# 《人工智能》实验五报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 孙潇桐 | 学号 | 2021117405 |
| 实验地点 | 机房321 | 实验日期 | 2023-11-13 |

**一、实验内容**

GANs have rapidly emerged as the state-of-the-art technique in realistic image generation. Its applications span realistic image editing that is omnipresent in popular app filters, enabling tumor classification under low data schemes in medicine, and visualizing realistic scenarios of climate change destruction. Building a simple Generative Adversarial Network (GAN) using pytorch. Visualizing the Training losses. （搭建GAN网络，生成MNIST手写体。Minist 手写字体数据库在课程群里下载）

1. **实验原理**

生成对抗网络（GAN）是一种由**生成器**和**判别器**组成的对抗性模型。生成器负责生成伪造的数据，而判别器则负责判断给定数据是真实的还是伪造的。生成器和判别器通过对抗训练的方式相互竞争，最终达到**生成逼真数据**的目的。我在训练了一个普通的GAN网络之后，发现这个网络没法满足我想生成**特定数字**的需求，于是我经过一番搜寻，找到了一个解决方案，我将生成器的模型稍加改进，使用One-hot编码，告知生成器这个数字的类别，变成了CGAN（条件 GAN），就能使得生成器可以根据我提供的提示进行生成，而不是纯随机生成。

在本实验中，生成器网络由三个全连接层（一个输入，一个输出，一个隐藏层）组成，接受一个随机噪声向量和条件类别标签作为输入，并**输出生成的图像**。下面是生成器的代码：

class Generator(nn.Module):

def \_\_init\_\_(self, input\_dim, output\_dim, num\_classes):

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

self.fc = nn.Sequential(

nn.Linear(input\_dim + num\_classes, 256),

nn.ReLU(),

nn.Linear(256, output\_dim),

nn.Tanh()

)

self.num\_classes = num\_classes

def forward(self, x, labels):

# 将条件类别标签转换为One-Hot编码

labels\_onehot = torch.zeros(labels.size(0), self.num\_classes).to(device)

labels\_onehot.scatter\_(1, labels.view(-1, 1), 1)

# 将噪声向量和条件类别标签进行拼接

x = torch.cat([x, labels\_onehot], dim=1)

x = self.fc(x)

return x

判别器网络也由三个全连接层（一个输出入层，一个输出层，一个隐藏层）组成，接受真实图像或生成图像以及条件类别标签作为输入，并输出**一个概率值**来表示输入图像的真实性。

class Discriminator(nn.Module):

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

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

self.fc = nn.Sequential(

nn.Linear(input\_dim + num\_classes, 256),

nn.LeakyReLU(0.2),

nn.Linear(256, 1),

nn.Sigmoid()

)

self.num\_classes = num\_classes

def forward(self, x, labels):

# 将条件类别标签转换为One-Hot编码

labels\_onehot = torch.zeros(labels.size(0), self.num\_classes).to(device)

labels\_onehot.scatter\_(1, labels.view(-1, 1), 1)

# 将图像数据和条件类别标签进行拼接

x = torch.cat([x, labels\_onehot], dim=1)

x = self.fc(x)

return x

训练过程中，首先通过生成器生成**伪造的图像**，然后使用真实图像和生成的图像分别计算判别器的损失。接下来，通过反向传播来更新判别器的参数。然后，再次通过生成器生成伪造的图像，并使用生成的图像计算生成器的损失。最后，通过反向传播来更新生成器的参数。通过反复迭代这个过程，生成器和判别器逐渐提升其性能，生成更加逼真的图像。

下面是训练使用的代码：

# 初始化生成器和判别器

generator = Generator(input\_dim, output\_dim, num\_classes).to(device)

discriminator = Discriminator(output\_dim, num\_classes).to(device)

# 定义损失函数和优化器

criterion = nn.BCELoss()

optimizer\_g = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))

optimizer\_d = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))

# 定义用于可视化训练损失的列表

losses = []

# 训练GAN

for epoch in range(1, num\_epochs + 1, 1):

progress\_bar = tqdm(train\_loader, total=len(train\_loader))

for batch\_idx, (real\_images, labels) in enumerate(progress\_bar):

real\_images = real\_images.view(-1, output\_dim).to(device)

labels = labels.to(device)

