```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
import os
```



```
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


# Hyperparameters
latent_dim = 100
batch_size = 64
epochs = 10
lr = 0.0002
image_dir = "gan_images"




# Create image directory
os.makedirs(image_dir, exist_ok=True)


# Data loader
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])  # Scale images to [-1, 1]
])


dataloader = torch.utils.data.DataLoader(
    datasets.MNIST(root='./data', train=True, download=True, transform=transform),
    batch_size=batch_size,
    shuffle=True
)




# Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 28 * 28),
            nn.Tanh()
```

```
            )

        def forward(self, z):
            img = self.model(z)
            return img.view(z.size(0), 1, 28, 28)
```



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

        def forward(self, img):
            flat = img.view(img.size(0), -1)
            return self.model(flat)



    # Initialize models
    generator = Generator().to(device)
    discriminator = Discriminator().to(device)



    # Loss and optimizers
    adversarial_loss = nn.BCELoss()
    optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))
    optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))



    # Training
    for epoch in range(epochs):
        for i, (imgs, _) in enumerate(dataloader):
            real_imgs = imgs.to(device)
            valid = torch.ones(imgs.size(0), 1, device=device)
            fake = torch.zeros(imgs.size(0), 1, device=device)

            # -----------------
            #  Train Generator
            # -----------------
            optimizer_G.zero_grad()
            z = torch.randn(imgs.size(0), latent_dim, device=device)
            gen_imgs = generator(z)
```

```
            g_loss = adversarial_loss(discriminator(gen_imgs), valid)
            g_loss.backward()
            optimizer_G.step()

            # ---------------------
            #  Train Discriminator
            # ---------------------
            optimizer_D.zero_grad()
            real_loss = adversarial_loss(discriminator(real_imgs), v
            fake_loss = adversarial_loss(discriminator(gen_imgs.deta
            d_loss = (real_loss + fake_loss) / 2
            d_loss.backward()
            optimizer_D.step()

            # Logging
            if i % 200 == 0:
                print(f"[Epoch {epoch}/{epochs}] [Batch {i}/{len(dataloader)}] "
                      f"[D loss: {d_loss.item():.4f}] [G loss: {g_loss.item():.4f}]")

        # Save generated images
        save_image(gen_imgs.data[:25], f"{image_dir}/{epoch:03d}.png", nrow=5, normalize=True)
    print("Training complete.")
```



```
[Epoch 0/10] [Batch 0/938] [D loss: 0.6758] [G loss: 0.7055]
[Epoch 0/10] [Batch 200/938] [D loss: 0.5246] [G loss: 0.9231]
[Epoch 0/10] [Batch 400/938] [D loss: 0.3715] [G loss: 1.6971]
[Epoch 0/10] [Batch 600/938] [D loss: 0.2356] [G loss: 1.7291]
[Epoch 0/10] [Batch 800/938] [D loss: 0.3525] [G loss: 0.9278]
[Epoch 1/10] [Batch 0/938] [D loss: 0.2339] [G loss: 1.0142]
[Epoch 1/10] [Batch 200/938] [D loss: 0.1229] [G loss: 3.4516]
[Epoch 1/10] [Batch 400/938] [D loss: 0.2676] [G loss: 1.3635]
[Epoch 1/10] [Batch 600/938] [D loss: 0.1214] [G loss: 1.7078]
[Epoch 1/10] [Batch 800/938] [D loss: 0.2959] [G loss: 0.9046]
[Epoch 2/10] [Batch 0/938] [D loss: 0.0538] [G loss: 3.5827]
[Epoch 2/10] [Batch 200/938] [D loss: 0.1134] [G loss: 1.9762]
[Epoch 2/10] [Batch 400/938] [D loss: 0.1443] [G loss: 4.3082]
[Epoch 2/10] [Batch 600/938] [D loss: 0.0858] [G loss: 3.5316]
[Epoch 2/10] [Batch 800/938] [D loss: 0.2866] [G loss: 4.8151]
[Epoch 3/10] [Batch 0/938] [D loss: 0.0336] [G loss: 2.7980]
[Epoch 3/10] [Batch 200/938] [D loss: 0.1631] [G loss: 3.1269]
[Epoch 3/10] [Batch 400/938] [D loss: 0.0846] [G loss: 4.1553]
[Epoch 3/10] [Batch 600/938] [D loss: 0.0763] [G loss: 2.2415]
[Epoch 3/10] [Batch 800/938] [D loss: 0.0465] [G loss: 3.5847]
[Epoch 4/10] [Batch 0/938] [D loss: 0.1011] [G loss: 2.8307]
[Epoch 4/10] [Batch 200/938] [D loss: 0.2182] [G loss: 1.3057]
[Epoch 4/10] [Batch 400/938] [D loss: 0.0677] [G loss: 3.5020]
[Epoch 4/10] [Batch 600/938] [D loss: 0.2145] [G loss: 5.2377]
[Epoch 4/10] [Batch 800/938] [D loss: 0.1208] [G loss: 9.0919]
[Epoch 5/10] [Batch 0/938] [D loss: 0.0308] [G loss: 3.1472]
[Epoch 5/10] [Batch 200/938] [D loss: 0.0615] [G loss: 3.7242]
[Epoch 5/10] [Batch 400/938] [D loss: 0.0200] [G loss: 8.1940]
[Epoch 5/10] [Batch 600/938] [D loss: 0.3862] [G loss: 11.7745]
[Epoch 5/10] [Batch 800/938] [D loss: 0.0543] [G loss: 3.2004]
[Epoch 6/10] [Batch 0/938] [D loss: 0.0696] [G loss: 2.7549]
[Epoch 6/10] [Batch 200/938] [D loss: 0.0301] [G loss: 3.3360]
[Epoch 6/10] [Batch 400/938] [D loss: 0.0552] [G loss: 2.5130]
[Epoch 6/10] [Batch 600/938] [D loss: 0.1271] [G loss: 5.9517]
[Epoch 6/10] [Batch 800/938] [D loss: 0.3405] [G loss: 0.8071]
[Epoch 7/10] [Batch 0/938] [D loss: 0.1256] [G loss: 12.5769]
```

```
[Epoch 7/10] [Batch 200/938] [D loss: 0.1168] [G loss: 3.8395]
[Epoch 7/10] [Batch 400/938] [D loss: 0.0165] [G loss: 3.9850]
[Epoch 7/10] [Batch 600/938] [D loss: 0.0473] [G loss: 3.3786]
[Epoch 7/10] [Batch 800/938] [D loss: 0.0621] [G loss: 2.86
[Epoch 8/10] [Batch 0/938] [D loss: 0.0486] [G loss: 2.733
[Epoch 8/10] [Batch 200/938] [D loss: 0.0988] [G loss: 7.1
[Epoch 8/10] [Batch 400/938] [D loss: 0.0211] [G loss: 3.8
[Epoch 8/10] [Batch 600/938] [D loss: 0.0639] [G loss: 3.5
[Epoch 8/10] [Batch 800/938] [D loss: 0.1091] [G loss: 4.1
[Epoch 9/10] [Batch 0/938] [D loss: 0.0365] [G loss: 3.431
[Epoch 9/10] [Batch 200/938] [D loss: 0.0731] [G loss: 2.6060]
[Epoch 9/10] [Batch 400/938] [D loss: 0.0401] [G loss: 2.9684]
[Epoch 9/10] [Batch 600/938] [D loss: 0.0768] [G loss: 3.6692]
[Epoch 9/10] [Batch 800/938] [D loss: 0.0305] [G loss: 3.4587]
Training complete.
```