03_variational_autoencoder

September 29, 2021

1 Variational Autoencoder on Fashion MNIST data using Feed-forward NN

Adapted from Building Autoencoders in Keras by Francois Chollet who created Keras.

1.1 Imports & Settings

[11]:
```
from keras.layers import Lambda, Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
from keras.losses import mse, binary_crossentropy
from keras.utils import plot_model
from keras import backend as K

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

1.2 Sampling

[12]:
```
# instead of sampling from Q(z|X), sample eps = N(0,I)
# z = z_mean + sqrt(var)*eps
def sampling(args):
    """Reparameterization trick by sampling fr an isotropic unit Gaussian.

    # Arguments
        args (tensor): mean and log of variance of Q(z|X)

    # Returns
        z (tensor): sampled latent vector
    """

    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
```

1.3 Load Fashion MNIST Data

[13]:
```
# MNIST dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

image_size = x_train.shape[1]
original_dim = image_size * image_size
x_train = np.reshape(x_train, [-1, original_dim])
x_test = np.reshape(x_test, [-1, original_dim])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

1.4 Define Variational Autoencoder Architecture

1.4.1 Network Parameters

[14]:
```
input_shape = (original_dim,)
intermediate_dim = 512
batch_size = 128
latent_dim = 2
epochs = 50
```

1.4.2 Encoder model

Define Layers

[15]:
```
inputs = Input(shape=input_shape, name='encoder_input')
x = Dense(intermediate_dim, activation='relu')(inputs)
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 Model

[16]:
```
encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
encoder.summary()
plot_model(encoder, to_file='vae_mlp_encoder.png', show_shapes=True)
```

```
--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
```

```
==================
encoder_input (InputLayer)      (None, 784)         0

--------------------------------------------------------------------------------
------------------
dense_4 (Dense)                 (None, 512)         401920
encoder_input[0][0]

--------------------------------------------------------------------------------
------------------
z_mean (Dense)                  (None, 2)           1026        dense_4[0][0]

--------------------------------------------------------------------------------
------------------
z_log_var (Dense)               (None, 2)           1026        dense_4[0][0]

--------------------------------------------------------------------------------
------------------
z (Lambda)                      (None, 2)           0           z_mean[0][0]
                                                                z_log_var[0][0]
================================================================================
==================
Total params: 403,972
Trainable params: 403,972
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

1.4.3 Decoder Model

Define Layers

[17]:
```python
latent_inputs = Input(shape=(latent_dim,), name='z_sampling')
x = Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = Dense(original_dim, activation='sigmoid')(x)
```

Instantiate model

[18]:
```python
decoder = Model(latent_inputs, outputs, name='decoder')
decoder.summary()
plot_model(decoder, to_file='vae_mlp_decoder.png', show_shapes=True)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 2)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               1536
_________________________________________________________________
dense_6 (Dense)              (None, 784)               402192
=================================================================
Total params: 403,728
Trainable params: 403,728
```

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

1.4.4 Combine Encoder and Decoder to VAE model

[19]:
```
outputs = decoder(encoder(inputs)[2])
vae = Model(inputs, outputs, name='vae_mlp')
```

[20]:
```
models = (encoder, decoder)
```

1.5 Train Model

[21]:
```
data = (x_test, y_test)

reconstruction_loss = mse(inputs, outputs)
reconstruction_loss *= original_dim

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 784)               0
_________________________________________________________________
encoder (Model)              [(None, 2), (None, 2), (N 403972
_________________________________________________________________
decoder (Model)              (None, 784)               403728
=================================================================
Total params: 807,700
Trainable params: 807,700
Non-trainable params: 0
_________________________________________________________________
```

[22]:
```
plot_model(vae,
           to_file='vae_mlp.png',
           show_shapes=True)
```

