assignment12_BasitAbdul

November 19, 2022

1 Assignment 12

1.0.1 Using section 8.4 in Deep Learning with Python as a guide, implement a variational autoencoder using the MNIST data set and save a grid of 15 x 15 digits to the results/vae directory. If you would rather work on a more interesting dataset, you can use the CelebFaces Attributes Dataset instead.

[2]:

```
# VAE encoder network
from tensorflow import keras
from tensorflow.keras import layers

import tensorflow.compat.v1 as tf

from keras import backend as K
from keras.models import Model
import numpy as np
```

[3]:

```
latent_dim = 2 # Dimensionality of the latent space: a 2D plane

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2,␣
 ↪padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)

# The input image ends up being encoded into these two parameters:
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

encoder = keras.Model(encoder_inputs, [z_mean, z_log_var], name="encoder")
```

[4]:

```
# View summary of encoder
encoder.summary()
```

```
Model: "encoder"
________________________________________________________________________________
__________________
```

```
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 28, 28, 1)]  0

--------------------------------------------------------------------------------
------------------
conv2d (Conv2D)                 (None, 14, 14, 32)   320         input_1[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_1 (Conv2D)               (None, 7, 7, 64)     18496       conv2d[0][0]

--------------------------------------------------------------------------------
------------------
flatten (Flatten)               (None, 3136)         0           conv2d_1[0][0]

--------------------------------------------------------------------------------
------------------
dense (Dense)                   (None, 16)           50192       flatten[0][0]

--------------------------------------------------------------------------------
------------------
z_mean (Dense)                  (None, 2)            34          dense[0][0]

--------------------------------------------------------------------------------
------------------
z_log_var (Dense)               (None, 2)            34          dense[0][0]
================================================================================
==================
Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0

--------------------------------------------------------------------------------
------------------
```

[5]:

```
# Latent-space-sampling layer
import tensorflow as tf

class Sampler(layers.Layer):
    def call(self, z_mean, z_log_var):
        batch_size = tf.shape(z_mean)[0]
        z_size = tf.shape(z_mean)[1]
        # Draw a batch of random normal vectors:
        epsilon = tf.random.normal(shape=(batch_size, z_size))
        # Apply the VAE sampling formula:
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

[6]:

```
# VAE decoder network, mapping latent space points to images
latent_inputs = keras.Input(shape=(latent_dim,)) # Input where we'll feed z
# Produce the same number of coefficients that we
# had at the level of the Flatten layer in the encoder:
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
```

```
x = layers.Reshape((7, 7, 64))(x) # Revert the Flatten layer of the encoder
# REvert the Conv2D layers of the encoder:
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2,␣
 ↪padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2,␣
 ↪padding="same")(x)
# The output ends up with shape (28, 28, 1):
decoder_outputs = layers.Conv2D(1, 3, activation="sigmoid", padding="same")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

[7]:
```
# View summary of decoder
decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_1 (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
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
# VAE model with custom `train_step()`
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super().__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.sampler = Sampler()
        # Use these metrics to keep track of the loss averages over each epoch
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss")
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")
```

```
    @property
    # List the metrics in the metrics property to enable the model to reset
    # them after each epoch (or between multiple calls to fit()/evaluate()):
    def metrics(self):
        return [self.total_loss_tracker,
                self.reconstruction_loss_tracker,
                self.kl_loss_tracker]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var = self.encoder(data)
            z = self.sampler(z_mean, z_log_var)
            reconstruction = decoder(z)
            # Sum the reconstruction loss over the spatial dimensions
            # (axes 1 and 2) and take its mean over the batch dimension:
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction),
                    axis=(1, 2)
                )
            )
            # Add the regularization term (Kullback-Leibler divergence):
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) - tf.
↪exp(z_log_var))
            total_loss = reconstruction_loss + tf.reduce_mean(kl_loss)
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "total_loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }
```

[9]:

```
# Training the VAE
import numpy as np

(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
# We train on all MNIST digits, so we concatenate the training/test samples:
mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") / 255

vae = VAE(encoder, decoder)
# Note: we don't pass loss argument in compile(), bc loss is in train_step():
vae.compile(optimizer=keras.optimizers.Adam(), run_eagerly=True)
```

```python
# Note: we don't pass targets in fit(), bc train_step() doesn't expect any:
vae.fit(mnist_digits, epochs=30, batch_size=128)
```

