《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：张玓璠 朱林雪 李梓诺 史开源 杨伟程

完成时间： 2023 年 5 月 30 日

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。同时要求：

- 支持随机产生输出图像
- 在 cpu 上有合理的运行时间
- 生成图像主观判断合理，尽可能少出现 OOD
- OOD 诱骗：尽量欺骗分类器，使其将正确生成图像识别成 OOD

2. 具体内容

2.1 实施方案——生成器

针对 MINST 数据集采用 Mehdi 提出的 CGAN 模型。CGAN 是在 GAN 基础上做的一种改进，通过给原始 GAN 的生成器 Generator（简记为 G）和判别器 Discriminator（简记为 D）添加额外的条件信息，实现条件生成模型。条件信息可以是类别标签或者其它的辅助信息 y，通过把 y 加入到 G 和 D 中新的输入层来改良模型：

- 若原始 GAN 生成器输入是噪声信号，类别标签可以和噪声信号组合作为隐空间表示；
- 若原始 GAN 判别器输入是图像数据（真实图像和生成图像），同样需要将类别标签和图像数据进行拼接作为判别器输入。

GAN 使用生成器 G 和判别器 D 进行极大极小博弈定义损失函数：

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

CGAN 使用额外信息 y 与 GAN 中的 x 与 z 合并即为其损失函数：

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

2.2 实施方案——OOD 诱骗

我们假定如果对手分类器对于正常标签("0"~"9")输出的概率很低，那么它会将输入图像判定为 OOD。

我们借鉴了对抗样本的思路：对抗样本的目标是让对手分类器对于一个错误标签("0"~"9")输出的概率很高；而 OOD 诱骗的目标是让对手分类器对于所有非 OOD 标签("0"~"9")输出的概率都很低，以至于将输入图像判定为 OOD。

据此我们将损失函数设定为“分类器对于所有非 OOD 标签("0"~"9")输出的概率的最大值”，且该损失函数越小越好。

有关对抗样本的训练过程，我们参考了 FGSM 对抗算法的思路，且做了一定的简化。算法可大致描述为：

1. 初始化噪声（置零）
2. 遍历噪声的像素点，试图对该像素点做微调，以降低加噪图像损失函数
3. 对上述两步做循环，直至损失函数达到满意的阈值或循环次数达到上限

而上述对抗样本的攻击目标是生成模型内嵌的分类器，我们希望该对抗样本对于其它分类器也有攻击效果，于是从借用了开源的 Mnist 手写数字分类器（不带 OOD 检测）进行测试。经测试，除非加很大的噪声，否则达不到攻击效果，所谓“伤敌一千，自损八百”。

上述现象的原因可能有：

1. 内嵌分类器有鉴别的成分在内，比较容易攻击。

2. 大小 28*28 且仅为单通道灰度图的图像空间较小，训练稳定、鲁棒的分类器较容易，而加噪的余地较小、实现对抗样本攻击较困难。
3. 我们的对抗样本算法还不够成熟。

最终我们做出权衡：决定在不影响生成图像质量的前提下，仅做尽力而为的 OOD 诱骗。

噪声大小的参数是可调的。若需要增强 OOD 诱骗功能，也可以通过调参，生成质量较低、人眼可辨、能够 OOD 诱骗的图像的。

（2）核心代码分析

判别器：其输入为图像张量，输出为数字类别（包括 OOD 一共 10 类），代码中定义了其卷积层、池化层、全连接层以及向前传播过程：

```python
#判别器，10分类，输入图像张量 x，输出类别
class Discriminator(nn.Module):

    #  卷积层和池化层
    def __init__(self):
        super(Discriminator, self).__init__()

        self.dis = nn.Sequential(
            nn.Conv2d(1, 32, 5, stride=1, padding=2),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((2, 2)),
            nn.Conv2d(32, 64, 5, stride=1, padding=2),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((2, 2))
        )

        # 全连接层
        self.fc = nn.Sequential(
            nn.Linear(7 * 7 * 64, 1024),
            nn.LeakyReLU(0.2, True),
            nn.Linear(1024, 10),
            nn.Sigmoid(),
            # nn.Softmax(dim=1)
        )

    def forward(self, x):
        x = self.dis(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

生成器：其输入为噪声向量，输出为大小为 batch_size×1×28×28 的张量，代码中定义了其序列生成以及向前传播过程：

