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：Conditional GAN		学号：201900301174
日期：2021.11.30	班级： 智能19	姓名： 韩旭
Email：hanx@mail.sdu.edu.cn		

实验目的：

Homework 7

Due: 2021-12-8 (Wed.) 8pm

Introduction

In this assignment, you will complete a conditional GAN (cGAN) . you will discover how to develop a conditional generative adversarial network for the targeted generation of items of clothing. You will know:

- The limitations of generating random samples with a GAN that can be overcome with a conditional generative adversarial network.
- How to develop and evaluate a conditional generative adversarial network for generating photos of items of clothing.

实验软件和硬件环境：

Xeon Gold 6226e
RTX 3090
Pytorch 1.10
CUDA 11.3
Cudnn 8.2.5
Jupyter notebook

实验原理和方法：

- **Experiments**

There are ### START CODE HERE/### END CODE HERE tags denoting the start and end of code sections you should fill out. Take care to not delete or modify these tags, or your assignment may not be properly graded.

- **Q1: Conditional GAN (100 points)**

  The Jupyter notebooks cGAN-PyTorch.ipynb will introduce the pipeline of developing and evaluate a conditional generative adversarial network for generating photos of items of clothing.

- **See the code file for details.**

实验步骤：（不要求罗列完整源代码）

生成器：按照推荐的结构，并且使用 BN 和 LeakyReLU

(100+50)--->128--->256--->512--->1024--->(1,28,28)

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.label_embedding = nn.Embedding(opt.n_classes, opt.label_dim)
        ## TODO: There are many ways to implement the model,  one alternative
        ## architecture is (100+50)--->128--->256--->512--->1024--->(1,28,28)

        ### START CODE HERE
        self.block1 = nn.Sequential(*[
            nn.Linear(150, 128),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.block2 = nn.Sequential(*[
            nn.Linear(128, 256),
            nn.BatchNorm1d(256, 0.8),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.block3 = nn.Sequential(*[
            nn.Linear(256, 512),
            nn.BatchNorm1d(512, 0.8),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.block4 = nn.Sequential(*[
            nn.Linear(512, 1024),
            nn.BatchNorm1d(1024, 0.8),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.fc = nn.Linear(1024, 784)
        self.tanh = nn.Tanh()
        ### END CODE HERE
```

```python
def forward(self, noise, labels):

    ### START CODE HERE
    img = torch.cat((self.label_embedding(labels),noise),dim=1)
    img = self.block1(img)
    img = self.block2(img)
    img = self.block3(img)
    img = self.block4(img)
    img = self.fc(img)
    img = self.tanh(img)
    img = img.view(img.size(0), -1)
    ### END CODE HERE
```

辨别器：网络结构为题中所推荐，并使用 LeakyReLU 和 Dropout

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.label_embedding = nn.Embedding(opt.n_classes, opt.n_classes)
        ## TODO: There are many ways to implement the discriminator, one alternative
        ## architecture is (100+784)--->512--->512--->512--->1

        ### START CODE HERE
        self.block1 = nn.Sequential(*[
            nn.Linear(opt.n_classes+int(np.prod(img_shape)),512),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.block2 = nn.Sequential(*[
            nn.Linear(512, 512),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.block3 = nn.Sequential(*[
            nn.Linear(512, 512),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True)
        ])
        self.fc = nn.Linear(512 ,1)
        self.softmax = nn.Softmax()
        ### END CODE HERE
```

```python
def forward(self, noise, labels):

    ### START CODE HERE
    img = torch.cat((self.label_embedding(labels),noise),dim=1)
    img = self.block1(img)
    img = self.block2(img)
    img = self.block3(img)
    img = self.block4(img)
    img = self.fc(img)
    img = self.tanh(img)
    img = img.view(img.size(0), -1)
    ### END CODE HERE

    return img
```

训练生成器:

要记得 Pytorch 训练的流程:

1,Model.zero_grad()

2,Calculate loss

3,Loss.backward()

