《人工智能导论》大作业报告

任务名称：Mnist条件生成器

小组成员：赵天航，史轩宇，吴铮，陈梓轩，郑祥

完成时间：2023/5/31

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。要求：

- 支持随机产生输出图像；
- 在cpu上有合理的运行时间。

在实验的基本目标上我们希望生成真实度更高的图像，选择更为合适的模型得出更好的训练、生成效果。

2、具体内容

（1）实施方案

本组选择的生成模型是CGAN，考虑到CGAN加入了条件信息，其可以生成更具结构性和真实性的图像，而在任务中，由于使用了数字标签作为条件，生成的图像可以更加准确地反映了所需数字的特征。如一般的训练过程，我们导入必要的库、加载MNIST数据集并设置了一些超参数，主要的可调节参数为*epochs*、*batch_size*、*lr*，分别表示训练的周期数、每次迭代中用于训练的样本数、学习率。*Epochs*的值越大，模型学习到数据集中的模式和特征的机会就越多，但是训练的时间和计算成本会增加，并且当*epochs*越来越大训练的效果会趋于饱和；*batch_size*越大可以加快训练速度，但是可能导致模型过拟合，

越小可以提高模型泛化能力，但会降低训练速度；*lr*越大训练速度越快，但可能会导致模型无法收敛或出现震荡现象，*lr*越小模型收敛性能越好，但会增加训练时间。我们在寻求较好的训练结果时对这三个参数进行调整尝试，通过控制变量法，在训练周期数上尽可能寻求一个效果趋于饱和的临界周期数在保证训练效果的情况下减少时间成本，同时通过调节*batch_size*和*lr*的值期望得到更好的训练效果。其中，我们将loss函数变化的曲线记录在results文件夹中以确定大致合适的训练周期数。接着我们定义生成器、判别器、优化器、损失函数等并进行训练，将训练结果保存在*cgan_generator.pth*中，最后在*test.py*中输出结果。我们使结果的输出实现更加面向对象，可以手动输入希望获得的生成数字也可以选择一次性全部输出。为了便于老师使用，我们还在文件夹中写了一份README.md文档，便于老师使用，直接运行`python aigcmn.py`即可得到生成的数字图像。

（2）核心代码分析

代码在*model.py*实现了导入库、设置超参数、定义生成器类和定义判别器类这四个功能，在*train.py*完成了具体的训练过程，在*aigcmn.py*实现了按序输出模型训练后产生的图片的功能。以下分别对代码中出现的关键参数、生成器和判别器的核心结构、具体的训练过程和模型训练后产生图片的相关代码进行解读：

1）关键参数

```
# 超参数设置
latent_dim = 100
condition_dim = 10
img_shape = (1, 28, 28)
epochs = 200
batch_size = 32
lr = 0.0002
device = 'cuda'
```

*condition_dim*指条件维度，本次采用的方案是one-hot编码来代表CGAN中生成器和判别器接受的条件输入，由于生成的范围是0-9共10个数字，因此条件维度为10。*img_shape*指图形形状，本次设定为(1,28,28)意思是生成图像为灰度图，像素为28*28。*epochs*指训练周期数，*batch_size*表示每次训练时采用的样本数量，*lr*表示优化器更新模型参数时的步长。

2）生成器和判别器的核心结构

```
# 定义生成器
class Generator(nn.Module):
    def __init__(self, latent_dim, condition_dim, img_shape):
        super(Generator, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, int(torch.prod(torch.tensor(img_shape)))),
            nn.Tanh()
        )

    def forward(self, noise, condition):
```

```
        x = torch.cat((noise, condition), -1)
        img = self.model(x)
        img = img.view(img.size(0), *img_shape)
        return img
```