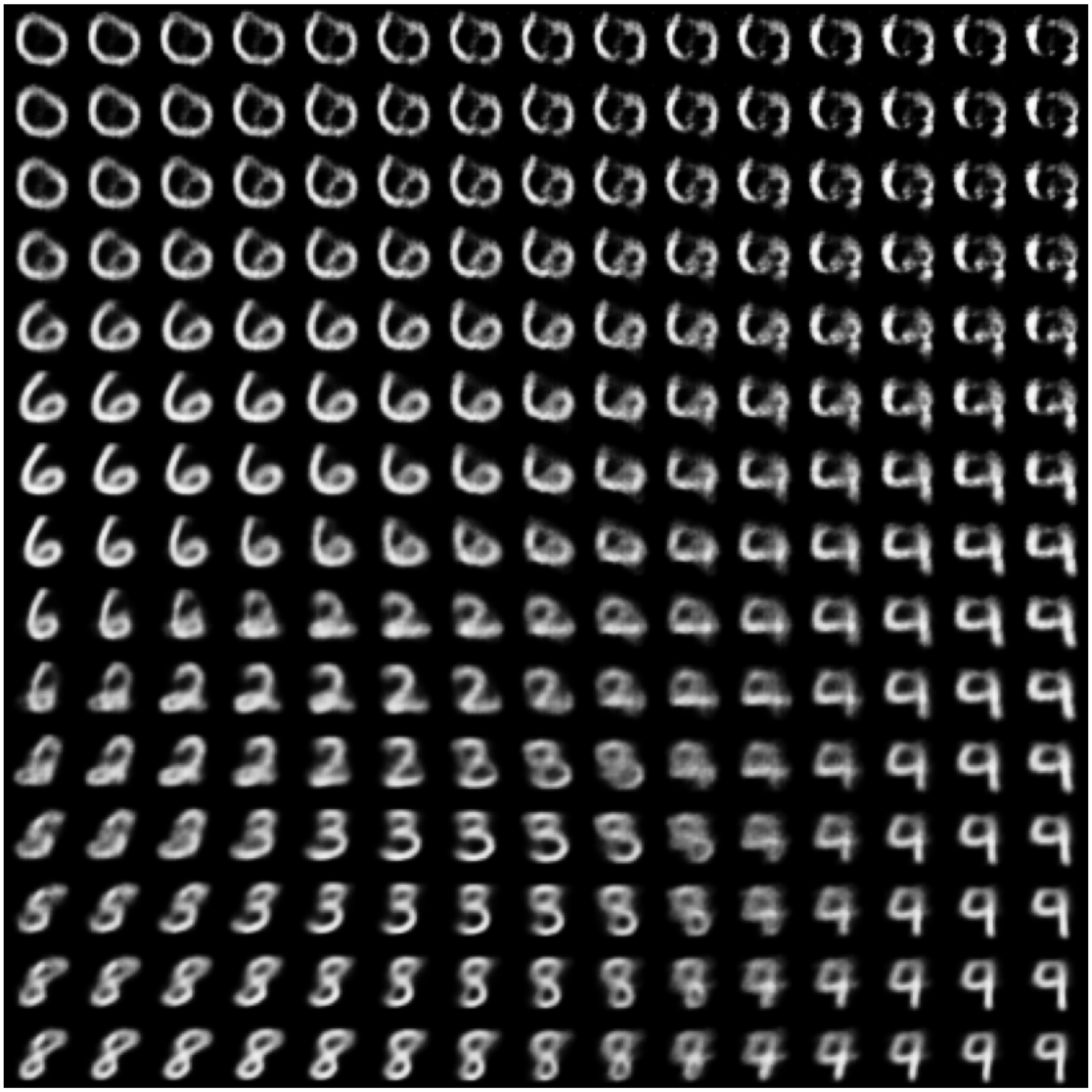
```
Epoch 1/30
547/547 [==============================] - 59s 109ms/step - total_loss: 947.4771
- reconstruction_loss: 939.5665 - kl_loss: 7.9099
Epoch 2/30
547/547 [==============================] - 59s 108ms/step - total_loss: 756.1326
- reconstruction_loss: 747.8571 - kl_loss: 8.2756
Epoch 3/30
547/547 [==============================] - 59s 107ms/step - total_loss: 722.3870
- reconstruction_loss: 715.8335 - kl_loss: 6.5531
Epoch 4/30
547/547 [==============================] - 59s 108ms/step - total_loss: 704.2607
- reconstruction_loss: 697.9865 - kl_loss: 6.2742
Epoch 5/30
547/547 [==============================] - 59s 108ms/step - total_loss: 693.9123
- reconstruction_loss: 687.8236 - kl_loss: 6.0890
Epoch 6/30
547/547 [==============================] - 59s 108ms/step - total_loss: 686.7798
- reconstruction_loss: 680.7572 - kl_loss: 6.0224
Epoch 7/30
547/547 [==============================] - 59s 108ms/step - total_loss: 681.6443
- reconstruction_loss: 675.6786 - kl_loss: 5.9660
Epoch 8/30
547/547 [==============================] - 58s 106ms/step - total_loss: 677.6512
- reconstruction_loss: 671.7861 - kl_loss: 5.8652
Epoch 9/30
547/547 [==============================] - 59s 107ms/step - total_loss: 674.2146
- reconstruction_loss: 668.3749 - kl_loss: 5.8403
Epoch 10/30
547/547 [==============================] - 59s 107ms/step - total_loss: 671.1721
- reconstruction_loss: 665.3798 - kl_loss: 5.7918
Epoch 11/30
547/547 [==============================] - 56s 102ms/step - total_loss: 669.2221
- reconstruction_loss: 663.5048 - kl_loss: 5.7183
Epoch 12/30
547/547 [==============================] - 56s 102ms/step - total_loss: 666.2900
- reconstruction_loss: 660.5960 - kl_loss: 5.6940
Epoch 13/30
547/547 [==============================] - 56s 103ms/step - total_loss: 664.7158
- reconstruction_loss: 659.0743 - kl_loss: 5.6422
Epoch 14/30
547/547 [==============================] - 57s 104ms/step - total_loss: 663.0807
- reconstruction_loss: 657.4609 - kl_loss: 5.6198
Epoch 15/30
547/547 [==============================] - 58s 106ms/step - total_loss: 661.0543
```

```
- reconstruction_loss: 655.4990 - kl_loss: 5.5558
Epoch 16/30
547/547 [==============================] - 58s 106ms/step - total_loss: 659.8758
- reconstruction_loss: 654.3108 - kl_loss: 5.5652
Epoch 17/30
547/547 [==============================] - 59s 107ms/step - total_loss: 658.7617
- reconstruction_loss: 653.2714 - kl_loss: 5.4908
Epoch 18/30
547/547 [==============================] - 58s 106ms/step - total_loss: 657.1764
