计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：GAN		学号：201600301148
日期：5.19	班级：人工智能	姓名：周阳
Email：862077860@qq.com		

实验目的：

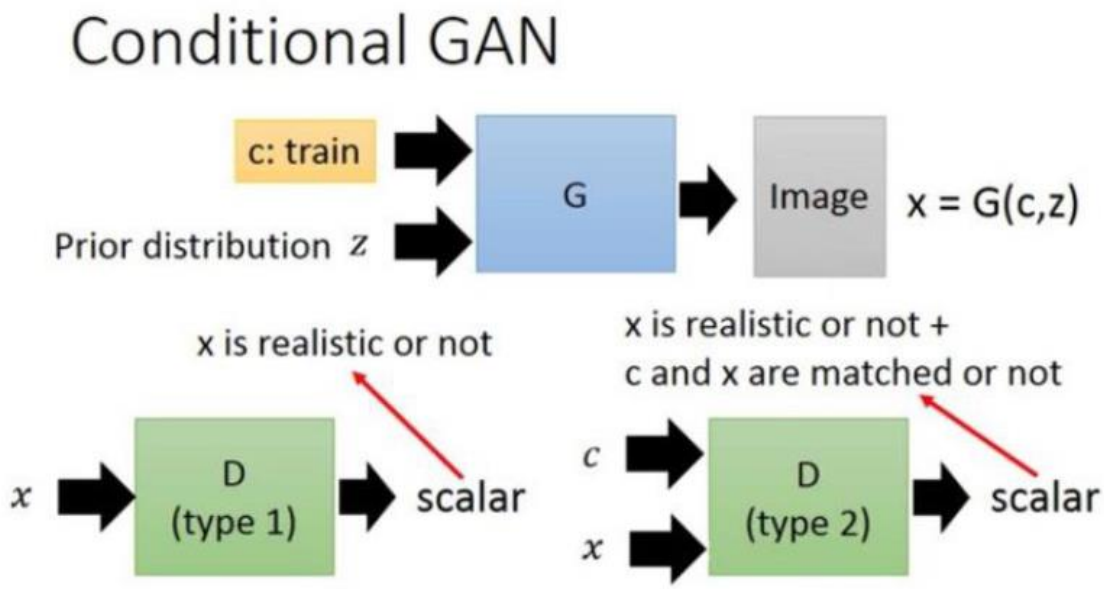
1，学习 使用 pandas 读取数据；
2，学习使用 torch vision 处理数据；
3，练习并使用 pytorch 搭建 GAN

实验软件和硬件环境：

CPU：英特尔至强 E5
GPU：NVIDIA GeForce 1060 6G
内存：16G
Pycharm
Python 3.6

实验原理和方法：

1，CGAN



Cgan 在 Gan 对抗生成的基础上增加了，条件对抗的限定，希望可以通过不同标签，生成不同的图像数据。

Gan 中的噪声输入：$\text{input} = z \sim N(\mu, \sigma^2)$

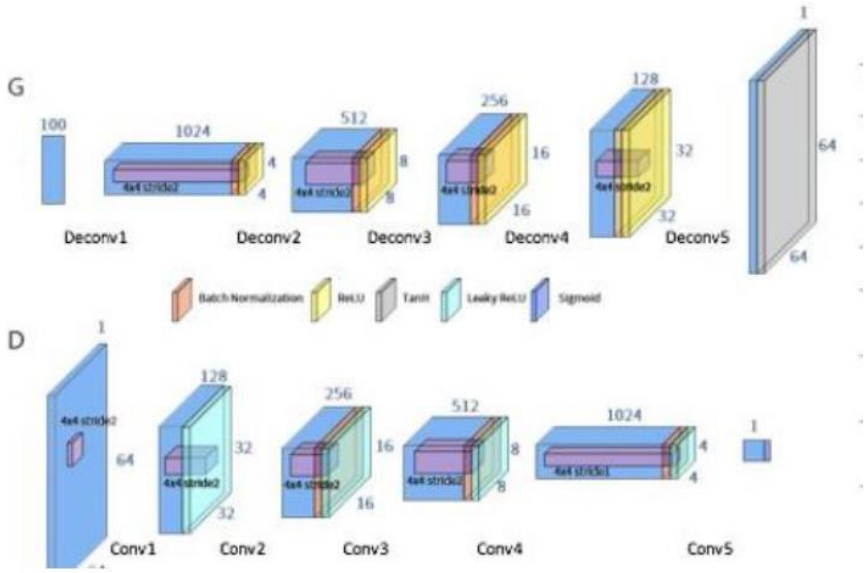
cGan 中的噪声输入：$\text{input} = \text{concat}(z, c), z \sim N(\mu, \sigma^2), c = \text{onehot}(\text{label})$