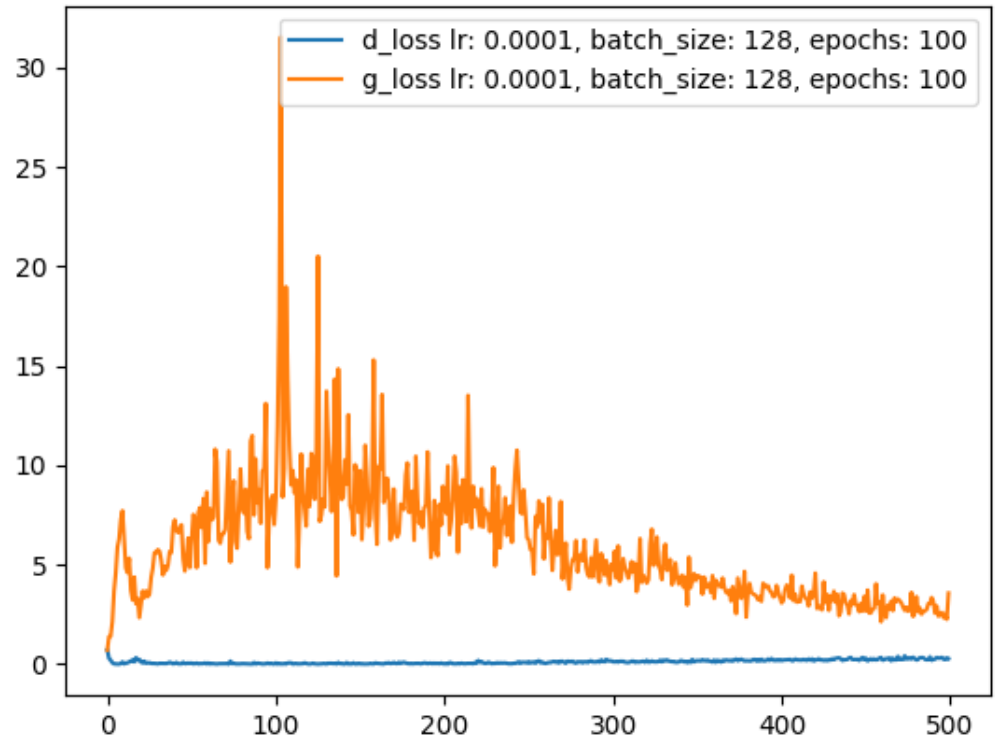
```
# 定义判别器
class Discriminator(nn.Module):
    def __init__(self, img_shape, condition_dim):
        super(Discriminator, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(int(torch.prod(torch.tensor(img_shape))) + condition_dim,
512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, img, condition):
        img_flat = img.view(img.size(0), -1)
        x = torch.cat((img_flat, condition), -1)
        validity = self.model(x)
        return validity
```

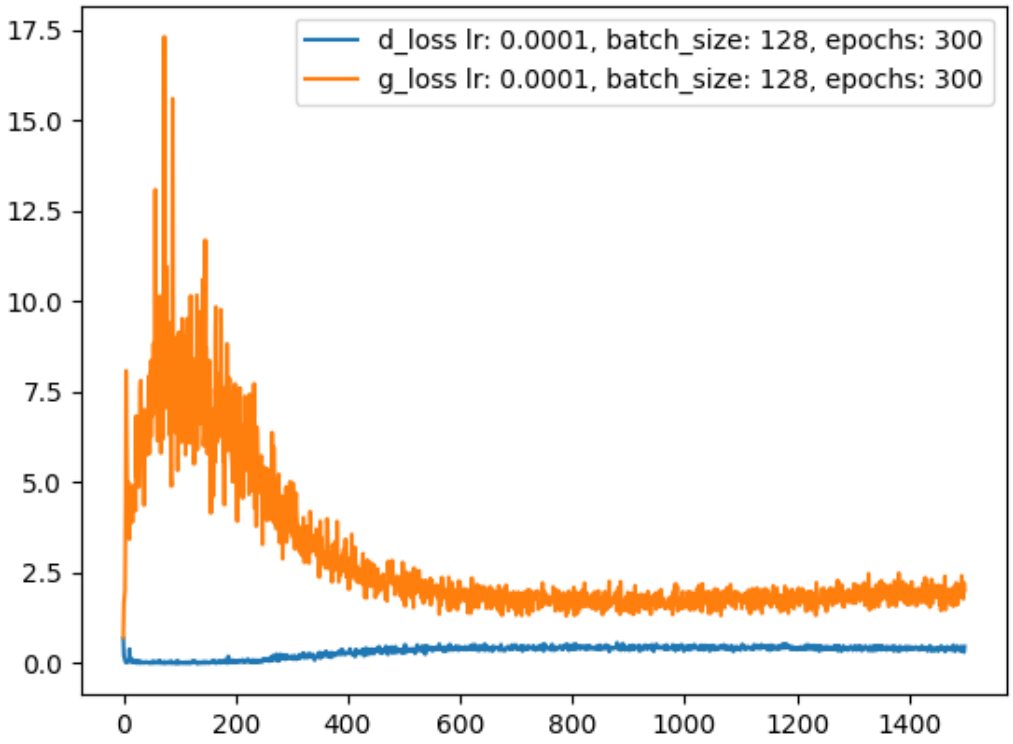
这里的生成器和判别器都是一个神经网络，其中生成器的输入是噪声维度*latent_dim*、条件维度*condition_dim*和图形形状*img_shape*，经过Linear这几个全连接层，并使用ReLU和Sigmoid两个函数激活，最后以pyTorch张量的形式用参数*img*返回。而判别器输入*condition_dim*和*img_shape*，同样经过全连接层和激活函数后，返回参量*validity*，这个参量反应了图像的有效性。生成器试图生成越来越逼真的图像以欺骗判别器，而判别器则试图更好地区分真实图像和生成图像。最终，生成器能够生成与真实图像非常相似的图像。