- reconstruction_loss: 651.6821 - kl_loss: 5.4941
Epoch 19/30
547/547 [==============================] - 58s 106ms/step - total_loss: 655.5926
- reconstruction_loss: 650.1417 - kl_loss: 5.4514
Epoch 20/30
547/547 [==============================] - 58s 106ms/step - total_loss: 654.9575
- reconstruction_loss: 649.5154 - kl_loss: 5.4420
Epoch 21/30
547/547 [==============================] - 58s 107ms/step - total_loss: 654.0459
- reconstruction_loss: 648.6295 - kl_loss: 5.4157
Epoch 22/30
547/547 [==============================] - 59s 108ms/step - total_loss: 652.9374
- reconstruction_loss: 647.5676 - kl_loss: 5.3697
Epoch 23/30
547/547 [==============================] - 58s 106ms/step - total_loss: 651.7272
- reconstruction_loss: 646.3416 - kl_loss: 5.3854
Epoch 24/30
547/547 [==============================] - 58s 106ms/step - total_loss: 651.0931
- reconstruction_loss: 645.7234 - kl_loss: 5.3700
Epoch 25/30
547/547 [==============================] - 58s 107ms/step - total_loss: 650.4158
- reconstruction_loss: 645.0604 - kl_loss: 5.35577s - t
Epoch 26/30
547/547 [==============================] - 58s 106ms/step - total_loss: 649.1877
- reconstruction_loss: 643.8411 - kl_loss: 5.3466
Epoch 27/30
547/547 [==============================] - 59s 107ms/step - total_loss: 648.9291
- reconstruction_loss: 643.6247 - kl_loss: 5.3045
Epoch 28/30
547/547 [==============================] - 58s 107ms/step - total_loss: 648.3905
- reconstruction_loss: 643.1008 - kl_loss: 5.2899
Epoch 29/30
547/547 [==============================] - 58s 106ms/step - total_loss: 647.3626
- reconstruction_loss: 642.0606 - kl_loss: 5.3024
Epoch 30/30
547/547 [==============================] - 58s 107ms/step - total_loss: 646.8358
- reconstruction_loss: 641.5722 - kl_loss: 5.2629
```

[9]: <tensorflow.python.keras.callbacks.History at 0x7f2f6c04ac40>

[10]:
```
# Sampling a grid of images from the 2D latent space
import matplotlib.pyplot as plt

# Display a 30x30 grid of digits (900 total):
n = 15
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))

# Sample points linearly on a 2D grid:
grid_x = np.linspace(-1, 1, n)
grid_y = np.linspace(-1, 1, n)[::-1]

# Iterate over grid locations:
for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        # For each location, sample a digit and add it to the figure:
        z_sample = np.array([[xi, yi]])
        x_decoded = vae.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=(15, 15))
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.axis("off")
plt.imshow(figure, cmap="Greys_r")
```

[10]: <matplotlib.image.AxesImage at 0x7f2f4042e0d0>



```
[ ]:
```