Generating new MNIST digits with VAE

Source: https://www.kaggle.com/code/vincentman0403/vae-with-convolution-on-mnist

How to use VAE(Variational Auto-Encoder) to generate different MNIST images? VAE use convolution layers in encoder and decoder. ¶

In [1]:
```
from keras.layers import Dense, Input
from keras.layers import Conv2D, Flatten, Lambda
from keras.layers import Reshape, Conv2DTranspose
from keras.models import Model
from keras.datasets import mnist
from keras.losses import mse, binary_crossentropy
from keras import backend as K

import numpy as np
import matplotlib.pyplot as plt
import argparse
import os
```

Step 1: Load and explore the dataset

In [2]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

image_size = x_train.shape[1]
# Only get some data to train and test
train_len = 10000
test_len = 1000
x_train = np.reshape(x_train[:train_len], [-1, image_size, image_size, 1])
x_test = np.reshape(x_test[:test_len], [-1, image_size, image_size, 1])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_test = y_test[:test_len]
```

In [3]:
```
x_train.shape
```

Out[3]:
```
(10000, 28, 28, 1)
```

In [4]:
```
x_test.shape
```

Out[4]:
```
(1000, 28, 28, 1)
```

In [6]:

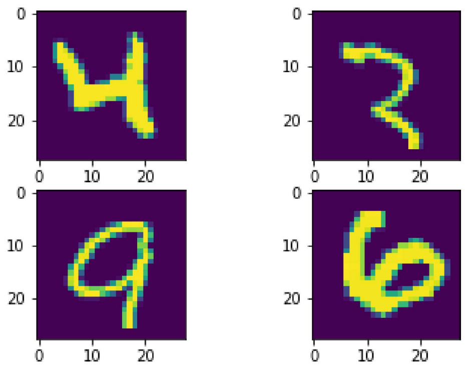
```
#Show MNIST images
x_train_show = x_train.reshape(-1,28,28)
plt.figure(1)
plt.subplot(221)
plt.imshow(x_train_show[20])
print('y_train[20] = ', y_train[20])

plt.subplot(222)
plt.imshow(x_train_show[500])
print('y_train[500] = ', y_train[500])

plt.subplot(223)
plt.imshow(x_train_show[3000])
print('y_train[3000] = ', y_train[3000])

plt.subplot(224)
plt.imshow(x_train_show[9000])
print('y_train[9000] = ', y_train[9000])
plt.show()
```

```
y_train[20] =  4
y_train[500] =  3
y_train[3000] =  9
y_train[9000] =  6
```



Step 2: Prepare functions to produce and plot the results (before we start)

In [7]:

```
# reparameterization trick: instead of sampling from Q(z|X), sample epsilon = N(0,1)
# then z = z_mean + sqrt(var)*eps
def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    # by default, random_normal has mean=0 and std=1.0
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon
```

In [26]:

```
def plot_results(encoder, decoder, x_test, y_test, batch_size=128):
    # display a 2D plot of the digit classes in the latent space
    z_mean, _, _ = encoder.predict(x_test,
                                   batch_size=batch_size)
    plt.figure(figsize=(12, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=y_test)
    plt.colorbar()
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.show()

    # display a 30x30 2D manifold of digits
    n = 30
    digit_size = 28
    figure = np.zeros((digit_size * n, digit_size * n))
    # linearly spaced coordinates corresponding to the 2D plot
    # of digit classes in the latent space
    grid_x = np.linspace(-4, 4, n)
    grid_y = np.linspace(-4, 4, n)[::-1]

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = decoder.predict(z_sample)
            digit = x_decoded[0].reshape(digit_size, digit_size)
            figure[i * digit_size: (i + 1) * digit_size,
                   j * digit_size: (j + 1) * digit_size] = digit

    plt.figure(figsize=(10, 10))
    start_range = digit_size // 2
    end_range = n * digit_size + start_range
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
    plt.yticks(pixel_range, sample_range_y)
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.imshow(figure, cmap='Greys_r')
    plt.show()
```

Step 3: Set up network parameters

In [12]:

```
input_shape = (image_size, image_size, 1)
batch_size = 32
kernel_size = 3
filters = 16
latent_dim = 2
epochs = 30
use_mse = True
load_weights = False
```

Step 4: Build encoder model

In [13]:

```
inputs = Input(shape=input_shape, name='encoder_input')
x = inputs
for i in range(2):
    filters *= 2
    x = Conv2D(filters=filters,
               kernel_size=kernel_size,
               activation='relu',
               strides=2,
               padding='same')(x)