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

# 训练判别器

discriminator.zero\_grad()

labels\_real = torch.ones(batch\_size, 1).to(device)

labels\_fake = torch.zeros(batch\_size, 1).to(device)

# 向生成器输入随机噪声

noise = torch.randn(batch\_size, input\_dim).to(device)

fake\_images = generator(noise, labels)

# 使用真实图像计算以下是对判别器和生成器的训练部分的代码：

# 使用真实图像计算判别器损失

outputs\_real = discriminator(real\_images, labels)

d\_loss\_real = criterion(outputs\_real, labels\_real)

# 使用生成的图像计算判别器损失

outputs\_fake = discriminator(fake\_images.detach(), labels)

d\_loss\_fake = criterion(outputs\_fake, labels\_fake)

# 合并判别器损失

d\_loss = d\_loss\_real + d\_loss\_fake

d\_loss.backward()

optimizer\_d.step()

# 训练生成器

generator.zero\_grad()

outputs = discriminator(fake\_images, labels)

g\_loss = criterion(outputs, labels\_real)

g\_loss.backward()

optimizer\_g.step()

# 记录损失值

if batch\_idx % 100 == 0:

losses.append((d\_loss.item(), g\_loss.item()))

progress\_bar.set\_description(f"Epoch [{epoch}/{num\_epochs}], "

f"Discriminator Loss: {d\_loss.item():.4f}, "

f"Generator Loss: {g\_loss.item():.4f}")

if epoch % 10 == 0:

torch.save(generator.state\_dict(), model\_path('generator', epoch))

