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

Experiment 6

实验题目：在这次作业中，我将完成一个conditional GAN (cGAN)。我将学会如何开发一个有条件的生成对抗网络目标的衣服项目。

实验目标和学习任务：

- 使用cGAN来克服使用GAN来生成随机样例的局限
- 开发和评估一个生成衣服照片的cGAN

实验学生基本信息：

- 姓名：陈佳睿
- 学号：201700301042
- 班级：17人工智能班

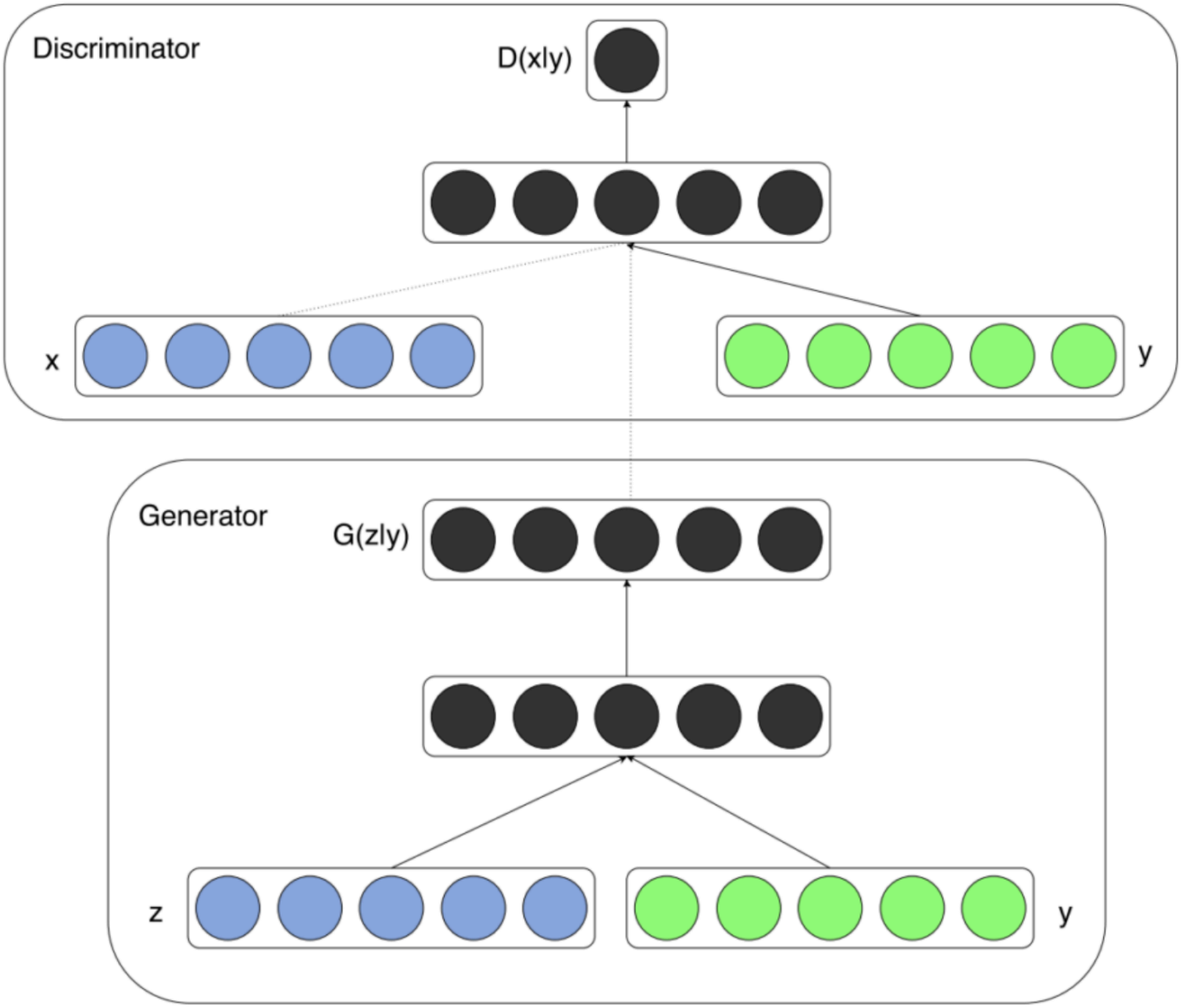
日期：2020.5.13

GAN主要由 `generator` 和 `discriminator` 组成
generator 负责生成新的合理示例，这些示例在理想情况下与数据集中的真实示例是无法区分的，而discriminator模型负责将给定图像分类为真实图像（从数据集提取）或伪图像（生成）

