《人工智能导论》大作业

任务名称：Mnist条件生成器

小组人员：傅以诚、马榕伯、赵宇航、谢浩哲

完成时间：2023.6.16

任务目标

基于Mnist数据集，构建一个条件生成模型，输入的条件为0~9的数字，输出对应条件的生成图像。

要求：

1. 支持随机产生输出图像；
2. 在cpu上有合理的运行时间。

具体内容

实施方案

1.数据准备

从`torchvision.datasetsd`中导入MNIST数据集，从从中提取0-9的数字图像。

2.构建生成器

使用深度卷积神经网络作为生成器，将条件作为输入，并生成对应条件的图像。

3.构建判别器

使用另一个深度卷积神经网络作为判别器，对生成的图像进行分类，以判断其是否真实。

4.构建GAN

将生成器和判别器组合起来，形成一个GAN模型。GAN模型的目标是让生成器生成的图像尽可能逼真，以欺骗判别器。损失函数使用交叉熵。损失函数使用二值交叉熵损失函数。

5.训练GAN

使用MNIST数据集来训练GAN模型。在训练期间，生成器将生成的图像传递给判别器进行分类。在训练过程中，有值函数

$$V(G,D)=\mathbb{E}_{x\sim P_{\text{data}}(x)}[\log D(x)]+\mathbb{E}_{z\sim p_{(z)}}[\log(1-D(G(z)))]$$

首先固定生成器，将生成器生成的样本和真实数据的样本输入判别器，并训练判别器。

训练完判别器后，训练生成器，将生成器生成的样本输入判别器，使得误差尽量小。

6.测试生成器

使用训练完毕的生成器生成图像，观察效果。

核心代码分析

1.生成器

```
class Generator(nn.Module):
    def __init__(self, num_classes, z_dim):
        super().__init__()

        self.embed = nn.Embedding(num_classes, z_dim) #将数字标签转换成向量形式

        self.dense = nn.Linear(z_dim, 7 * 7 * 256)  #将随机向量 z 与标签向量
embedded_label 进行线性组合

        self.trans1 = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 3, stride=2, padding=1,
output_padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU())

        self.trans2 = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU())

        self.trans3 = nn.Sequential(
            nn.ConvTranspose2d(64, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Tanh())  #使用三个转置卷积层进行逐步上采样，从而将线性组合后的向量转换为图
像。

    def forward(self, z, label):

        embedded_label = self.embed(label)  #将label传递给nn.Embedding层，以将其转换
为一个向量。

        x = embedded_label * z  #将随机向量z与embedded_label逐元素相乘，得到一个新的向
量x。

        x = self.dense(x).view(-1, 256, 7, 7)
        x = self.trans1(x)
        x = self.trans2(x)
        x = self.trans3(x)  #传递给三个转置卷积层，以获得最终的生成图像
        return x
```

2.判别器

```
class Discriminator(nn.Module):
    def __init__(self, num_classes, img_size):
        super().__init__()

        self.embed = nn.Embedding(num_classes, img_size[0] * img_size[1])

        self.conv1 = nn.Sequential(
            nn.Conv2d(2, 64, 3, stride=2, padding=1),
```

```
            nn.LeakyReLU())

        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 64, 3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU())

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, stride=2, padding=0),
            nn.BatchNorm2d(128),
            nn.LeakyReLU())

        self.dense = nn.Sequential(
            nn.Flatten(),
            nn.Linear(3 * 3 * 128, 1),
            nn.Sigmoid())

    def forward(self, x, label):
        embedded_label = self.embed(label).view_as(x) #将label传递给 nn.Embedding 层，将其转换为一个向量。

        x = torch.cat([x, embedded_label], dim=1) #将将图像数据 x 和标签向量 embedded_label 拼接在一起。

        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x) #将拼接后的向量 x 传递给三个卷积层，以提取特征。
        x = self.dense(x) #将特征展平，并使用线性层 nn.Linear 将其转换为一个标量值。最后，将标量值转换为0到1之间的概率值

        return x
```