# shape info needed to build decoder model
shape = K.int_shape(x)

# generate latent vector Q(z|X)
x = Flatten()(x)
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)

# use reparameterization trick to push the sampling out as input
# note that "output_shape" isn't necessary with the TensorFlow backend
z = Lambda(sampling, output_shape=(latent_dim,), name='z')([z_mean, z_log_var])

# instantiate encoder model
encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
encoder.summary()
```

```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)     [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
conv2d_2 (Conv2D)              (None, 14, 14, 32)   320         encoder_input[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)              (None, 7, 7, 64)     18496       conv2d_2[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)            (None, 3136)         0           conv2d_3[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                (None, 16)           50192       flatten_1[0][0]
__________________________________________________________________________________________________
z_mean (Dense)                 (None, 2)            34          dense_1[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)              (None, 2)            34          dense_1[0][0]
__________________________________________________________________________________________________
z (Lambda)                     (None, 2)            0           z_mean[0][0]
                                                                z_log_var[0][0]
```

Step 5: Build decoder model

In [14]:

```
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)

# use Conv2DTranspose to reverse the conv layers from the encoder
for i in range(2):
    x = Conv2DTranspose(filters=filters,
                        kernel_size=kernel_size,
                        activation='relu',
                        strides=2,
                        padding='same')(x)
    filters //= 2

outputs = Conv2DTranspose(filters=1,
                          kernel_size=kernel_size,
                          activation='sigmoid',
                          padding='same',
                          name='decoder_output')(x)

# instantiate decoder model
decoder = Model(latent_inputs, outputs, name='decoder')
decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      [(None, 2)]               0
_________________________________________________________________
dense_2 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
decoder_output (Conv2DTransp (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

Step 6: Build VAE model = encoder + decoder

In [15]:

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

Loss = reconstruction loss + KL loss

In [16]:

```
if use_mse:
    reconstruction_loss = mse(K.flatten(inputs), K.flatten(outputs))
else:
    reconstruction_loss = binary_crossentropy(K.flatten(inputs),
                                              K.flatten(outputs))

reconstruction_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(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
```

Compile model

In [17]:

```
vae.compile(optimizer='rmsprop')
vae.summary()
```

```
Model: "vae"
__________________________________________________________________________________________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)     [(None, 28, 28, 1)]  0

encoder (Functional)           [(None, 2), (None, 2 69076       encoder_input[0][0]

decoder (Functional)           (None, 28, 28, 1)    65089       encoder[0][2]

conv2d_2 (Conv2D)              (None, 14, 14, 32)   320         encoder_input[0][0]

conv2d_3 (Conv2D)              (None, 7, 7, 64)     18496       conv2d_2[0][0]

flatten_1 (Flatten)            (None, 3136)         0           conv2d_3[0][0]

dense_1 (Dense)                (None, 16)           50192       flatten_1[0][0]

z_log_var (Dense)              (None, 2)            34          dense_1[0][0]

z_mean (Dense)                 (None, 2)            34          dense_1[0][0]

tf.reshape_1 (TFOpLambda)      (None,)              0           decoder[0][0]

tf.reshape (TFOpLambda)        (None,)              0           encoder_input[0][0]

tf.__operators__.add (TFOpLambd (None, 2)           0           z_log_var[0][0]

tf.math.square (TFOpLambda)    (None, 2)            0           z_mean[0][0]

tf.convert_to_tensor (TFOpLambd (None,)             0           tf.reshape_1[0][0]

tf.cast (TFOpLambda)           (None,)              0           tf.reshape[0][0]

tf.math.subtract (TFOpLambda)  (None, 2)            0           tf.__operators__.add[0][0]
                                                                 tf.math.square[0][0]

tf.math.exp (TFOpLambda)       (None, 2)            0           z_log_var[0][0]

tf.math.squared_difference (TFO (None,)             0           tf.convert_to_tensor[0][0]
                                                                 tf.cast[0][0]

tf.math.subtract_1 (TFOpLambda) (None, 2)           0           tf.math.subtract[0][0]
                                                                 tf.math.exp[0][0]

tf.math.reduce_mean (TFOpLambda ()                  0           tf.math.squared_difference[0][0]

tf.math.reduce_sum (TFOpLambda) (None,)             0           tf.math.subtract_1[0][0]

tf.math.multiply (TFOpLambda)  ()                   0           tf.math.reduce_mean[0][0]

tf.math.multiply_1 (TFOpLambda) (None,)             0           tf.math.reduce_sum[0][0]

tf.__operators__.add_1 (TFOpLam (None,)             0           tf.math.multiply[0][0]
                                                                 tf.math.multiply_1[0][0]

tf.math.reduce_mean_1 (TFOpLamb ()                  0           tf.__operators__.add_1[0][0]

add_loss (AddLoss)             ()                   0           tf.math.reduce_mean_1[0][0]
==================================================================================================
Total params: 134,165
Trainable params: 134,165
Non-trainable params: 0
__________________________________________________________________________________________________
```

