2022-2023学年

第2学期

《深度学习》

课程实验报告

|  |  |
| --- | --- |
| 姓名：XXX | 学号：XXXXXXX |

**实验五：基于生成对抗网络的手写数字生成**

# 实验目的

本次实验通过使用PyTorch完成基于生成对抗网络的手写数字生成，掌握生成对抗网络使用方法。

# 实验内容

1、了解生成对抗网络

2、生成器、判别器实现方案

3、损失函数的实现和理解

4、完成基于生成对抗网络的手写数字生成

# 实验原理

生成对抗网络（Generative Adversarial Networks，GANs）是一类机器学习模型，由生成器（Generator）和判别器（Discriminator）两个神经网络组成。GAN的原理是基于博弈论的思想，生成器和判别器进行一种两人零和博弈的最小最大化游戏。

生成器网络接受随机噪声作为输入，并生成合成数据样本。它的目标是学习从噪声分布到数据分布的映射，从而生成与训练数据相似的真实样本。生成器的目标是产生的样本与真实数据在判别器中难以区分。

判别器网络则被训练用于区分真实样本和生成样本。它接受输入数据，并预测其为真实样本还是伪造样本（生成的）。判别器的目标是尽可能准确地区分真实样本和伪造样本。

GAN的训练过程涉及生成器和判别器网络之间的竞争性交互。以下是GAN的训练过程的概述：

1. 生成器以随机权重初始化，并使用真实数据和生成的数据样本对判别器进行训练。生成器生成随机的伪造样本。
2. 判别器使用来自训练集的真实样本和生成器生成的样本进行训练。判别器学习将真实样本分类为“真实”（分配高概率）和生成样本分类为“伪造”（分配低概率）。
3. 生成器通过生成更逼真的样本来欺骗判别器。生成器以随机噪声为输入，生成新的伪造样本。
4. 判别器再次使用更新后的生成器和真实样本进行训练。判别器不断提高区分真实和伪造样本的能力。
5. 训练过程在更新判别器和生成器之间迭代。目标是找到一个平衡点，使生成器生成的样本逼真到判别器无法区分真伪。

通过反复以对抗的方式训练生成器和判别器，GAN学习生成样本以捕捉潜在的数据分布。生成器逐渐学会生成更逼真的样本，而判别器则变得更加准确地区分真实和伪造样本。最终目标是生成的样本与真实数据无法区分。

GAN在各个领域取得了显著的成功，如图像合成、文本生成和音乐创作等。它们有潜力生成高度逼真且多样化的数据，为创造性应用和数据增强在机器学习中开辟了新的可能性。

# 实验过程

1. 生成器和判别器网络

生成器与判别器的定义代码编写如下：

|  |
| --- |
| # 判决网络 def discriminator():  net = nn.Sequential(  nn.Linear(784, 256),  nn.LeakyReLU(0.2),  nn.Linear(256, 256),  nn.LeakyReLU(0.2),  nn.Linear(256, 1)  )  return net   # 生成网络 def generator(noise\_dim=NOISE\_DIM):  net = nn.Sequential(  nn.Linear(noise\_dim, 1024),  nn.ReLU(True),  nn.Linear(1024, 1024),  nn.ReLU(True),  nn.Linear(1024, 784),  nn.Tanh()  )  return net |

其中判别器网络结构的输入层是一个线性网络，输入维度大小为784，对应于MNIST数据集展平后的图像尺寸，输出维度大小为256。之后引入了非线性激活函数，之后再连接上两个线性层，使用非线性激活函数。

生成器网络的结构与判别器相似，其输入层的输入维度采用了噪声数据的维度，激活函数采用ReLU函数，在最后一个线性层后再对其使用一Tanh函数。

1. 损失函数

损失函数的代码编写如下：

|  |
| --- |
| bce\_loss = nn.BCEWithLogitsLoss()  def discriminator\_loss(logits\_real, logits\_fake): # 判别器的 loss  size = logits\_real.shape[0]  true\_labels = Variable(torch.ones(size, 1)).float().cuda()  false\_labels = Variable(torch.zeros(size, 1)).float().cuda()  loss = bce\_loss(logits\_real, true\_labels) + bce\_loss(logits\_fake, false\_labels)  return loss  def generator\_loss(logits\_fake): # 生成器的 loss  size = logits\_fake.shape[0]  true\_labels = Variable(torch.ones(size, 1)).float().cuda()  loss = bce\_loss(logits\_fake, true\_labels)  return loss |

