**深度学习实验报告**

实验5：生成式对抗网络

姓名：姚舜宇

学号：1190202107

班级：1903602

**实验环境：**

Windows11

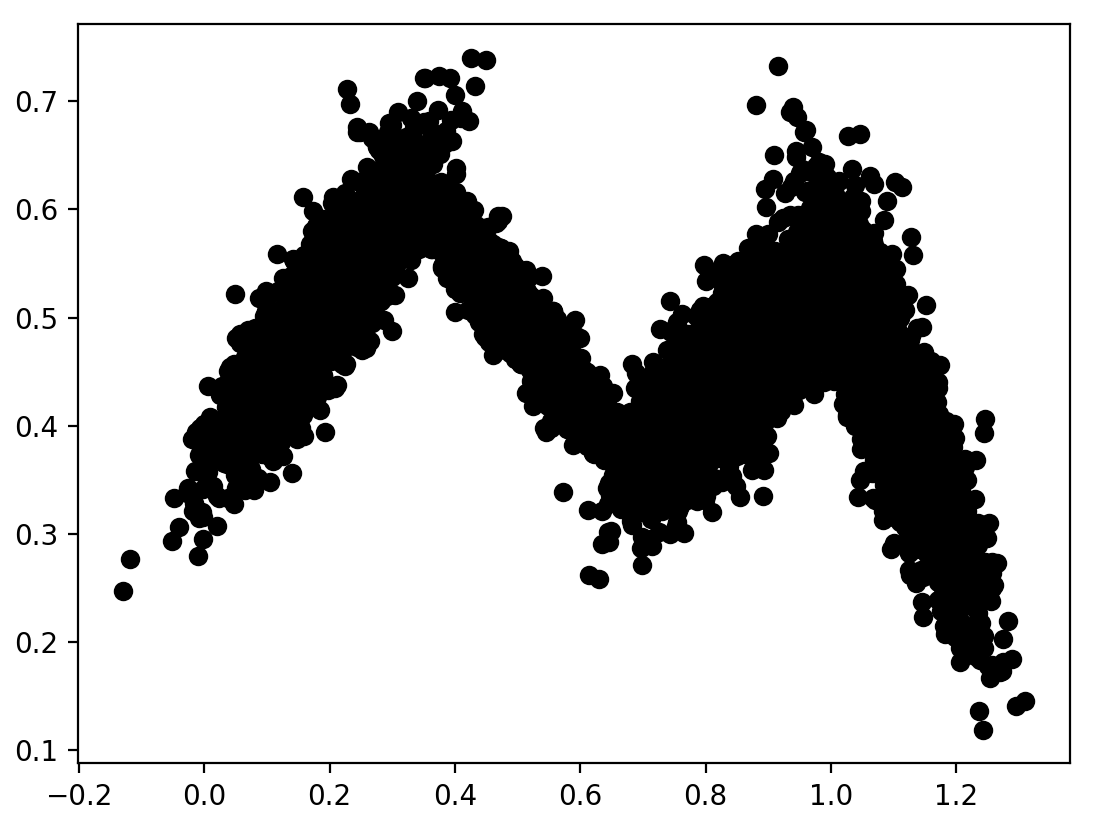
Python3.8

PyTorch1.10.2

**实验内容：**

**生成对抗网络实现**

给定分布：points.mat，绘制出如下图：



**Gan：**

定义生成器、判别器：

把每个点的坐标作为输入，生成器用于输出一个点的坐标，判别器输出一个0~1的值作为判别结果，是否是给定分布中的点。

