ASSIGNEMENT 5 : VAE USING MNIST DATASET

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

```
Start coding or generate with AI.
```

```
# Load data
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype("float32") / 255.
x_test = x_test.astype("float32") / 255.
x_train = np.reshape(x_train, (-1, 784))
x_test = np.reshape(x_test, (-1, 784))

latent_dim = 2
```

```
# Encoder
encoder_inputs = tf.keras.Input(shape=(784,))
x = layers.Dense(256, activation="relu")(encoder_inputs)
z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)

def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=tf.shape(z_mean))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])
encoder = tf.keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
```

```
# Decoder
latent_inputs = tf.keras.Input(shape=(latent_dim,))
x = layers.Dense(256, activation="relu")(latent_inputs)
decoder_outputs = layers.Dense(784, activation="sigmoid")(x)
decoder = tf.keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

```
# VAE Model with custom training
class VAE(tf.keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            recon_loss = tf.keras.losses.binary_crossentropy(data, reconstruction)
            recon_loss = tf.reduce_sum(recon_loss, axis=-1)  # now shape: (batch,)
            kl_loss = -0.5 * tf.reduce_sum(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var), axis=1)
            total_loss = tf.reduce_mean(recon_loss + kl_loss)
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        return {"loss": total_loss}

vae = VAE(encoder, decoder)
vae.compile(optimizer="adam")
vae.fit(x_train, x_train, epochs=30, batch_size=128)
```

Show hidden output

```
def generate_digits(decoder, latent_dim=2, n=15):
    digit_size = 28
    grid = np.zeros((digit_size * n, digit_size * n))
    for i, yi in enumerate(np.linspace(-2, 2, n)):
        for j, xi in enumerate(np.linspace(-2, 2, n)):
            z_sample = np.array([[xi, yi]])
            x_decoded = decoder.predict(z_sample)
            digit = x_decoded[0].reshape(digit_size, digit_size)
```

```
                grid[i * digit_size:(i + 1) * digit_size,
                     j * digit_size:(j + 1) * digit_size] = digit
        plt.figure(figsize=(10, 10))
        plt.imshow(grid, cmap="gray")
        plt.axis("off")
        plt.show()

generate_digits(decoder)
```

Show hidden output