[23]:
```
vae.fit(x_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

```python
vae.save_weights('vae_mlp_mnist.h5')
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 5s 84us/step - loss: 43.3090 -
val_loss: 34.3889
Epoch 2/50
60000/60000 [==============================] - 5s 77us/step - loss: 33.4821 -
val_loss: 32.6602
Epoch 3/50
60000/60000 [==============================] - 5s 79us/step - loss: 32.2131 -
val_loss: 32.0505
Epoch 4/50
60000/60000 [==============================] - 5s 81us/step - loss: 31.5358 -
val_loss: 31.1049
Epoch 5/50
60000/60000 [==============================] - 5s 76us/step - loss: 31.0607 -
val_loss: 30.9470
Epoch 6/50
60000/60000 [==============================] - 5s 79us/step - loss: 30.6998 -
val_loss: 30.5644
Epoch 7/50
60000/60000 [==============================] - 5s 82us/step - loss: 30.3913 -
val_loss: 30.1875
Epoch 8/50
60000/60000 [==============================] - 5s 78us/step - loss: 30.1415 -
val_loss: 29.9968
Epoch 9/50
60000/60000 [==============================] - 5s 78us/step - loss: 29.9107 -
val_loss: 29.9642
Epoch 10/50
60000/60000 [==============================] - 5s 78us/step - loss: 29.6511 -
val_loss: 29.7572
Epoch 11/50
60000/60000 [==============================] - 5s 80us/step - loss: 29.4334 -
val_loss: 29.2960
Epoch 12/50
60000/60000 [==============================] - 5s 83us/step - loss: 29.1718 -
val_loss: 28.9645
Epoch 13/50
60000/60000 [==============================] - 5s 83us/step - loss: 28.9931 -
val_loss: 28.8535
Epoch 14/50
60000/60000 [==============================] - 5s 84us/step - loss: 28.8639 -
val_loss: 28.7241
Epoch 15/50
60000/60000 [==============================] - 5s 79us/step - loss: 28.7483 -
```

```
val_loss: 28.6740
Epoch 16/50
60000/60000 [==============================] - 5s 80us/step - loss: 28.6502 -
val_loss: 28.6064
Epoch 17/50
60000/60000 [==============================] - 5s 79us/step - loss: 28.5263 -
val_loss: 28.4739
Epoch 18/50
60000/60000 [==============================] - 5s 79us/step - loss: 28.4683 -
val_loss: 28.6913
Epoch 19/50
60000/60000 [==============================] - 5s 80us/step - loss: 28.3919 -
val_loss: 28.6425
Epoch 20/50
60000/60000 [==============================] - 5s 85us/step - loss: 28.3120 -
val_loss: 28.3970
Epoch 21/50
60000/60000 [==============================] - 5s 81us/step - loss: 28.2618 -
val_loss: 28.2174
Epoch 22/50
60000/60000 [==============================] - 5s 80us/step - loss: 28.1734 -
val_loss: 28.3498
Epoch 23/50
60000/60000 [==============================] - 5s 81us/step - loss: 28.1509 -
val_loss: 28.1824
Epoch 24/50
60000/60000 [==============================] - 5s 81us/step - loss: 28.0902 -
val_loss: 28.2143
Epoch 25/50
60000/60000 [==============================] - 5s 82us/step - loss: 28.0262 -
val_loss: 28.1908
Epoch 26/50
60000/60000 [==============================] - 5s 84us/step - loss: 28.0190 -
val_loss: 28.3770
Epoch 27/50
60000/60000 [==============================] - 5s 88us/step - loss: 28.0110 -
val_loss: 28.1026
Epoch 28/50
60000/60000 [==============================] - 5s 81us/step - loss: 27.9182 -
val_loss: 28.0952
Epoch 29/50
60000/60000 [==============================] - 5s 86us/step - loss: 27.9236 -
val_loss: 27.9516
Epoch 30/50
60000/60000 [==============================] - 5s 81us/step - loss: 27.8752 -
val_loss: 27.9971
Epoch 31/50
60000/60000 [==============================] - 5s 79us/step - loss: 27.8571 -
```