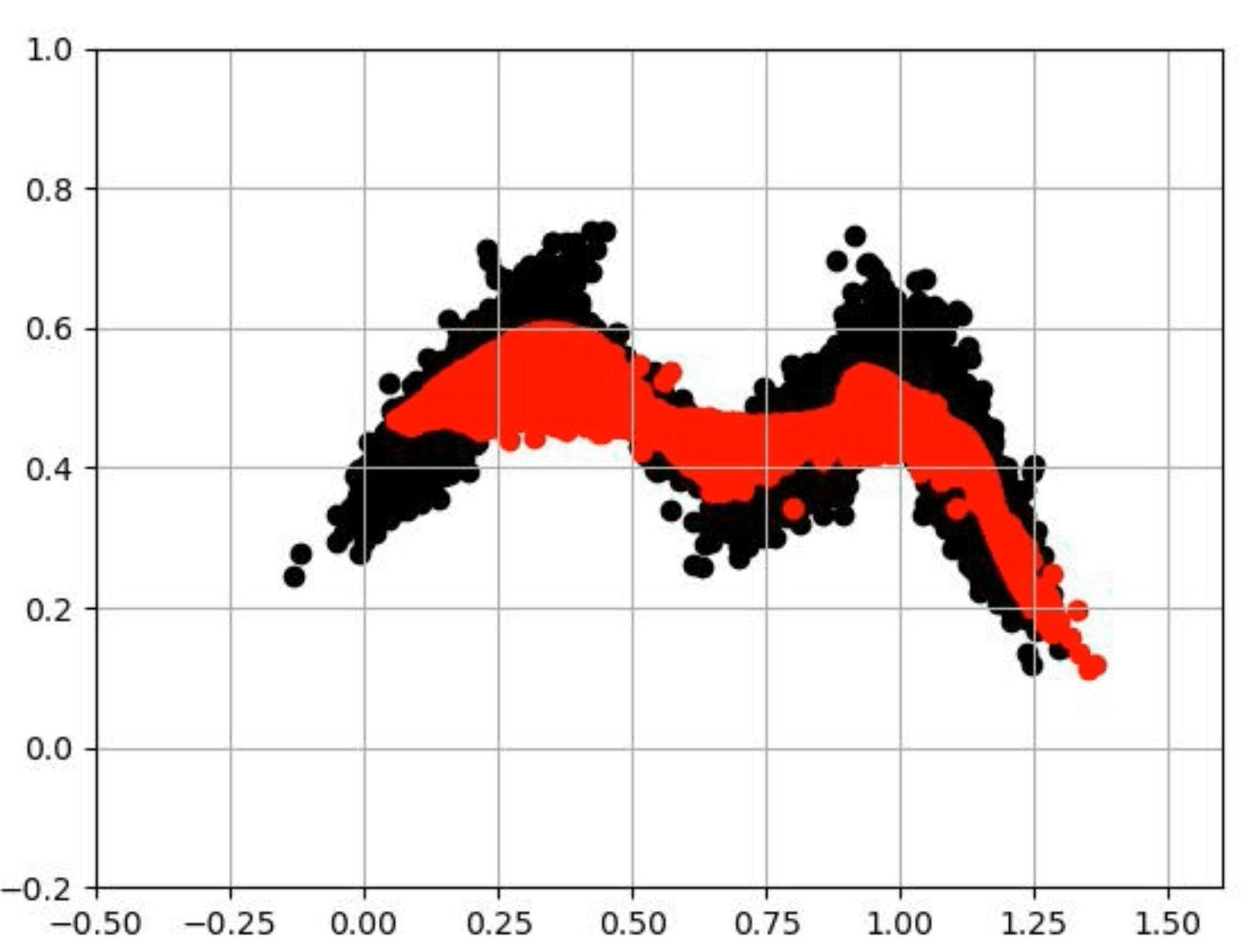
每次训练中，首先生成一个固定的高斯分布：

|  |
| --- |
| fixed\_noise = torch.randn((opt.points\_size, opt.dim), device=device) |

用于测试生成器的效果。

每次epoch中，生成器迭代8次，每次生成一个同样的高斯分布，使用生成器的输出送入判别器计算分为正类BCE损失函数；判别器迭代1次，生成一个同样的高斯分布，并从给定分布中进行采样，分别把两个分布送入判别器，计算分为负类和正类的BCE损失函数。

