GAN

> 2013551 雷贺奥

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

GAN基本原理

生成对抗模型可以拆分为两个模块：一个是判别模型，另一个是生成模型。

- Generator：生成网络，即输入一个随机噪声，通过网络生成图片。
- Discriminator：判别网络，判别一张图片是不是真的。最终输出的是一个属于0~1间的概率，概率越大表示越有可能是真的。

生成网络训练过程就是让生成的图片“越来越真”，从而使得判别网络无法判断真假。

判别网络训练过程就是要尽可能分别出真假，损失值由两部分组成，一部分是真实图片与真实标签的损失，另一部分是生成器生成图片与虚假图片的损失。

本实验默认的GAN是用MLP搭建的，**作者在此基础上也尝试了使用简单CNN搭建。**

MLP_GAN

网络结构

notebook中提供的Generator和Discriminator，均为采用线性连接的MLP网络结构。

- Generator：将输入的噪声先采用线性连接，将其从(1×128)扩展到(1×784)，再经过，激活函数，最后将其reshape成为$(1 \times 28 \times 28)$的图片。

```
class Generator(nn.Module):
    def __init__(self, z_dim=100):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(z_dim, 128)
        self.nonlin1 = nn.LeakyReLU(0.2)
        self.fc2 = nn.Linear(128, 784)
    def forward(self, x):
        h = self.nonlin1(self.fc1(x))
        out = self.fc2(h)
        out = torch.tanh(out) # range [-1, 1]
        # convert to image
        out = out.view(out.size(0), 1, 28, 28)
        return out
```

- Discriminator：将输入的$(1 \times 28 \times 28)$图片，先展平为(1×784)，再使用线性连接层将其转为(1×128)，经过激活函数，再将其转为(1×1)，最终使用sigmoid函数使得输出的结果处于0~1之间。

```
class Discriminator(torch.nn.Module):
    def __init__(self, inp_dim=784):
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(inp_dim, 128)
        self.nonlin1 = nn.LeakyReLU(0.2)
        self.fc2 = nn.Linear(128, 1)
    def forward(self, x):
        x = x.view(x.size(0), 784) # flatten (bs x 1 x 28 x 28) -> (bs x 784)
        h = self.nonlin1(self.fc1(x))
        out = self.fc2(h)
        out = torch.sigmoid(out)
        return out
```

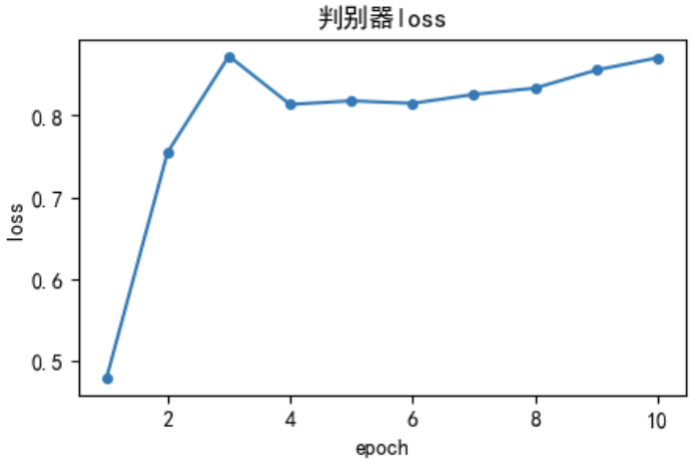
使用print()函数打印网络结构，结果如下所示：

```
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)
)
```

FashionMNIST数据集loss曲线

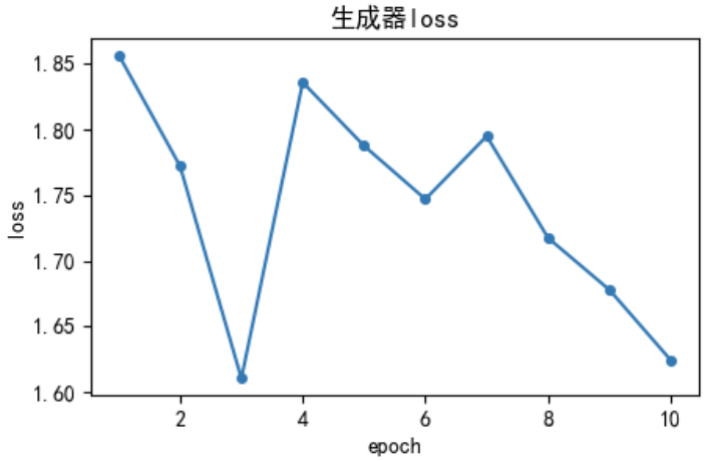
Discriminator、Generator均使用 BCELoss 作损失函数。