3.GAN训练

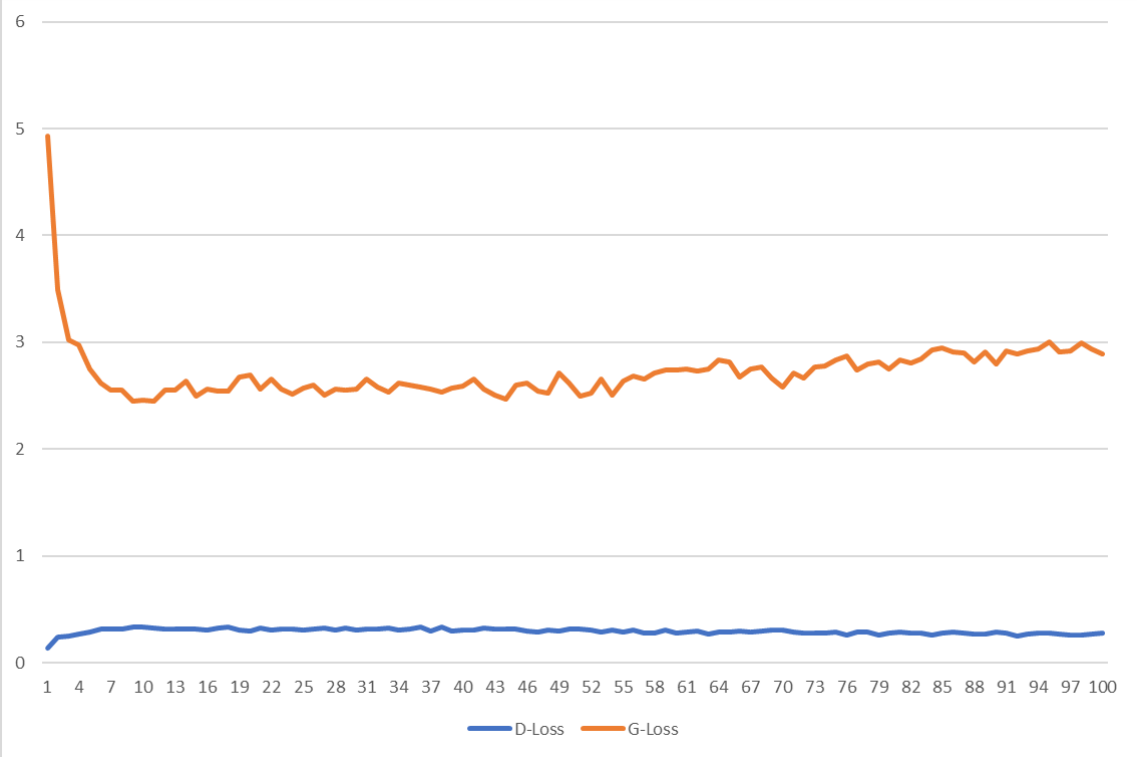
判别器训练

```
#将真实图像及其对应的label传入判别器并计算损失
d_out = discriminator(x, labels)
d_loss_real = criterion(d_out, y_real)
#根据损失进行优化判别器
d_optim.zero_grad()
d_loss_real.backward()
d_optim.step()

#生成一批与标签相关的假图像，并将其和label传入计算损失
x_gen = generator(get_z_vector(batch_size, Z_DIM).to(device), labels)
d_out = discriminator(x_gen, labels)
d_loss_fake = criterion(d_out, y_fake)
#根据损失进行优化判别器
d_optim.zero_grad()
d_loss_fake.backward()
d_optim.step()
```

生成器训练

```
#生成一批与标签相关的假图像，并将其和label传入判别器计算损失
x_gen = generator(get_z_vector(batch_size, Z_DIM).to(device), labels)
d_out = discriminator(x_gen, labels)
g_loss_real = criterion(d_out, y_real)

#根据损失进行优化生成器
g_optim.zero_grad()
g_loss_real.backward()
g_optim.step()
```

D-Loss和G-Loss变化图：



工作总结

收获、心得

我们深入了解了生成对抗网络（GAN）的工作原理和基本概念。通过阅读相关文献和参考资料，我们对GAN的生成器和判别器的作用以及它们如何通过对抗学习的方式相互提升有了更深入的理解。
其次，我们学会了如何处理和预处理图像数据集。在这个项目中，我们使用了MNIST数据集，并对图像数据进行了归一化处理，以适应生成器的输出范围。
实现MNIST生成器的过程中，我们掌握了使用深度学习框架（如TensorFlow、Keras等）构建模型的方法。我们学习了如何定义生成器和判别器的网络结构，并使用适当的层和激活函数来实现所需的功能。
最后，我们对生成的结果进行了评估和分析。我们生成了一些图像样本，并与真实的MNIST图像进行了比较。通过观察和比较生成的图像，我们可以评估生成器的性能并判断其生成图像的逼真程度。

通过这个小组作业，我们实现了MNIST生成器并使用进行训练，提高了我们在深度学习和生成对抗网络方面的技能。我们深入了解了GAN的工作原理，掌握了使用深度学习框架构建模型的技巧，并学会了通过调整超参数来优化模型性能。这个项目也加强了我们的团队合作能力和问题解决能力，让我们更好地理解和应用人工智能技术。

