实验五：生成式对抗网络

李旻翀
1190200208

日期：June 13, 2022

摘　　要

本实验为模式识别与深度学习课程的实验五：生成式对抗网络。在本次实验中，我利用 Pytorch 自己实现了 GAN，WGAN，WGAN-GP 三种网络的结构，拟合 points.mat 中的数据分布，然后对比了三种模型的效果。除此之外，我基于给定的 ProGAN 代码和模型，实现 ProGAN 模型的 SeFa 部分，完成了隐空间语义方向搜索的任务。

关键词：生成式对抗网络，GAN，WGAN，WGAN-GP，ProGAN

1　分布拟合任务

1.1　网络结构

在本次实验中，我利用 Pytorch 自己实现了 GAN，WGAN，WGAN-GP 三种网络的结构（分别保存在根目录下的 gan.py，wgan.py，wgan_gp.py 中）。GAN 的 Generator 和 Discriminator 结构如下：

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_size, 64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(64, 256),
            nn.BatchNorm1d(256, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 2)
        )

    def forward(self, x):
        return self.net(x)
```

图 1: generator 结构

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(2, 64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(64, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.net(x)
```

图 2: discriminator 结构

相较于普通的 GAN，WGAN 主要有 4 点改进：

1. 去掉 discriminator 的最后一层 Sigmoid
2. 将 discriminator 的 w 取值限制在 [-c,c] 区间内, 确保 lipschitz 连续
3. 使用不带有 log 函数的 loss
4. 不使用具有动量的优化方法（如 Adam）

而 WGAN-GP 相较 WGAN，引入了梯度惩罚项。

1.2 训练过程

实验中，首先读取 points.mat 的'xx' 维度，并将样本随机打乱顺序，然后取其中的 7000 个点进行训练；剩余的 1192 个点用来画分布图，来验证生成的数据是否拟合该分布。

在训练时，batch size 设定为 256，学习率设定为 0.00005，生成器输入噪声维度设定为 2，优化器均采用 RMSProp。进行 320 个 epoch 的训练，结果如下：

1.2.1 GAN 训练结果

GAN 的训练结果如图所示:

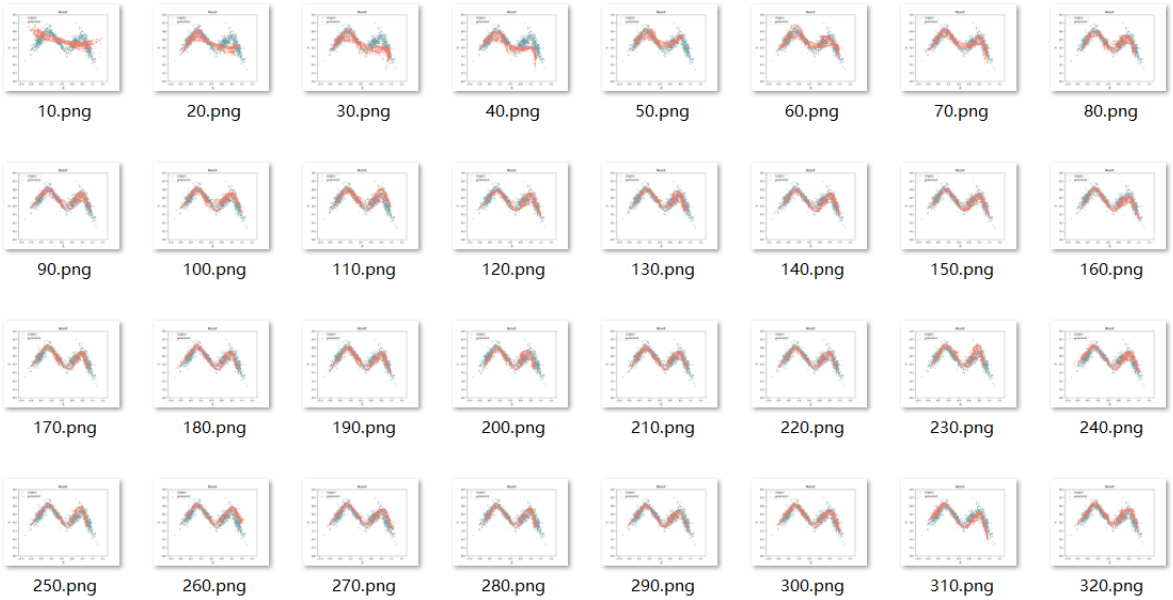


图 3: GAN 训练结果

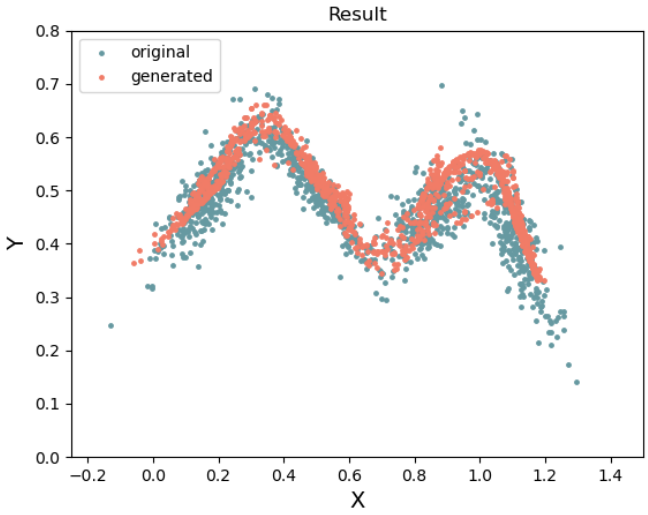


图 4: 320 个 epoch 后的 GAN 训练结果

可以看出，GAN 的训练结果较好，在训练约 50 个 epoch 之后就有较好的效果，且最终比较能够保持稳定。

1.2.2 WGAN 训练结果

WGAN 的训练结果如图所示：

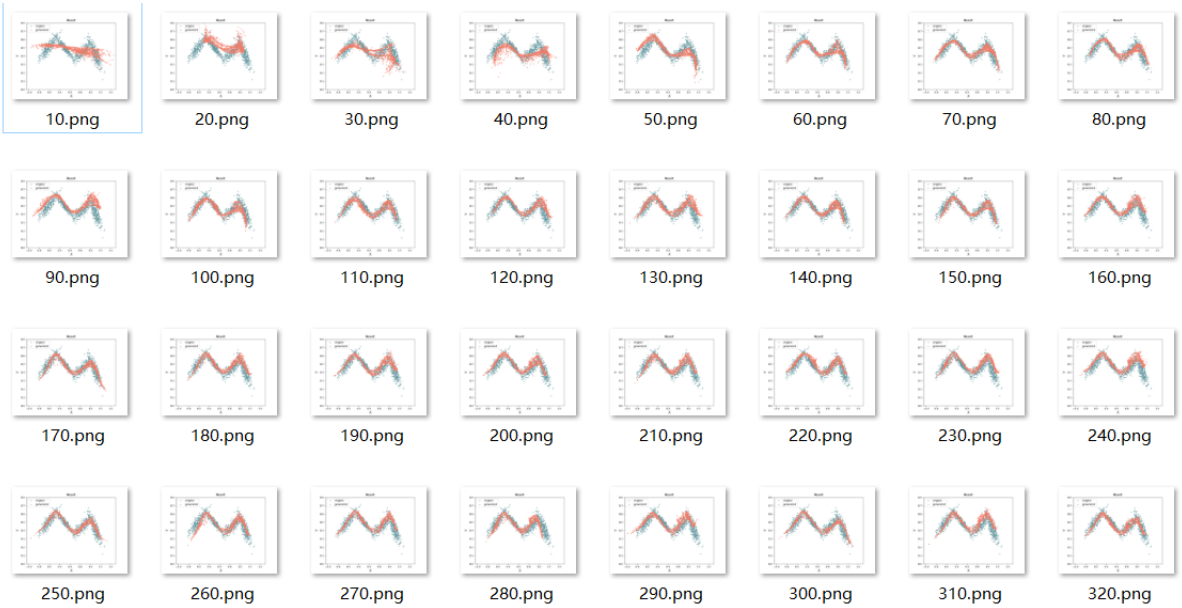


图 5: WGAN 训练结果

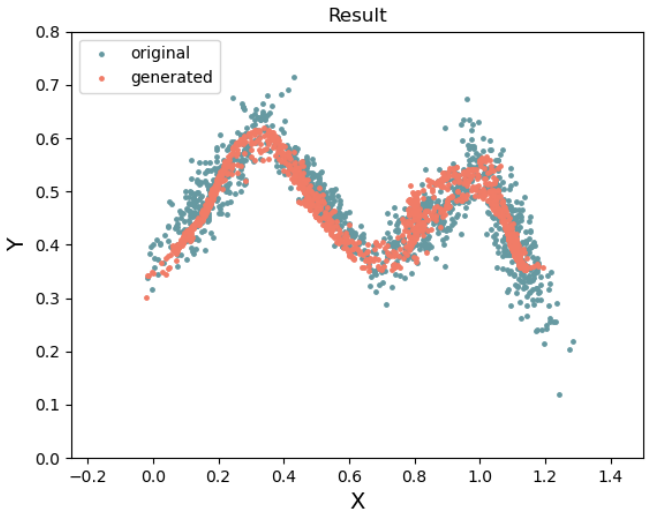


图 6: 320 个 epoch 后的 WGAN 训练结果

可以看出，WGAN 的训练结果和 GAN 类似较好，在训练几十个 epoch 之后就有较好的效果，且最终比较能够保持稳定。

1.2.3 WGAN-GP 训练结果

WGAN-GP 的训练结果如图所示：

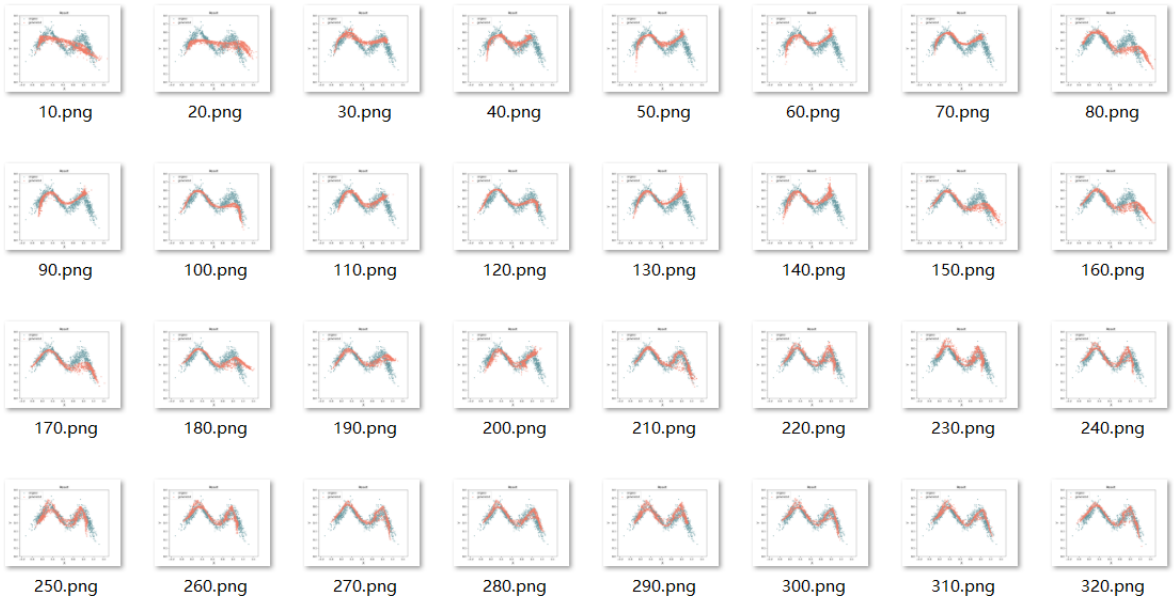


图 7: WGAN-GP 训练结果

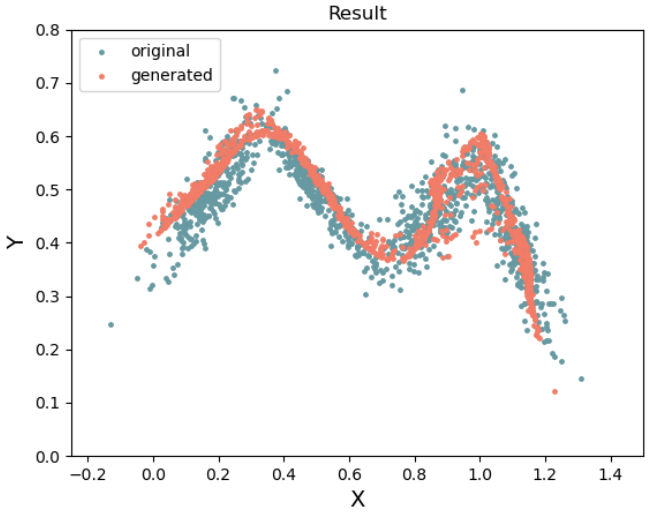
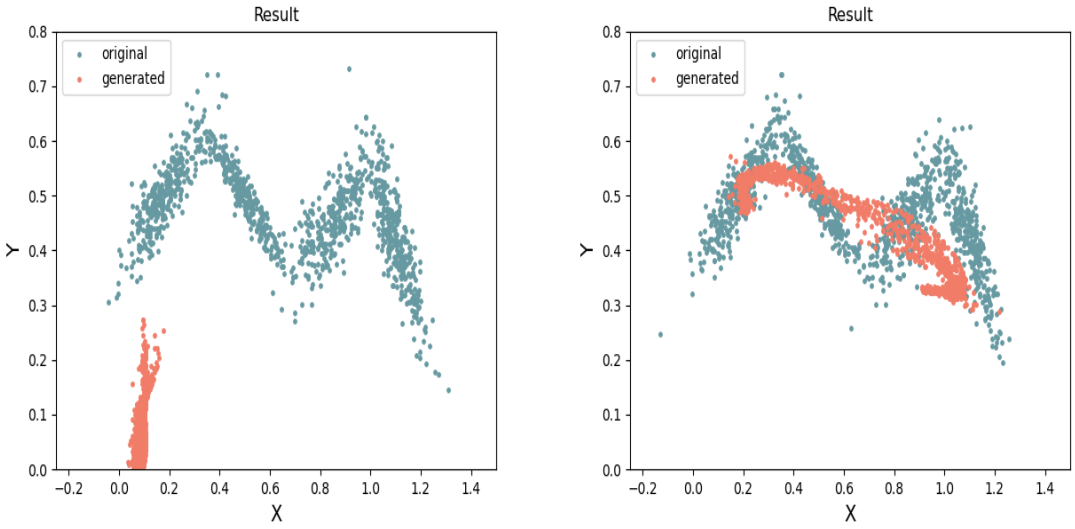