测试两种优化器RMSProp和Adam，分别的最终生成效果如下：



**Wgan：**

在gan的基础上，做以下修改：

判别器最后一层去掉sigmoid；

生成器和判别器的损失函数不取对数；

每次更新判别器的参数之后把它们的绝对值截断到不超过一个固定的常数；

不用基于动量的优化算法。

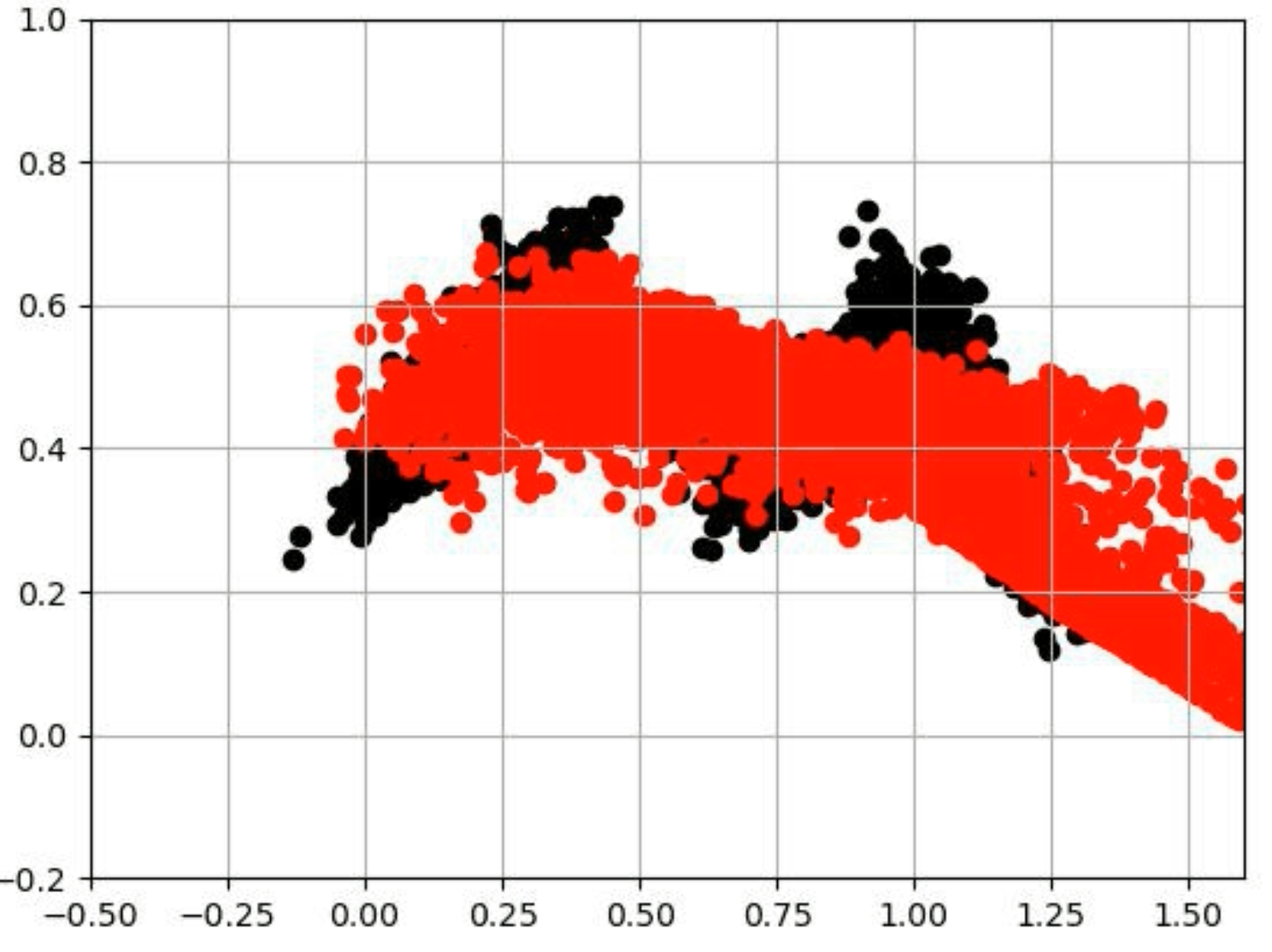