```
val_loss: 27.9477
Epoch 32/50
60000/60000 [==============================] - 5s 80us/step - loss: 27.7946 -
val_loss: 27.8683
Epoch 33/50
60000/60000 [==============================] - 5s 82us/step - loss: 27.7705 -
val_loss: 27.9689
Epoch 34/50
60000/60000 [==============================] - 5s 82us/step - loss: 27.7620 -
val_loss: 27.9068
Epoch 35/50
60000/60000 [==============================] - 5s 86us/step - loss: 27.6971 -
val_loss: 27.8580
Epoch 36/50
60000/60000 [==============================] - 5s 85us/step - loss: 27.7102 -
val_loss: 27.9967
Epoch 37/50
60000/60000 [==============================] - 5s 81us/step - loss: 27.6690 -
val_loss: 27.8570
Epoch 38/50
60000/60000 [==============================] - 5s 83us/step - loss: 27.6489 -
val_loss: 27.7641
Epoch 39/50
60000/60000 [==============================] - 5s 90us/step - loss: 27.6315 -
val_loss: 27.8103
Epoch 40/50
60000/60000 [==============================] - 5s 83us/step - loss: 27.6079 -
val_loss: 27.6965
Epoch 41/50
60000/60000 [==============================] - 5s 86us/step - loss: 27.5654 -
val_loss: 27.7521
Epoch 42/50
60000/60000 [==============================] - 5s 85us/step - loss: 27.5823 -