```python
#生成器，输入的噪声向量x,输出的值为大小为 batch_size x 1 x 28 x 28 的张量，表示生成的图像
class Generator(nn.Module):

    def __init__(self, input_size, num_feature):
        super(Generator, self).__init__()
        self.fc = nn.Linear(input_size, num_feature)  # 1*56*56
        self.br = nn.Sequential(
            nn.BatchNorm2d(1),
            nn.ReLU(True)
        )

        self.gen = nn.Sequential(
            nn.Conv2d(1, 50, 3, stride=1, padding=1),
            nn.BatchNorm2d(50),
            nn.ReLU(True),
            nn.Conv2d(50, 25, 3, stride=1, padding=1),
            nn.BatchNorm2d(25),
            nn.ReLU(True),
            nn.Conv2d(25, 1, 2, stride=2),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.fc(x)
        x = x.view(x.size(0), 1, 56, 56)
        x = self.br(x)
        x = self.gen(x)
        return x
```

二者的训练过程：设定生成器和判别器的训练周期为 gen_epoch，dis_epoch。在每一轮判别器的训练周期中训练生成器。在两个训练过程中，将得到的损失函数 loss、生成器在判别器中的得分和真实图像在判别器中的得分引入到新一轮的训练当中：

```
for i in range(dis_epoch):

    for (img, label) in train_loader:
        labels = np.zeros((Batch_size, 10))
        labels[np.arange(Batch_size), label.numpy()] = 1
        img = Variable(img).cuda()

        real_label = Variable(torch.from_numpy(labels).float()).cuda()
        fake_label = Variable(torch.zeros(Batch_size, 10)).cuda()

        # loss & score of real
        real_out = dis(img)
        dis_loss_real = loss_function(real_out, real_label)
        real_score = real_out

        # loss & score of fake
        noise = Variable(torch.randn(Batch_size, z_dimension)).cuda()
        fake_img = gen(noise)
        fake_out = dis(fake_img)
        dis_loss_fake = loss_function(fake_out, fake_label)
        fake_score = fake_out

        # Back Propagation & Optimize
        dis_loss = dis_loss_real + dis_loss_fake
        dis_optimizer.zero_grad()
        dis_loss.backward()
        dis_optimizer.step()

        # train the Generator
        for j in range(gen_epoch):
            noise = torch.randn(Batch_size, 100)
            noise = np.concatenate((noise.numpy(), labels), axis=1)
            noise = Variable(torch.from_numpy(noise).float()).cuda()
            fake_img = gen(noise)
            output = dis(fake_img)
            gen_loss = loss_function(output, real_label)

