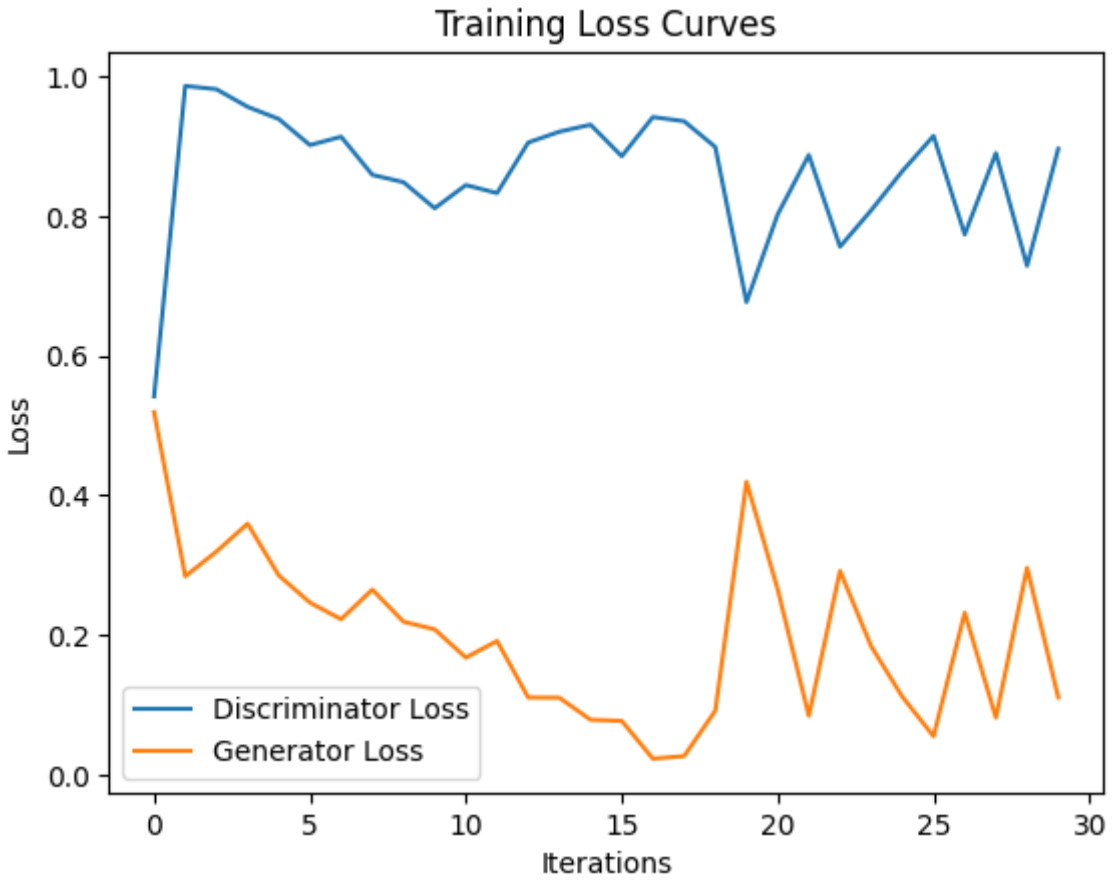
深度学习报告w5：生成对抗网络

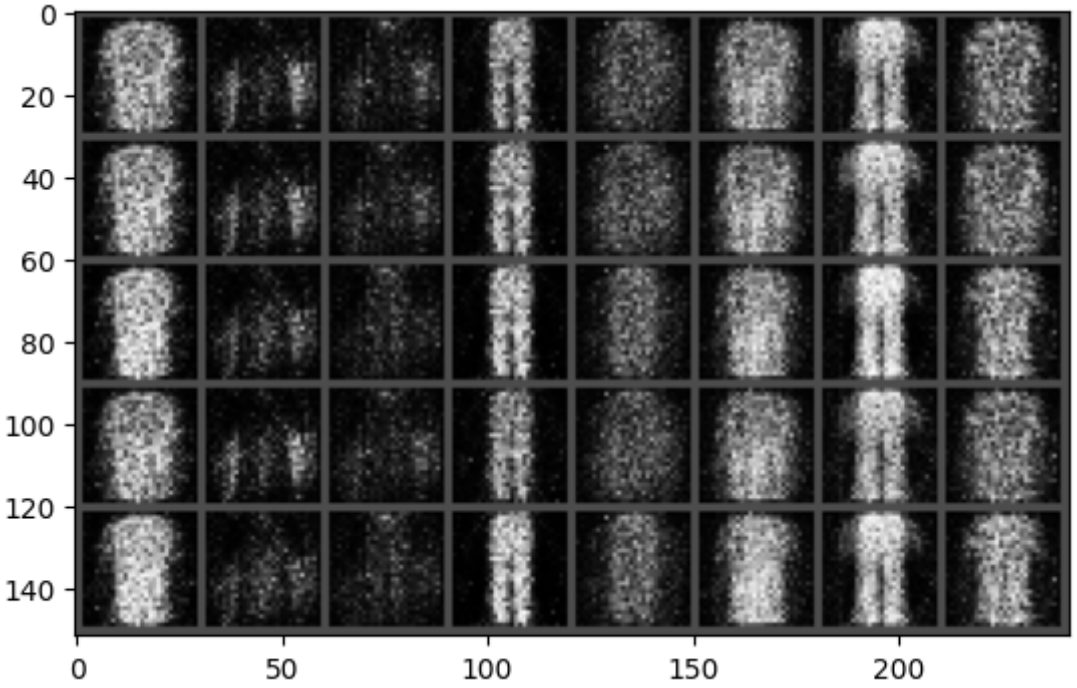
1 模型复现

```
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)
)
```



