《人工智能导论》大作业

任务名称：Mnist 条件生成器

完成组号：

小组人员：徐宇飞、陈冠宇、林中阳、邹若钦、闫梓涵

完成时间：2023.6.17

1．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。满足要求：

①支持随机产生输出图像；

②在 cpu 上有合理的运行时间.

2．具体内容

（1）实施方案

①采用cDCGAN模型，通过带有条件输入的生成器和判别器相互对抗训练不断提高生成器生成手写图像的真实性和判别器判别图像真伪的能力，最终获得能够输出较为真实的手写数字图像的生成器模型。

②包括导入、权重初始化、生成器的构造、损失函数调用、训练参数设定和图像生成等。

（2）核心代码分析

①网络模型

生成器网络：

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.deconv1_1 = nn.ConvTranspose2d(100, 256, 4, 1, 0)
        self.deconv1_1_bn = nn.BatchNorm2d(256)
        self.deconv1_2 = nn.ConvTranspose2d(10, 256, 4, 1, 0)
        self.deconv1_2_bn = nn.BatchNorm2d(256)
        self.deconv2 = nn.ConvTranspose2d(512, 256, 4, 2, 1)
        self.deconv2_bn = nn.BatchNorm2d(256)
        self.deconv3 = nn.ConvTranspose2d(256, 128, 4, 2, 1)
        self.deconv3_bn = nn.BatchNorm2d(128)
        self.deconv4 = nn.ConvTranspose2d(128, 1, 4, 2, 1)

    # 参数初始化
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    def forward(self, input, label):
        x = fun.relu(self.deconv1_1_bn(self.deconv1_1(input)))
        y = fun.relu(self.deconv1_2_bn(self.deconv1_2(label)))
        x = torch.cat([x, y], 1)
        x = fun.relu(self.deconv2_bn(self.deconv2(x)))
        x = fun.relu(self.deconv3_bn(self.deconv3(x)))
        x = torch.tanh(self.deconv4(x))
        return x
```

生成器使用了四个反卷积层，输入为一个100维的随机正态分布噪声向量和一个10维的标签向量，将其分别映射为256维的特征图并融合，然后先映射为256维后映射为128维的特征图，最终映射为1维的输出图像，并经过torch.tanh()函数的激活操作，将生成的图像数据映射到[-1,1]之间的范围，避免生成极端的图像数据。

判别器网络：

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1_1 = nn.Conv2d(1, 64, 4, 2, 1)
        self.conv1_2 = nn.Conv2d(10, 64, 4, 2, 1)
        self.conv2 = nn.Conv2d(128, 256, 4, 2, 1)
        self.conv2_bn = nn.BatchNorm2d(256)
        self.conv3 = nn.Conv2d(256, 512, 4, 2, 1)
        self.conv3_bn = nn.BatchNorm2d(512)
        self.conv4 = nn.Conv2d(512, 1, 4, 1, 0)

    # 参数初始化
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    def forward(self, input, label):
        x = fun.leaky_relu(self.conv1_1(input), 0.2)
        y = fun.leaky_relu(self.conv1_2(label), 0.2)
        x = torch.cat([x, y], 1)
        x = fun.leaky_relu(self.conv2_bn(self.conv2(x)), 0.2)
        x = fun.leaky_relu(self.conv3_bn(self.conv3(x)), 0.2)
        x = torch.sigmoid(self.conv4(x))
        return x
```

判别器使用了四个卷积层，输入为生成器输出的1维灰度图像和10维的标签向量，将其分别映射为64维的特征图并融合，然后先映射为256维的特征图、后映射为512维的特征图，最终降维为1维的数值并经过sigmoid激活处理，表示输入属于真实图像的概率。

②接口类

```
class AiGcMn:
    def __init__(self, gen_path):
        # 初始化生成器
        self.gen = Generator()
        self.gen.load_state_dict(torch.load(gen_path,
map_location=torch.device('cpu')))
        self.resize = Resize(28)
        # 初始化独热编码表
        self.onehot = fun.one_hot(torch.arange(10),
num_classes=10).float().view(10, 10, 1, 1)

    def generate(self, labels):
        # 获取批次大小
        batch_size = labels.size(0)
        # 进行独热编码
        hot_label = self.onehot[labels.type(torch.LongTensor)]
        # 生成随机噪声
        rd_noise = torch.randn(batch_size, 100, 1, 1)
        # 生成图像
        gen_output = self.gen(rd_noise, hot_label)
        # 处理为 28*28 大小
        output = torch.stack([self.resize(img) for img in gen_output])
        return output
```