Step 7: Fit model

In [20]:

```
if load_weights:
    vae = vae.load_weights(args.weights)
else:
    # train the autoencoder
    vae.fit(x_train,
            epochs=epochs,
            batch_size=batch_size,
            validation_data=(x_test, None))
    vae.save_weights('vae_cnn_mnist.h5')
```

```
Epoch 1/30
313/313 [==============================] - 6s 20ms/step - loss: 37.5350 - val_loss: 39.5232
Epoch 2/30
313/313 [==============================] - 7s 21ms/step - loss: 37.4733 - val_loss: 39.1956
Epoch 3/30
313/313 [==============================] - 7s 23ms/step - loss: 37.4376 - val_loss: 39.1108
Epoch 4/30
313/313 [==============================] - 7s 21ms/step - loss: 37.3805 - val_loss: 39.0958
Epoch 5/30
313/313 [==============================] - 6s 20ms/step - loss: 37.2182 - val_loss: 39.4293
Epoch 6/30
313/313 [==============================] - 7s 22ms/step - loss: 37.1992 - val_loss: 39.8790
Epoch 7/30
313/313 [==============================] - 7s 21ms/step - loss: 37.0978 - val_loss: 39.2028
Epoch 8/30
313/313 [==============================] - 7s 22ms/step - loss: 37.0922 - val_loss: 39.0676
Epoch 9/30
313/313 [==============================] - 7s 21ms/step - loss: 37.0021 - val_loss: 39.1928
Epoch 10/30
313/313 [==============================] - 7s 21ms/step - loss: 36.9596 - val_loss: 39.1371
Epoch 11/30
313/313 [==============================] - 6s 20ms/step - loss: 36.9170 - val_loss: 39.2005
Epoch 12/30
313/313 [==============================] - 6s 20ms/step - loss: 36.8380 - val_loss: 40.0566
Epoch 13/30
313/313 [==============================] - 6s 20ms/step - loss: 36.7989 - val_loss: 39.4482
Epoch 14/30
313/313 [==============================] - 7s 24ms/step - loss: 36.7226 - val_loss: 39.1888
Epoch 15/30
313/313 [==============================] - 7s 22ms/step - loss: 36.7134 - val_loss: 39.1914
Epoch 16/30
313/313 [==============================] - 7s 22ms/step - loss: 36.6395 - val_loss: 39.0419
Epoch 17/30
313/313 [==============================] - 6s 20ms/step - loss: 36.6394 - val_loss: 39.0287
Epoch 18/30
313/313 [==============================] - 6s 20ms/step - loss: 36.5277 - val_loss: 39.0719
Epoch 19/30
313/313 [==============================] - 6s 21ms/step - loss: 36.5322 - val_loss: 39.8637
Epoch 20/30
313/313 [==============================] - 7s 21ms/step - loss: 36.4604 - val_loss: 39.8507
Epoch 21/30
313/313 [==============================] - 7s 21ms/step - loss: 36.4415 - val_loss: 39.5375
Epoch 22/30
313/313 [==============================] - 7s 21ms/step - loss: 36.4530 - val_loss: 39.4899
Epoch 23/30
313/313 [==============================] - 6s 20ms/step - loss: 36.3539 - val_loss: 39.3501
Epoch 24/30
313/313 [==============================] - 7s 21ms/step - loss: 36.3014 - val_loss: 39.6465
Epoch 25/30
313/313 [==============================] - 7s 22ms/step - loss: 36.2951 - val_loss: 39.4382
Epoch 26/30
313/313 [==============================] - 7s 22ms/step - loss: 36.2488 - val_loss: 39.1172
Epoch 27/30
313/313 [==============================] - 8s 25ms/step - loss: 36.2677 - val_loss: 39.2752
Epoch 28/30
313/313 [==============================] - 7s 21ms/step - loss: 36.1682 - val_loss: 39.2472
Epoch 29/30
313/313 [==============================] - 6s 20ms/step - loss: 36.1328 - val_loss: 39.2380
Epoch 30/30
313/313 [==============================] - 7s 22ms/step - loss: 36.1408 - val_loss: 39.7729
```

Step 8: Plot new MNIST images

In [27]:
```
plot_results(encoder, decoder, x_test, y_test, batch_size=batch_size)
```