4,Model.step()

```python
# -----------------
#  Train Generator
# -----------------

### START CODE HERE
optimizer_G.zero_grad()
noise = Variable(FloatTensor(np.random.normal(0, 1, (batch_size, opt.latent_dim))))
g_label = Variable(LongTensor(np.random.randint(0, opt.n_classes, batch_size)))

g_img = generator(noise, g_label)
tem_validity = discriminator(g_img, g_label)

g_loss = adversarial_loss(tem_validity, valid)

g_loss.backward()
optimizer_G.step()
### END CODE HERE
```

训练辨别器：

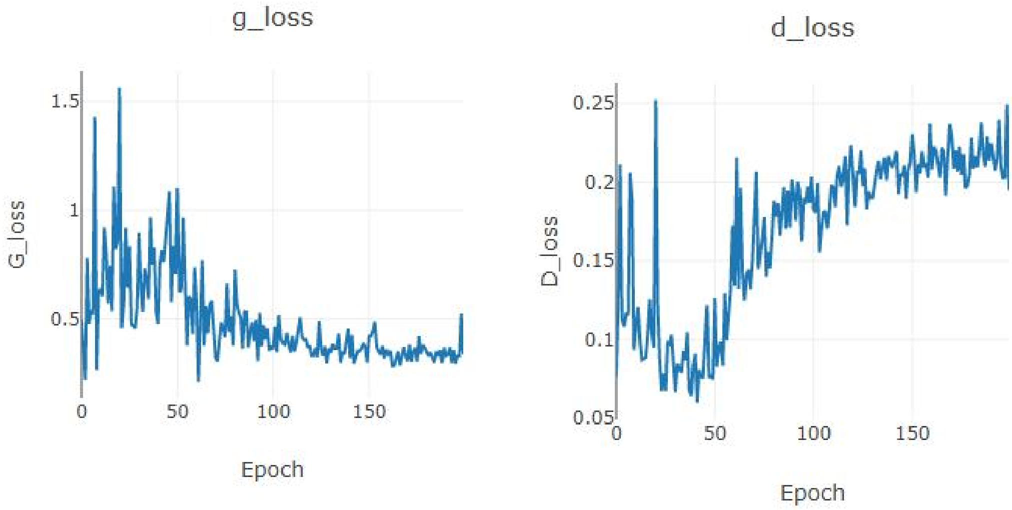
注意辨别器使用两部分 loss，一部分是辨别真实图片，一部分是辨别生成器生成的假图片，两部分 loss 直接取平均。

```python
# ---------------------
#  Train Discriminator
# ---------------------

### START CODE HERE
optimizer_D.zero_grad()
d_validity = discriminator(real_imgs, labels)
d_loss = adversarial_loss(d_validity, valid)

d_validity = discriminator(g_img.detach(), g_label)
d_loss += adversarial_loss(d_validity, fake)
d_loss /= 2

d_loss.backward()
optimizer_D.step()
### END CODE HERE
```

部分训练过程：

```
executed in 32m 33s, finished 01:21:43 2021-11-30
[Epoch 195/200] [Batch 0/235] [G loss: 0.428614] [D loss: 0.206805]
[Epoch 195/200] [Batch 100/235] [G loss: 0.413901] [D loss: 0.185027]
[Epoch 195/200] [Batch 200/235] [G loss: 0.420093] [D loss: 0.205255]
[Epoch 196/200] [Batch 0/235] [G loss: 0.360762] [D loss: 0.202741]
[Epoch 196/200] [Batch 100/235] [G loss: 0.426333] [D loss: 0.190696]
[Epoch 196/200] [Batch 200/235] [G loss: 0.373713] [D loss: 0.192984]
[Epoch 197/200] [Batch 0/235] [G loss: 0.441131] [D loss: 0.192596]
[Epoch 197/200] [Batch 100/235] [G loss: 0.454266] [D loss: 0.197696]
[Epoch 197/200] [Batch 200/235] [G loss: 0.388484] [D loss: 0.197512]
[Epoch 198/200] [Batch 0/235] [G loss: 0.337241] [D loss: 0.209550]
[Epoch 198/200] [Batch 100/235] [G loss: 0.461028] [D loss: 0.207623]
[Epoch 198/200] [Batch 200/235] [G loss: 0.362166] [D loss: 0.187764]
[Epoch 199/200] [Batch 0/235] [G loss: 0.405142] [D loss: 0.196623]
[Epoch 199/200] [Batch 100/235] [G loss: 0.407082] [D loss: 0.199535]
[Epoch 199/200] [Batch 200/235] [G loss: 0.370894] [D loss: 0.203177]
[Epoch 200/200] [Batch 0/235] [G loss: 0.385040] [D loss: 0.193255]


[Epoch 200/200] [Batch 100/235] [G loss: 0.386506] [D loss: 0.198166]
[Epoch 200/200] [Batch 200/235] [G loss: 0.414721] [D loss: 0.194395]
```

