Lab 2: Advanced Generative Adversarial Networks (GANs)

Course: Generative AI & LLM
Level: M2 (Master's 2)
Duration: 4 hours

Objective

In this lab, you will:

1. Implement a **CNN-based GAN** for generating realistic images.
2. Explore the use of **Transformer-based architectures** in GANs for generative modeling.
3. Compare the performance and visual results of different GAN architectures.
4. Reflect on the strengths and challenges of using CNNs vs. Transformers in GANs.

Background

Variational Autoencoders (VAEs)

A **VAE** is a generative model that reconstructs input data by encoding it into a latent space and then decoding it back. The latent space is a compressed representation that captures the most important features of the input.
The key elements of a VAE:

- **Encoder:** Compresses input data into a latent distribution (mean and variance).
- **Latent Space:** The encoded representation space.
- **Reparameterization Trick:** Allows gradients to flow through the stochastic latent space.
- **Decoder:** Reconstructs the input from the latent space.

Generative Adversarial Networks (GANs)

A **GAN** is a generative model consisting of two components:

- **Generator:** Produces fake data from random noise (latent space).
- **Discriminator:** Distinguishes between real and fake data.

GANs are trained using an adversarial process where the generator tries to fool the discriminator, and the discriminator tries to detect fake data.
Reversing VAEs into GANs
In a conceptual sense:

- A **VAE decoder** can be seen as a **GAN generator**.
- The **GAN discriminator** replaces the **VAE encoder** by determining the quality of generated samples instead of encoding input data.

CNN-based GANs

Convolutional Neural Networks (CNNs) are effective in image processing tasks. In GANs:

- **Generator:** Uses transpose convolutions to upsample noise into images.
- **Discriminator:** Uses convolutions to classify images as real or fake.

Transformer-based GANs

Transformers, traditionally used for NLP, have been adapted for vision tasks (e.g., Vision Transformers). They model global dependencies between pixels:

- **Generator:** Uses multi-head self-attention layers to synthesize images.
- **Discriminator:** Uses similar attention mechanisms to evaluate the generated images.

Part 1: CNN-based GAN

Instructions

1. Implement a **CNN Generator:**
   - Transpose convolution layers to upsample noise into an image.
   - Use ReLU activation for hidden layers and Tanh for the output layer.
2. Implement a **CNN Discriminator**:
   - Convolution layers to downsample the input.
   - Use LeakyReLU for hidden layers and Sigmoid for the output layer.
3. Train the GAN on the MNIST dataset and generate images.

Questions

1. What is Transpose Convolution and why do we use it Generator**?**
2. What are LeakyReLU and sigmoid and why do we use them?
3. Read the code and comment every line.

Part 2: Transformer-based GAN

Instructions

1. Implement a **Transformer Generator**:
   - Use Multi-Head Self-Attention (MHSA) and positional encodings.
   - Upsample the latent space using feedforward layers.
2. Implement a **Transformer Discriminator**:
   - Use MHSA to analyze global relationships in the input.
   - Classify real vs. fake images.
3. Train the Transformer-based GAN on MNIST and compare its performance to the CNN-based GAN.

