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： Homework 7		学号：201900130024
日期：2021.11.29	班级： 数据 19	姓名： 刘士渤
Email：liuburger@qq.com		

实验目的：

complete a cGAN.discover how to develop a conditional generative adversarial network for the targeted generation of items of clothing

实验软件和硬件环境：

VScode JupyterNoteBook

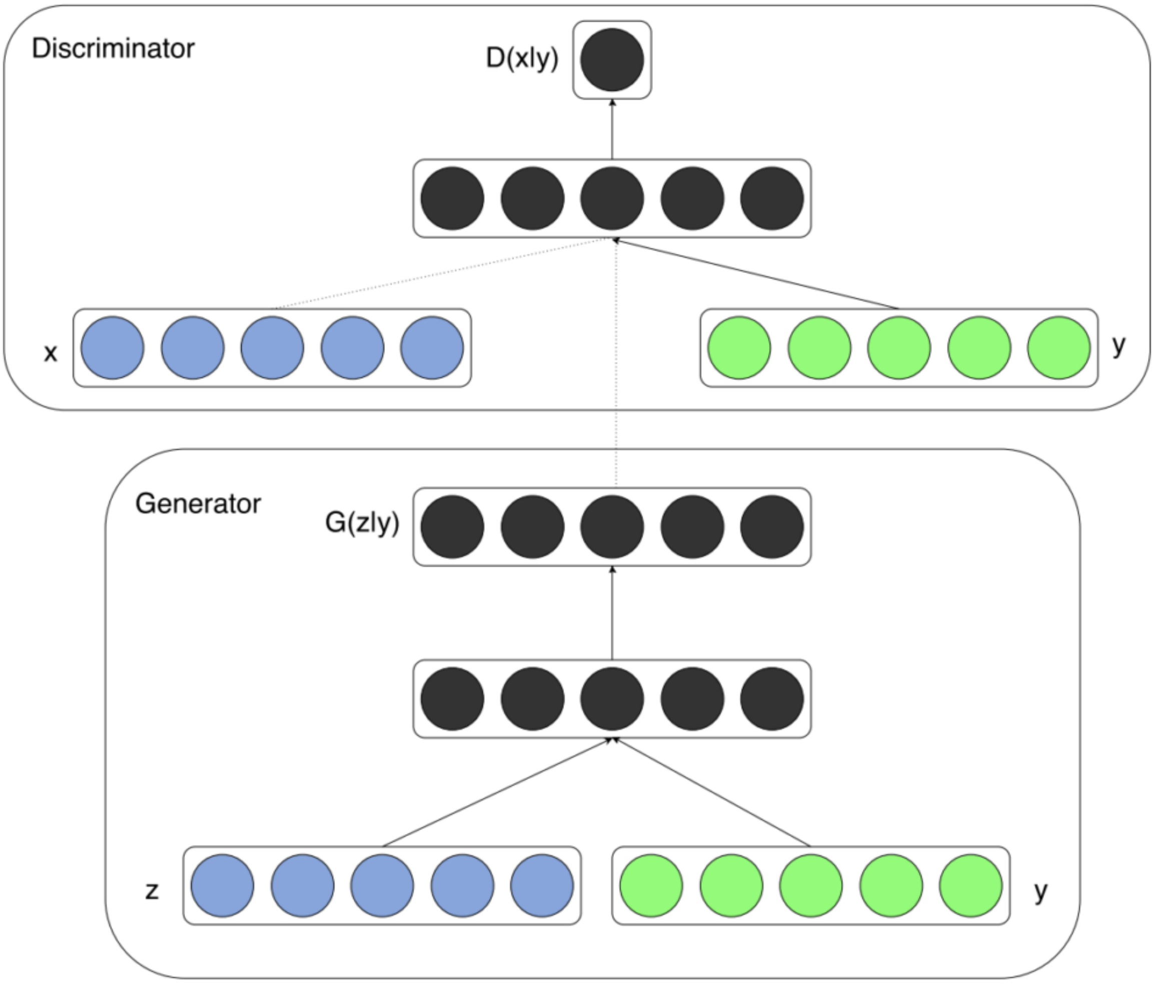
联想拯救者 Y7000p

实验原理和方法：

GAN 主要由 generator 和 discriminator 组成，generator 负责生成新的图片，这些图片在理想情况下与真实图片无法区分，而 discriminator 负责辨别图像真伪。