具体的两部分 loss 变化可视化结果：



可以看到生成器的 loss 一开始很高，后来降低最后小幅度抖动，而辨别器的 loss 一开始低，后来升高最后小幅度抖动。我们可以得出结论，生成器一开始生成的图片很简单，于是辨别器就很容易辨别出来，但是随着训练的进行，生成器的准确度逐渐提高，生成的图片更加逼真，使辨别器很难辨别出来，于是生成器 loss 下降而辨别器 loss 升高，而最后网络趋于收敛后生成器和辨别器虽然仍在对抗但 loss 浮动不大。

generate_latent_points():

实验结果：

```python
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples, n_classes):
    # Sample noise

    ### START CODE HERE
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    z = x_input.reshape(n_samples, latent_dim)
    # generate labels
    labels = randint(0, n_classes, n_samples)

    ### END CODE HERE

    return z,labels
```

直接使用训练好的模型进行测试：

```python
# # load model
generator=Generator()
generator.cuda()
generator.load_state_dict(torch.load('./cgan_generator_1_200.pth'))
generator.eval()



z, labels = generate_latent_points(100, 100, 10)
labels = asarray([x for _ in range(10) for x in range(10)])
z = torch.tensor(z).cuda().to(torch.float32)
labels = torch.tensor(labels).cuda()
X  = generator(z, labels).cpu().detach()
X = torch.reshape(X, [100, 28, 28])
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
save_plot(X, 10)
```

实验结果：

这里训练了三个模型 cgan_generator_0_XXX.pth、

cgan_generator_XXX.pth、cgan_generator_1_XXX.pth

这里三个模型的不同点就在于生成器和辨别器的 BN 或者 Dropout 位置或者参数稍微不同，其他网络结构相同，本实验报告中代码所示的具体结果是 cgan_generator_1_XXX.pth，因为训练 200 epoch 后这个模型效果最好。

具体三个模型的实验结果：

1，cgan_generator_0_200.pth

这个实验结果可以比较清晰地看到生成的衣服图片，较为逼真。

2，本实验报告中对应结构 cgan_generator_1_XX.pth 的结果：

可以看到训练 20 个 epoch 后的结果很差，只大概学到了轮廓：

2.1 cgan_generator_1_20.pth

2.2 cgan_generator_1_200.pth

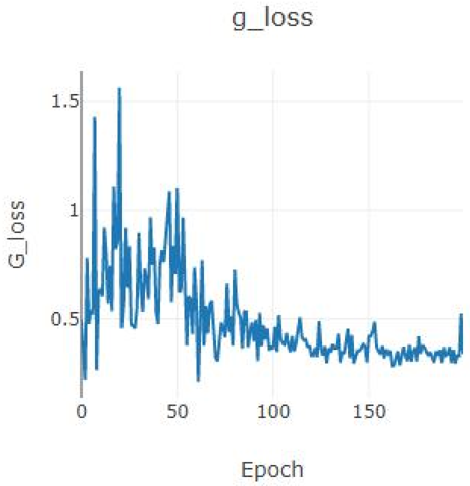
可以看到训练 200 个 epoch 后的模型准确率大大提升，可以非常精确的生成衣服的图像。