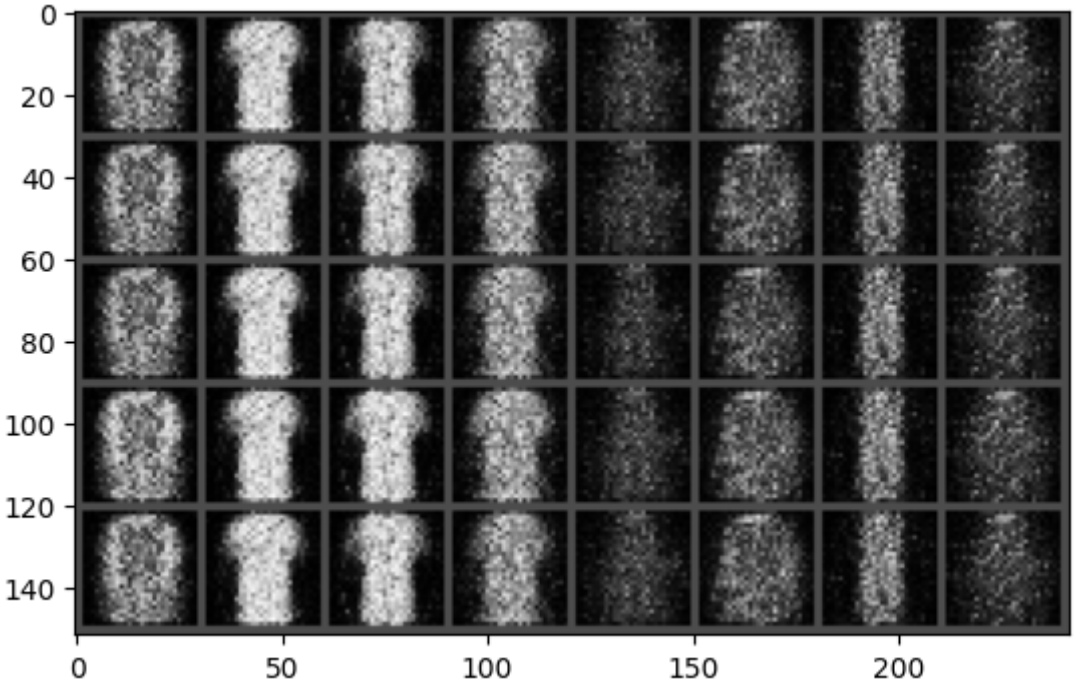
本人认为看gan的loss意义不大，对抗学习的loss波动是比较剧烈的。

2 自定义随机数，生成8张图

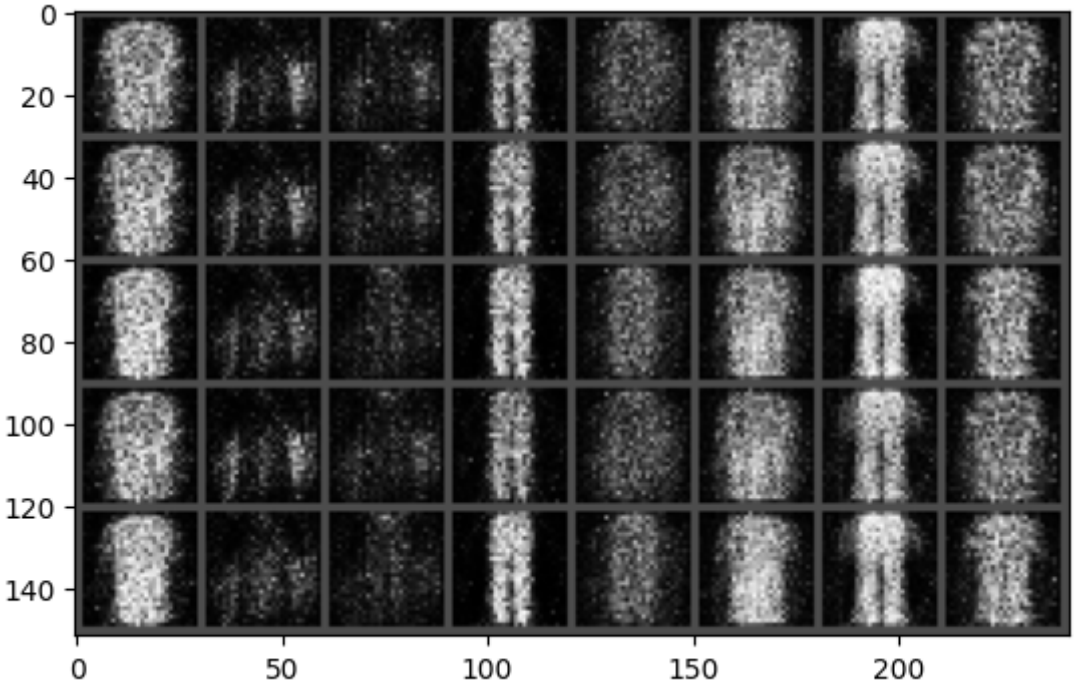


3 调整每个随机数

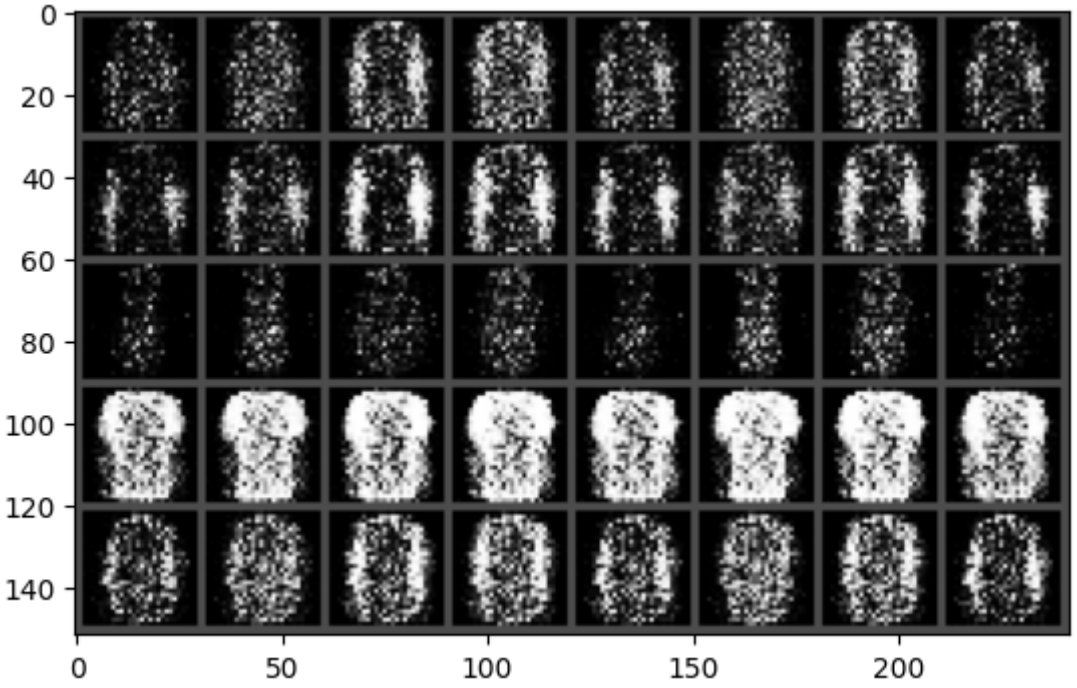
随机数为1



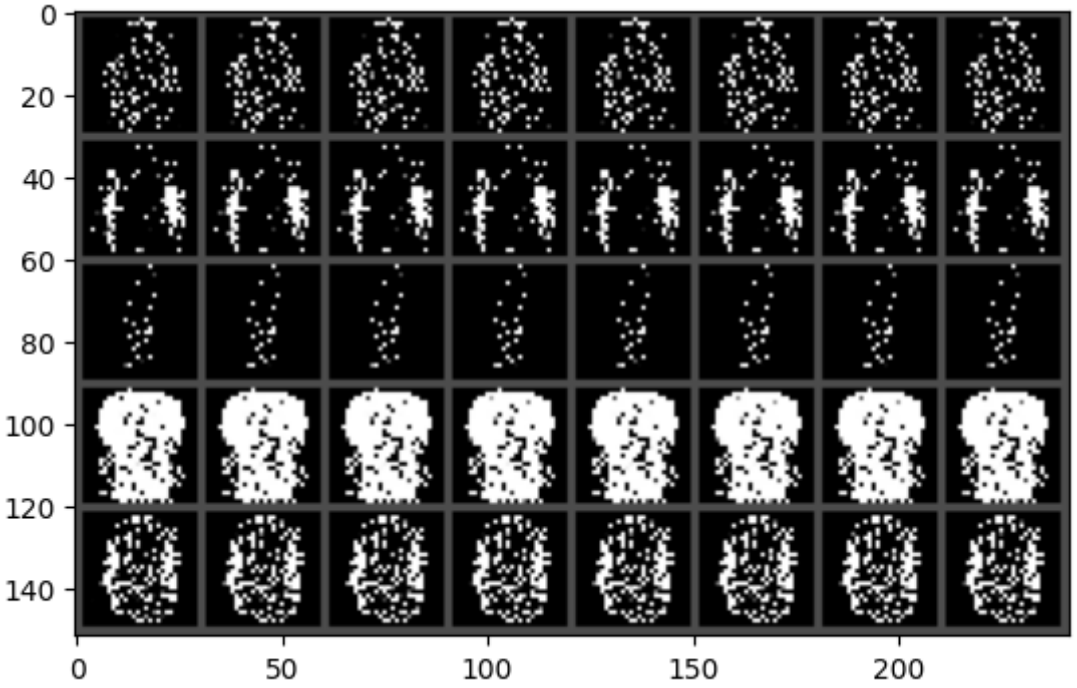
随机数为3



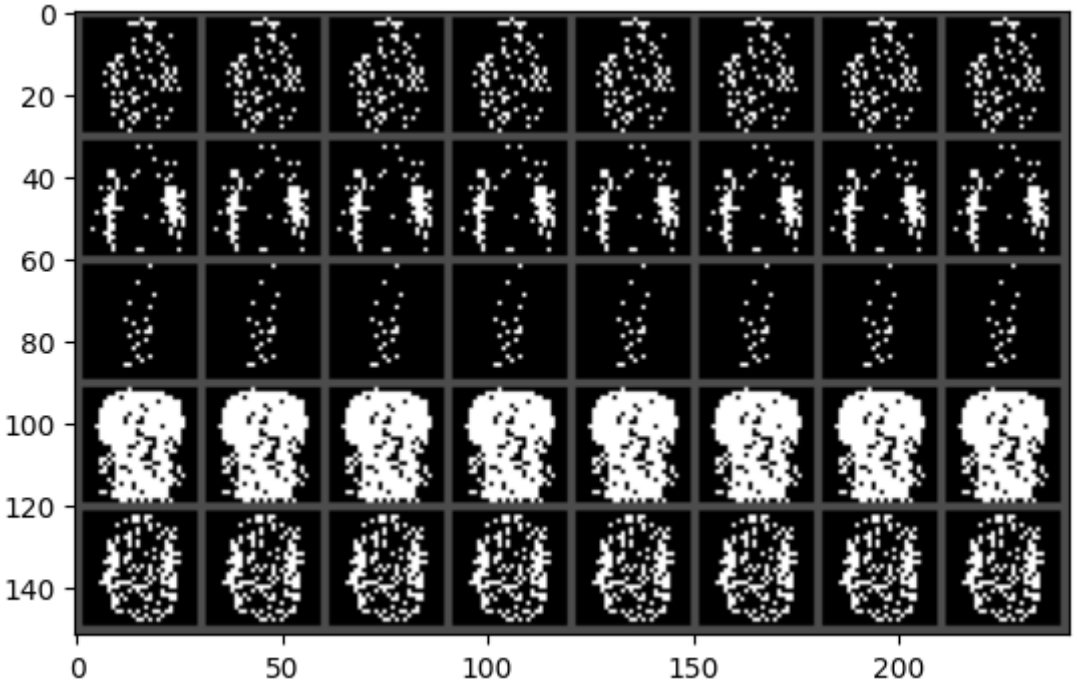
随机数为30



随机数为810



随机数为114514



当随机数设置得很小（如 0.3）时，在不同位置改变一个这样的小随机数会产生相对较小的影响，甚至肉眼无法观察到任何变化。

