4. Implement a simple Generative Adversarial Network (GAN) to generate synthetic images similar to MNIST digits. Learn how to build Generator and Discriminator, and train them adversarially.

GAN to Generate MNIST Handwritten Digits

□ What is a GAN?

A GAN has **two neural networks**:

Network	Role
Generator (G)	Creates fake images
Discriminator (D)	Distinguishes real vs fake images

They are trained **against each other** like a game.

□ Workflow

```
Noise → Generator → Fake Images → Discriminator → Real/Fake
Real Images → Discriminator → Real/Fake
```

Generator improves to fool Discriminator
Discriminator improves to catch Generator

Step 1: Import Libraries

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

Step 2: Load MNIST Dataset

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_data = datasets.MNIST(root="./data", train=True, transform=transform,
download=True)
dataloader = DataLoader(train_data, batch_size=128, shuffle=True)
```

Step 3: Build Generator

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 28*28),
            nn.Tanh()
        )

    def forward(self, z):
        return self.model(z).view(-1,1,28,28)
```

Step 4: Build Discriminator

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x.view(-1, 28*28))
```

Step 5: Initialize Models

```
G = Generator()
D = Discriminator()

criterion = nn.BCELoss()
```

```
g_optimizer = optim.Adam(G.parameters(), lr=0.0002)
d_optimizer = optim.Adam(D.parameters(), lr=0.0002)
```

Step 6: GAN Training Loop

```
epochs = 50

for epoch in range(epochs):
    for real_images, _ in dataloader:

        # Train Discriminator
        real_labels = torch.ones(real_images.size(0),1)
        fake_labels = torch.zeros(real_images.size(0),1)

        outputs = D(real_images)
        d_loss_real = criterion(outputs, real_labels)

        z = torch.randn(real_images.size(0), 100)
        fake_images = G(z)
        outputs = D(fake_images.detach())
        d_loss_fake = criterion(outputs, fake_labels)

        d_loss = d_loss_real + d_loss_fake
        d_optimizer.zero_grad()
        d_loss.backward()
        d_optimizer.step()

        # Train Generator
        z = torch.randn(real_images.size(0), 100)
        fake_images = G(z)
        outputs = D(fake_images)
        g_loss = criterion(outputs, real_labels)

        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()

    print(f"Epoch [{epoch+1}/{epochs}]  D Loss: {d_loss.item():.4f}  G
Loss: {g_loss.item():.4f}")
```

Step 7: Generate New Digits

```
z = torch.randn(16, 100)
fake_images = G(z).detach()

plt.figure(figsize=(6,6))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(fake_images[i][0], cmap='gray')
    plt.axis('off')
plt.show()
```
