实验五：生成式对抗网络实现

120L021716 蔡泽栋

一、实验目的：

基于 Pytorch 实现 GAN、WGAN、WGAN-GP，拟合给定分布（points.mat）。要求可视化训练过程

隐空间语义方向搜索

二、实验环境：

Python 3.9.13; torch =2.0.0+cu117; torchvision=0.15.0+cu117; GPU

```
python                    3.9.13          h6244533_2
python-dateutil           2.8.2             pypi_0
pytorch-mutex             1.0                 cpu

pytz                      2023.3            pypi_0
regex                     2023.3.23         pypi_0
requests                  2.28.1       py39haa95532_1
scikit-learn              1.2.2             pypi_0
scipy                     1.10.1            pypi_0
setuptools                65.6.3       py39haa95532_0
six                       1.16.0         pyhd3eb1b0_1
sqlite                    3.41.1          h2bbff1b_0
sympy                     1.11.1       py39haa95532_0
threadpoolctl             3.1.0             pypi_0
tk                        8.6.12          h2bbff1b_0
torch                     2.0.0+cu117       pypi_0
torchvision               0.15.0+cu117      pypi_0
tqdm                      4.65.0            pypi_0
```

三、实验内容：

1. 网络结构实现：

对抗生成网络有生成器和判别器两个部分，生成器通过训练使得生成的图片逼近训练集，判别器通过训练分类器判断输入图片的真假。本次实验拟合分布二维分布，生成器最后通过全连接层输出一个二维的张量。判别器最后通过 sigmoid 函数生成一个概率值。三个网络结构本质上是一样的，只在训练过程中计算损失的方法不一样。

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            # z: [b, 2] => [b, 2]
            nn.Linear(10, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 2),
        )

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

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            # [b, 2] => [b, 1]
            nn.Linear(2, 128),
            nn.LeakyReLU(),
            nn.Linear(128, 256),
            nn.LeakyReLU(),
            nn.Linear(256, 128),
            nn.LeakyReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        output = self.net(x).view(-1)
        return output
```

2. 数据读取与预处理：

从 point.mat 文件中读入，转换为 tensor 后再封装在 DataLoader 中。

```
class Points(Dataset):
    def __init__(self):
        self.data = mat4py.loadmat("./data/points.mat")['xx']

    def __getitem__(self, idx):
        xy = torch.tensor(np.array(self.data[idx])).to(torch.float32)
        return xy

    def __len__(self):
        return len(self.data)
```

3. 定义优化器并训练

本次实验优化器选用 Adam、Sgd 和 RMSprop。对于每个模型，进行 1000 个 epoch 训练，每 50 轮保存一次当前训练情况。

本次实验所用的损失函数并不是真正意义上的损失函数，因为生成模型希望生成的概率期望越大越好，当于其负数越小越好。

GAN 的训练过程中，通过交替训练生成器和判别器来优化模型。生成器和判别器的损失函数是通过最大化判别器对真实样本的概率和最小化对生成样本的概率来定义的。

```
# 计算判别器损失
D_loss = - (torch.log(real_outputs) + torch.log(1. - fake_outputs)).mean()
```

WGAN 的训练过程中，使用 Wasserstein 距离作为判别器的损失函数，而不是传统 GAN 中的对数似然损失。判别器的输出是一个实数，而不是通过激活函数输出概率。通过剪裁（clipping）判别器的权重来满足 Lipschitz 连续性条件，以确保梯度的稳定性。

```
# 限制判别器的权重范围
if args.model == 'WGAN':
    for p in D.parameters():
        p.data.clamp_(-args.CLAMP, args.CLAMP)
```

WGAN-GP 是在 WGAN 的基础上引入了梯度惩罚项，用于改进梯度的稳定性和生成样本的质量。梯度惩罚项通过对判别器输入真实样本和生成样本之间的插值样本进行梯度计算，并对梯度范数与 1 之间的差异进行惩罚。梯度惩罚项的权重由超参数越大，梯度惩罚的影响越大，本次实验取 0.2

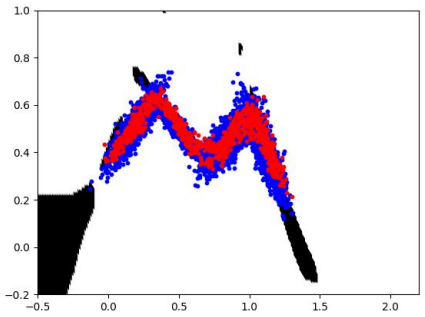
```
def gradient_penalty(D, real_samples, fake_samples, batchsz, args):