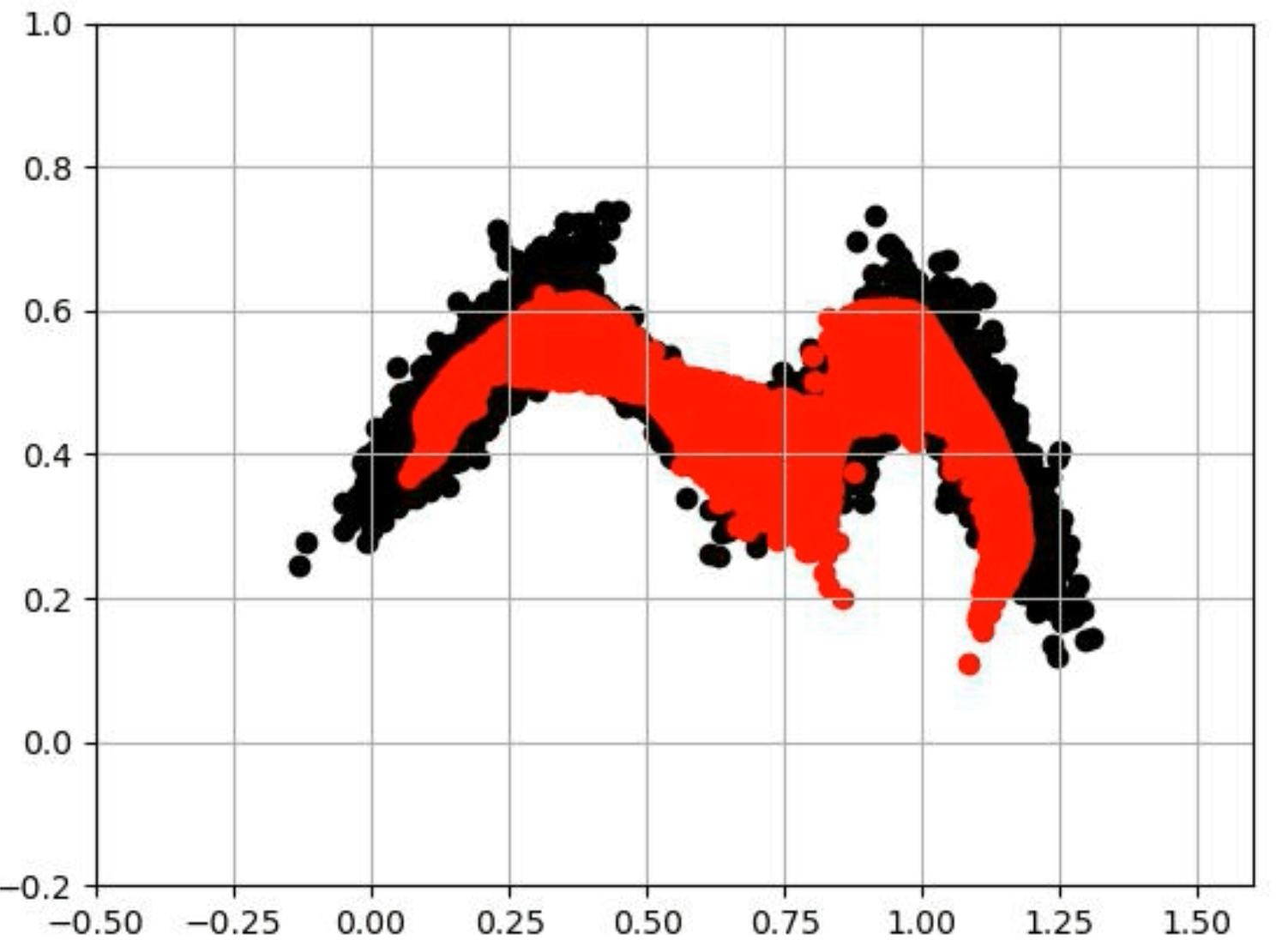
损失函数的定义：

|  |
| --- |
| g\_loss = -torch.mean(discriminator(fake)) # 生成器  d\_loss = torch.mean(discriminator(fake))-torch.mean(discriminator(real)) # 判别器 |

