生成对抗网络实验报告

姓名：张三　　学号：xxxxxxxxx

实验要求：

- 掌握 GAN 原理
- 学会使用 PyTorch 搭建 GAN 网络来训练 FashionMNIST 数据集

报告内容：

- 老师提供的原始版本 GAN 网络结构（也可以自由调整网络）在 FashionMNIST 上的训练 loss 曲线，生成器和判别器的模型结构（print(G)、print(D)）
- 自定义一组随机数，生成 8 张图
- 针对自定义的 100 个随机数，自由挑选 5 个随机数，查看调整每个随机数时，生成图像的变化（每个随机数调整 3 次，共生成 15x8 张图），总结调整每个随机数时，生成图像发生的变化。
- 解释不同随机数调整对生成结果的影响（重点部分）
- 格式不限

作业提交：

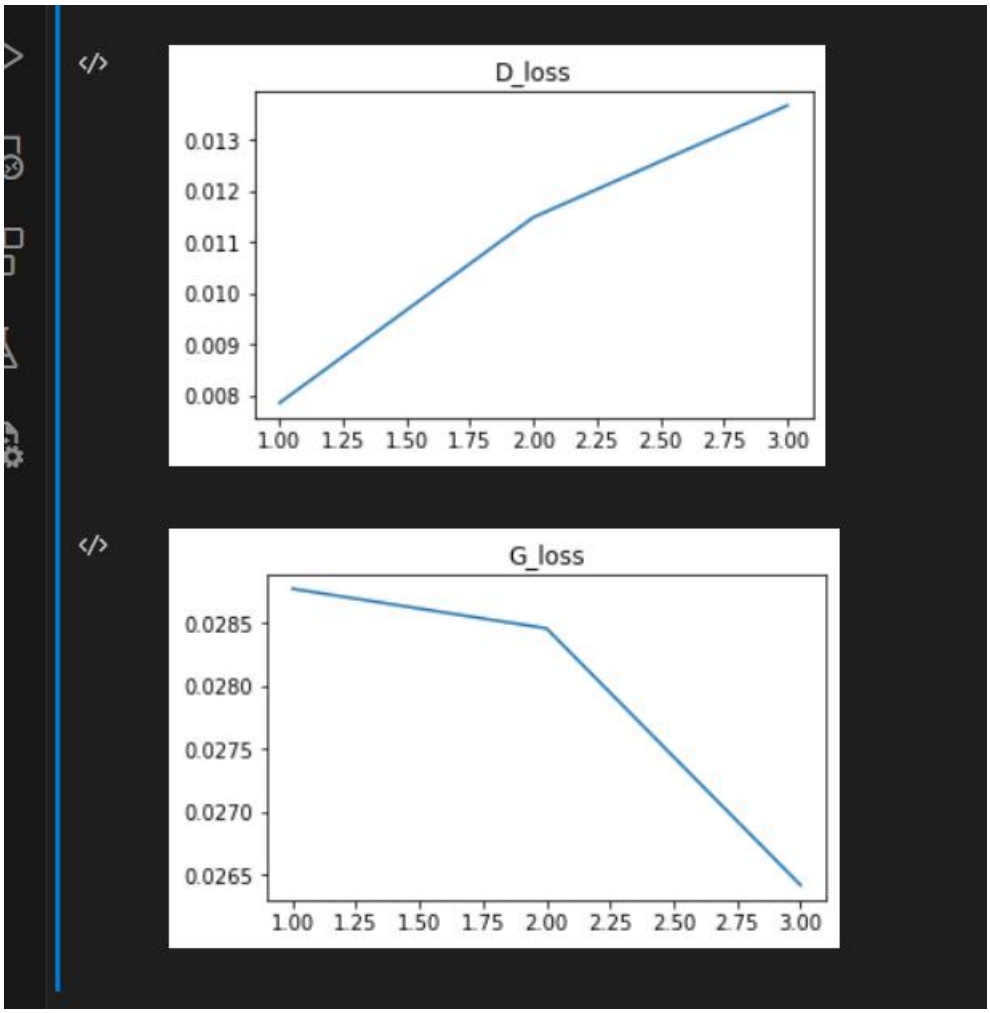
- 期末前将报告和代码（可将 jupyter notebook 里代码复制到一个 xxx.py 文件中）打包（学号+姓名.zip），提交方式另行通知
- 实验报告内容应工整
- 加分项：用卷积实现生成器和判别器。