3）具体的训练过程

```
    # 保存loss变化图
    save_dir="./results"
    cur = time.strftime('%Y-%m-%d-%H-%M-%S', time.localtime(time.time()))
    # 保存d_loss
    label = 'd_loss lr: {}, batch_size: {}, epochs: {}'.format( lr, batch_size,
epochs)
    plt.plot(d_losses,label=label)
    plt.legend(loc='upper right')
    plt.savefig(os.path.join(save_dir, 'loss'+ cur +'.png'))
    # 保存g_loss
    label = 'g_loss lr: {}, batch_size: {}, epochs: {}'.format( lr, batch_size,
epochs)
    plt.plot(g_losses,label=label)
    plt.legend(loc='upper right')
    plt.savefig(os.path.join(save_dir, 'loss'+ cur +'.png'))
```

在*train.py*里，首先设置了几个超参数，并使用DataLoader把MNIST数据集封装成迭代器，然后对生成器和判别器进行初始化并设置优化器后，选择了二值交叉熵损失作为对抗性损失，开启了多轮训练：这个过程里，生成器和判别器的参数会不断更新，使得生成器尽可能生成逼真图像，判别器尽可能区分真实图像和生成图像。最后在保存了参数后生成一个条件图像并展示出来。由于我们希望确定训练周期数为多少最为合适，以上代码用于保存loss函数损失变化曲线，便于我们判断合适的周期数。下图为几个示例:





4）模型训练后产生图片的方法

```
def test():
    generator = get_model()
    generator.eval()

    flag = input("需要顺序输出0-9请输入：1 \n需要按要求输出指定数字请输入：0\n")
    if flag == '1':
        print("现在开始为您按序输出")
        fig, axs = plt.subplots(10, 10, figsize=(10, 10))
        for i in range(10):
            for j in range(10):
                condition = i
                image = generate_image(condition, generator)
                axs[i, j].imshow(image, cmap='gray')
                axs[i, j].axis('off')
        plt.tight_layout()
        plt.show()
    if flag == '0':
        while True:
            print("")
            print("如果想退出请输入-1")
            condition = int(input("请输入你想要输出的数字："))
            if condition == -1:
                return
            fig, axs = plt.subplots(1, 10, figsize=(10, 1))
            for j in range(10):
                image = generate_image(condition, generator)
                axs[j].imshow(image, cmap='gray')
                axs[j].axis('off')
            plt.tight_layout()
            plt.show()
```

这一部分在*aigcmn.py*中实现，generate_image()函数将生成器设置为评估模式并禁用梯度计算后，生成随机噪声向量noise和条件向量condition,将噪声和条件输入到生成器中，得到生成图像gen_img，再把转换为NumPy数组的形式返回。然后test()函数从get_model()函数获取生成器模型并设置为评估模式，并提示用户输入，输入'1'则函数按序生成数字0到9；输入'0'则将提示用户再输入一个数字，随后生成该数字的10张图像；输入'-1'则函数结束。

3、工作总结

（1）收获、心得

我和组员共同参与了《人工智能导论》大作业，并在项目中承担了不同的角色和任务。我们的工作内容主要包括模型构建、超参数调整和训练过程的尝试。

在项目中，我们深入学习了条件生成模型和CGAN，并成功构建了一个基于MNIST数据集的条件生成模型。通过共同努力，我们对模型的结构和原理有了更深入的理解，并能够灵活运用它们。

在模型构建过程中，我们一起研究了CGAN的架构，并根据需求设计了生成器和判别器的核心结构。通过不断优化网络层次、激活函数和损失函数的选择，我们努力提高模型的生成能力和判别准确性。

