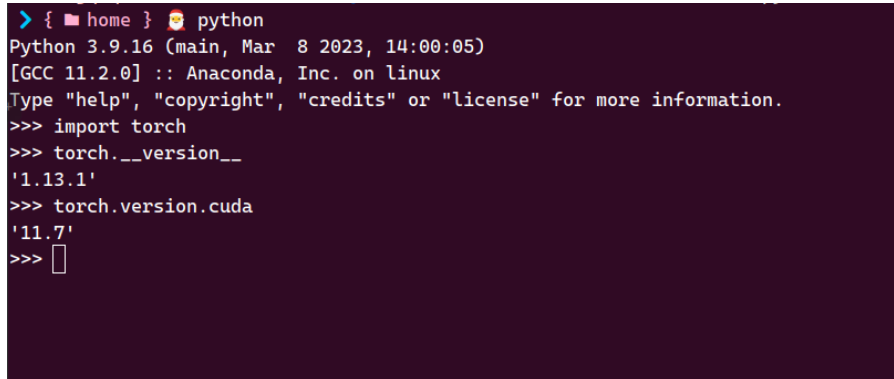


# 实验 5-生成式对抗网络实现

符兴 7203610316

## 1 实验环境

Ubuntu 20.04, Python 3.9, PyTorch1.13, Cuda11.7, TensorBoard;



```
> { home } python
Python 3.9.16 (main, Mar 8 2023, 14:00:05)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1'
>>> torch.version.cuda
'11.7'
>>> 
```

图 1 Python 环境

## 2 实验过程

### 2.1 模型构建

```
class Discriminator(nn.Module):
    def __init__(self, isSigmoid = False):
        super(Discriminator, self).__init__()
        self.isSigmoid = isSigmoid
        self.sigmoid = nn.Sigmoid()
        self.net = nn.Sequential(
            nn.Linear(2, 128),
            nn.LeakyReLU(),
            nn.Linear(128, 256),
            nn.LeakyReLU(),
            nn.Linear(256, 128),
            nn.LeakyReLU(),
            nn.Linear(128, 1),
        )
    def forward(self, x):
        output = self.net(x)
        if self.isSigmoid == True:
            output = self.sigmoid(output)
        return output.view(-1)
```

图 2 判别器模型

在本次实验中，需要基于 Pytorch 实现 GAN、WGAN、WGAN-GP。这几个模型有两个基本的模型，一个是判别器模型，一个是生成器模型；判别器模型输入的是一个点坐标，输出的是该点坐标是否真实的概率；在 WGAN 和 WGAN-GP 中，去掉了 sigmoid()，因为判别器要拟合的是 Wasserstein 距离，它不是一个 0 或 1 的分类问题，而是回归问题，取值不限于 0 到 1。

```

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 2),
        )

    def forward(self, x):
        output = self.net(x)
        return output

```

图 3 生成器模型

生成器模型是一个生成图片的网络，它接收一个随机的噪声  $z$ ，通过这个噪声生成图片。在训练过程中，生成器  $G$  的目标就是尽量生成真实的图片去欺骗判别器  $D$ 。而  $D$  的目标就是尽可能把  $G$  生成的图片和真实的图片分别开来。这样， $G$  和  $D$  构成了一个动态的“博弈过程”。

在最理想的状态下， $G$  可以生成“以假乱真”的图片  $G(z)$ ，对于  $D$  而言，它难以判定  $G$  生成的图片究竟是不是真实的，因此  $D(G(z)) = 0.5$ 。

## 2.2 模型训练

GAN、WGAN、WGAN-GP 三个模型在结构上没有太大的区别，三者的本质区别在于 Loss 的计算上。

```

if args.model == "GAN":
    lossD = -(torch.log(labelPred)+torch.log(1-fakePred)).mean()
if args.model == 'WGAN':
    # 限制判别器参数大小
    fakePred = D(fakeX)
    if args.model == "GAN":
        lossG = torch.log(1- fakePred).mean()
    else:
        lossG = - fakePred.mean()