参数截断：

|  |
| --- |
| for p in discriminator.parameters():  p.data.clamp\_(-opt.clip\_value, opt.clip\_value) |

测试两种优化器RMSProp和Adam，分别的最终生成效果如下：



发现使用基于动量的优化算法无法收敛。

**Wgan-gp：**

在wgan的基础上，做以下改动：

使用梯度惩罚代替梯度裁剪；

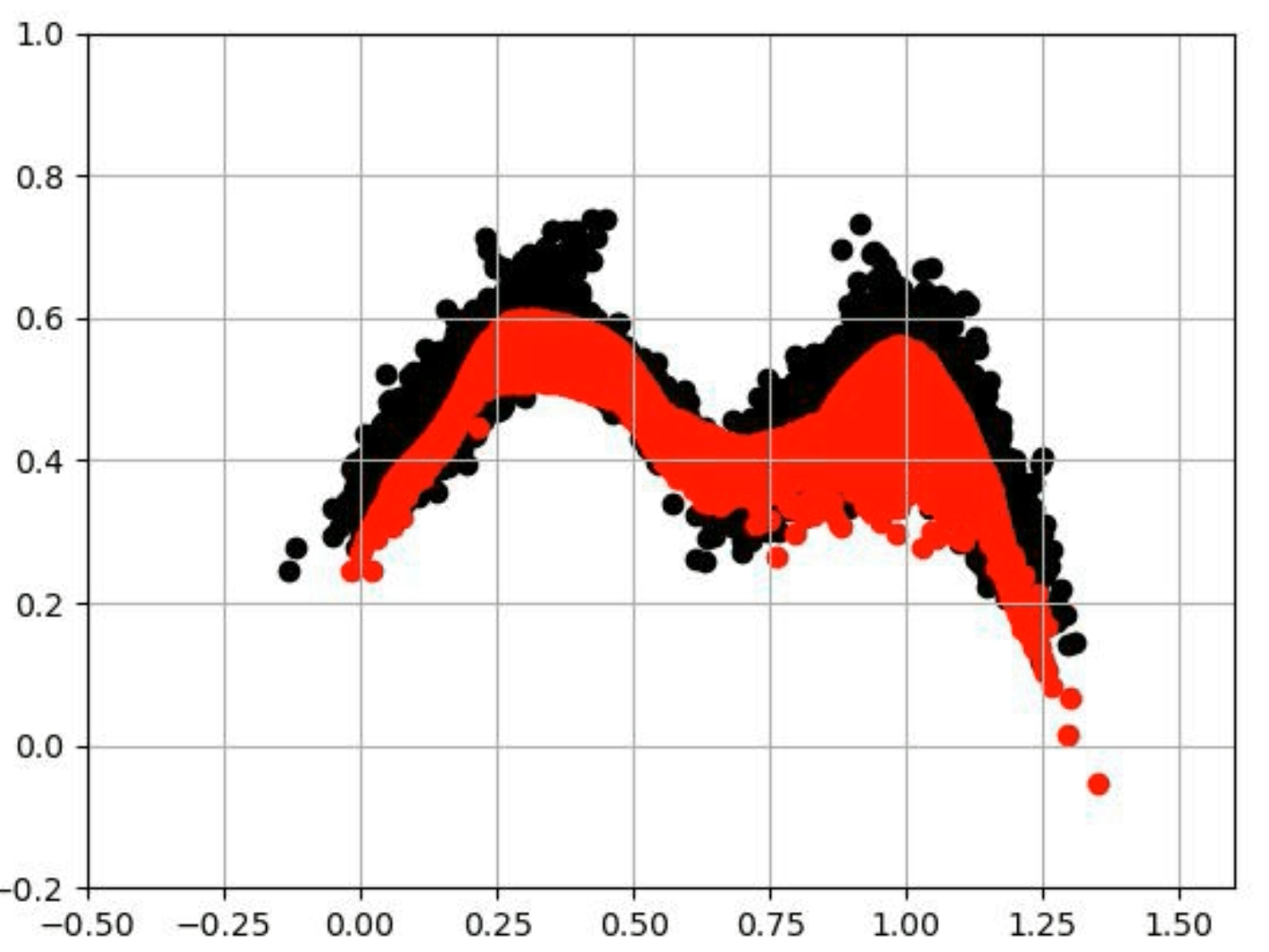