原始版本 GAN

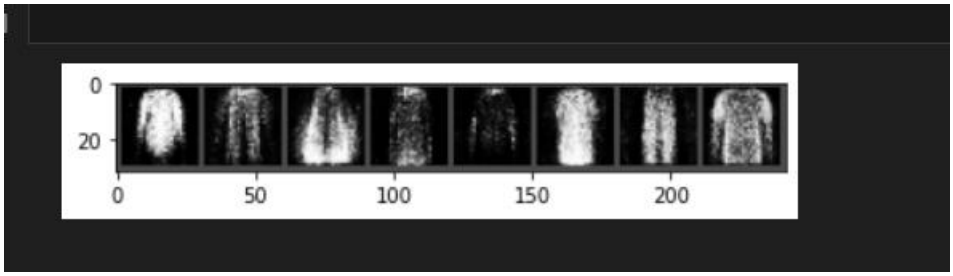
网络结构：

```
Discriminator(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (nonlin1): LeakyReLU(negative_slope=0.2)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
)
Generator(
  (fc1): Linear(in_features=100, out_features=128, bias=True)
  (nonlin1): LeakyReLU(negative_slope=0.2)
  (fc2): Linear(in_features=128, out_features=784, bias=True)
)
```

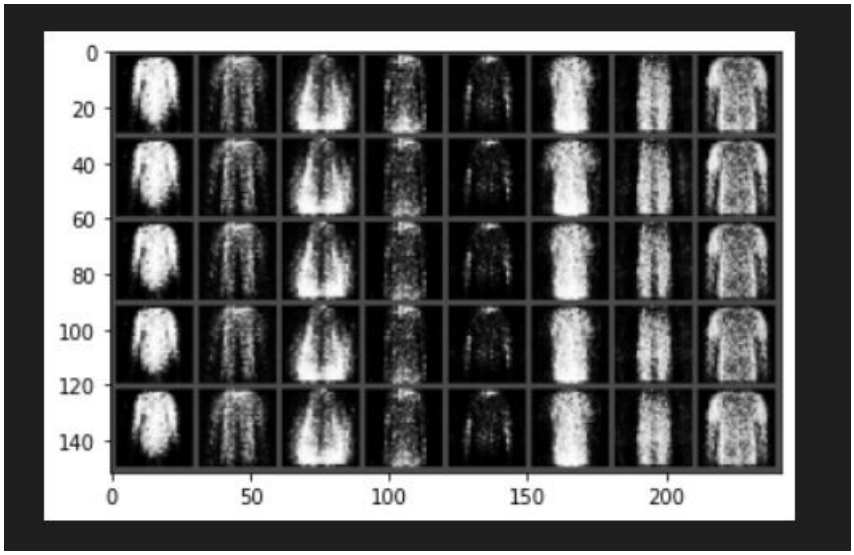
训练三轮的损失曲线

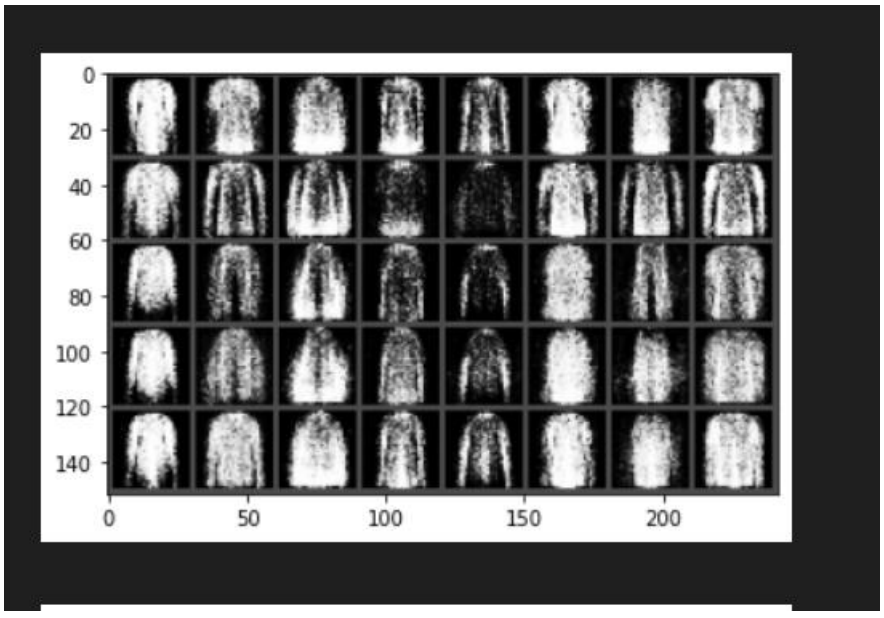


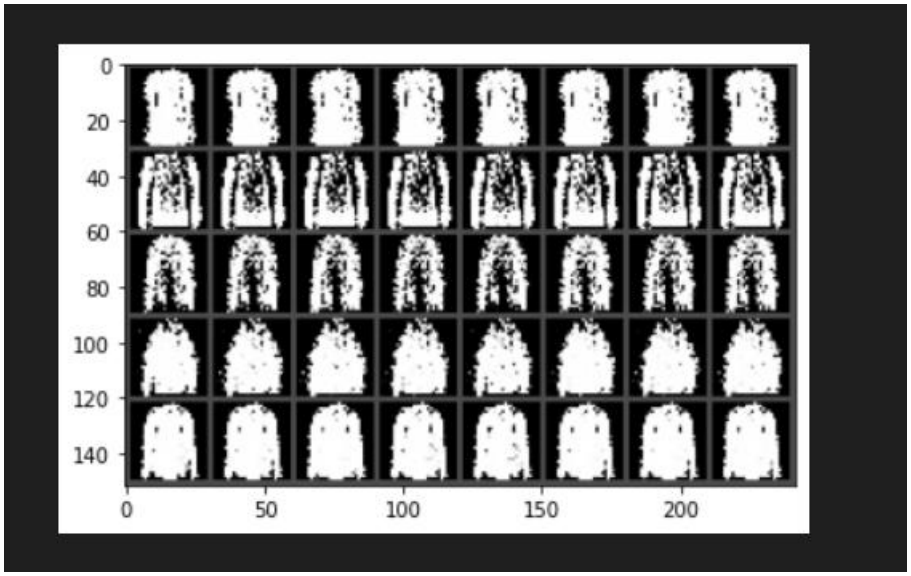
自定义一组随机数，生成 8 张图



15x8 张图


0
20
40
60
80
100
120
140
0
50
100
150
200


0
20
40
60
80
100
120
140
0
50
100
150
200



对调整参数生成的 15*8 张图的解释

我们每隔 20 个数更改一次，每张图的第一行代表的是更改第一个位置的随机数，第二行是代表更改第 20 个位置的随机数，以此类推

三幅图分别对应的是分别把他们改为 1,10,100 的情况，我们可以看到，在第一幅图每一行跟最开始的时候基本没肉眼可见的变化，因为比较小的时候他的值乘上他的权重最后带来的影响并不大。

然后我们改为 10 的时候就有了比较明显的变化，而且改在不同的位置变化也有所不同，这是因为不同位置的权重有所不同。

最后我们改为 100 的时候可以看到同一行基本都变成接近一样的图片了，这是因为数据太大，其他位置的随机参数加权之后跟这个参数的加权比仍然太小了，所以基本图像就是只由这个参数主导了，然后因为我们把这些图像的此参数都设置为了一样的所以图片也就近似了。

卷积实现生成器和判别器

网络结构

```
Discriminator(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc): Linear(in_features=6272, out_features=1, bias=True)
)
Generator(
  (l1): Linear(in_features=100, out_features=12544, bias=True)
  (bn1): BatchNorm1d(12544, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv1): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
)
```