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

论文中提出的CGAN模型的架构：

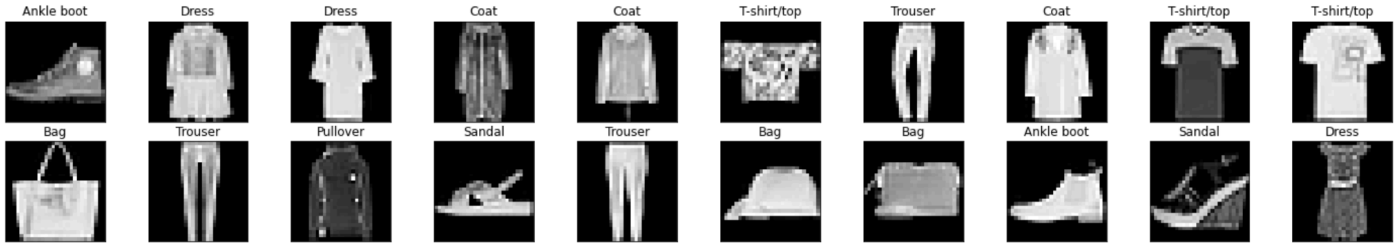


与输入图像相关的其他信息（例如类别标签）可用于改善GAN。这种改进可以实现更稳定、更快的训练效果或生成的图像以更好的质量出现

GAN模型的局限性：潜在空间中的点和生成的图像之间存在关系这种关系复杂且难以映射。

我们使用 `Fashion-Mnist` 数据集来进行CGAN模型的训练，数据集一共有10个种类的不同衣物，比如t恤，鞋子，包之类的，训练数据集规模：60000

打印20张数据集的照片以及对应的label（图像已经通过cmap函数进行了灰度化处理，我们只处理一个通道的信息）



具体来说，`generator` 将学习如何使用 `discriminator` 生成新的服装项目，该区分器将尝试区分来自Fashion MNIST训练数据集的真实图像和生成器模型使用噪声来生成的新图像。

我们的 `opt参数` 设置如下：

```
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)
```

对于 `generator` 我们有两个输入： `latent space` 中的一个点和图像的类标签，我们输出单个 `32*32` 灰度图像。

超参数调整结果：

```
import argparse
##TODO: you can tune the Hyperparameters as you need
parser = argparse.ArgumentParser()
parser.add_argument("--n_epochs", type=int, default=200, help="number of epochs of training")
parser.add_argument("--batch_size", type=int, default=64, help="size of the batches")
parser.add_argument("--lr", type=float, default=0.0002, help="adam: learning rate")
parser.add_argument("--b1", type=float, default=0.5, help="adam: decay of first order momentum of gradient")
parser.add_argument("--b2", type=float, default=0.999, help="adam: decay of first order momentum of gradient")
parser.add_argument("--n_cpu", type=int, default=8, help="number of cpu threads to use during batch generation")
parser.add_argument("--label_dim", type=int, default=50, help="dimensionality of the label embedding")
parser.add_argument("--latent_dim", type=int, default=100, help="dimensionality of the noise embedding")
parser.add_argument("--n_classes", type=int, default=10, help="number of classes for dataset")
parser.add_argument("--img_size", type=int, default=32, help="size of each image dimension")
parser.add_argument("--channels", type=int, default=1, help="number of image channels")
```

对于数据集的加载，是继承自 Dataset，使用 transform
尝试了如果调整 `normalize` 为真实的 `mean` 和 `方差 std` 会导致结果非常差

```
# Configure data loader
dataloader = torch.utils.data.DataLoader(
    FashionMNIST(root='./data',
        train=True,
        download=False,
        transform=transforms.Compose(
            [transforms.Resize(opt.img_size), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]
        ),
    ),
    batch_size=opt.batch_size,
    shuffle=True,
)
```

