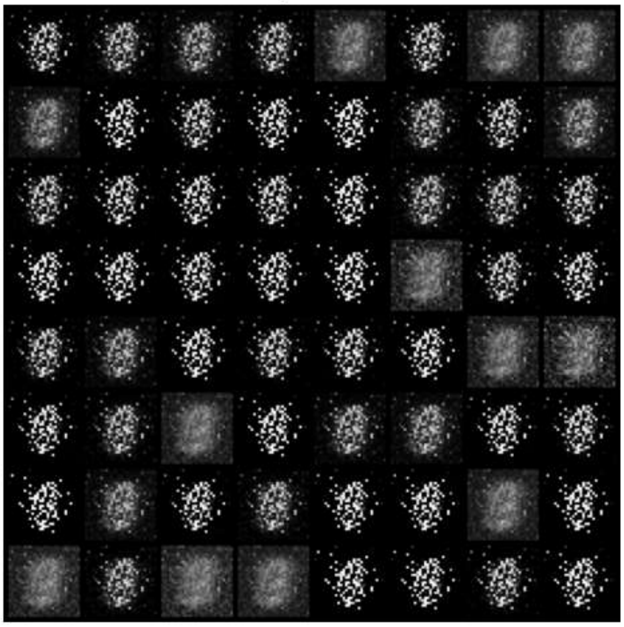
```
# ✅ 1. Install & import required libraries
!pip install torchvision --quiet

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torchvision.utils import save_image, make_grid
import matplotlib.pyplot as plt
import os

# ✅ 2. Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
bs = 100  # Batch size

# ✅ 3. MNIST dataset & DataLoader
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='./mnist_data', train=True, transform=transform, download=True)
train_loader = DataLoader(dataset=train_dataset, batch_size=bs, shuffle=True)

# ✅ 4. Generator model
class Generator(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(input_dim, 256)
        self.fc2 = nn.Linear(256, 512)
        self.fc3 = nn.Linear(512, 1024)
        self.fc4 = nn.Linear(1024, output_dim)

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        return torch.tanh(self.fc4(x))

# ✅ 5. Discriminator model
class Discriminator(nn.Module):
    def __init__(self, input_dim):
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(input_dim, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 256)
        self.fc4 = nn.Linear(256, 1)

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        return torch.sigmoid(self.fc4(x))

# ✅ 6. Initialize models
z_dim = 100
mnist_dim = 28 * 28

G = Generator(z_dim, mnist_dim).to(device)
D = Discriminator(mnist_dim).to(device)

# ✅ 7. Loss and Optimizers
criterion = nn.BCELoss()
lr = 0.0002
G_optimizer = optim.Adam(G.parameters(), lr=lr)
D_optimizer = optim.Adam(D.parameters(), lr=lr)

# ✅ 8. Training functions
def D_train(x):
    D.zero_grad()
    x_real, y_real = x.view(-1, mnist_dim).to(device), torch.ones(bs, 1).to(device)
    D_output = D(x_real)
    D_real_loss = criterion(D_output, y_real)

    z = torch.randn(bs, z_dim).to(device)
    x_fake = G(z)
```

```
        y_fake = torch.zeros(bs, 1).to(device)
        D_output_fake = D(x_fake)
        D_fake_loss = criterion(D_output_fake, y_fake)

        D_loss = D_real_loss + D_fake_loss
        D_loss.backward()
        D_optimizer.step()
        return D_loss.item()

    def G_train():
        G.zero_grad()
        z = torch.randn(bs, z_dim).to(device)
        y = torch.ones(bs, 1).to(device)
        G_output = G(z)
        D_output = D(G_output)
        G_loss = criterion(D_output, y)
        G_loss.backward()
        G_optimizer.step()
        return G_loss.item()

    # ✅ 9. Sample display function
    def show_generated_images(generator, epoch):
        generator.eval()
        with torch.no_grad():
            z = torch.randn(64, z_dim).to(device)
            generated = generator(z).view(-1, 1, 28, 28)
            grid = make_grid(generated, nrow=8, normalize=True)
            plt.figure(figsize=(6,6))
            plt.title(f"Epoch {epoch}")
            plt.imshow(grid.permute(1, 2, 0).cpu())
            plt.axis('off')
            plt.show()
        generator.train()

    # ✅ 10. Training loop
    n_epochs = 50
    os.makedirs('./samples', exist_ok=True)

    for epoch in range(1, n_epochs + 1):
        D_losses, G_losses = [], []
        for batch_idx, (real_batch, _) in enumerate(train_loader):
            D_loss = D_train(real_batch)
            G_loss = G_train()
            D_losses.append(D_loss)
            G_losses.append(G_loss)

        print(f"Epoch [{epoch}/{n_epochs}], D_loss: {torch.tensor(D_losses).mean():.4f}, G_loss: {torch.tensor(G_losses).mean():.4f}")

        # Save and show sample images
        if epoch % 10 == 0 or epoch == 1:
            with torch.no_grad():
                z = torch.randn(bs, z_dim).to(device)
                generated = G(z).view(bs, 1, 28, 28)
                save_image(generated, f'./samples/sample_epoch_{epoch}.png', nrow=10, normalize=True)
            show_generated_images(G, epoch)
```