超参数调整是我们取得良好结果的关键步骤之一。我们仔细调整了训练周期数、批次大小和学习率等超参数，并通过控制变量法进行对比实验。这样的尝试使我们更加了解各个超参数对模型训练的影响，也提高了我们对调参过程的技巧和经验。

在训练过程中，我们认真记录了损失函数变化的曲线，并进行了交流和讨论，以确定合适的训练周期数。我们共同分析了训练结果，并根据实验数据进行调整和改进。这种合作和团队协作的精神使我们能够共同克服困难，并最终取得了令人满意的结果。

通过这个项目，我们不仅学会了具体的技术和方法，还提升了团队合作和解决问题的能力。我们体验到了探索和实践的重要性，也体会到了在人工智能领域持续学习和改进的重要性。这个项目为我们今后在人工智能领域的发展打下了坚实的基础，同时也培养了我们的团队意识和解决问题的能力。我们希望在未来能够继续探索和进步。

（2）遇到问题及解决思路

原先的方法中，当Generator的dense层，即

```
self.dense = nn.Linear(z_dim, 7 * 7 * 256)
```

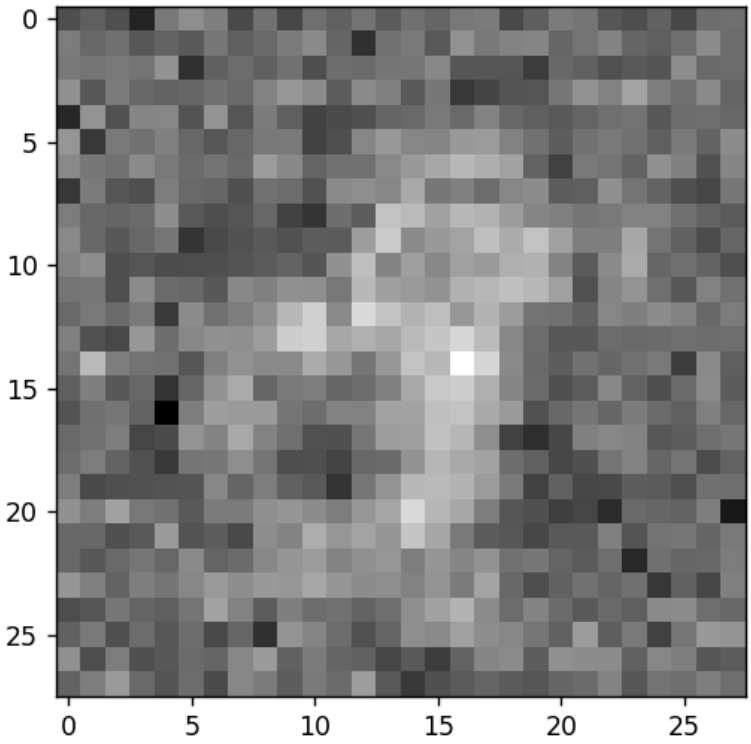
最终返回结果的维度，即第二个参数过大时，会使得整个模型难以收敛到满意的结果。

考虑卷积、转置卷积都主要利用空间不变性来减少参数，但本题中由于最后生成的图片不是特别大，故考虑取消转置卷积层直接使用全连接层。问题在于可能会有较多杂点的出现，曾考虑在Generator最后添加一层汇聚层AvgPool或卷积层Conv2d进行处理，但最终生成图像的风格与原来的数据集并不一致，整体更加模糊。

后来考虑加深dense层网络，即如今的实际方案

```
self.model = nn.Sequential(
    nn.Linear(latent_dim + condition_dim, 128),
    nn.ReLU(),
    nn.Linear(128, 256),
    nn.ReLU(),
    nn.Linear(256, 512),
    nn.ReLU(),
    nn.Linear(512, int(torch.prod(torch.tensor(img_shape)))),
    nn.Tanh()
)
```

来尝试解决，但最终效果也并不理想，依然存在杂点，且当层数过深时，模型也很难收敛。



4、课程建议

- 可以在平时的课堂中多布置一些实际操作的实验作业，提高动手能力；
- 可以在课堂中多一些作业的交流讨论机会；
- 如果能有机会参观人工智能实验室或者有关企业会更加激发学习兴趣。

5、致谢

感谢黄征老师、王士林老师的悉心指导！感谢杨磊助教、邓鑫瑞助教的关心！感谢人工智能学导论课程组的辛勤付出！