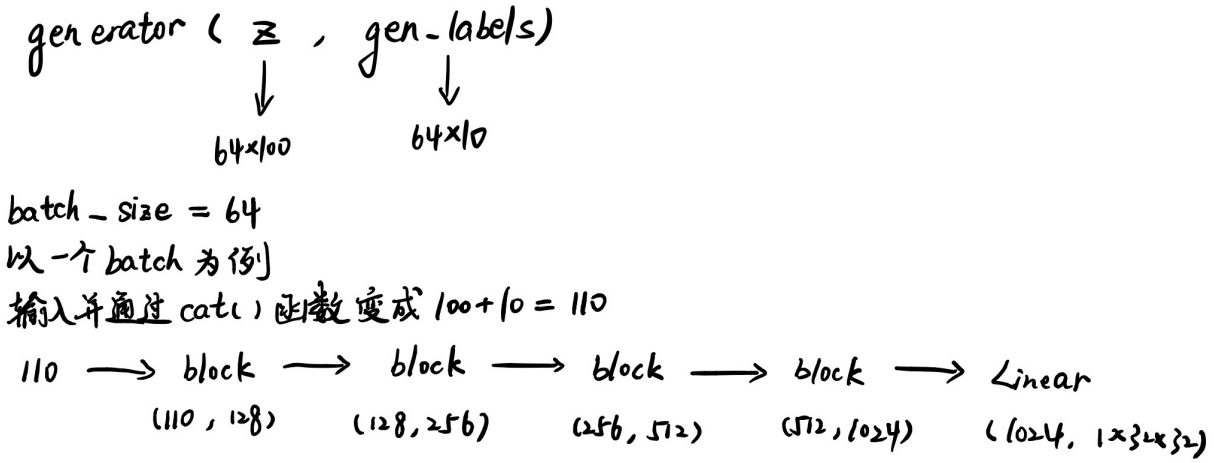
我们自己调整后的 `Generator` 实现代码：

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

        self.label_embedding = nn.Embedding(opt.n_classes, opt.n_classes)

        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()
        )
```

`Generator` 训练 `for-loop` 中的实现：

