import torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt

import numpy as np

# Generator Network

class Generator(nn.Module):

def \_init\_(self, latent\_dim, output\_dim):

super(Generator, self).\_init\_()

self.main = nn.Sequential(

nn.Linear(latent\_dim, 256),

nn.LeakyReLU(0.2),

nn.Linear(256, 512),

nn.LeakyReLU(0.2),

nn.Linear(512, 1024),

nn.LeakyReLU(0.2),

nn.Linear(1024, output\_dim),

nn.Tanh()

)

def forward(self, x):

return self.main(x)

# Discriminator Network

class Discriminator(nn.Module):

def \_init\_(self, input\_dim):

super(Discriminator, self).\_init\_()

self.main = nn.Sequential(

nn.Linear(input\_dim, 512),

nn.LeakyReLU(0.2),

nn.Linear(512, 256),

nn.LeakyReLU(0.2),

nn.Linear(256, 1)

)

def forward(self, x):

return self.main(x)

class LSWGAN:

def \_init\_(self, latent\_dim, output\_dim):

self.latent\_dim = latent\_dim

self.generator = Generator(latent\_dim, output\_dim)

self.discriminator = Discriminator(output\_dim)

self.g\_optimizer = optim.Adam(self.generator.parameters(), lr=0.0002, betas=(0.5, 0.999))

self.d\_optimizer = optim.Adam(self.discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))

def train\_step(self, real\_data):

batch\_size = real\_data.size(0)

# Train Discriminator

self.d\_optimizer.zero\_grad()

z = torch.randn(batch\_size, self.latent\_dim)

fake\_data = self.generator(z)

real\_validity = self.discriminator(real\_data)

fake\_validity = self.discriminator(fake\_data.detach())

# Least squares loss

d\_loss = 0.5 \* (torch.mean((real\_validity - 1)\*2) + torch.mean(fake\_validity\*2))

d\_loss.backward()

self.d\_optimizer.step()

# Train Generator

self.g\_optimizer.zero\_grad()

gen\_validity = self.discriminator(fake\_data)

g\_loss = 0.5 \* torch.mean((gen\_validity - 1)\*\*2)

g\_loss.backward()

self.g\_optimizer.step()

return d\_loss.item(), g\_loss.item()

def train\_and\_plot(model, train\_data, test\_data, epochs=100):

train\_d\_losses = []

train\_g\_losses = []

test\_d\_losses = []

test\_g\_losses = []

for epoch in range(epochs):

# Training phase

d\_loss, g\_loss = model.train\_step(train\_data)

train\_d\_losses.append(d\_loss)

train\_g\_losses.append(g\_loss)

# Testing phase

with torch.no\_grad():

d\_loss, g\_loss = model.train\_step(test\_data)

test\_d\_losses.append(d\_loss)

test\_g\_losses.append(g\_loss)

if epoch % 10 == 0:

print(f"Epoch {epoch}: D\_loss: {d\_loss:.4f}, G\_loss: {g\_loss:.4f}")

# Plot results

plt.figure(figsize=(12, 8))

# Training Phase

plt.subplot(2, 2, 1)

plt.plot(train\_d\_losses, label='LSWGAN', color='purple')

plt.plot(train\_g\_losses, label='SGAN', color='red')

plt.title('LSWGAN - Training Phase')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(2, 2, 2)

plt.plot(train\_d\_losses, label='LSWGAN', color='purple')

plt.plot(train\_g\_losses, label='SGAN', color='red')

plt.title('LSWGAN - Training Phase')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

# Testing Phase

plt.subplot(2, 2, 3)

plt.plot(test\_d\_losses, label='LSWGAN', color='purple')

plt.plot(test\_g\_losses, label='SGAN', color='red')

plt.title('LSWGAN - Testing Phase')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(2, 2, 4)

plt.plot(test\_d\_losses, label='LSWGAN', color='purple')

plt.plot(test\_g\_losses, label='SGAN', color='red')

plt.title('LSWGAN - Testing Phase')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.tight\_layout()

plt.show()

# Example usage

latent\_dim = 100

output\_dim = 784 # for UCSD Pedestrian dataset, the Shanghai Tech dataset

# Create synthetic data for demonstration

train\_data = torch.randn(1000, output\_dim)

test\_data = torch.randn(200, output\_dim)

# Initialize and train model

model = LSWGAN(latent\_dim, output\_dim)

train\_and\_plot(model, train\_data, test\_data, epochs=100)