当随机数被设置得很小（如 0.3）时，在不同位置改变一个这样的小随机数会产生相对较小的影响，甚至肉眼无法观察到任何变化；然而，在不同位置改变一个这样的小随机数会产生相对较小的影响，甚至肉眼无法观察到任何变化；然而，在不同位置改变一个这样的小随机数会产生相对较小的影响。

然而，当随机数设置为较大的数字（如 10）时，如果我们在不同位置改变其中一个小随机数，收获的效果还是比较明显的，可以说是非常明显。

效果还是比较明显的，从最后一张图中可以清楚地看到，与位置 60 到 10 相比，位置 80 的变化直接让裤子变成了两条。

从最后一张图中可以清楚地看到，与位置 60 到 10 的变化相比，位置 80 的变化直接使裤子变成了两件

衣服。在观察同一位置的随机数变化时，我们会发现随机数太小（0.3）或太小（0.3）都会导致裤子数量的变化。
(0.3）或过大的随机数（10）会导致图像亮度较低，甚至有些图像接近全黑。
当我们使用适中的随机数时，我们的模型生成的图像相对更亮、更好。

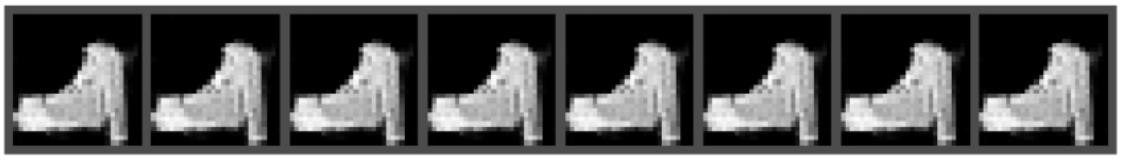
4 用CNN实现生成器和判别器

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 1, 7, 1, 0, bias=False),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.main(x).view(-1, 1).squeeze(1)

class Generator(nn.Module):
    def __init__(self, z_dim=100):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(z_dim, 128, 7, 1, 0, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 1, 4, 2, 1, bias=False),
            nn.Tanh()
        )
    def forward(self, x):
        return self.main(x.view(x.size(0), x.size(1), 1, 1))

Discriminator(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 1, kernel_size=(7, 7), stride=(1, 1), bias=False)
    (6): Sigmoid()
  )
)
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 128, kernel_size=(7, 7), stride=(1, 1), bias=False)
```

```
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
    (7): Tanh()
  )
)
```



实验结果如上