迭代优化公式：

$$\max_{D}\{\mathbb{E} \sim P_{data} log D(G(x|y)) + \mathbb{E}_{x \sim P_G} \log\Big(1 - \big(D(x|y)\big)\Big)\}$$

训练过程中，G_loss 使 D 判断结果（sigmoid 后的结果）趋向于 1
D_loss 使 G 生成 D 判断结果（sigmoid 后的结果）趋向于 0
D_loss 使真实图像 D 判断结果（sigmoid 后的结果）趋向于 1

2，cDCGAN



cDCGAN 在 CGAN 的基础上引入了 Conv 结构和 Deconv 结构。
使拟合能力更加强大。

实验步骤：（不要求罗列完整源代码）

1，数据加载与数据处理

数据加载:

```
class FashionMnists(Dataset):
    def __len__(self) -> int:
        return len(self.x)
    def __init__(self, csv_file: str,  transform=None) -> None:
        super().__init__()
        self.landmarks_frame = pd.read_csv(csv_file).values
        self.x =  self.landmarks_frame[:,1:].astype('uint8').reshape(-1,28,28)
        self.y =  self.landmarks_frame[:,:1]
        self.transform = transform

    def __getitem__(self, index:int):
        image = self.x[index]
        if self.transform:
            image = self.transform(image)
        image= image[0].reshape(-1,)
        label = self.y[index]
        return image, label
```

```
preprocess = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.repeat(3,1,1)),
        transforms.Normalize(mean=(mean,mean,mean), std=(std,std,std))
])
```

构建 FashionMnist 数据集，继承 Dataset，使用 transform

注意：这里发现使用 transform，如果调整 normalize 为真实的 mean 和 方差 std 会导致结果非常差，在这里卡了很久。最终使用 mean = 0.5，std = 0.5

2，cGAn 生成判别网络

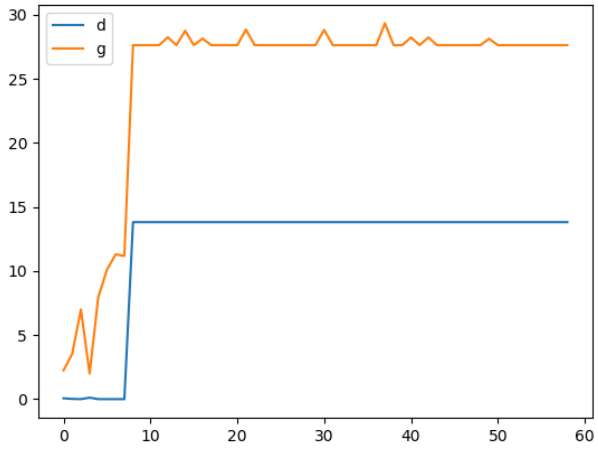
```
class Discriminator(nn.Module):
    '''全连接判别器，用于1x28x28的MNIST数据,输出是数据和类别'''
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(794, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

        #your code
```

判别网络

```
class Generator(nn.Module):
    '''全连接生成器，用于1x28x28的MNIST数据，输入是噪声和类别'''
    def __init__(self, z_dim):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(10 + z_dim, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )
        #your code
```

生成网络

注意：这里的网络结构不能太复杂，之前试过很多网上公布的网络超参数，大多不 work，因为没能非常好的 balance，G 与 D 之间的能力。造成梯度爆炸或者消失，这也是 sigmoid 的弊端。



失败的训练结果

3，训练

训练使用 bce，训练过程如实验原理中分析相同，这里代码过长，不再贴上，可在 github 中看到。

4,DCGAN 生成对抗网络

网络结构:

```
class Generator(nn.Module):
    '''全连接生成器，用于1x28x28的MNIST数据，输入是噪声和类别'''
    def __init__(self,z_dim, d = 128):
        super(Generator, self).__init__()
        self.deconv1_1 = nn.ConvTranspose2d(100, d*2, 4, 1, 0)
        self.deconv1_1_bn = nn.BatchNorm2d(d*2)
        self.deconv1_2 = nn.ConvTranspose2d(10, d*2, 4, 1, 0)
        self.deconv1_2_bn = nn.BatchNorm2d(d*2)
        self.deconv2 = nn.ConvTranspose2d(d*4, d*2, 4, 2, 1)
        self.deconv2_bn = nn.BatchNorm2d(d*2)
        self.deconv3 = nn.ConvTranspose2d(d*2, d, 4, 2, 1)
        self.deconv3_bn = nn.BatchNorm2d(d)
        self.deconv4 = nn.ConvTranspose2d(d, 1, 4, 2, 1)
        #your code

    def forward(self, z, c):
        ## z 是 noise
        x = F.relu(self.deconv1_1_bn(self.deconv1_1(z)))
        y = F.relu(self.deconv1_2_bn(self.deconv1_2(c)))
        x = torch.cat([x, y], 1)
        x = F.relu(self.deconv2_bn(self.deconv2(x)))
        x = F.relu(self.deconv3_bn(self.deconv3(x)))
        x = F.tanh(self.deconv4(x))

        return x
        #your code
```

