Lab Sheet 9: Generative Adversarial Networks (GANs)

9.1 Objective

The aim of this lab exercise is to provide an understanding of Generative Adversarial Networks (GANs) and their application in image generation.

9.2 Problem Statement

You are required to implement a GAN using the MNIST dataset, which is a public domain dataset of handwritten digits. The MNIST dataset consists of 70,000 grayscale images of handwritten digits, each of size 28x28 pixels.

9.3 Tasks

GAN Architecture: Implement a GAN. The GAN should consist of a Generator and a Discriminator. The Generator should take a random noise vector as input and output an image. The Discriminator should take an image as input and output a probability indicating whether the image is real (from the MNIST dataset) or fake (generated by the Generator).

Training: Train the GAN on the MNIST dataset. The Generator should improve at generating realistic images of handwritten digits, while the Discriminator should improve at distinguishing between real and fake images.

Evaluation: Evaluate the GAN. This can be done qualitatively by visualizing the images generated by the GAN. You may also consider implementing quantitative metrics such as the Inception Score or the Frechet Inception Distance.

9.4 Implementation

Complete the portions marked as **#todo.** Then submit your jupyter notebook.

imports

```
import torch
from torch import nn
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
```

Define the Generator

```
class Generator(nn.Module):
    def __init__(self, z_dim=10, img_dim=784):
        super().__init__()
        self.gen = nn.Sequential(
            nn.Linear(z_dim, 256),
            nn.ReLU(),
            nn.Linear(256, img_dim),
            nn.Tanh(),
        )

    def forward(self, x):
        #todo: return the gen object
```

Define the Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, img_dim=784):
        super().__init__()
        self.disc = nn.Sequential(
            nn.Linear(img_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        #todo: return disc object
```

Hyperparameters

```
lr = #todo
z_dim = 64
image_dim = 28 * 28  # 784
batch_size = #todo
num_epochs = #todo
```

Initialize the Generator and the Discriminator

```
gen = #todo
disc = #todo
```

Load MNIST dataset

```
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
dataloader = DataLoader(datasets.MNIST('.', download=True,
transform=transform), batch_size=batch_size, shuffle=True)
```

Loss function and optimizers

```
criterion = nn.BCELoss()
opt_gen = torch.optim.Adam(gen.parameters(), lr=lr)
opt_disc = torch.optim.Adam(disc.parameters(), lr=lr)
```

Training loop

```
for epoch in range(num_epochs):
    print(epoch)
    for batch_idx, (real, _) in enumerate(dataloader):
        real = real.view(-1, 784)
        batch_size = real.shape[0]

        # Train Discriminator: max log(D(x)) + log(1 - D(G(z)))
        noise = torch.randn(batch_size, z_dim)
        fake = gen(noise)
        disc_real = disc(real).view(-1)
        lossD_real = criterion(disc_real,
torch.ones_like(disc_real))
        disc_fake = disc(fake).view(-1)
        lossD_fake = criterion(disc_fake,
torch.zeros_like(disc_fake))
        lossD = (lossD_real + lossD_fake) / 2
        disc.zero_grad()
        lossD.backward(retain_graph=True)
        opt_disc.step()

        # Train Generator: min log(1 - D(G(z))) (equivalent to)
max log(D(G(z))
        output = disc(fake).view(-1)
        lossG = criterion(output, torch.ones_like(output))
        gen.zero_grad()
        lossG.backward()
        opt_gen.step()
```

```
      # Visualize the generated images after each epoch
    with torch.no_grad():
        fake = gen(torch.randn(batch_size, z_dim)).view(-1, 28,
28).detach().cpu().numpy()
    fig, ax = plt.subplots(1, 5)
    for i in range(5):
        ax[i].imshow(fake[i], cmap='gray')
        ax[i].axis('off')
    plt.show()

    print(f"Epoch [{epoch}/{num_epochs}] Loss D: {lossD:.4f},
loss G: {lossG:.4f}")
```

Additional Exercise

Since you will be working for the GAN based assignment already, we implemented the training loop for you. You just need to comment about this implementation here, and what it is that we are plotting. Also comment about what you observed about this GAN training process. (Please don't use ChatGPT etc. to write your answers, it's very obvious when you do. 🙏)