- Discriminator：损失值由两部分组成，一部分是真实图片与真实标签的损失，另一部分是生成器生成图片与虚假图片的损失。



判别器的loss最后趋向于逐渐提升，说明生成的图片越来越像真实的图片，可以以假乱真了，效果良好。

- Generator: `lossG = criterion(D_G_z, lab_real )`，计算生成的图片，经Discriminator判别后，与真实图片的差距。



前几个 epoch 生成器的 loss 不稳定，而后面的epoch，loss整体为下降趋势，说明生成的图片越来越像真实的图片，可以以假乱真。

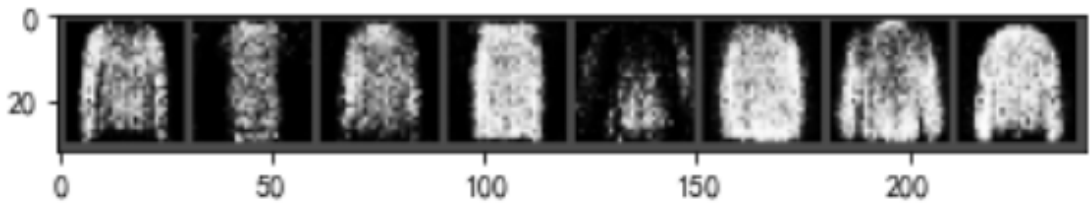
随机数改变

自定义一组随机数，生成8张图

先设置随机数种子，由于需要生成八张图片，输入8个长度为100的随机数向量到Generator中。

```
torch.manual_seed(23231)
z = torch.randn(8, 100, device=device) # random noise,8 samples, z_dim=100
x_gen = G(z)
x_gen = G(z).detach()
show_imgs(x_gen)
```

生成结果如下：



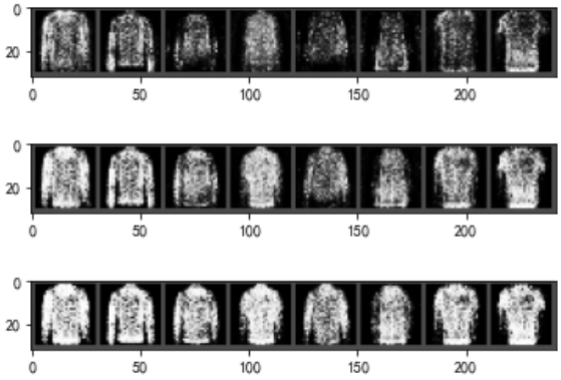
针对自定义的100个随机数，自由挑选5个随机数

作者选取的五个位置分别为[0, 20, 40, 60, 80]，每个位置都会从[-1,5,10]进行调整。

```
mylist1 = [0, 20, 40, 60, 80]
mychange = [-1,5,10]

for position in mylist1:
    for change in mychange:
        print("position %d \t change %d \n"%(position,change))
        temp = z
        for i in range(0,8):
            temp[i][mylist1] = change
        x_gen = G(temp)
        show_imgs(x_gen, new_fig=True)
```

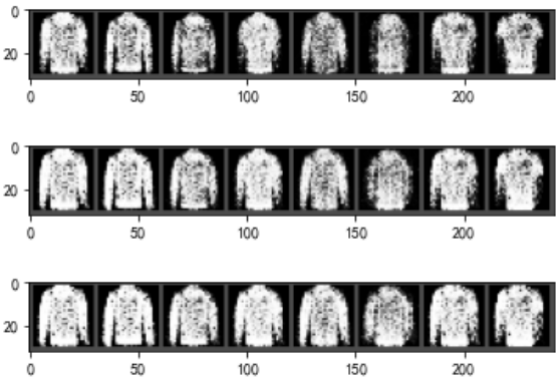
- $position = 0$



从上到下，依次为$change = -1$、$change = 5$、$change = 10$，可以看出图像越来越亮，灰度值越来越大。

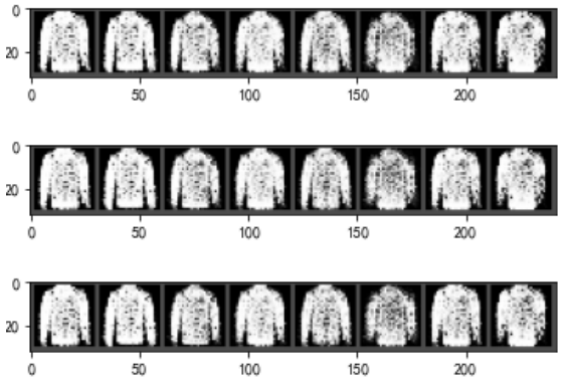
同时，第三列、第五列的衣服样式也发生了较大的改变，可能是由于改变值，使得图像灰度越来越大，从而使得原来是黑色的像素格变为白色。