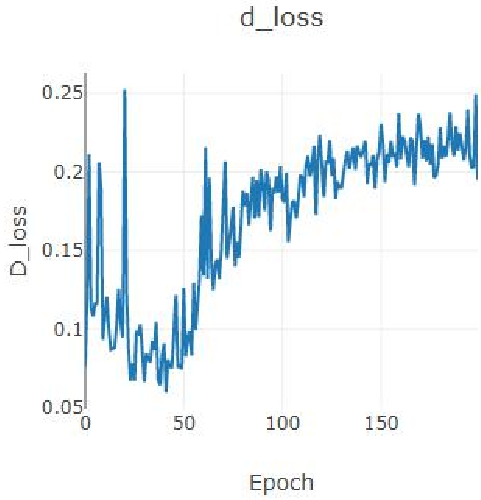
3，cgan_generator_200.pth

最后一个模型结果也是较为准确。

结论分析与体会：

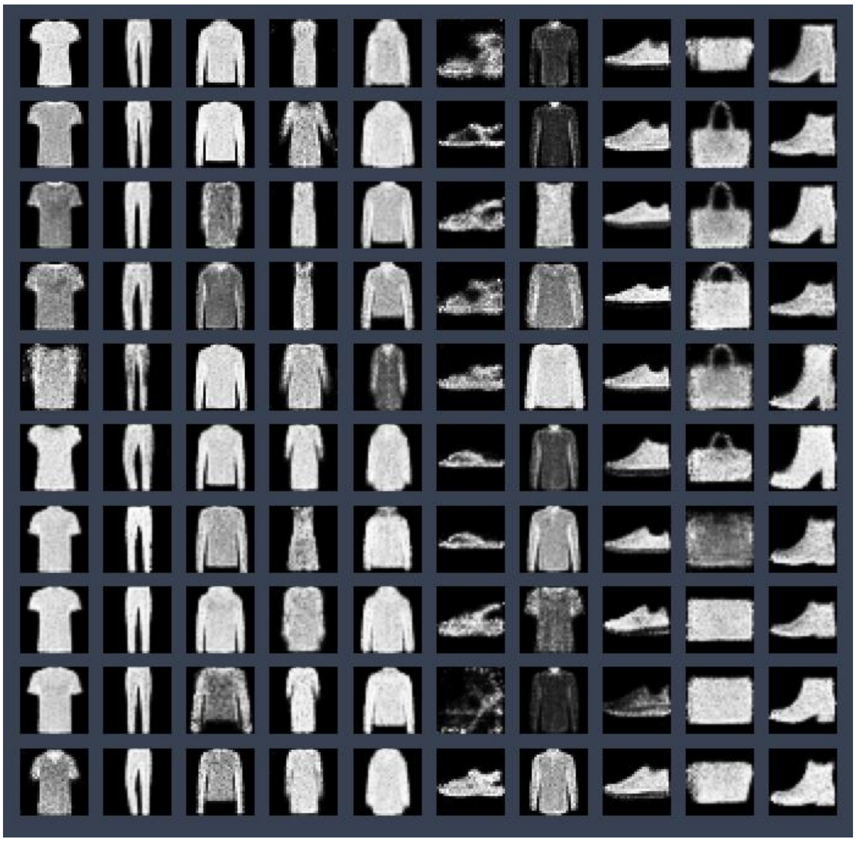
1，可以根据可视化训练过程观察效果：





可以看到生成器的 loss 一开始很高，后来降低最后小幅度抖动，而辨别器的 loss 一开始低，后来升高最后小幅度抖动。我们可以得出结论，生成器一开始生成的图片很简单，于是辨别器就很容易辨别出来，但是随着训练的进行，生成器的准确度逐渐提高，生成的图片更加逼真，使辨别器很难辨别出来，于是生成器 loss 下降而辨别器 loss 升高，而最后网络趋于收敛后生成器和辨别器虽然仍在对抗但 loss 浮动不大。

2，文中代码所示结构经过训练 200 epoch 后效果最准确，可以生成非常逼真的图片。



本实验的网络结构参考：https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/cgan/cgan.py

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， GAN 的训练过程可能会很有迷惑性，不能只看生成器或辨别器 loss 变化不大后就停止训练，最好再继续训练一段时间，而且可以通过其他工具使实验过程可视化，这里使用 visdom。

2， 对于训练过程慢的问题，一是可以改变优化器，使用例如 SGD 等收敛更快的优化器，这里实验过 SGD 发现辨别器 loss 下降太快，直接使辨别器在很少的 epoch 后达到收敛，而这时候生成器的能力还很弱就已经被辨别器打败了于是 loss 一直上升，无法收敛。于是这里使用 Adam 的优化方法 AdamW，发现结果较好。