首先，代码中创建了一个二分类交叉熵损失函数（BCEWithLogitsLoss），表示为bce\_loss。该损失函数用于度量模型输出与目标标签之间的差异。

接下来，定义了判别器损失函数（discriminator\_loss）。该函数接受判别器对真实样本（logits\_real）和生成样本（logits\_fake）的预测结果作为输入。函数首先确定了样本的大小，然后创建了真实标签（true\_labels）和伪造标签（false\_labels），这些标签用于计算判别器的损失。

损失的计算包括两部分，第一部分是将真实样本的预测结果与真实标签进行比较，计算真实样本的损失。第二部分是将生成样本的预测结果与伪造标签进行比较，计算生成样本的损失。最终的判别器损失是这两部分损失的总和。

然后，定义了生成器损失函数（generator\_loss）。该函数接受生成器对生成样本（logits\_fake）的预测结果作为输入。函数首先确定了生成样本的大小，然后创建了真实标签（true\_labels）。生成器的损失是将生成样本的预测结果与真实标签进行比较得到的。

这些损失函数用于衡量判别器和生成器在训练过程中的性能，并用于优化GAN模型的参数。判别器的目标是最小化判别器损失，使其能够准确区分真实和生成的样本。生成器的目标是最小化生成器损失，使其能够生成更逼真的样本以欺骗判别器。通过优化这些损失函数，GAN模型可以逐步提升生成样本的质量。

1. 训练与手写数字生成

网络训练函数代码编写如下：

|  |
| --- |
| def train\_a\_gan(D\_net, G\_net, D\_optimizer, G\_optimizer, discriminator\_loss, generator\_loss, show\_every=250, noise\_size=96, num\_epochs=10):  iter\_count = 0  for epoch in range(num\_epochs):  for x, \_ in train\_data:  bs = x.shape[0]  # 判别网络  real\_data = Variable(x).view(bs, -1).cuda() # 真实数据  logits\_real = D\_net(real\_data) # 判别网络得分   sample\_noise = (torch.rand(bs, noise\_size) - 0.5) / 0.5 # -1 ~ 1 的均匀分布  g\_fake\_seed = Variable(sample\_noise).cuda()  fake\_images = G\_net(g\_fake\_seed) # 生成的假的数据  logits\_fake = D\_net(fake\_images) # 判别网络得分   d\_total\_error = discriminator\_loss(logits\_real, logits\_fake) # 判别器的 loss  D\_optimizer.zero\_grad()  d\_total\_error.backward()  D\_optimizer.step() # 优化判别网络   # 生成网络  g\_fake\_seed = Variable(sample\_noise).cuda()  fake\_images = G\_net(g\_fake\_seed) # 生成的假的数据   gen\_logits\_fake = D\_net(fake\_images)  g\_error = generator\_loss(gen\_logits\_fake) # 生成网络的 loss  G\_optimizer.zero\_grad()  g\_error.backward()  G\_optimizer.step() # 优化生成网络   if (iter\_count % show\_every == 0):  print('Iter: {}, D: {:.4}, G:{:.4}'.format(iter\_count, d\_total\_error.item(), g\_error.item()))  imgs\_numpy = deprocess\_img(fake\_images.data.cpu().numpy())  show\_images(imgs\_numpy[0:16])  plt.show()  print()  iter\_count += 1 |

函数通过迭代训练判别器和生成器来训练生成对抗网络。它接受判别器网络（D\_net）、生成器网络（G\_net）、判别器优化器（D\_optimizer）、生成器优化器（G\_optimizer）、判别器的损失函数（discriminator\_loss）和生成器的损失函数（generator\_loss）作为输入。

在每个epoch循环中，函数遍历训练数据集（train\_data）。对于每个批次的输入数据x，首先计算真实数据的判别器得分（logits\_real），然后生成噪声数据（sample\_noise）并通过生成器网络生成假数据（fake\_images）。接着，计算生成数据的判别器得分（logits\_fake）。

然后，根据判别器的损失函数计算判别器的损失（d\_total\_error）。将判别器优化器（D\_optimizer）的梯度置零，进行反向传播（backpropagation）计算判别器的梯度，并进行优化（D\_optimizer.step()）。