Deliverables

1. Code Implementation for both GAN architectures (CNN and Transformer-based).
2. Generated Images from both models.
3. Answers to the Questions, comparing CNNs and Transformers in GANs.

```
# For pedagogic purpose, the code is sequential here, it should be of form : function definitions and archtitecture in different files.

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

# Load MNIST dataset
(x_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)

latent_dim = 100

# CNN Generator
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(7 * 7 * 256, input_dim=latent_dim),
        layers.Reshape((7, 7, 256)),
        layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding="same", activation="relu"),
        layers.Conv2DTranspose(64, kernel_size=4, strides=2, padding="same", activation="relu"),
        layers.Conv2DTranspose(1, kernel_size=7, activation="tanh", padding="same")
    ])
    return model

# CNN Discriminator
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, kernel_size=4, strides=2, padding="same", input_shape=(28, 28, 1)),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid")
    ])
    return model

# GAN Model
generator = build_generator()
discriminator = build_discriminator()
```

```
discriminator.compile(optimizer=tf.keras.optimizers.Adam(0.0002),
loss="binary_crossentropy", metrics=["accuracy"])
discriminator.trainable = False

gan_input = layers.Input(shape=(latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer=tf.keras.optimizers.Adam(0.0002), loss="binary_crossentropy")

# Training the GAN
def train_gan(generator, discriminator, gan, epochs, batch_size=128):
    for epoch in range(epochs):
        for _ in range(batch_size):
            # Train discriminator
            noise = tf.random.normal([batch_size, latent_dim])
            fake_images = generator.predict(noise)
            real_images = x_train[np.random.randint(0, x_train.shape[0], batch_size)]

            real_labels = tf.ones((batch_size, 1))
            fake_labels = tf.zeros((batch_size, 1))

            d_loss_real = discriminator.train_on_batch(real_images, real_labels)
            d_loss_fake = discriminator.train_on_batch(fake_images, fake_labels)

            # Train generator
            misleading_labels = tf.ones((batch_size, 1))
            g_loss = gan.train_on_batch(noise, misleading_labels)

        print(f"Epoch {epoch + 1}/{epochs}, D Loss: {d_loss_real[0] + d_loss_fake[0]}, G Loss:
{g_loss}")

train_gan(generator, discriminator, gan, epochs=50)

# Generate and visualize images
def generate_images(generator, n_images):
    noise = tf.random.normal([n_images, latent_dim])
    generated_images = generator.predict(noise)
    fig, axes = plt.subplots(1, n_images, figsize=(20, 4))
    for i, img in enumerate(generated_images):
        axes[i].imshow(img.squeeze(), cmap="gray")
        axes[i].axis("off")
    plt.show()

generate_images(generator, 10)
```

1. Transformer Generator

The Generator takes random noise as input and uses Multi-Head Self-Attention (MHSA) and positional encodings to generate images.

```
from tensorflow.keras.layers import Input, Dense, Reshape, Flatten, Dropout
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization
from tensorflow.keras.models import Model
import tensorflow as tf

latent_dim = 100

def transformer_generator(latent_dim):
    inputs = Input(shape=(latent_dim,))

    # Initial dense layer to project noise into a higher-dimensional space
    x = Dense(7 * 7 * 128, activation="relu")(inputs)
    x = Reshape((49, 128))(x)  # Reshape to (7x7 patches, 128 features)

    # Positional Encoding
    position_encoding = tf.range(start=0, limit=49, delta=1)
    position_embedding = tf.keras.layers.Embedding(input_dim=49,
output_dim=128)(position_encoding)
    x += position_embedding

    # Multi-Head Self-Attention
    x = MultiHeadAttention(num_heads=4, key_dim=128)(x, x)
    x = LayerNormalization()(x)

    # Feedforward network
    x = Dense(128, activation="relu")(x)
    x = LayerNormalization()(x)

    # Reshape and upsample to image dimensions
    x = Reshape((7, 7, 128))(x)
    x = layers.Conv2DTranspose(64, kernel_size=4, strides=2, padding="same",
activation="relu")(x)
    x = layers.Conv2DTranspose(1, kernel_size=4, strides=2, padding="same",
activation="tanh")(x)

    model = Model(inputs, x, name="Transformer_Generator")
    return model

generator = transformer_generator(latent_dim)
generator.summary()
```

2. Transformer Discriminator

The Discriminator uses MHSA to analyze global relationships in the input image and classify it as real or fake.

```
def transformer_discriminator():
    inputs = Input(shape=(28, 28, 1))
    x = Flatten()(inputs)  # Flatten the image into a sequence
    x = Dense(128, activation="relu")(x)
    x = Reshape((49, 128))(x)  # Reshape into (49 patches, 128 features)

    # Positional Encoding
    position_encoding = tf.range(start=0, limit=49, delta=1)
    position_embedding = tf.keras.layers.Embedding(input_dim=49,
output_dim=128)(position_encoding)
    x += position_embedding

    # Multi-Head Self-Attention
    x = MultiHeadAttention(num_heads=4, key_dim=128)(x, x)
    x = LayerNormalization()(x)

    # Classification layer
    x = Flatten()(x)
    x = Dense(128, activation="relu")(x)
    x = Dense(1, activation="sigmoid")(x)

    model = Model(inputs, x, name="Transformer_Discriminator")
    return model

discriminator = transformer_discriminator()
discriminator.summary()
```

3. GAN Model and Training

The GAN combines the Transformer Generator and Discriminator, with adversarial training to improve both.

TO DO: Implement the Gan model using both Generator and Discriminator.

Hint : get inspired from the solution given for CNN-based GAN.