1） 初始化Generator示例，通过预训练的生成器生成图像，并加载保存的模型参数到CPU，使用Resize类将图像调整为28*28大小，以便后续处理。

2） 对数字标签进行独热编码，通过初始化独热编码表并转换输入的标签，得到一个大小为(10,10,1,1)的四维张量，更好地区分不同数字标签之间的差异。

3） 将n维tensor传入generate，根据tensor的标签数组编码为独热编码。使用torch.randn()方法随机生成n个100维的正态分布噪声向量和独热编码传入生成器中，经过Resize类的处理，使生成图像大小调整为28*28，最终返回处理后的图像数据。

③模型训练过程

模型训练采取判别器和生成器同步训练的方式，先训练判别器一次再训练生成器一次，并以此循环形成二者相互对抗的训练过程。

```
discriminator.zero_grad()

mini_batch = image.size()[0]

labels_fill_ = fill[labels_]
image, labels_fill_ = image.to(device), labels_fill_.to(device)

D_result = discriminator(image, labels_fill_).squeeze()
D_real_loss = BCE_loss(D_result, labels_real_)

noise = torch.randn((mini_batch, 100)).view(-1, 100, 1, 1)
labels_ = (torch.rand(mini_batch, 1) *
10).type(torch.LongTensor).squeeze()
labels_label_ = onehot[labels_]
labels_fill_ = fill[labels_]
noise, labels_label_, labels_fill_ = noise.to(device),
labels_label_.to(device), labels_fill_.to(device)

G_result = generator(noise, labels_label_)
D_result = discriminator(G_result, labels_fill_).squeeze()
D_fake_loss = BCE_loss(D_result, labels_fake_)

D_train_loss = D_real_loss + D_fake_loss

D_train_loss.backward()
D_optimizer.step()

# 更新数据
with torch.no_grad():
    D_loss += D_train_loss.item()
```

判别器训练的大致流程：

1） 重置判别器的梯度。

2） 将来自数据集的真实图像和相应的标签传递给判别器。

3） 计算判别器对真实图像的损失。

4） 将由生成器生成的伪造图像和相应的标签传递给判别器。

5） 计算判别器对伪造图像的损失。

6） 总体判别器损失通过将真实和伪造图像的损失相加得到。

7） 使用反向传播计算和更新判别器的梯度。

```
generator.zero_grad()

noise = torch.randn((mini_batch, 100)).view(-1, 100, 1, 1)
labels_ = (torch.rand(mini_batch, 1) *
10).type(torch.LongTensor).squeeze()
labels_label_ = onehot[labels_]
labels_fill_ = fill[labels_]
noise, labels_label_, labels_fill_ = noise.to(device),
labels_label_.to(device), labels_fill_.to(device)

G_result = generator(noise, labels_label_)
D_result = discriminator(G_result, labels_fill_).squeeze()

G_train_loss = BCE_loss(D_result, labels_real_)

G_train_loss.backward()
G_optimizer.step()

# 更新数据
with torch.no_grad():
    G_loss += G_train_loss.item()

if (step + 1) % 10 == 0 or (step + 1) == dataloader_len:
    print(f"Epoch[{epoch + 1}/{epoch_num}] , Batch[{step +
1}/{dataloader_len}] , 生成器损失: {G_loss/(step + 1)} , 判别器损失:
{D_loss/(step + 1)}")
```

生成器训练的大致流程：

1） 重置生成器的梯度。

2） 使用随机噪声和标签生成由生成器生成的伪造图像。

3） 将伪造图像和相应的标签传递给判别器。

4） 根据判别器的输出计算生成器的损失。

5） 使用反向传播计算和更新生成器的梯度。

6） 定期跟踪和打印损失值。

训练输出的结果：

3．工作总结

（1）收获、心得

①对GAN模型有更深入的理解。

通过老师的引导和网上搜集的相关资料，我们学习了神经网络模型的具体搭建过程和一些库与相关工具，首次实操搭建了GAN模型并了解到GAN模型的适

用场景和一些其他模型的优劣。

②提高了数据预处理能力和解决困难的能力。

在构建数据集时需要首先去噪声，对数据进行缩放、归一化等处理，包括在图像上进行随机翻转或旋转操作，以增加数据集的多样性。在实现过程中，我们的模型也面临了一些挑战和困难，如模型过拟合、损失函数梯度爆炸等问题，这也锻炼了我们解决问题的能力，一次次的重复性实验增强了我们的耐心，也更好地理解了模型训练的过程和各种技巧。

（2）遇到问题及解决思路

①网络模型的选择。

工作开始时我们选择了cGAN模型进行测试，输出的数字图像结果为图中所示。经过多次参数包括Adam优化器的学习率、训练循环次数等的修改和测试发现输出的图像效果未能达到期望效果。