```

图 4 GAN 的 Loss

在 GAN 中，先训练判别器  $D$ ，判别器  $D$  是希望  $V(G, D)$  越大越好，所以是加上梯度。第二步训练生成器  $G$  时， $V(G, D)$  越小越好，所以是减去梯度。整个训练过程交替进行。

```

    lossD = -torch.log(labelPred) - torch.log(1 - fakePred).mean()
if args.model == 'WGAN':
    # 限制判别器参数大小
    for p in D.parameters():
        p.data.clamp_(-args.clamp, args.clamp)
    lossD = (fakePred - labelPred).mean()
if args.model == "GAN":
    lossG = torch.log(1 - fakePred).mean()
else:
    lossG = -fakePred.mean()
optimizer.zero_grad()

```

图 5 WGAN 的 Loss

WGAN 和 GAN 的区别在于, WGAN 的 Loss 计算都不再去 log, 用 Wasserstein 距离代替 JS 散度; Wasserstein 距离相比 KL 散度、JS 散度的优越性在于, 即便两个分布没有重叠, Wasserstein 距离仍然能够反映它们的远近。除此之外, 还限制了判别器的参数大小, 将其限制在预设的区间内。

```

def gradPenalty(D, x, fake, device):
    alpha = torch.rand(x.shape[0], 1).to(device)
    alpha = alpha.expand_as(x)
    mid = alpha*x+(1-alpha)*fake
    mid.requires_grad_()
    pred = D(mid)
    grad = torch.autograd.grad(pred, mid, grad_outputs=torch.ones_like(pred),
                                create_graph=True, retain_graph=True,
                                only_inputs=True)[0]
    gp = torch.pow(grad.norm(2, dim=1), -1, 2).mean()
    return gp

```

图 6 WGAN-GP 的 Gradient Penalty

WGAN 在处理 Lipschitz 限制条件时直接采用了权重截断的方式, 这样会限制模型参数固定在一个范围之内, 超出这个范围要么取最大值要么取最小值, 但是随着层数加深可能会出现梯度消失和梯度爆炸的现象。WGAN-GP 在 WGAN 的基础上, 通过 Gradient Penalty 的方式, 设置一个额外的 loss 项来实现梯度与截断阈值之间的联系。

## 2.3 隐空间语义方向搜索

GAN 中的生成器通常以随机采样的隐向量  $z$  作为输入, 生成高保真图像。通过改变隐向量  $z$ , 我们可以改变输出图像。然而, 为了改变输出图像中某些特定的属性(如面部表情、姿势、肤色等), 我们需要知道隐向量  $z$  的特定移动方向。在本次实验中, SeFa 是一种无监督的方式, 无需数据采样和模型训练就能

找出这些方向向量来改变输出图像中的属性。

SeFa 对图像的操作，可以看做是将  $d$  维潜在空间中的对应向量  $z$  沿着  $n$  的方向进行移动。

$$\text{edit}(G(Z)) = G(Z') = G(Z + \alpha n)$$

而 GAN 还会将  $z$  映射到另一个  $m$  维空间的  $y$ 。

$$G(Z) = y = Az + b$$

最终，将这一问题转化为：

$$N^* = \arg \max_{N \in R^{N*d}} \sum_{i=1}^k \|An_i\|_2^2$$

根据拉格朗日法，求出的特征向量，找出  $k$  个使图像进行变换的信息，即选择特征值最大的  $k$  个特征向量作为方向。然后，给定一定的步长，将其信息加入生成器的输入向量  $z$  中进行训练，得到特定的图像语义变换。

