Hyperparamaters

```
z_dir = 64
lr = 0.0002
batch_size = 128
epochs =  60
```

Generator

```
#@title Generator
class Generator(nn.Module):
  def __init__(self, z_dim):
    super().__init__()
    self.model = nn.Sequential(
        nn.Linear(z_dim, 128),
        nn.ReLU(True),
        nn.Linear(128, 256),
        nn.ReLU(True),
        nn.Linear(256, 784),
        nn.Tanh(),
    )
  def forward(self, z):
    return self.model(z)
```

Discriminator

```
#@title Discriminator
class Discriminator(nn.Module):
  def __init__(self):
    super().__init__()
    self.model = nn.Sequential(
        nn.Linear(784, 256),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Linear(256, 128),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Linear(128, 1),
        nn.Sigmoid(),
    )
  def forward(self, x):
    return self.model(x)
```

```
# ---setup---
device = "cuda" if torch.cuda.is_available() else "cpu"
Generator = Generator(z_dir).to(device)
Discriminator = Discriminator().to(device)

optimizer_G = optim.Adam(Generator.parameters(), lr=lr)
optimizer_D = optim.Adam(Discriminator.parameters(), lr=lr)
loss_fn = nn.BCELoss()

# --- Data ---
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5],[0.5]),
])
train_loader = DataLoader(
    datasets.MNIST(root="data", train=True, download=True, transform=transform),
    batch_size=batch_size,
    shuffle=True,
)
```

Training

```
# @title Training
for epoch in range(epochs):
    for batch, (real, _) in enumerate(train_loader):
        real = real.view(-1, 784).to(device)
        batch_size_curr = real.size(0)

        ones = torch.ones(batch_size_curr, 1).to(device)
        zeros = torch.zeros(batch_size_curr, 1).to(device)

        # Train Discriminator
        z = torch.randn(batch_size_curr, z_dir).to(device)
        fake = Generator(z).detach()
        d_loss = (loss_fn(Discriminator(real), ones) +
                  loss_fn(Discriminator(fake), zeros)) / 2
        optimizer_D.zero_grad(); d_loss.backward(); optimizer_D.step()

        # Train Generator
        z = torch.randn(batch_size_curr, z_dir).to(device)
        fake = Generator(z)
        g_loss = loss_fn(Discriminator(fake), ones)
        optimizer_G.zero_grad(); g_loss.backward();optimizer_G.step()

    print(f"Epoch [{epoch+1}/{epochs}] | D Loss: {d_loss:.3f} | G Loss: {g_loss:.3f}")
```

```
#-- generating samples
z = torch.randn(16, z_dir).to(device)
fake = Generator(z).view(-1,28,28).cpu().detach()

fig,axes =  plt.subplots(4,4,figsize = (6,6))
for i,ax in enumerate(axes.flatten()):
  ax.imshow(fake[i],cmap ='gray')
  ax.axis('off')
plt.show()
```

```
Start coding or generate with AI.
```