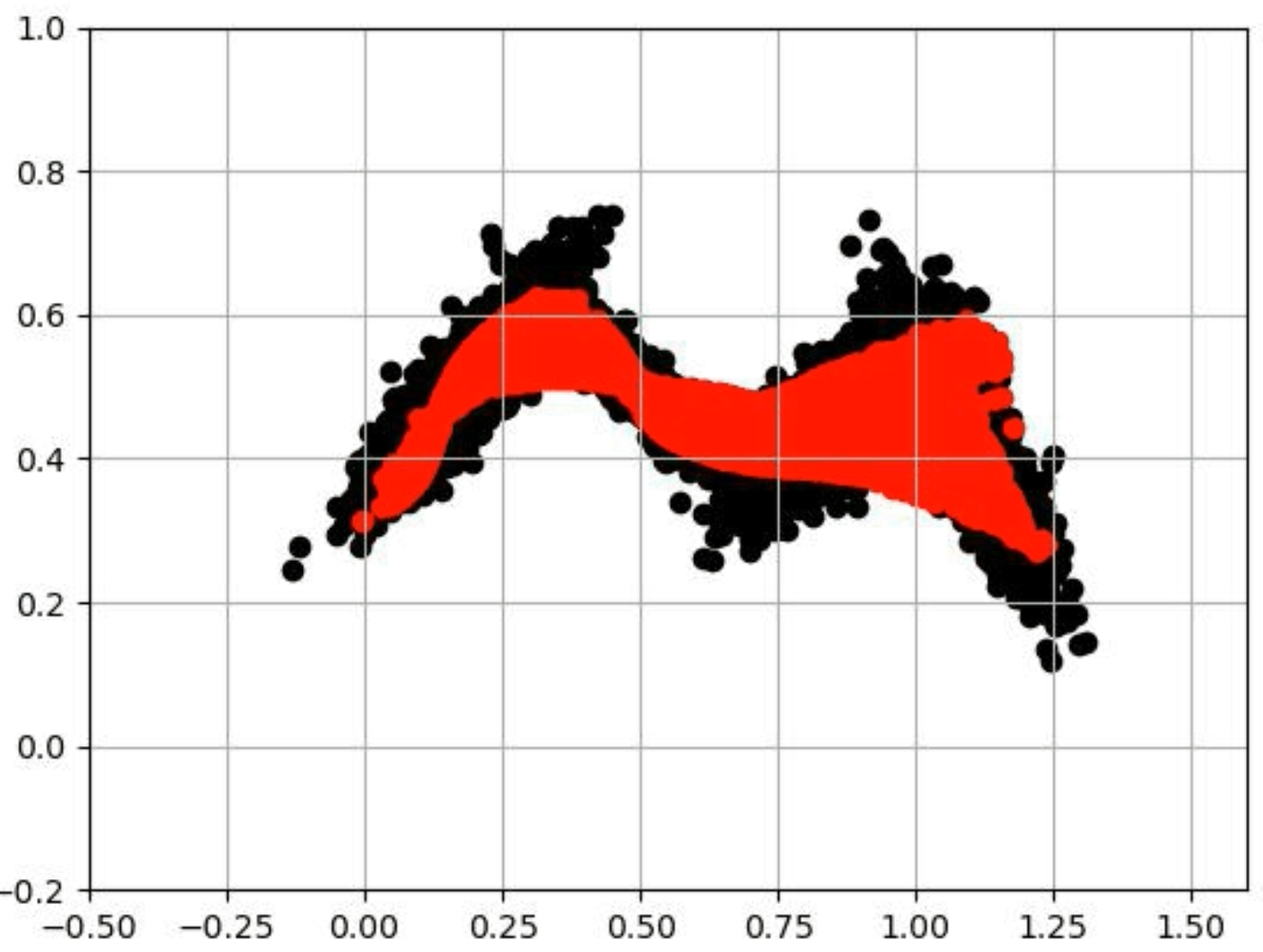
|  |
| --- |
| gradient\_penalty = compute\_gradient\_penalty(discriminator, real.data, fake.data, device)  d\_loss=torch.mean(discriminator(fake))-torch.mean(discriminator(real))+0.01\*gradient\_penalty |