- $position = 20$



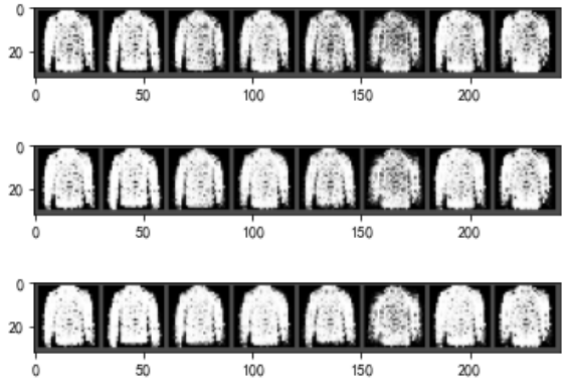
从上到下，依次为$change = -1$、$change = 5$、$change = 10$，可以看出图像越来越亮，灰度值越来越大。但是，衣服的样式没有明显改变。

- $position = 40$



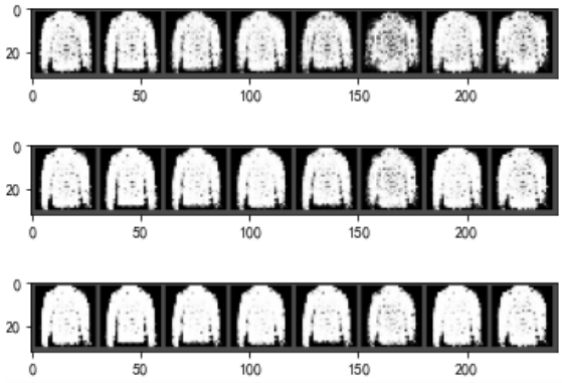
从上到下，依次为$change=-1$、$change=5$、$change=10$，可以看出图像越来越亮，灰度值越来越大，同时衣服的样式，几乎看不出改变。

- $position=60$



从上到下，依次为$change=-1$、$change=5$、$change=10$，可以看出图像越来越亮，灰度值越来越大,但也不明显，同时衣服的样式，几乎看不出改变。

- $position=80$



从上到下，依次为$change=-1$、$change=5$、$change=10$，可以看出图像越来越亮，灰度值越来越大。在值较小时，衣服的中间部分有些黑色部分，在值较大时，衣服的中间部分，黑色变为白色。

综上:

（1）不同postion主要改变的衣服的样式，这点非常好分析，因为不同的position对应生成模型中的线性连接的权重不一样，导致特征向量提取的不同，从而使得衣服样式发生较大的改变。

（2）不同change值主要改变图像的灰度，因为position相同，即权重相同，同一位置上采用不同值可能只会影响灰度值，从而改变图像的亮度。

CNN_GAN

网络结构

作者在前面作业的基础上，同样尝试了使用CNN来搭建Discriminator、Generator。采用较为简单的卷积神经网络。

```
class myDiscriminator(torch.nn.Module):
    def __init__(self, inp_dim=784):
        super(myDiscriminator, self).__init__()
        # [64, 1, 28, 28]->[64, 64, 6, 6]
        self.conv1 = nn.Sequential(
            nn.Conv2d(1,1,kernel_size=2,stride=2),
            nn.LeakyReLU(0.2),
        )
        self.fc=nn.Sequential(
            nn.Linear(14*14, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 1),
            nn.LeakyReLU(0.2),
        )

    def forward(self, x):
        # flatten (bs x 1 x 28 x 28) -> (bs x 784)
        x = self.conv1(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.fc(x)
        x = torch.sigmoid(x)
        return x

class myGenerator(nn.Module):
    def __init__(self, z_dim=100):
        super(myGenerator, self).__init__()
        self.fc1 = nn.Sequential(
            nn.Linear(z_dim, 900),
            nn.LeakyReLU(0.2),
        )
        self.conv1 = nn.Sequential(
            nn.Conv2d(1,64,kernel_size=2,stride=1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(64,1,kernel_size=2,stride=1),
            nn.LeakyReLU(0.2),
        )
    def forward(self, x):
        x = self.fc1(x)
        #print(x.shape)
        # convert to image
        x = x.view(x.size(0), 1, 30, 30)
        x = self.conv1(x)
        return x
```

打印网络结构如下：

- Discriminator

  先使用一个简单的卷积层提取特征，后面再接上两个线性连接层，每一层后面都接上激活函数。这个CNN的结构很简单，作者并没有进行反复的调参，只是尝试CNN。

```
myDiscriminator(
  (conv1): Sequential(
    (0): Conv2d(1, 1, kernel_size=(2, 2), stride=(2, 2))
    (1): LeakyReLU(negative_slope=0.2)
  )
  (fc): Sequential(
    (0): Linear(in_features=196, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=128, out_features=1, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
  )
)
```