我还设置了没10个epoch记录一次生成器模型，后面调用这些模型就能直观的看到生成器训练的效果。

1. **实验过程以及结果分析**

先设置好超参数：

batch\_size = 64

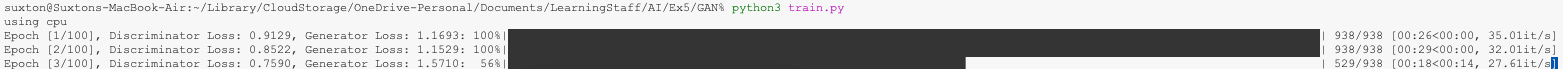
input\_dim = 100

output\_dim = 784

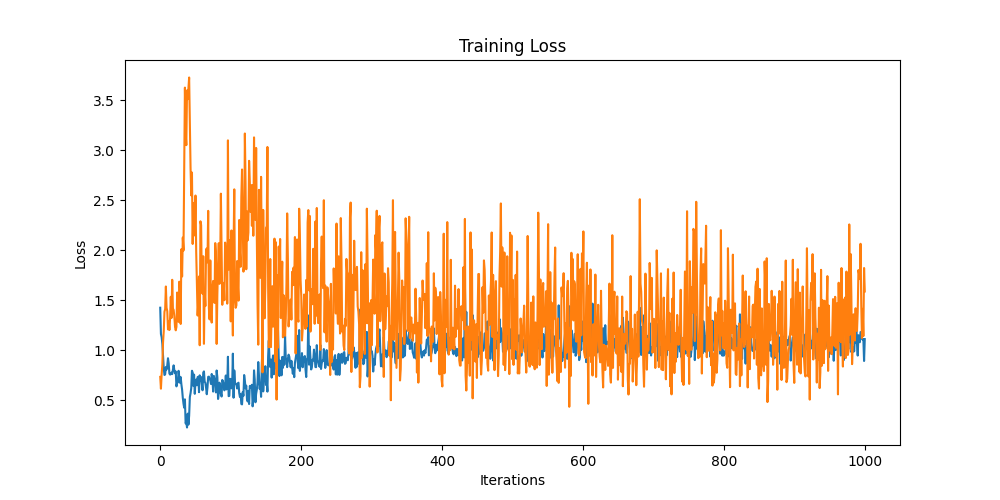
num\_epochs = 100

num\_classes = 10

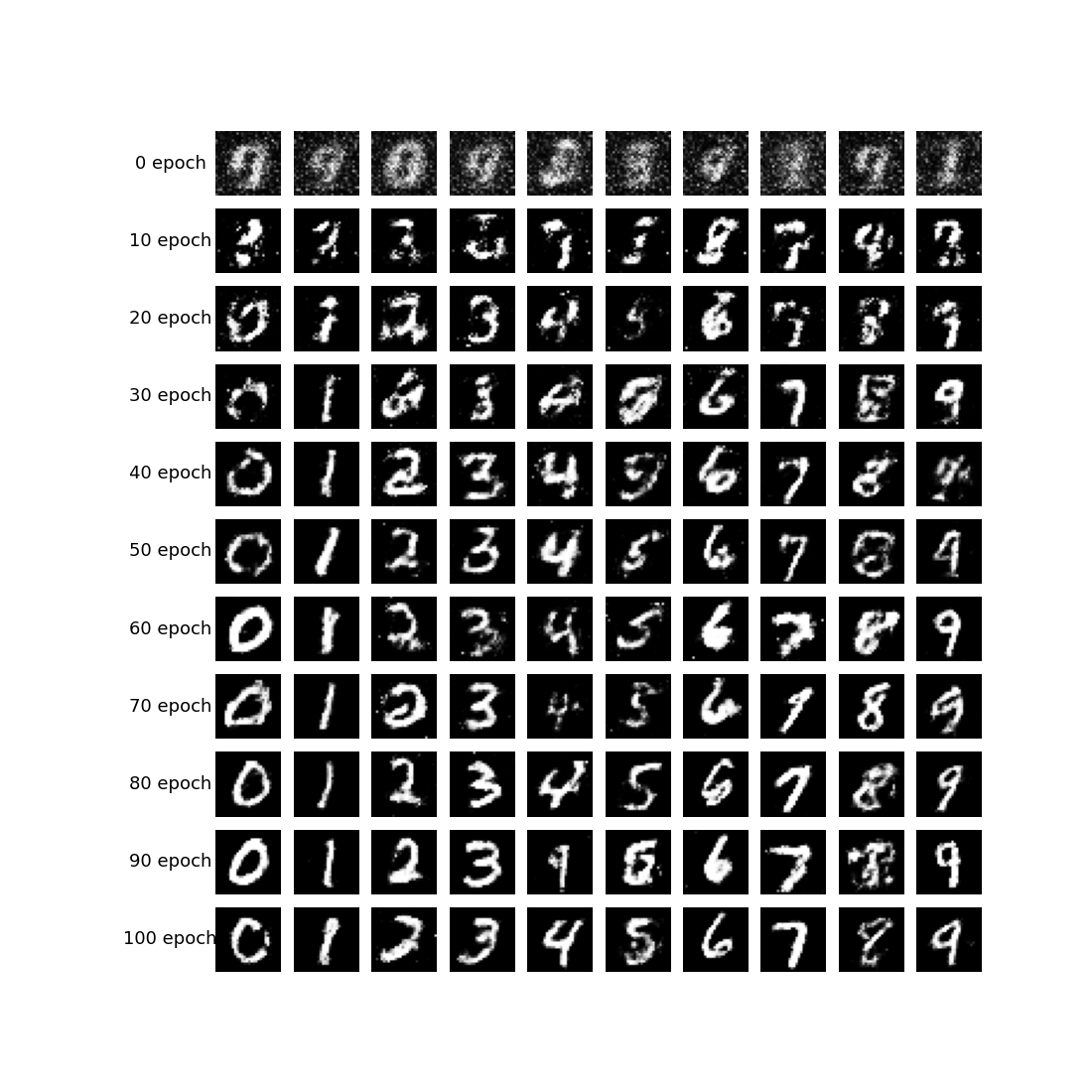
然后开始训练：



下面是训练时候记录的生成器和分类器的loss值：其中橙色的时候生成器的loss，蓝色的时候辨别器的loss。



下面是我用每10次迭代保存的模型生成的0-9的数字，可以看到，随着迭代次数的增加，模型生成的效果显著提高。



1. **实验总结**

本实验通过构建一个简单的生成对抗网络（GAN）并使用PyTorch生成手写数字图像，展示了GAN在图像生成领域的强大能力。通过对生成器和判别器网络的定义和训练过程的描述，展示了GAN的基本原理和训练方法。

我对我的模型最后生成的数据还是比较满意的，感觉机器学习相当有意思。这次实验帮助了我深入理解生成对抗网络的原理和训练过程，并为进一步探索和应用GAN提供了基础。

可惜我的电脑性能不足，不然的话我再让我可以尝试使用更加复杂的生成器和判别器，应该能使我的模型的性能更进一步。