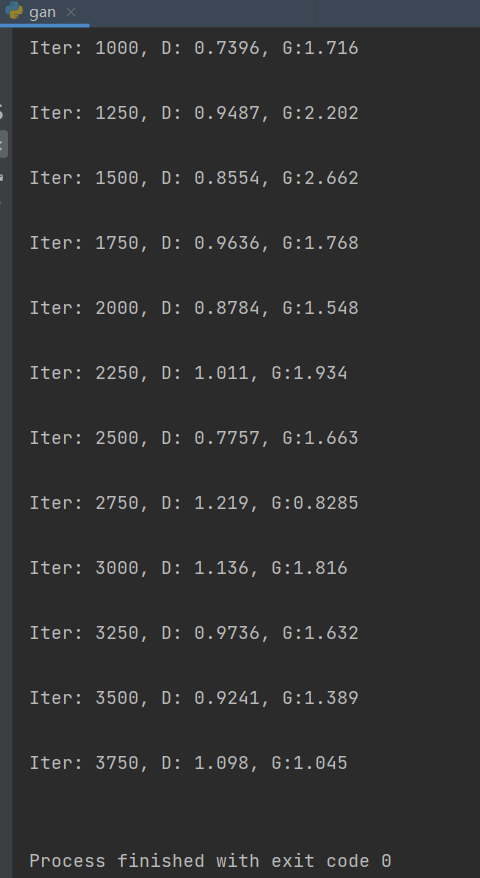
接下来，再次通过生成器网络生成假数据，并计算生成数据在判别器中的得分（gen\_logits\_fake）。然后，根据生成器的损失函数计算生成器的损失（g\_error）。将生成器优化器（G\_optimizer）的梯度置零，进行反向传播计算生成器的梯度，并进行优化。

如果迭代次数（iter\_count）能被show\_every整除，则打印当前迭代次数、判别器的损失和生成器的损失。同时，将生成的假图片转换为NumPy数组（imgs\_numpy），使用show\_images函数显示部分生成的图片，并通过plt.show()显示图片。最后，迭代次数（iter\_count）加1。

通过这个训练函数，判别器和生成器网络可以相互博弈并相互提升，从而达到生成逼真图片的目标。函数中的优化器、损失函数和网络架构的选择与GAN的训练策略密切相关，可以根据具体的任务和数据集进行调整和优化。

# 实验结果与分析

设置随机数种子为本人学号后四位5631，启动GAN网络训练函数，运行截图如图1所示，代码正常退出，顺利完成GAN网络训练过程。



**图1 代码运行截图**

在网络训练完成后生成了一幅手写数字图像`data.png`，如图2所示。

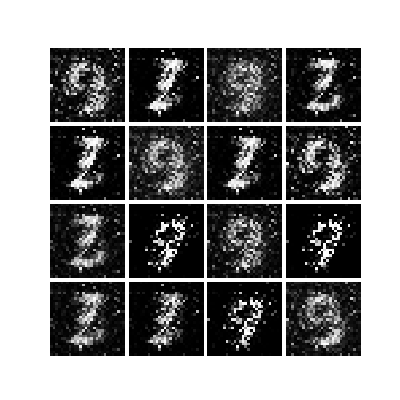


图2 生成手写数字图片

在整体的训练过程中，生成器和判别器的损失都较为接近。初期时判别器的损失高于生成器，此时说明生成器的性能优于判别器，判别器难以有效地区分真实样本和生成样本。在后续的训练过程中，判别器和生成器的性能接近平衡，损失相近，判别器能够辨别出大部分真实样本和生成样本，而生成器能够生成逼真的样本，难以被判别器辨别出来。

# 心得体会

通过本次实验，对生成对抗网络的模型结构、损失函数、训练方式等有了初步的了解，并通过MNIST数据集的生成简单体验了解了GAN，是一次有益的尝试。

GAN的核心思想是通过让生成器网络和判别器网络相互对抗来实现生成逼真样本的目标。生成器网络试图生成逼真的样本，而判别器网络则试图区分真实样本和生成样本。

GAN的训练过程是一个博弈过程，判别器和生成器相互竞争并不断提升。通过反复迭代的训练，判别器和生成器的性能逐渐达到平衡，生成器可以生成逼真的样本。

判别器的目标是最大化判别准确性，即正确区分真实样本和生成样本。生成器的目标是最小化被判别器误判为生成样本的概率，从而生成更逼真的样本。

GAN的训练过程中需要注意平衡判别器和生成器的能力，避免出现失衡现象。如果判别器和生成器的损失相差较大，可能需要调整模型结构、超参数或训练策略来实现平衡。