    # 对判别器的参数进行梯度惩罚
    epsilon = torch.rand(batchsz, 1).to(args.device)
    epsilon = epsilon.expand_as(real_samples)
    interpolated_samples = real_samples + epsilon * (fake_samples - real_samples)
    interpolated_samples.requires_grad_()
    interpolated_outputs = D(interpolated_samples)
    grad_outputs = torch.ones_like(interpolated_outputs).to(args.device)
    gradients = autograd.grad(outputs=interpolated_outputs, inputs=interpolated_samples,
                              grad_outputs=grad_outputs, create_graph=True,
                              retain_graph=True, only_inputs=True)[0]
    gradient_penalty = torch.pow(gradients.norm(2, dim=1) - 1, 2).mean()

    return gradient_penalty
```

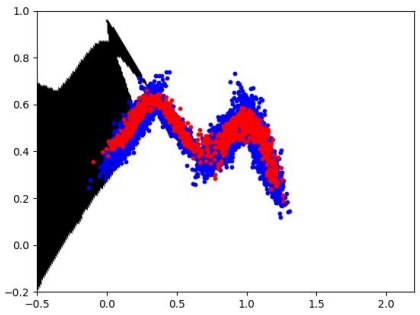
四、实验结果分析：

GAN + RMSprop:



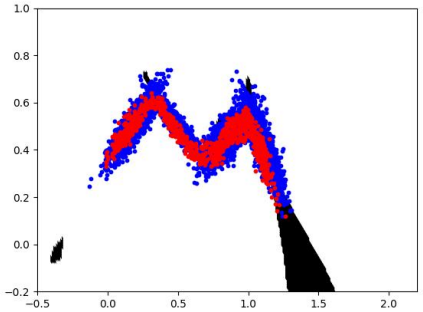
(epoch=1000)

WGAN + RMSprop:



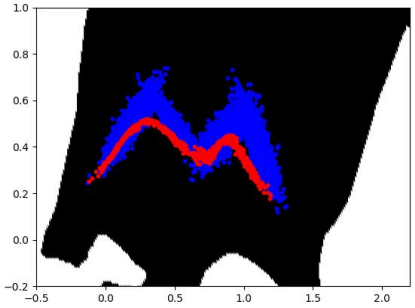
(epoch=1000)

WGAN-GP + RMSprop:



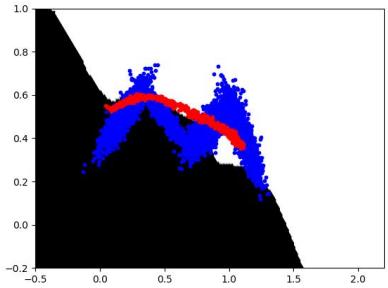
(epoch=1000)

WGAN-GP + Adam:



(epoch=1000)

WGAN-GP + SGD:



(epoch=1000)

对比优化器，选用 RMSprop 的效果最好，而 Adam 在收敛相对较慢。 SGD 收敛速度最慢，在 1000 轮时还没有收敛。

对比不同模型，GAN、WGAN 和 WGAN-GP 的性能都很好，在选用 RMSprop 作为优化器时 1000 轮之内都得到了较好的拟合结果。其中 GAN 收敛速度最快，在 400 轮时已经收敛，WGAN 和 WGAN-GP 都在 550 轮以后收敛。

五、隐空间语义方向搜索实验内容：

隐空间是生成器的输入向量空间，通常是一个低维度的向量空间，例如 100 维。这些向量被称为隐向量或潜在向量。生成器将这些隐向量映射为生成图像的特定表示。隐空间语义方向是隐空间中的特定方向，沿着该方向进行微调可以改变生成图像的特定属性或特征。例如，在人脸生成中，可能存在表示年龄、性别、表情等语义方向。为了找到特定的语义方向，可以使用不同的技术和方法。根据论文中的方法，求出A^TA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。

一旦找到了特定的语义方向，可以通过在隐空间中微调隐向量来修改生成器的输入，从而控制生成图像的属性。例如，在人脸生成中，增加或减少对应年龄方向的隐向量分量，可以使生成的人脸显得年轻或年长。

附：相关文件说明
result 文件夹保存对应模型的拟合过程的图像
models 文件夹保存三种网络的模型
work_dirs 文件夹保存隐空间语义方向的生成结果
sefa.py 为隐空间语义方向的 todo 文件
其余.py 文件为生成式对抗网络实现的代码