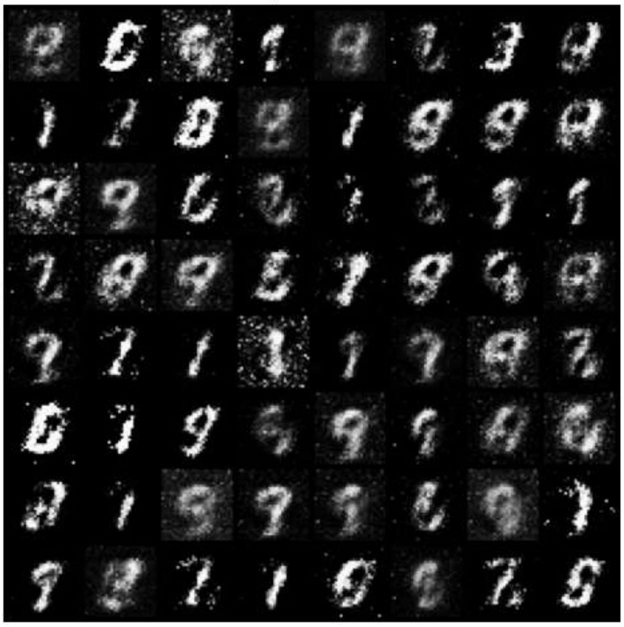
```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 54.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 38.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 28.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 869.8 kB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 12.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 35.9 MB/s eta 0:00:00
100%|██████████| 9.91M/9.91M [00:00<00:00, 16.4MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 494kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.60MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.95MB/s]
Epoch [1/50], D_loss: 0.7537, G_loss: 3.9932
```

Epoch 1



```
Epoch [2/50], D_loss: 0.7336, G_loss: 3.8226
Epoch [3/50], D_loss: 0.8792, G_loss: 2.3457
Epoch [4/50], D_loss: 0.8286, G_loss: 2.3013
Epoch [5/50], D_loss: 0.5400, G_loss: 2.7459
Epoch [6/50], D_loss: 0.4163, G_loss: 3.1530
Epoch [7/50], D_loss: 0.4308, G_loss: 3.2661
Epoch [8/50], D_loss: 0.4925, G_loss: 2.9714
Epoch [9/50], D_loss: 0.5021, G_loss: 3.0034
Epoch [10/50], D_loss: 0.5644, G_loss: 2.6605
```

Epoch 10



```
Epoch [11/50], D_loss: 0.6296, G_loss: 2.5011
Epoch [12/50], D_loss: 0.6759, G_loss: 2.4019
Epoch [13/50], D_loss: 0.7046, G_loss: 2.0770
Epoch [14/50], D_loss: 0.6907, G_loss: 2.2474
Epoch [15/50], D_loss: 0.7289, G_loss: 2.0638
Epoch [16/50], D_loss: 0.7742, G_loss: 1.9427
Epoch [17/50], D_loss: 0.7924, G_loss: 1.9267
Epoch [18/50], D_loss: 0.8210, G_loss: 1.7987
```

```
Epoch [19/50], D_loss: 0.8378, G_loss: 1.7860
Epoch [20/50], D_loss: 0.8689, G_loss: 1.7109
```

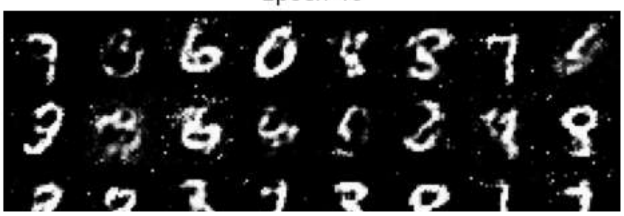
Epoch 20

```
Epoch [21/50], D_loss: 0.8946, G_loss: 1.6408
Epoch [22/50], D_loss: 0.9287, G_loss: 1.5538
Epoch [23/50], D_loss: 0.9281, G_loss: 1.5349
Epoch [24/50], D_loss: 0.9369, G_loss: 1.5615
Epoch [25/50], D_loss: 0.9237, G_loss: 1.5596
Epoch [26/50], D_loss: 0.9467, G_loss: 1.5349
Epoch [27/50], D_loss: 0.9561, G_loss: 1.5228
Epoch [28/50], D_loss: 0.9841, G_loss: 1.4483
Epoch [29/50], D_loss: 0.9588, G_loss: 1.4941
Epoch [30/50], D_loss: 0.9793, G_loss: 1.4559
```

Epoch 30

```
Epoch [31/50], D_loss: 0.9903, G_loss: 1.4499
Epoch [32/50], D_loss: 1.0021, G_loss: 1.3952
Epoch [33/50], D_loss: 1.0073, G_loss: 1.3921
Epoch [34/50], D_loss: 1.0490, G_loss: 1.3137
Epoch [35/50], D_loss: 1.0498, G_loss: 1.3209
Epoch [36/50], D_loss: 1.0521, G_loss: 1.2966
Epoch [37/50], D_loss: 1.0643, G_loss: 1.2803
Epoch [38/50], D_loss: 1.0780, G_loss: 1.2553
Epoch [39/50], D_loss: 1.0791, G_loss: 1.2452
Epoch [40/50], D_loss: 1.0941, G_loss: 1.2266
```

Epoch 40



```
Epoch [41/50], D_loss: 1.0933, G_loss: 1.2021
Epoch [42/50], D_loss: 1.1010, G_loss: 1.1871
Epoch [43/50], D_loss: 1.1061, G_loss: 1.1934
Epoch [44/50], D_loss: 1.1332, G_loss: 1.1396
Epoch [45/50], D_loss: 1.1224, G_loss: 1.1493
Epoch [46/50], D_loss: 1.1333, G_loss: 1.1448
Epoch [47/50], D_loss: 1.1492, G_loss: 1.1204
Epoch [48/50], D_loss: 1.1574, G_loss: 1.0874
Epoch [49/50], D_loss: 1.1714, G_loss: 1.0741
Epoch [50/50], D_loss: 1.1737, G_loss: 1.0781
```

Epoch 50