            # Back Propagation & Optimize
            gen_optimizer.zero_grad()
            gen_loss.backward()
            gen_optimizer.step()
            final = real_label
```

测试结果：测试结果如图 1-6 所示。从结果可以看出训练周期在约 40 轮后，判别器和生成器的损失函数小。生成器生成图像在判别器中的得分接近 0，真实图像在判别器中的得分接近 0.1，表明能较好地区分生成图像和真实图像。

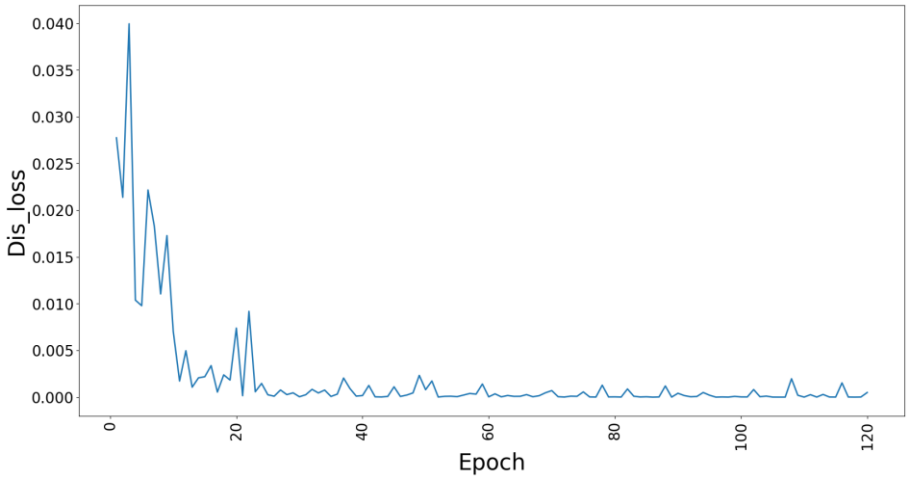

图 1 判别器的损失函数值

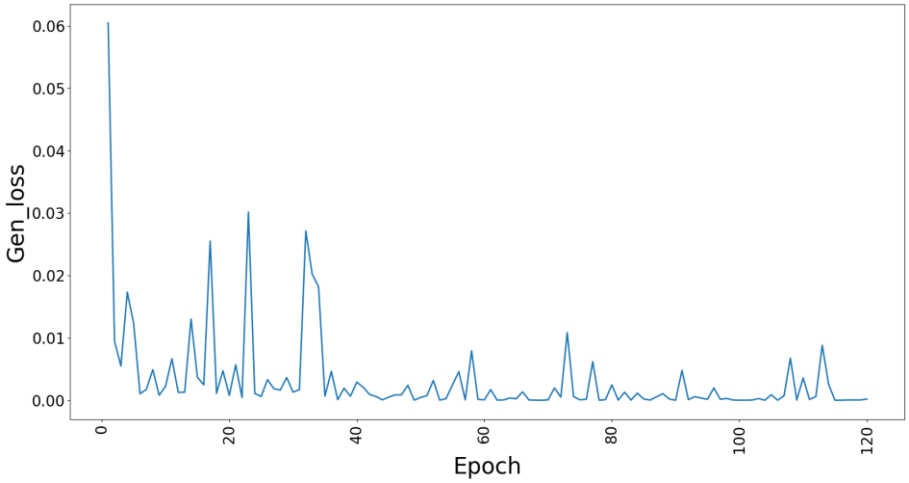

图 2 生成器的损失函数值

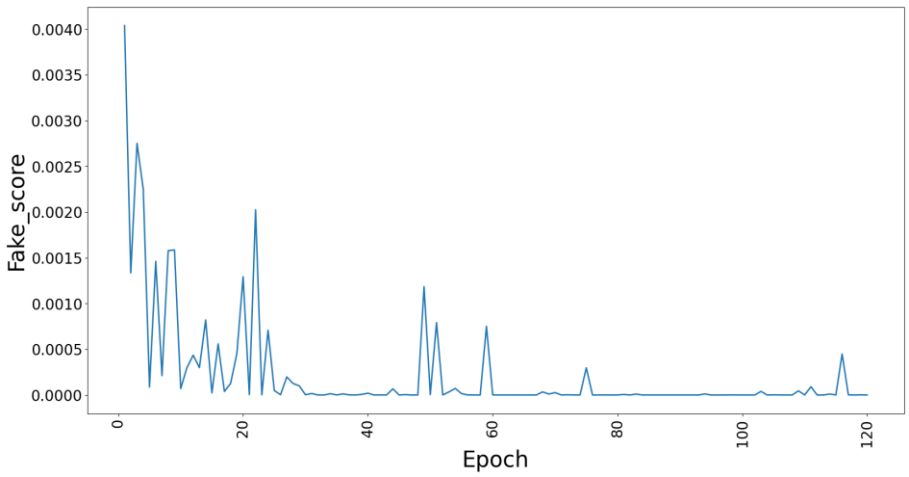

图 3 生成图像在判别器中的得分

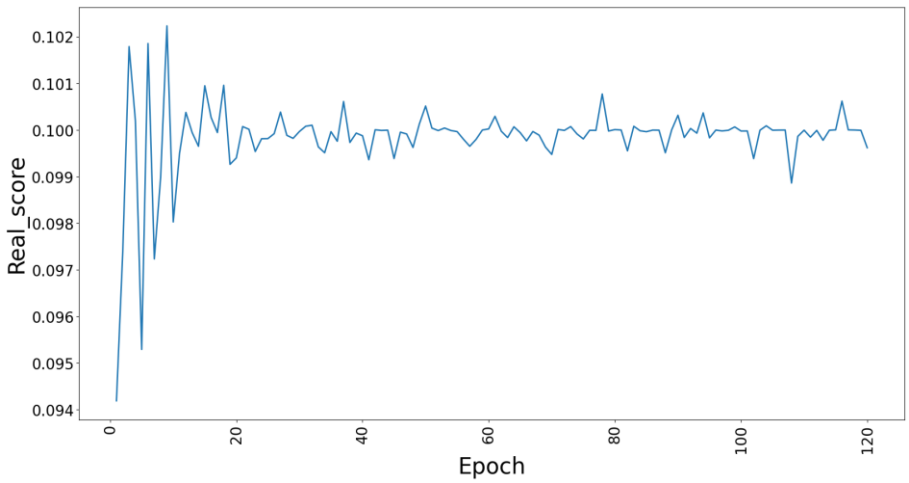

图 4 真实图像在判别器中的得分

图 5 前 60 轮生成图像结果

图 6 第 32 轮训练得到生成图像结果

OOD 诱骗模块：设计为单个函数，输入原始图像和目标分类器，输出对抗样本图像。