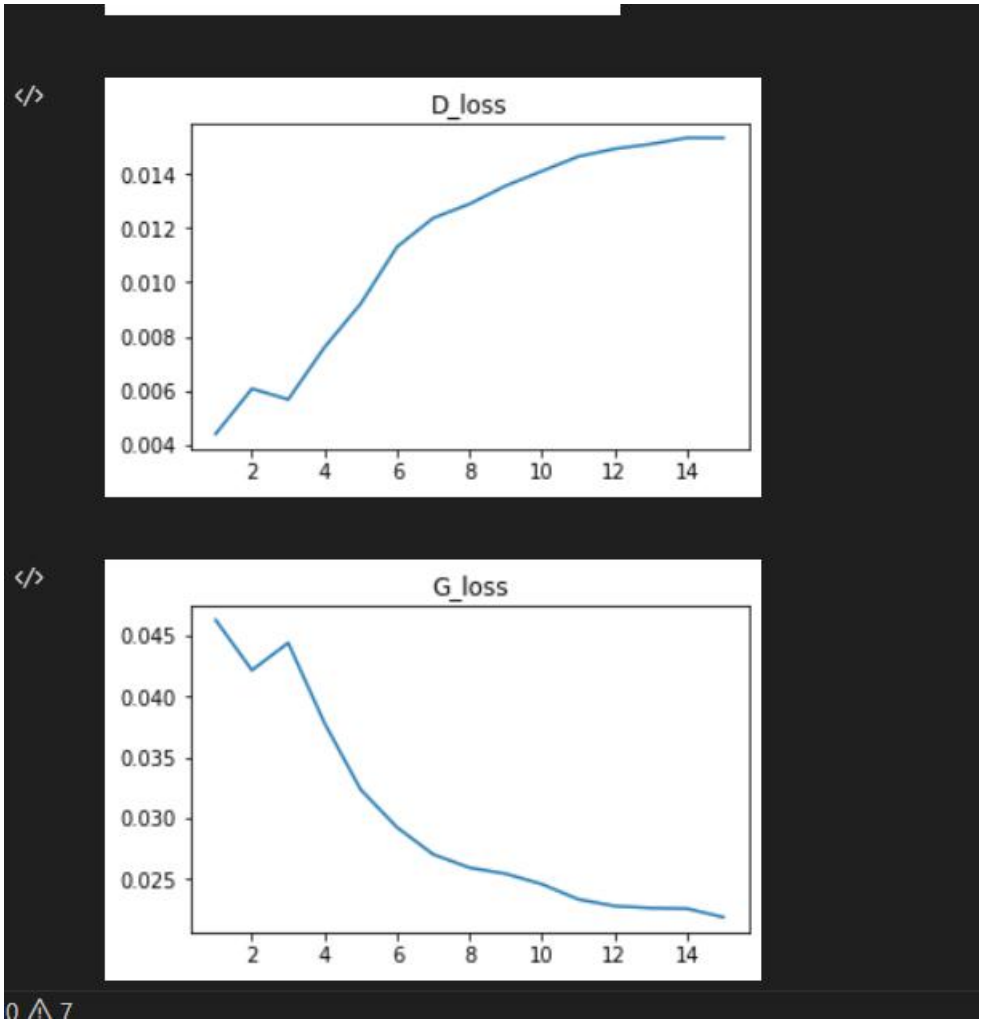
```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=2, padding=1, bias=False)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(128)
        self.fc= nn.Linear(128*7*7, 1)

    def forward(self, x):
        x = F.leaky_relu(self.conv1(x))
        x=self.conv2(x)
        x=self.bn1(x)
        x = F.leaky_relu(x)
        x = x.view(-1, 128*7*7)
        x = self.fc(x)
        x=torch.sigmoid(x)
        return x
```

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator,self).__init__()
        self.linear1 = nn.Linear(100,256*7*7)
        self.bn1=nn.BatchNorm1d(256*7*7)
        self.deconv1 = nn.ConvTranspose2d(256,128,
                                          kernel_size=(3,3),
                                          stride=1,
                                          padding=1)
        self.bn2=nn.BatchNorm2d(128)
        self.deconv2 = nn.ConvTranspose2d(128,64,
                                          kernel_size=(4,4),
                                          stride=2,
                                          padding=1)
        self.bn3=nn.BatchNorm2d(64)
        self.deconv3 = nn.ConvTranspose2d(64,1,
                                          kernel_size=(4,4),
                                          stride=2,
                                          padding=1)

    def forward(self,x):
        x=F.relu(self.linear1(x))
        x=self.bn1(x)
        x=x.view(-1,256,7,7)
        x=F.relu(self.deconv1(x))
        x=self.bn2(x)
        x=F.relu(self.deconv2(x))
        x=self.bn3(x)
        x=torch.tanh(self.deconv3(x))
        return x
```

训练曲线


D_loss
0.014
0.012
0.010
0.008
0.006
0.004
2
4
6
8
10
12
14
G_loss
0.045
0.040
0.035
0.030
0.025
2
4
6
8
10
12
14

训练 15 轮后的生成效果