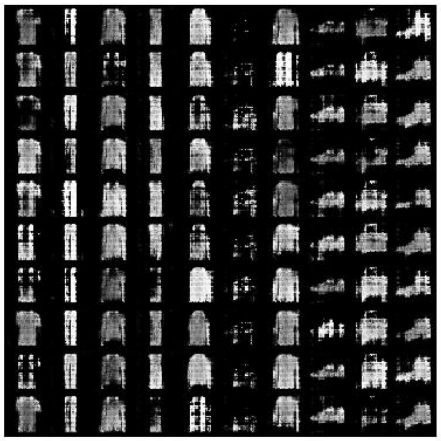
```python
class Discriminator(nn.Module):
    '''全连接判别器，用于1x28x28的MNIST数据,输出是数据和类别'''
    def __init__(self,d = 128):
        super(Discriminator, self).__init__()
        self.conv1_1 = nn.Conv2d(1, d//2, 4, 2, 1)
        self.conv1_2 = nn.Conv2d(10, d//2, 4, 2, 1)
        self.conv2 = nn.Conv2d(d, d*2, 4, 2, 1)
        self.conv2_bn = nn.BatchNorm2d(d*2)
        self.conv3 = nn.Conv2d(d*2, d*4, 4, 2, 1)
        self.conv3_bn = nn.BatchNorm2d(d*4)
        self.conv4 = nn.Conv2d(d * 4, 1, 4, 1, 0)

        #your code

    def forward(self, x, c):#c = class
        x = F.leaky_relu(self.conv1_1(x), 0.2)
        y = F.leaky_relu(self.conv1_2(c), 0.2)
        x = torch.cat([x, y], 1)
        x = F.leaky_relu(self.conv2_bn(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.conv3_bn(self.conv3(x)), 0.2)
        x = F.sigmoid(self.conv4(x))

        return x
        #your code
```

对于 cDCGAN 存在一个非常大的问题：**需要大量的向量维度变换**，使输入 Conv 层的维度一致，并且需要精确地计算，使用 padding 和 view，这中间出了很多 bug。

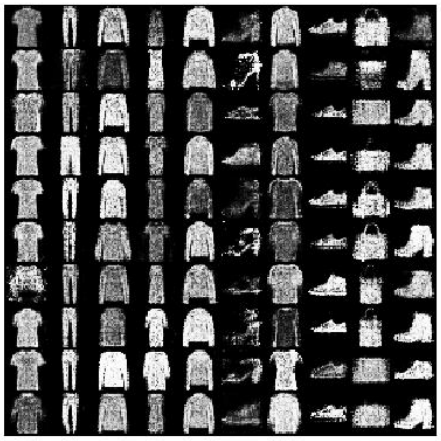
同样 cDCGAN 也非常看 normalize 和网络能力

5，结果讨论

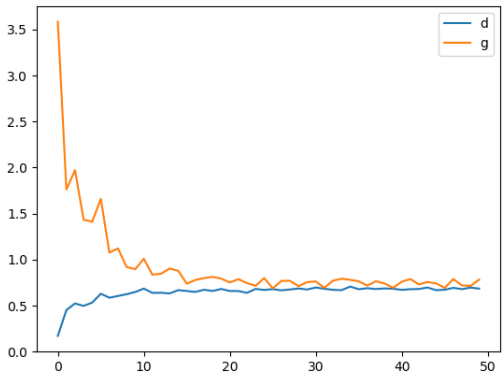
cGAN 生成结果：



cDCGAN 生成结果：



可以看出 cDCGAN 的结果更加清晰。生成质量较高。
正常的优化结果:



正常情况下 GAN 中的 G 和 D 会收敛与一个较小的值。

结论分析与体会：

1，数据处理中的归一化，均值化操作对训练的影响非常大。

2，GAN 的训练超参数非常多，并且影响大的超参数也非常多，调起来非常麻烦，并且 GAN 的训练非常“脆弱”稍有不注意就会导致无法拟合好的结果。

3，本次实验使用 Sigmoid 作为 D 的最后一层，并且使用 bceloss，极其容易梯度爆炸或者梯度消失。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

1，如何选超参数，尤其在本次实验如此“脆弱”的网络前提下：

先借鉴网上，或者论文中的超参数，对于很久才能看到结果的超参数调整，需要我们分析预测训练趋势（是否有梯度爆炸或者梯度消失的趋势），来进行调整。

2，GAN 的训练注意事项：

注意网络能力，注意 Loss Function，注意两个网络的能力。

3，代码服务器训练问题：

①服务器需要设置 matplotlib 的模式进行显示图像

②torchvision save_image 用不了->使用 grid + plt