val_loss: 28.0018
Epoch 43/50
60000/60000 [==============================] - 5s 88us/step - loss: 27.5378 -
val_loss: 27.7956
Epoch 44/50
60000/60000 [==============================] - 5s 83us/step - loss: 27.4946 -
val_loss: 27.7793
Epoch 45/50
60000/60000 [==============================] - 5s 83us/step - loss: 27.4760 -
val_loss: 27.6293
Epoch 46/50
60000/60000 [==============================] - 5s 84us/step - loss: 27.4670 -
val_loss: 27.6876
Epoch 47/50
60000/60000 [==============================] - 5s 88us/step - loss: 27.4831 -
```

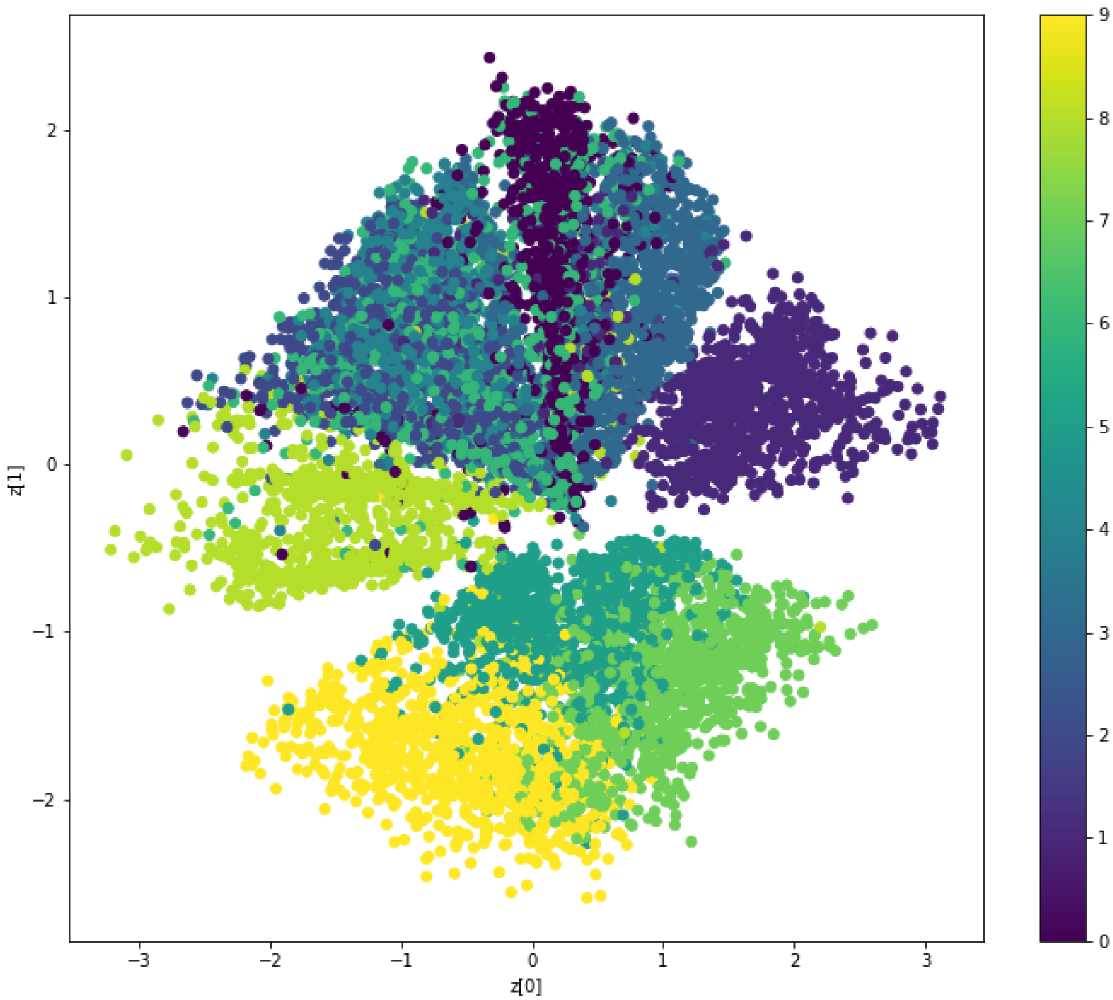
```
val_loss: 27.6400
Epoch 48/50
60000/60000 [==============================] - 5s 87us/step - loss: 27.4264 -
val_loss: 27.5798
Epoch 49/50
60000/60000 [==============================] - 5s 84us/step - loss: 27.4224 -
val_loss: 27.6857
Epoch 50/50
60000/60000 [==============================] - 5s 80us/step - loss: 27.3895 -
val_loss: 27.5600
```

1.6 Plot Results

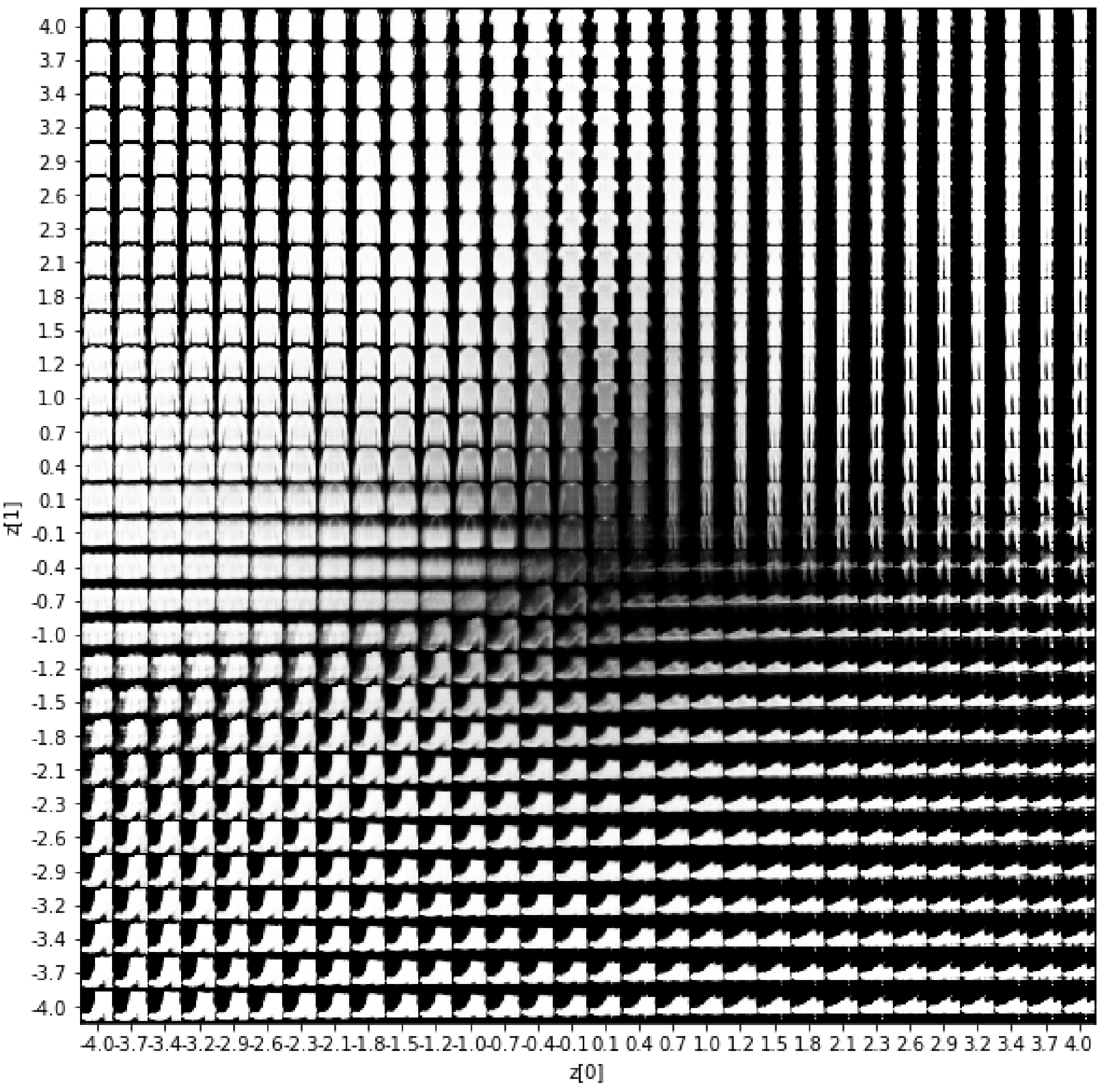
[24]:
```python
def plot_results(models,
                 data,
                 batch_size=128,
                 model_name="vae_mnist"):
    """Plots labels and MNIST digits as function of 2-dim latent vector

