32, kernel_size=3, activation='relu',
strides=2,padding='same')(x)
x=Conv2DTranspose(16, kernel_size=3, activation='relu',
strides=2,padding='same')(x)
dec_output = Conv2DTranspose(1, kernel_size=3,
activation='relu',padding='same')(x)

decoder = Model(latent_inputs, dec_output, name='decoder')
decoder.summary()


# autoencoder definition
outputs = decoder(encoder(input_img)[2])
vae = Model(input_img, outputs, name='vae')

reconst_loss = binary_crossentropy(K.flatten(input_img), K.flatten(outputs))
reconst_loss *= image_size * image_size
kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconst_loss + kl_loss)

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

vae.fit(x_train,epochs=20,batch_size=128,shuffle=True,validation_data=(x_test
,None))

z_mean, _, _ = encoder.predict(x_test)
```

```
decoded_imgs = decoder.predict(z_mean)

n = 10
plt.figure(figsize=(20, 4))
for i in range(10):
        plt.gray()
        ax = plt.subplot(2, n, i+1)
        plt.imshow(x_test[i].reshape(28, 28))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        ax = plt.subplot(2, n, i +1+n)
        plt.imshow(decoded_imgs[i].reshape(28, 28))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
plt.show()
```