在 GAN 模型中使用类标签信息有两种动机：改善 GAN、目标图像生成。

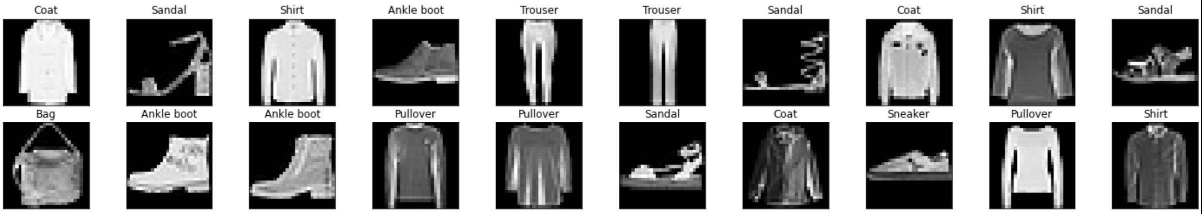
cGAN 结构：



实验步骤：（不要求罗列完整源代码）

1. 补全 cGAN_Pytorch.ipynb：

真实图像如下：



opt 参数如下：

```
Namespace(b1=0.5, b2=0.999, batch_size=64, channels=1, img_size=28,
label_dim=50, latent_dim=100, lr=0.0002, n_classes=10, n_cpu=8,
n_epochs=200)
```

- Generator：
  为了简化代码定义一个 block 函数（主要是为了方便正则化），然后进行(100+10)--->128--->256--->512--->1024--->(1,28,28)的 linear 和 ReLU：

```
### START CODE HERE
def block(in_feat,out_feat,normalize=True):
    layers=[nn.Linear(in_feat,out_feat)]
    if normalize:
        layers.append(nn.BatchNorm1d(out_feat,0.8))
    layers.append(nn.LeakyReLU(0.2,inplace=True))
    return layers

self.model=nn.Sequential(
    *block(opt.latent_dim+opt.n_classes,128,normalize=False),
    *block(128, 256),
    *block(256, 512),
    *block(512, 1024),
    nn.Linear(1024, int(np.prod(img_shape))),
    nn.Tanh()
)
### END CODE HERE
```

generator 的输入是噪声和标签，将它们拼接到一起作为输入：

```
### START CODE HERE
gen_input = torch.cat((self.label_embedding(labels), noise), -1)
img = self.model(gen_input)
img = img.view(img.size(0), *img_shape)
### END CODE HERE
```

- Discriminator：
  进行(10+784)--->512--->512--->512--->1 的 linear、ReLU 和 Dropout：

```
### START CODE HERE
self.model = nn.Sequential(
    nn.Linear(opt.n_classes + int(np.prod(img_shape)), 512),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Linear(512, 512),
    nn.Dropout(0.4),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Linear(512, 512),
    nn.Dropout(0.4),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Linear(512, 1),
)
### END CODE HERE
```

discriminator 的输入是 generator 生成的图像（或真实图像）和标签，将它们拼接到一起作为输入：

```
### START CODE HERE
d_in = torch.cat((img.view(img.size(0), -1), self.label_embedding(labels)), -1)
validity = self.model(d_in)
### END CODE HERE
```

- training process：
  利用写好的 Generator 类和 Discriminator 类初始化：

```
# Initialize generator and discriminator
generator = Generator()
discriminator = Discriminator()
```

训练 Generator，用 np.random 生成噪声 z 和 labels，然后作为 generator 的输入生成 img；
用 discriminator 计算生成图像的准确率，并计算 Generator 的 loss。

```
### START CODE HERE
optimizer_G.zero_grad( )
z = FloatTensor(np.random.normal(0, 1, (batch_size, opt.latent_dim)))
gen_labels = LongTensor(np.random.randint(0, opt.n_classes, batch_size))
gen_imgs = generator(z, gen_labels)
# 计算生成器loss
validity = discriminator(gen_imgs, gen_labels)
g_loss = adversarial_loss(validity, valid)
g_loss.backward()
optimizer_G.step()
### END CODE HERE
```

训练 Discriminator，分别计算真实图像和生成图像的准确率和 loss，然后求出二者的平均 loss。

```
### START CODE HERE
optimizer_D.zero_grad( )
validity_real = discriminator(real_imgs, labels)
d_real_loss = adversarial_loss(validity_real, valid)
validity_fake = discriminator(gen_imgs.detach(), gen_labels)
d_fake_loss = adversarial_loss(validity_fake, fake)
# 计算总loss
d_loss = (d_real_loss + d_fake_loss) / 2
d_loss.backward()
optimizer_D.step()
### END CODE HERE
```

- generate_latent_points:
  仍然是用 np.random 生成噪声 z 和 labels 。

```
### START CODE HERE
z = FloatTensor(np.random.normal(0, 1, (n_samples, latent_dim)))
labels = LongTensor(randint(0,n_classes,n_samples))
### END CODE HERE
```