### 3 实验结果

#### 3.1 不同网络结构、不同优化器实验结果对比

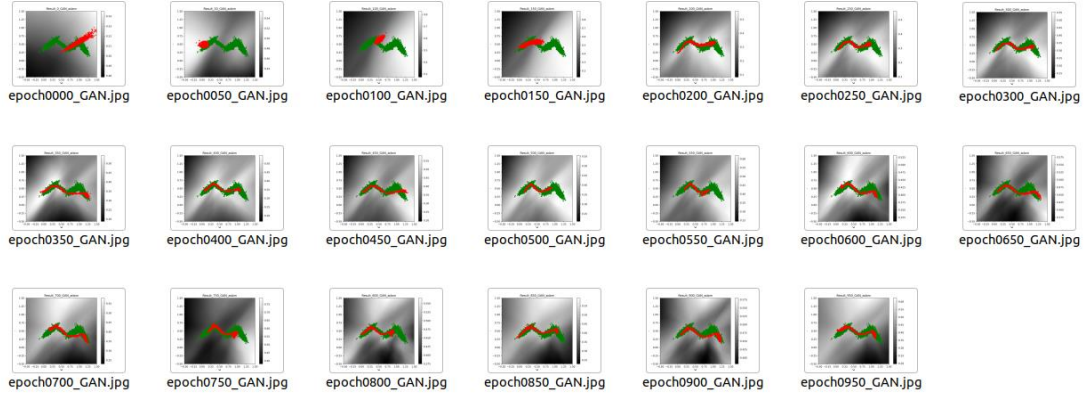


图 7 GAN-Adam 的训练结果

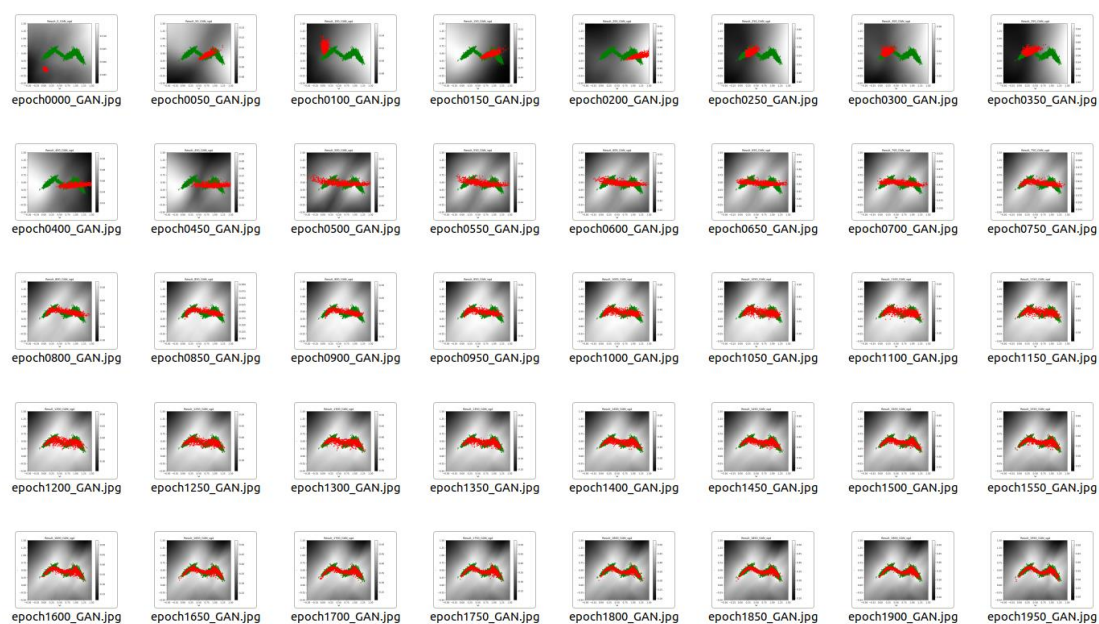


图 8 GAN-SGD 的训练结果

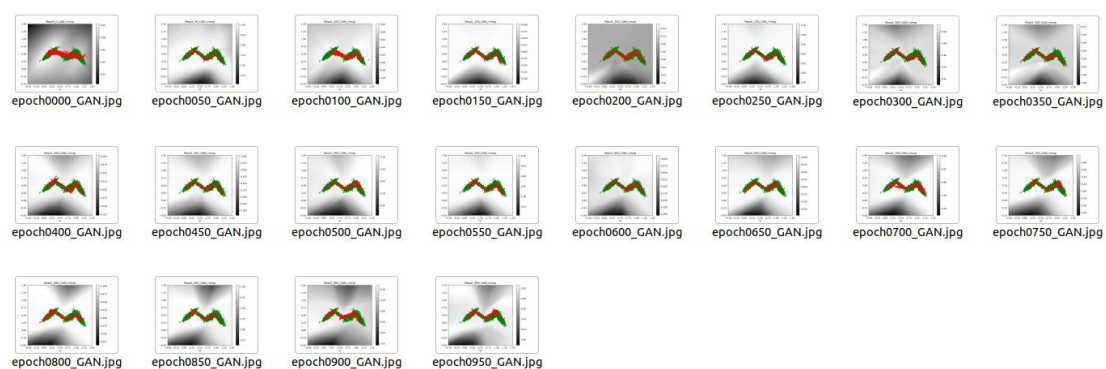


图 9 GAN-RMSprop 的训练结果

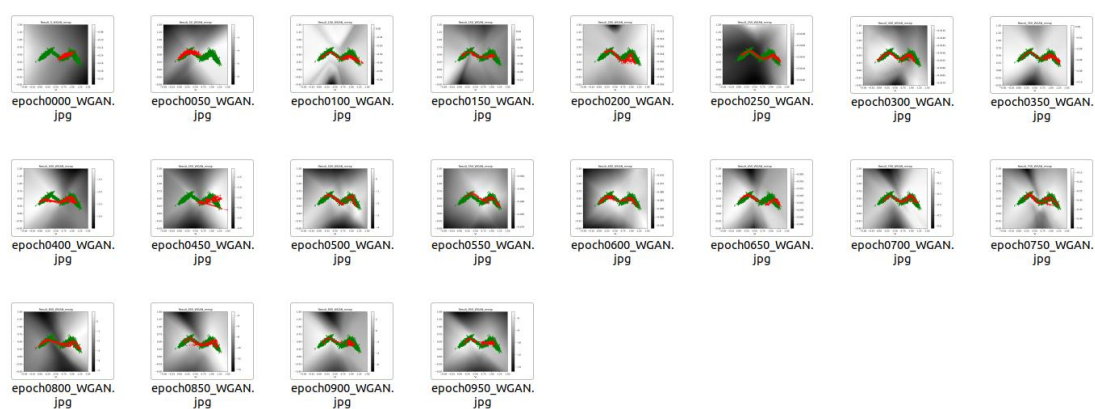


图 10 WGAN-RMSprop 的训练结果

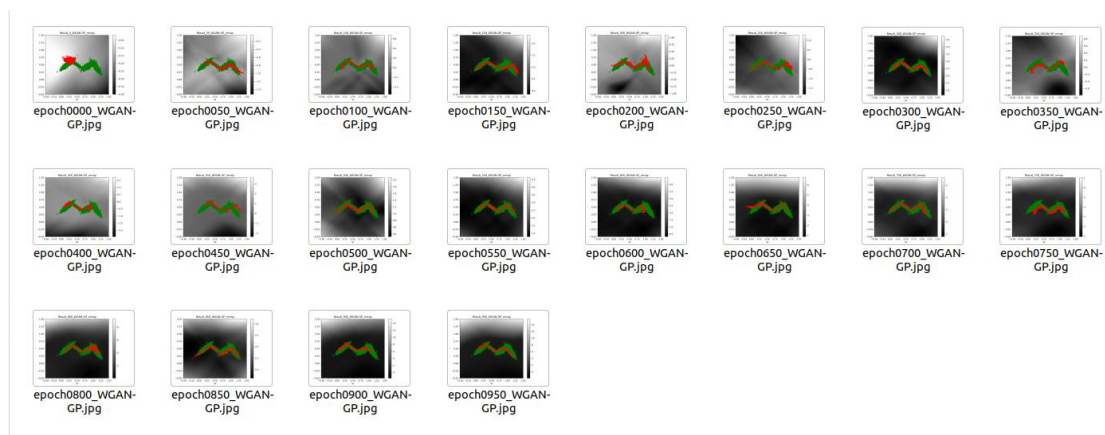


图 11 WGAN-GP-RMSprop 的训练结果

对比上面的实验结果可以发现，WGAN、WGAN-GP 模型的收敛速度较快于 GAN 网络；除此之外，在训练 GAN 时不同的优化器也具有明显的差异：SGD 优化器收敛速度最慢，需要接近 2000 个 Epoch 才收敛；Adam 收敛速度次之，RMSprop 收敛速度最快，效果也是最好的。

### 3.2 隐空间语义方向搜索实验结果

在运行程序后，保存了五张效果图，可以发现找到的语义方向分别为：性别、脸部朝向、表情等。

## 4 心得与体会

在进行本次实验后，我深刻体会到了其在计算机视觉领域的巨大潜力。在实验中，我深入了解了 GAN、WGAN、WGAN-GP 等模型的基本原理和工作机制。在这个过程中也遇到了很多问题，比如学习率过大，由于 Loss 的不稳定导致结果非常丑陋等等。