机器学习与数据挖掘

期末课程报告

一、研究问题的背景和动机

本次课程报告的主题是生成模型，生成模型可以描述成一个生成数据的模型，属于一种概率模型。区分于判别模型，生成模型关注点在于样本的概率分布本身，是对样本分布进行建模，判别模型往往带有需要解决的任务，比如划分样本聚类。通过生成模型能够生成不包含在原有数据集中的新的数据，能够有效用于半监督学习中，降低了获取数据样本的难度。在之前的课程作业中我们主要都是训练模型进行判别，没有涉及到生成数据的部分，也有使用过生成模型，如使用混合高斯模型进行聚类，包括朴素贝叶斯这类生成模型都属于浅层的生成模型，结构相对固定。深度生成模型包括 VAE、GAN 等，这类模型通过学习数据分布生成新的数据样本，因此主要研究问题就在于如何提高生成样本的质量和多样性，研究方向包括图像生成与增强，自然语言处理，音频合成处理这几大领域，本次报告我将就图像生成这一部分展开具体研究与实验。

二、解决问题的主要方法

基本的深度生成模型都能够用于图像生成，深度生成模型的本质都是缩小数据分布与模型分布的距离，求解这一问题可以将深度生成模型分为三类。一般问题通过处理极大似然函数来求解，但深度生成模型的结构相对复杂，在此基础上进行一定变通。

一是通过变分或抽样的方法求似然函数的近似分布，典型代表是变分自编码器（VAEs）。VAEs 用似然函数的变分下界作为目标函数，使用编码器将输入映射到潜在空间，然后使用解码器从潜在空间中生成样本。

二是对似然函数进行变形来简化计算，主要有流模型和自回归模型这两种模型。流模型利用可逆网络构造似然函数之后直接优化模型参数，训练出的编码器利用可逆结构的特点直接得到生成模型。自回日模型则是将目标函数分解为条件概率乘积的形式。

三是避开求极大似然过程的隐式方法，代表模型为生成对抗网络（GANs）。利用神经网络的学习能力来拟合两个分布之间的距离，具有很多代表性的模型，如 DCGAN、StackGAN、PGGAN、StyleGAN 和 BigGAN；神经网络的泛化能力、鲁棒性具有很大的研究价值，GAN 主要有生成器和判别器两部分网络模型组成，生成器的目标是生成真实样本欺骗判别器，判别器则是锻炼区分真实样本和生成样本的能力，通过对抗训练不断提高各自的能力，达到纳什均衡的状态。生成器能够实现从噪声到样本的直接转换，具有强大的图像生成能力，使其成为当前图像处理领域的主流选择，本次实验内容就是使用 DCGAN 进行简单的图像生成。

另外还有近期兴起的扩散模型，相对 GAN 具有更加灵活的模型架构和精确的似然计算，主要包括前向扩散把随机噪声加入样本，逆向扩散从噪声中生成样本，该方法在发展之初就展现出优异的图像生成效果，相信有关扩散模型的研究将会进一步推进图像生成领域的发展。