    # Arguments
        models (tuple): encoder and decoder models
        data (tuple): test data and label
        batch_size (int): prediction batch size
        model_name (string): which model is using this function
    """

    encoder, decoder = models
    x_test, y_test = data
    os.makedirs(model_name, exist_ok=True)

    filename = os.path.join(model_name, "vae_mean.png")
    # 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.savefig(filename)
    plt.show()

    filename = os.path.join(model_name, "digits_over_latent.png")
    # 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 + 1
    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.savefig(filename)
    plt.show()
```

[25]:
```
plot_results(models,
             data,
             batch_size=batch_size,
             model_name="vae_mlp")
```



z[0]
z[1]
−3
−2
−1
0
1
2
3
0
1
2
3
4
5
6
7
8
9


z[1]
4.0
3.7
3.4
3.2
2.9
2.6
2.3
2.1
1.8
1.5
1.2
1.0
0.7
0.4
0.1
-0.1
-0.4
-0.7
-1.0
-1.2
-1.5
-1.8
-2.1
-2.3
-2.6
-2.9
-3.2
-3.4
-3.7
-4.0
-4.0 -3.7 -3.4 -3.2 -2.9 -2.6 -2.3 -2.1 -1.8 -1.5 -1.2 -1.0 -0.7 -0.4 -0.1 0.1 0.4 0.7 1.0 1.2 1.5 1.8 2.1 2.3 2.6 2.9 3.2 3.4 3.7 4.0
z[0]