结论分析与体会：

1. Epoch 结果：

✓ 63m 10.9s

```
[Epoch 0/200] [Batch 0/938] [D loss: 0.480817] [G loss: 0.972361]
[Epoch 0/200] [Batch 1/938] [D loss: 0.325524] [G loss: 0.949323]
[Epoch 0/200] [Batch 2/938] [D loss: 0.190867] [G loss: 0.923571]
[Epoch 0/200] [Batch 3/938] [D loss: 0.101495] [G loss: 0.890251]
[Epoch 0/200] [Batch 4/938] [D loss: 0.033324] [G loss: 0.865778]
[Epoch 0/200] [Batch 5/938] [D loss: 0.023246] [G loss: 0.844929]
[Epoch 0/200] [Batch 6/938] [D loss: 0.043621] [G loss: 0.817807]
[Epoch 0/200] [Batch 7/938] [D loss: 0.020990] [G loss: 0.824161]
[Epoch 0/200] [Batch 8/938] [D loss: 0.022063] [G loss: 0.822055]
[Epoch 0/200] [Batch 9/938] [D loss: 0.025033] [G loss: 0.810219]
[Epoch 0/200] [Batch 10/938] [D loss: 0.019088] [G loss: 0.797587]
[Epoch 0/200] [Batch 11/938] [D loss: 0.017300] [G loss: 0.771797]
[Epoch 0/200] [Batch 12/938] [D loss: 0.025198] [G loss: 0.758500]
[Epoch 0/200] [Batch 13/938] [D loss: 0.023753] [G loss: 0.739448]
[Epoch 0/200] [Batch 14/938] [D loss: 0.021051] [G loss: 0.745617]
[Epoch 0/200] [Batch 15/938] [D loss: 0.025097] [G loss: 0.725289]
[Epoch 0/200] [Batch 16/938] [D loss: 0.027201] [G loss: 0.707708]
[Epoch 0/200] [Batch 17/938] [D loss: 0.026181] [G loss: 0.680913]
[Epoch 0/200] [Batch 18/938] [D loss: 0.026075] [G loss: 0.668485]
[Epoch 0/200] [Batch 19/938] [D loss: 0.031924] [G loss: 0.649388]
[Epoch 0/200] [Batch 20/938] [D loss: 0.033294] [G loss: 0.654303]
[Epoch 0/200] [Batch 21/938] [D loss: 0.034389] [G loss: 0.659844]
[Epoch 0/200] [Batch 22/938] [D loss: 0.033328] [G loss: 0.644243]
[Epoch 0/200] [Batch 23/938] [D loss: 0.034868] [G loss: 0.614190]
[Epoch 0/200] [Batch 24/938] [D loss: 0.033376] [G loss: 0.624410]
show more (open the raw output data in a text editor) ...

[Epoch 199/200] [Batch 933/938] [D loss: 0.238823] [G loss: 0.277628]
[Epoch 199/200] [Batch 934/938] [D loss: 0.241450] [G loss: 0.258911]
[Epoch 199/200] [Batch 935/938] [D loss: 0.257828] [G loss: 0.239909]
[Epoch 199/200] [Batch 936/938] [D loss: 0.244199] [G loss: 0.256798]
[Epoch 199/200] [Batch 937/938] [D loss: 0.224027] [G loss: 0.287491]
```

D loss 是一个由小到大的过程（除刚开始的几个很大），G loss 是一个从大到小的过程；

G loss 变小的原因是：随着训练次数增加，Generator 生成的图片真实度越来越高；

D loss 变大的原因是：由于 Generator 以假乱真的程度越来越高，所以 Discriminator 的误判也高了；

而一开始 D loss 较高的原因是训练少，容易判断错误。

2. 由 Generator 生成的图片（如果不放大追究细节的话真实度还是挺高的）：

3. 我觉得 GAN 的 Generator 和 Discriminator 有点像生物中的捕食与被捕食关系，如豹子和梅花鹿的关系，二者互相选择、互相进化：为了获取食物，只有跑得快的豹子才能生存；为了逃脱豹子的追捕，只有跑的快的梅花鹿才能生存；以至于豹子和梅花鹿的奔跑速度都很快。

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. 这次实验同样要用到 cuda 加速，如果没有 cuda 加速可能会使 1 个小时的 Epoch 变成 3 个小时。

2. 代码框架略有问题，需要将 Generator 和 Discriminator 这两个类的__init__中的 nn.Embedding 的第二个参数 opt.label_dim 改为 opt. n_classes 才可以正确运行之后的 Epoch。