```python
def ad(self, img): # img是生成器的输出
    epsilon = 0.04
    noise = torch.zeros([1, 1, 28, 28])
    noises = torch.zeros([28*28, 1, 28, 28])
    for i in range(28):
        for j in range(28):
            noises[28*i+j, 0, i, j] = epsilon
    loss_0 = torch.max(self.discriminator(img))
    # print(loss_0)
    loss_now = loss_0
    ad = img + noise
    for i in range(5):
        if float(loss_now) < 0.5:
            break
        losses = torch.max(self.discriminator(ad + noises),dim=1).values
        noise += ((losses <= loss_0) * epsilon).reshape(1, 1, 28, 28)
        noise -= ((losses > loss_0) * epsilon).reshape(1, 1, 28, 28)
        ad = img + noise
        # 我觉得想出下面两行有点费劲，不知道有没有方便的函数可以调用
        ad *= (ad >= torch.zeros(1, 1, 28, 28)) # 将小于0的像素置0
        ad = ad - ad * (ad > torch.ones(1, 1, 28, 28)) + (ad > torch.ones(1, 1, 28, 28)) # 将大于1的像素置1

        loss_now = torch.max(self.discriminator(ad))

    return ad
```

测试结果如图 7、8 所示，可以看出分类器的最大输出概率（即损失函数）较小，同时生成图像在主观上合理。

图 7 生成的若干对抗样本

```
Input your number: 0 1 2 3 4 5 6 7 8 9
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
tensor(0.0058, grad_fn=<MaxBackward1>)
tensor(0.0887, grad_fn=<MaxBackward1>)
tensor(0.9987, grad_fn=<MaxBackward1>)
tensor(0.4756, grad_fn=<MaxBackward1>)
tensor(0.1923, grad_fn=<MaxBackward1>)
tensor(0.0450, grad_fn=<MaxBackward1>)
tensor(0.0137, grad_fn=<MaxBackward1>)
tensor(0.1566, grad_fn=<MaxBackward1>)
tensor(0.1288, grad_fn=<MaxBackward1>)
tensor(0.1968, grad_fn=<MaxBackward1>)
```

图 8 分类器对于上述图像的最大输出概率

3. 工作总结

（1）收获、心得

在本次大作业中，我们加深了对条件生成对抗网络的理解：通过

一步步构建 CGAN 模型，我们更深入地了解了 GAN 与 CGAN 模型的原理和实现方式，以及在进行模型训练过程中的对抗样本对模型的影响。

总的来说，本次大作业为我们提供了一个深入了解人工智能技术的机会，让我们深入地理解了其在未来的重要性。

（2）遇到问题及解决思路

在开发过程中我们遇到了以下问题：

- 环境配置过程中，自主安装 pytorch 由于显卡硬件问题报错，使用 anaconda 后统一配置环境解决问题。
- 模型训练过程中，由于体量比较大，训练过程较慢。改用硬件配置较高的设备一定程度上解决了该问题。
- 对于代码管理，由于此前使用 git 机会不多，故而在初期熟练度不高。

4．课程建议

- 希望针对每次上课讲的模型可以开设 tutorial 或者阅读材料，以从实际编写程序的角度更清晰地理解模型。
- 针对大作业可以提供相关的阅读材料，以减少学生浪费在筛选信息的时间。
- 可以提供一个 FAQ 板块，有针对性地回答在做大作业时可能遇到的问题（例如环境配置，提供实验室设备）。