Natural Language Processing

Assignment-10

Name: Adithi
No:2203A54032

1.Generative models are a class of machine learning models that generate new data instances that resemble your training data. These models can be used for various tasks, such as creating realistic images, text, or even music. In this lab assignment, we'll focus on implementing and training generative models, particularly a Variational Autoencoder (VAE) and a Generative Adversarial Network (GAN), using a simple dataset like MNIST or a text dataset for generating synthetic samples. [CO5]

(i) Implement Variational Autoencoder (VAE)

1.**Import Libraries**:

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

2.**Load the Dataset**:

```
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

3.**Define the VAE Model**:

```python
latent_dim = 2

# Encoder
encoder_inputs = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation='relu', padding='same')(encoder_inputs)
x = layers.MaxPooling2D()(x)
x = layers.Conv2D(64, 3, activation='relu', padding='same')(x)
x = layers.MaxPooling2D()(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation='relu')(x)

# Latent Space
z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)

# Sampling Function
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])

# Decoder
decoder_inputs = layers.Input(shape=(latent_dim,))
x = layers.Dense(7 * 7 * 64, activation='relu')(decoder_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, activation='relu', padding='same')(x)
x = layers.UpSampling2D()(x)
```

```python
decoder_outputs = layers.Conv2DTranspose(1, 3, activation='sigmoid', padding='same')(x)

# VAE Model
vae = models.Model(encoder_inputs, decoder_outputs)
```

4.**Define the Loss Function**:

```python
def vae_loss(y_true, y_pred):
    reconstruction_loss = tf.keras.losses.binary_crossentropy(y_true, y_pred)
    reconstruction_loss *= 28 * 28
    kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
    kl_loss = tf.reduce_mean(kl_loss) * -0.5
    return tf.reduce_mean(reconstruction_loss + kl_loss)

vae.compile(optimizer='adam', loss=vae_loss)
```

(ii) Implement the GAN

GAN Implementation Steps

1.Define the GAN Model:

```
# Generator Model
generator = models.Sequential([
    layers.Dense(128, activation='relu', input_dim=latent_dim),
    layers.Dense(784, activation='sigmoid'),
    layers.Reshape((28, 28, 1))
])

# Discriminator Model
discriminator = models.Sequential([
    layers.Flatten(input_shape=(28, 28, 1)),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

discriminator.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# GAN Model
discriminator.trainable = False
gan_input = layers.Input(shape=(latent_dim,))
generated_image = generator(gan_input)
gan_output = discriminator(generated_image)
gan = models.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pref
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential model
  super().__init__(**kwargs)
```

2.Train the GAN:

```
def train_gan(epochs, batch_size):
    for epoch in range(epochs):
        # Generate random noise
        noise = np.random.normal(0, 1, size=[batch_size, latent_dim])
        generated_images = generator.predict(noise)

        # Create labels
        real_labels = np.ones(batch_size)
        fake_labels = np.zeros(batch_size)

        # Train Discriminator
        d_loss_real = discriminator.train_on_batch(x_train[np.random.randint(0, x_train.shape[0], size=batch_size)], real_labels)
        d_loss_fake = discriminator.train_on_batch(generated_images, fake_labels)

        # Train Generator
        noise = np.random.normal(0, 1, size=[batch_size, latent_dim])
        g_loss = gan.train_on_batch(noise, real_labels)

        # Print the progress
        if epoch % 100 == 0:
            print(f"Epoch: {epoch}, D Loss: {d_loss_real[0] + d_loss_fake[0]}, G Loss: {g_loss}")

train_gan(epochs=10000, batch_size=128)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/trainer.py:75: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
Epoch: 0, D Loss: 1.5629907846450806, G Loss: [array(0.81829923, dtype=float32), array(0.81829923, dtype=float32), array(0.19140625, dtype=float32)]
```

(iii) Visualize the latent space and generated images to understand how well the model captures the data distribution.

Visualize Generated Images:

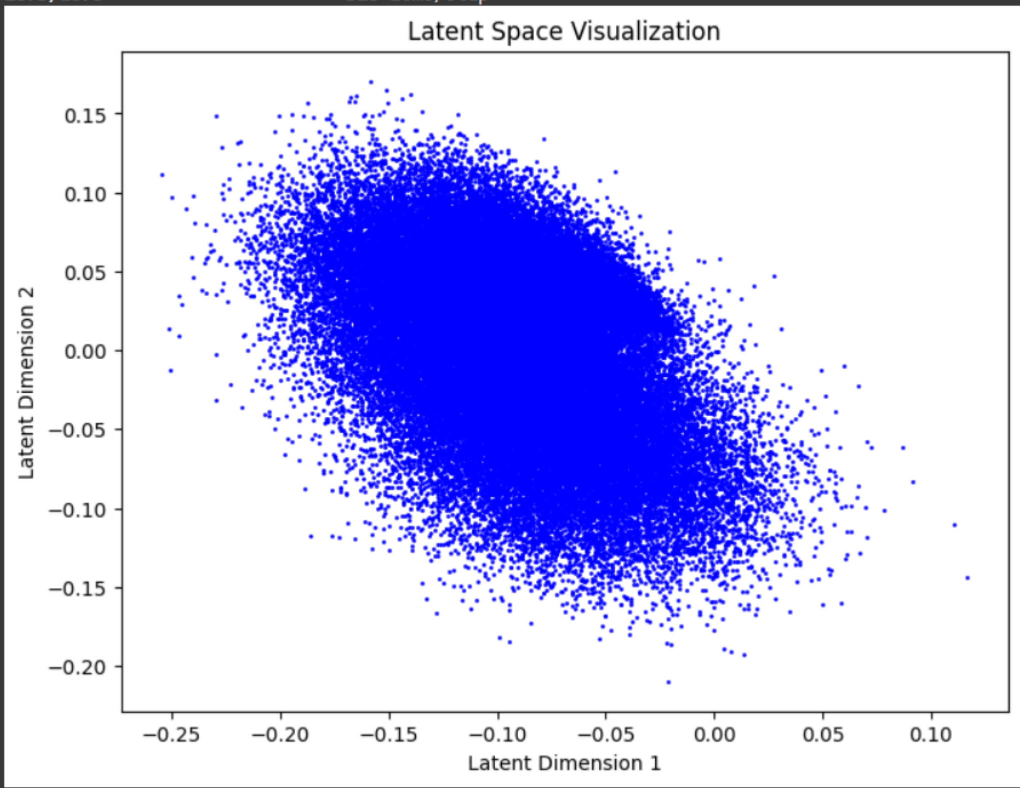
For VAE:

```
encoder = models.Model(encoder_inputs, z_mean)

z_mean_train = encoder.predict(x_train)

plt.figure(figsize=(8, 6))
plt.scatter(z_mean_train[:, 0], z_mean_train[:, 1], c='blue', s=1)
plt.xlabel("Latent Dimension 1")
plt.ylabel("Latent Dimension 2")
plt.title("Latent Space Visualization")
plt.show()
```

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 16ms/step



For GAN:

```python
generated_images = generator.predict(np.random.normal(0, 1, size=(15, latent_dim)))

plt.figure(figsize=(10, 10))
for i in range(15):
    ax = plt.subplot(5, 5, i + 1)
    plt.imshow(generated_images[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step