Experiment No. 4
Train a Variational Autoencoder on Fashion MNIST
Date of Performance:
Date of Submission:

Aim: Train a Variational Autoencoder on Fashion MNIST

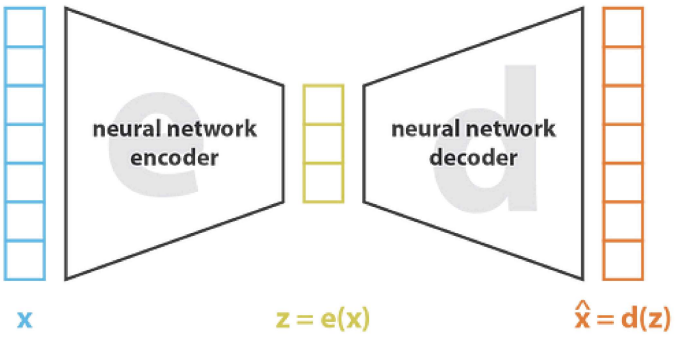
Objective: Ability to implement Variational Autoencoder.

Theory:

Autoencoders:

The general idea of autoencoders is pretty simple and consists in setting an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimisation process. So, at each iteration we feed the autoencoder architecture (the encoder followed by the decoder) with some data, we compare the encoded-decoded output with the initial data and backpropagate the error through the architecture to update the weights of the networks.

Thus, intuitively, the overall autoencoder architecture (encoder+decoder) creates a bottleneck for data that ensures only the main structured part of the information can go through and be reconstructed. Looking at our general framework, the family E of considered encoders is defined by the encoder network architecture, the family D of considered decoders is defined by the decoder network architecture and the search of encoder and decoder that minimise the reconstruction error is done by gradient descent over the parameters of these networks.



$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$

Variational Autoencoders:

A variational autoencoder can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

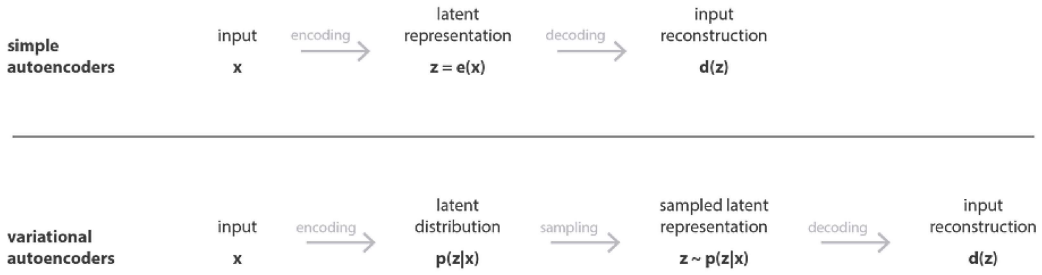
Just as a standard autoencoder, a variational autoencoder is an architecture composed of both an encoder and a decoder and that is trained to minimise the reconstruction error between the encoded-decoded data and the initial data. However, in order to introduce some regularisation of the latent space, we proceed to a slight modification of the encoding-decoding process: instead of encoding an input as a single point, we encode it as a distribution over the latent space. The model is then trained as follows:

first, the input is encoded as distribution over the latent space

second, a point from the latent space is sampled from that distribution

third, the sampled point is decoded and the reconstruction error can be computed

finally, the reconstruction error is backpropagated through the network



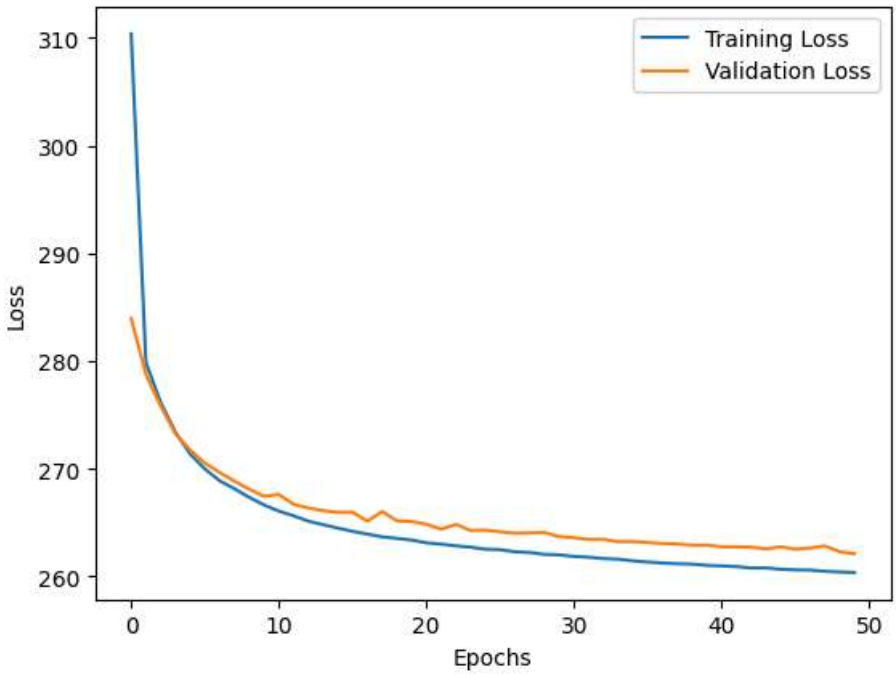
Code:

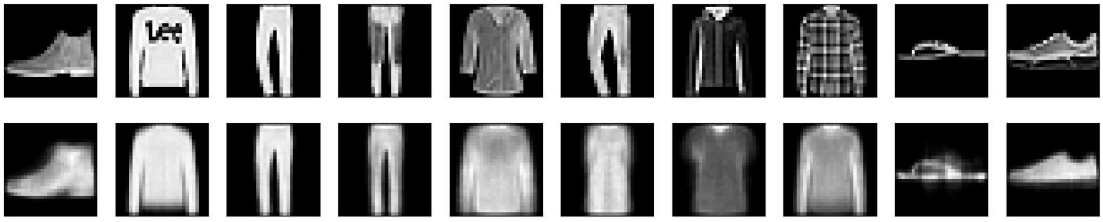
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import fashion_mnist
import tensorflow as tf

# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize pixel values between 0 and 1
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Reshape images to flatten them
```

```
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:])))

# Define VAE architecture
original_dim = 784
intermediate_dim = 256
latent_dim = 2

# Encoder
inputs = keras.Input(shape=(original_dim,))
h = layers.Dense(intermediate_dim, activation='relu')(inputs)
z_mean = layers.Dense(latent_dim)(h)
z_log_var = layers.Dense(latent_dim)(h)

# Sampling function
def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], latent_dim))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])

# Decoder
decoder_h = layers.Dense(intermediate_dim, activation='relu')
decoder_mean = layers.Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)

# VAE model
vae = keras.Model(inputs, x_decoded_mean)

# Define loss
reconstruction_loss = keras.losses.binary_crossentropy(inputs, x_decoded_mean) * original_dim
kl_loss = -0.5 * tf.reduce_sum(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var), axis=-1)
vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)

# Compile VAE model
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')
```

```
# Train VAE model
history = vae.fit(x_train, x_train,
                  epochs=50,
                  batch_size=128,
                  validation_data=(x_test, x_test))

# Plot loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Generate new images
decoded_imgs = vae.predict(x_test)

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

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

Output:





Conclusion:

The accuracy of a VAE is typically assessed qualitatively by inspecting the quality of the reconstructed images. In this case, we can observe how well the VAE is able to reconstruct the input images from the Fashion MNIST dataset. Higher quality reconstructions indicate better performance.Overall, the VAE network architecture is relatively simple, with a single hidden layer in both the encoder and decoder. The choice of the latent dimension (2 in this case) influences the compactness of the latent representation and the quality of the generated images. Increasing the latent dimension may lead to more expressive latent representations but may also require a more complex decoder network.