- Generator

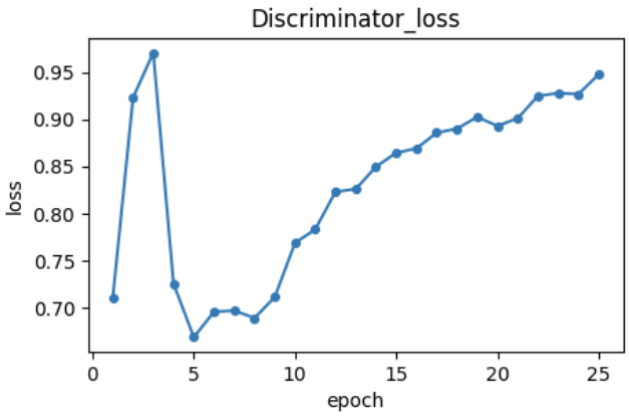
  将1×100经过线性连接层扩展到1×900，再将其reshape为30×30，再经过两层卷积层，刚好结果的shape为28×28,只需要将其作为结果图像，直接输出即可。

```
myGenerator(
  (fc1): Sequential(
    (0): Linear(in_features=100, out_features=900, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
  )
  (conv1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(2, 2), stride=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(64, 1, kernel_size=(2, 2), stride=(1, 1))
    (3): LeakyReLU(negative_slope=0.2)
  )
)
```

FashionMNIST数据集loss曲线

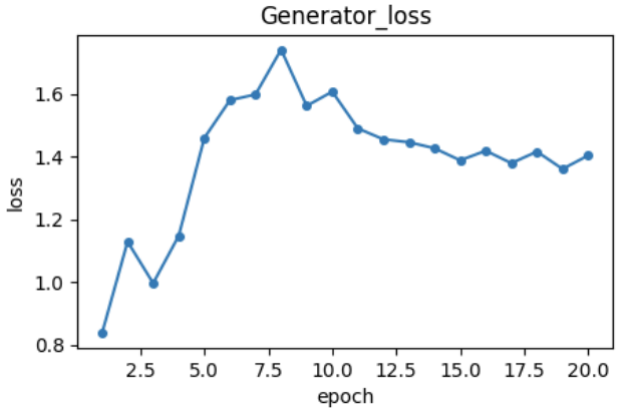
Discriminator、Generator均使用 BCELoss 作损失函数。

- Discriminator



  判别器的loss最后趋向于逐渐提升，说明生成的图片越来越像真实的图片。

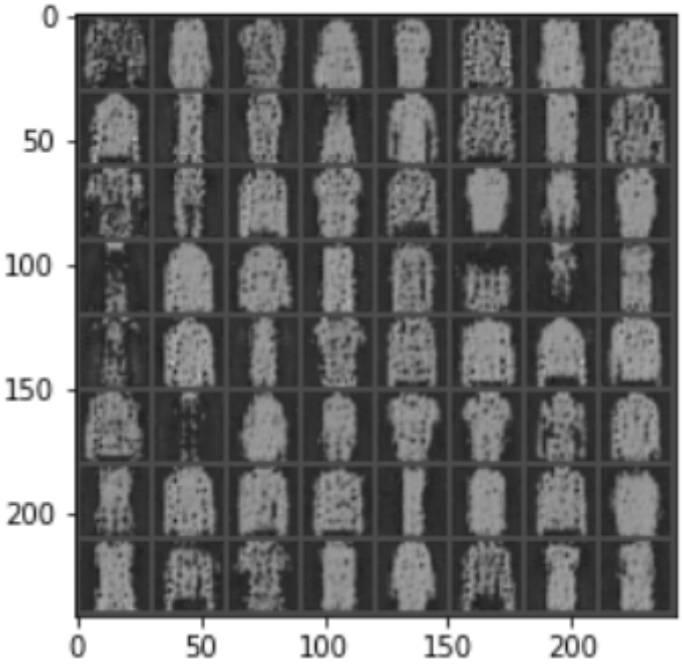
- Generator



判别器的loss最后趋向于逐渐提升，说明生成的图片越来越像真实的图片。

CNN生成结果

最终，在epoch=20时，Generator生成的结果如下所示：



可以看出，相较于MLP，使用CNN搭建GAN会使得生成的图像更暗，灰度发生改变；但是，CNN的优点在于，例如"衣服的袖子"的细节部分更加详细，从而可以推断出，CNN对于提取关键特征的效果总体由于MLP。

综上，使用CNN搭建GAN效果总体上更加优秀。