图 8: 320 个 epoch 后的 WGAN-GP 训练结果

可以看出，WGAN-GP 的训练结果收敛较慢，在训练大约 200 个 epoch 之后才能较为贴合原数据分布。

1.2.4 更换优化器

使用 GAN 模型，将优化器换为 SGD 与 Adam，对比其效果如下：



(a) 使用 SGD　　(b) 使用 Adam

图 9: 换用不同优化器，训练 320 个 epoch 后的实验结果

可以看出，换用 SGD 优化器后，拟合图形几乎从开始到结束都没有移动，说明训练速度很慢，而 Adam 优化器的拟合图形则波动非常大，十分不稳定。两种优化器的效果都不如 RMSProp。

1.2.5 实验结果对比分析

综上所述，三种网络模型在使用 RMSProp 优化器时，都能在 320 个 epoch 内得到很好的拟合效果，其中，WGAN-GP 的速度稍慢，但点的分布更均匀（表现为拟合图形更加纤细）。而换用其他优化器则很难得到很好的效果。

最终结果的相关动图可以运行 draw.py 生成，保存在./gif 下。

2 隐空间语义方向搜索

在这一部分中，我补全了 sefa.py 中的相关代码，主要完成的任务是分解 layer0 的权重，从而得到 directions 的参数。具体的补全代码如下：

```
################
# factorize the weight of layer0 to get the directions
# run: python sefa.py pggan_celebahq1024 --cuda false/true

temp = torch.mm(weight, weight.T)
evals, evecs = torch.eig(temp, eigenvectors=True)
evals_arg = torch.argsort(evals[:, 0], descending=True)

for i in range(num_sem):
    v = evecs[:, evals_arg[i]]
    for j in range(num_sam):
        directions[i][j] = v
################
```

图 10: sefa.py 中补全的代码部分

补全完成后，下载模型文件，运行 sefa.py，可以得到如下结果：

图 11: 模型训练结果

视频保存在./work_dirs/synthesis/pggan_celebahq1024_N5_K5_seed0 下。

3 说明文档

本次实验由两部分组成，数据拟合相关的文件保存在./GAN 下，隐空间语义方向搜索相关文件保存在./genforce 下。