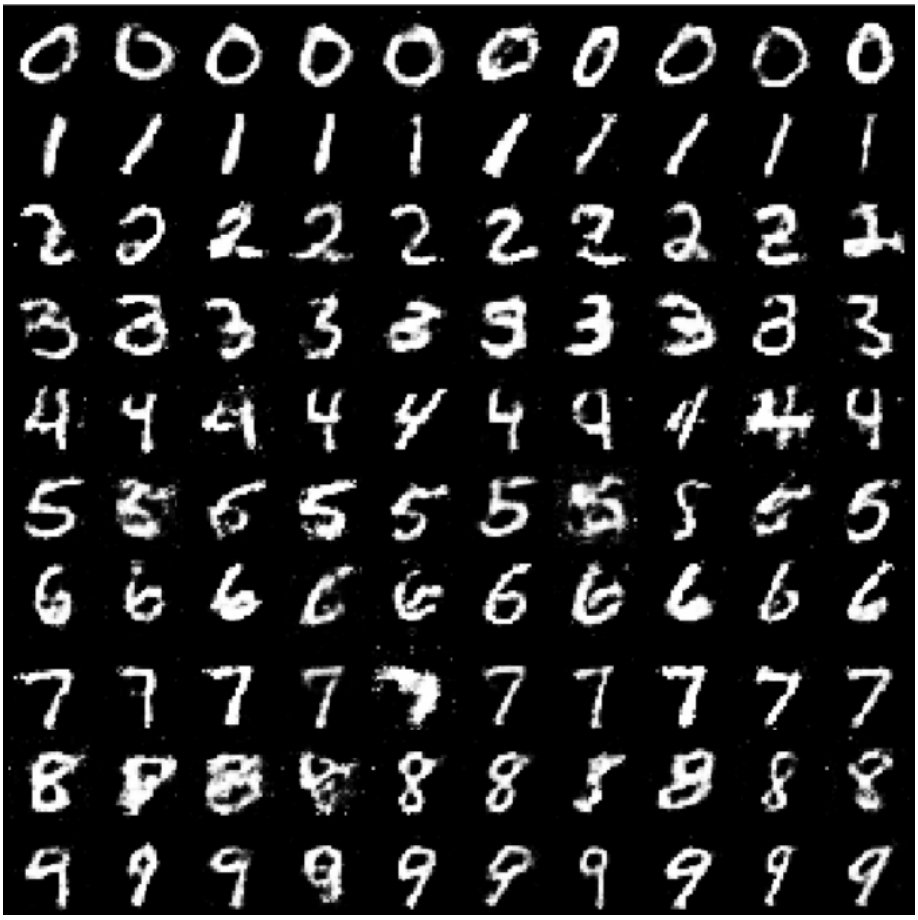
遇到问题及解决思路

在确立模型阶段：
首先我们尝试了cgan模型，并成功得到图案
此外我们尝试了vae模型，因为在网上寻找资料时发现该模型展示的图像的质量较好，然后我们学习了网上对应的代码：

```
for epoch in range(1, 51):
    train(epoch)
    test()

with torch.no_grad():
    z = torch.randn(64, 2).cpu()
    print(z)
    sample = vae.decoder(z).cpu()

    save_image(sample.view(64, 1, 28, 28), './samples/sample_' + '.png')
```

从上述代码发现其给出的是x重构后的数据集图案，并没有与1到9数字对应，然后我们找到了与数字对应的vae模型，却发现生成图像反而很差

同时，我们也有想法将cgan和vae模型结合起来，也就是使用vae模型重构后的图案交给cgan分析，其效果如下图：



经过组员判断，其生成图像不如直接使用cgan，分析原因在于vae的重构放到gan看来也只是一种随机打乱。
结论：最后采用cgan模型

我们尝试了在0.0001-0.01区间内对学习率的大小进行调整，观察最终结果。在初始阶段，我们使用较小的学习率（如0.0001）进行训练，生成的图像质量不够理想，缺乏清晰的边缘和细节。这表明学习率过小，限制了梯度下降的步长，导致生成器难以有效地学习。随后，我们尝试了较大的学习率（如0.01），希望能够加快模型的收敛速度。然而，我们观察到训练不稳定的现象，生成的图像出现了明显的模糊和失真。这是因为学习率过大，导致梯度下降过快，生成器和判别器之间的平衡失调，无法达到良好的对抗学习效果。最终，我们选择了学习率为0.001。通过多次实验和观察，我们发现这个学习率可以取得较好的训练效果。生成器能够较快地学习到图像的特征和分布，生成的图像质量得到了明显的提高。此外，训练过程相对稳定，避免了梯度爆炸或消失的问题。

lr=0.005

lr=0.001

lr=0.0005

在确定学习率后，我们增加了训练的次数为10倍，得到了更好的效果。

课程建议

学习课程内容之后提供相关代码以供学习。