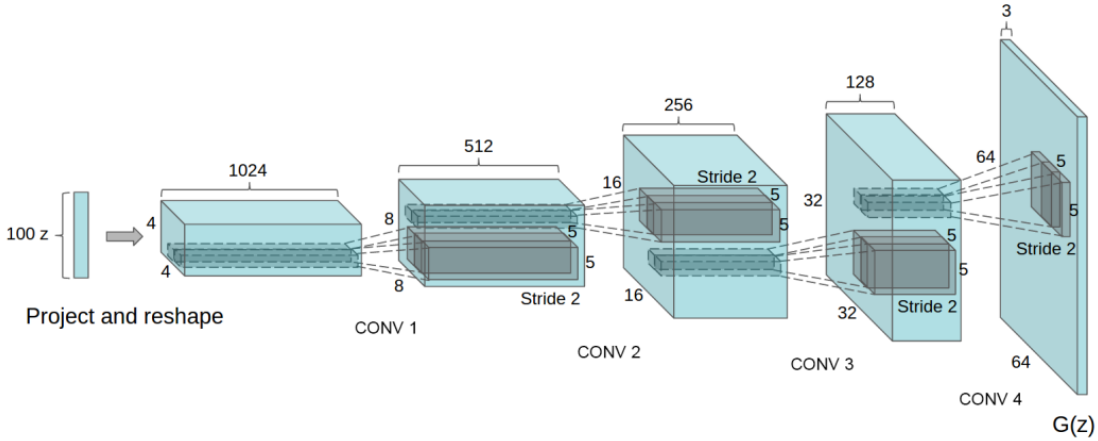
三、DCGAN 模型

传统 GAN 的目标公式如下

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

x 表示真实图像，G(**z**)表示生成图像
D 表示判别器，目标是最大化 V(D, G)，即令 D(**x**)逼近 1，D(G(**z**))逼近 0
G 表示生成器，目标是最小化 V(D, G)，即令生成的 G(**z**)在 D 中逼近 1

DCGAN 是 GAN 的变体，将 DNN 应用到了 GAN 上，主要思想与 GAN 相同，DCGAN 利用卷积神经网络来构造生成器和判别器，如下图，前向表示生成器，是卷积的逆过程，逆向则是判别器，是一个典型的卷积神经网络。



生成器：输入均值为 0 方差为 1 的随机噪声，通过多个连续 ConvTranspose2d- BatchNorm2d-ReLU 组成，最后通过 Tanh 输出图像
判别器：即生成器的逆过程，卷积层使用 Conv2d，激活函数使用 Leaky ReLU， 同样使用 BatchNorm2d 对特征图进行归一化。不断重复，直到输出 1x1x1 的特征图，使用 Sigmoid 激活函数将特征图压缩到[0,1]，表示图像真实的概率

主要代码如下：使用了 torch 深度学习框架
生成器

```
class Generator(nn.Module):
    def __init__(self, latent_dim, img_channels):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, 64, 3, 1, 0),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 32, 3, 2, 0),
            nn.BatchNorm2d(32),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 16, 4, 2, 1),
            nn.BatchNorm2d(16),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, img_channels, 4, 2, 1),
            nn.Tanh()
        )
    def forward(self, x):
        return self.main(x)
```

判别器:

```
class Discriminator(nn.Module):
    def __init__(self, img_channels):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(img_channels, 16, 4, 2, 1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(16, 32, 4, 2, 1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(32, 64, 3, 2, 0),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 1, 3, 1, 0),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.main(x)
```

主要训练过程:

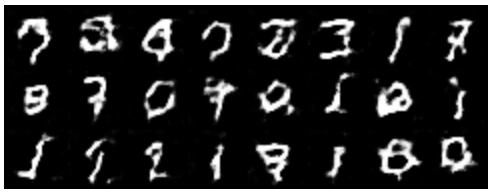
```
for epoch in range(epochs):
    for i, data in enumerate(dataloader, 0):
        real_imgs, _ = data
        real_imgs = real_imgs.to(device)
        # 训练判别器
        optimizer_D.zero_grad()
        real_labels = torch.full((batch_size, 1), 1.0, device=device)
        fake_labels = torch.full((batch_size, 1), 0.0, device=device)
        # 真实图像的损失
        output_real = discriminator(real_imgs).resize(batch_size,1)
        loss_real = criterion(output_real, real_labels)
        # 假图像损失
        noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)
        fake_imgs = generator(noise)
        output_fake = discriminator(fake_imgs.detach()).resize(batch_size,1)
        loss_fake = criterion(output_fake, fake_labels)
        # 总损失
        loss_D = loss_real + loss_fake
        loss_D.backward()
        optimizer_D.step()
        # 训练生成器
        optimizer_G.zero_grad()
        output = discriminator(fake_imgs).resize(batch_size,1)
        loss_G = criterion(output, real_labels)
        loss_G.backward()
        optimizer_G.step()
```