在生成的图像上加入噪声；

|  |
| --- |
| interpolates = (alpha \* real\_samples + ((1 - alpha) \* fake\_samples)).requires\_grad\_(True)  d\_interpolates = D(interpolates) |

使用Adam优化器代替RMSProp。

测试两种优化器RMSProp和Adam，分别的最终生成效果如下：

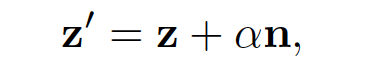
**结论**

比较三种gan，发现对于该任务，在稳定性方面，gan优于其他生成对抗网络，收敛较快，性能也较好。Wgan-gp的收敛性也较好，使用adam优化的收敛速度最快。对于优化器，使用RMSProp优化器训练三种gan的效果都似乎好于使用Adam训练。具体原因可能是由于动量因素，导致模型的优化很大程度上收到上一次迭代的影响，使得收敛速度变慢甚至不收敛。

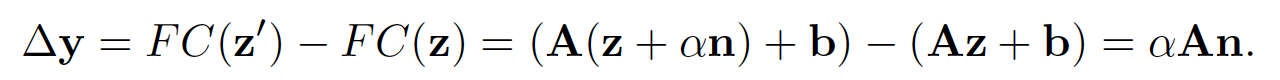
**隐空间语义方向搜索**

该工作的目的是通过改变潜在向量，来按照人为意愿改变生成器输出的效果。为了改变输出图像中的特定属性，需要计算出移动潜在向量的特定方向。

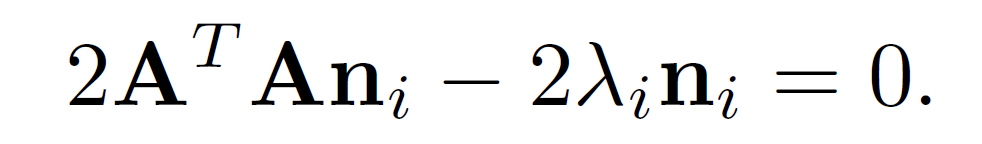
先进的gan模型往往由多个层组成，每一层都学习一个空间到另一个空间的转换。Sefa中重点研究了第一个变换：



如果将上式应用于输入潜在代码，可以得到：



为让Δy最大，通过拉格朗日乘子法，最终即求方程



的解。可以通过求矩阵的特征向量得到最终解。

在程序中，加入代码如下。

|  |
| --- |
| \_, evecs = torch.eig(weight @ weight.T, eigenvectors=True)  directions = evecs.T |

模型训练结果如下。



其他结果与视频参见目录/checkpoint/和附录。

。

**加载预训练模型**

|  |
| --- |
| import torch  import matplotlib.pyplot as plt  from train\_gan import Generator # 需要import对应的generator模型  if \_\_name\_\_ == '\_\_main\_\_':  noise = torch.randn((8191, 2)).cuda()  generator = torch.load('path/generator\_gan\_rmsprop').cuda()  fake = generator(noise).detach().cpu().numpy()  plt.scatter(fake[:, 0], fake[:, 1])  plt.show() |

**附录**