图 3.2.1.1 cGAN模型测试结果

因此小组决定更换为cDGAN模型，在给出条件的基础上加入卷积和反卷积操作，生成的图像更符合真实情况。

②将数字条件转化为模型所需的参数

如何将数字标签输入到生成器和判别器中并使得模型便于分辨不同数字之间的区别，是工作进行中的一个明显的问题。通过资料查阅和权衡，我们选择了使用one-hot编码将数字标签转换为10维的one-hot编码向量。例如，相对于3和8，模型更能通过两个one-hot编码向量[0,0,0,1,0,0,0,0,0,0]和[0,0,0,0,0,0,0,0,1,0]来区分两类数字之间的区别。

③平衡学习率与训练批次大小

优化器可以自适应地调整学习率，快速地收敛到最优解。经过多次尝试，通过控制其他参数不变，我们从lr即学习率分别为0.0001、0.0002、0.0003中分别测试并不断细化梯度，最终选择了0.0002的学习率。

同样地，训练批次对最终结果也有影响。控制其他参数不变，我们从循环次数5、10、15、20分别测试并不断减小梯度，最终选择了12的训练批次。

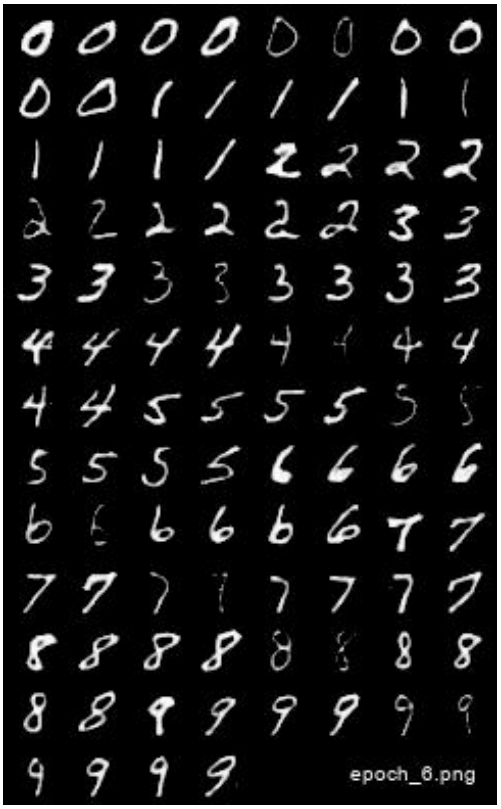

图 3.2.3.1　6次训练结果

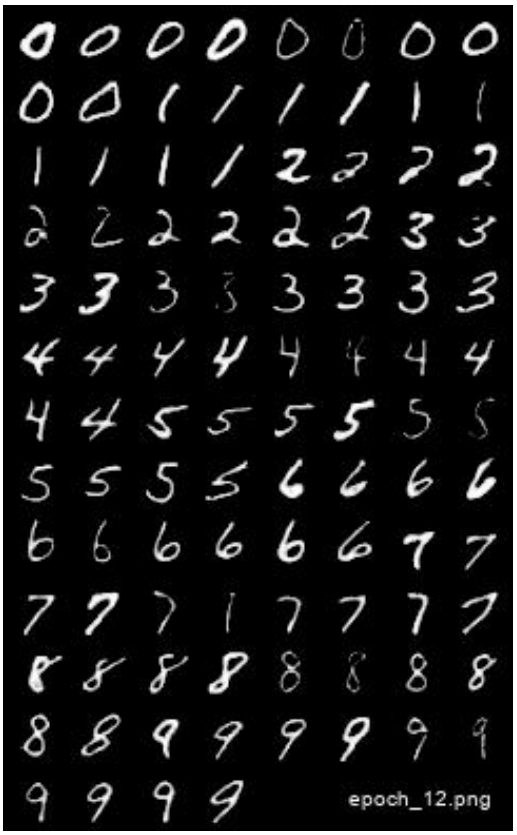

图 3.2.3.2　12次训练结果

④tensor拼接大小问题

在对tensor进行拼接时，出现维度不同导致不能拼接的问题。我们选择使用广播机制，例如labels_fill_ = fill[labels_]中将labels_张量广播扩展为与fill相同的维度，并基于labels_的值从fill的第一个维度中选择相应的值。labels_fill_具有与fill相同的维度后就可以进行拼接操作。

4．课程建议

小组在初次完成大作业时上手有些迷茫，花了很多时间学习基础知识。建议老师在以后的授课中添加一些与课程大作业相关的小作业，比如相关知识的应用、算法理解等到十六周的课程中，这样完成课程大作业的压力可以适当减轻。