训练判别器时不需要对生成器进行优化，因此 fake_imgs 后加了 detach

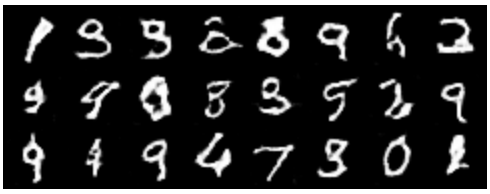
训练生成器的时候直接将判别器 label 设为真，即可计算反向的损失，不需要将梯度反向。

四、实验结果及分析

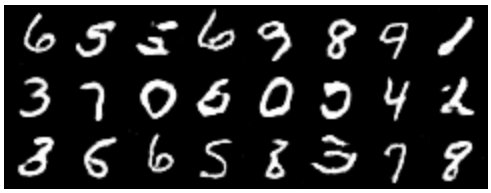
使用 MNIST 手写数字集进行训练，生成随机手写数字，效果如下



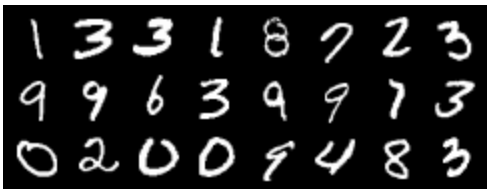
迭代 1 轮图片



迭代 20 轮图片

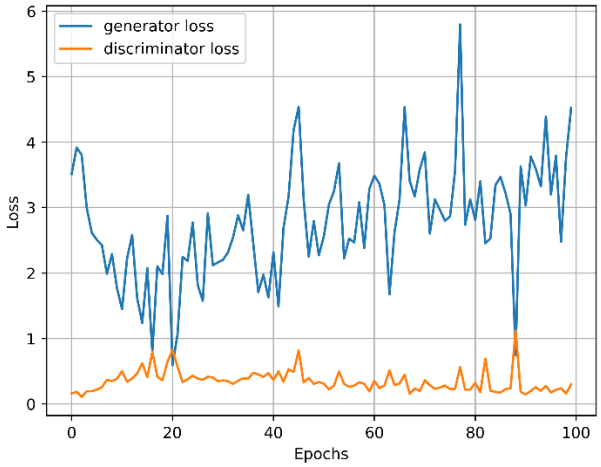


迭代 100 轮图片



原图

训练过程生成器和判别器的 loss 如下图



可以看出生成器模型在前 20 轮保持下降趋势，但第 20 轮的效果并不是最好的，可推断随着判别器的优化，生成器的优化速度低于判别器，损失反而上升，但已经基本呈现数字的形状，比对迭代 100 轮图片与原图，生成模型基本能够生成与原图类似的手写数字。

五、结论

本次课程报告对生成模型进行一定的探索研究，并对 DCGAN 生成图像进行了实验尝试，完成了手写数字的生成，主要遇到问题出现在软件包调用不够熟练以及调整参数上面，整体的代码量不大，就能够实现图像的生成，并且对各类图像都由一定使用性，这也得益于卷积神经网络的泛化能力，将卷积神经网络与生成模型结合这个创新点具有很大的意义。时间问题，对于 DCGAN 还有许多值得实践尝试的地方，比如本次实验是直接使用随机噪声作为输入随机生成手写数字，而手写数字集本来就存在 10 个不同类别的标签，统一作为真实图片进行学习难免会在不同类别间互相干扰，将类别作为输入也许会有更好的效果。但也仍然存在着一些问题，比如还未找出模型训练不收敛的问题。总的来说，本次课程实验主要将重心放在了 GAN 的原理学习和实验上，学到了有用的知识和实验技巧，但也仍停留在认知浅薄的阶段，希望今后能在此方面作出更多的探索实践。