```
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()
```

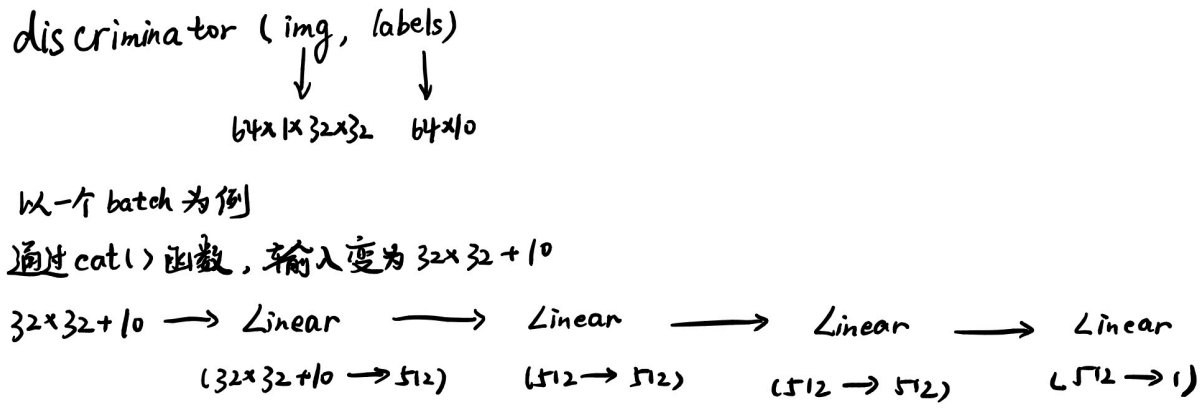
所以可以得到整个的网络结构如下：



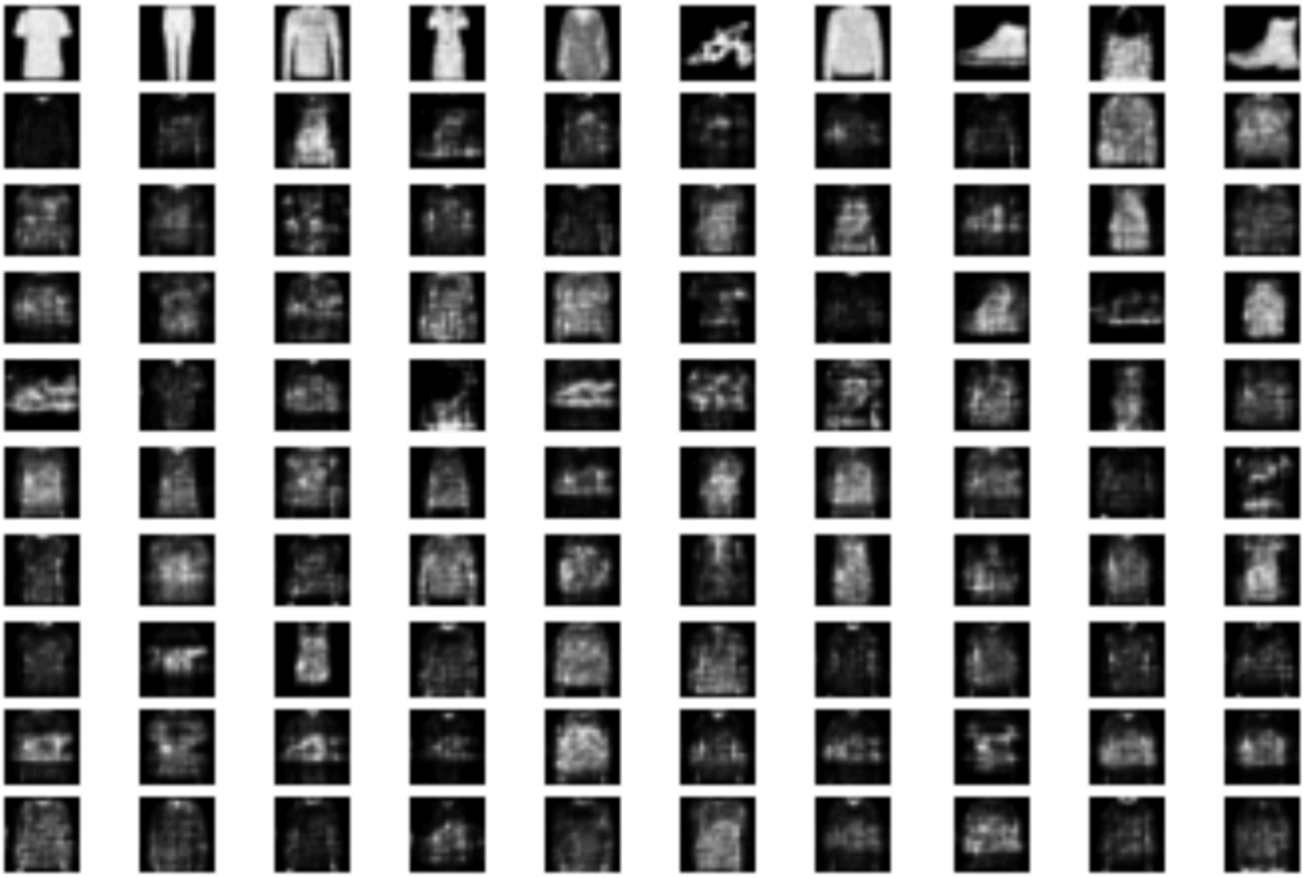
最后得到 `discriminator` 的实现代码：

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

        self.label_embedding = nn.Embedding(opt.n_classes, opt.n_classes)
        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),
        )
        ###


    def forward(self, img, labels):
        # Concatenate label embedding and image to produce input
        d_in = torch.cat((img.view(img.size(0), -1), self.label_embedding(labels)), -1)
        validity = self.model(d_in)
        return validity
        ###
```

`discriminator` 网络结构推导：

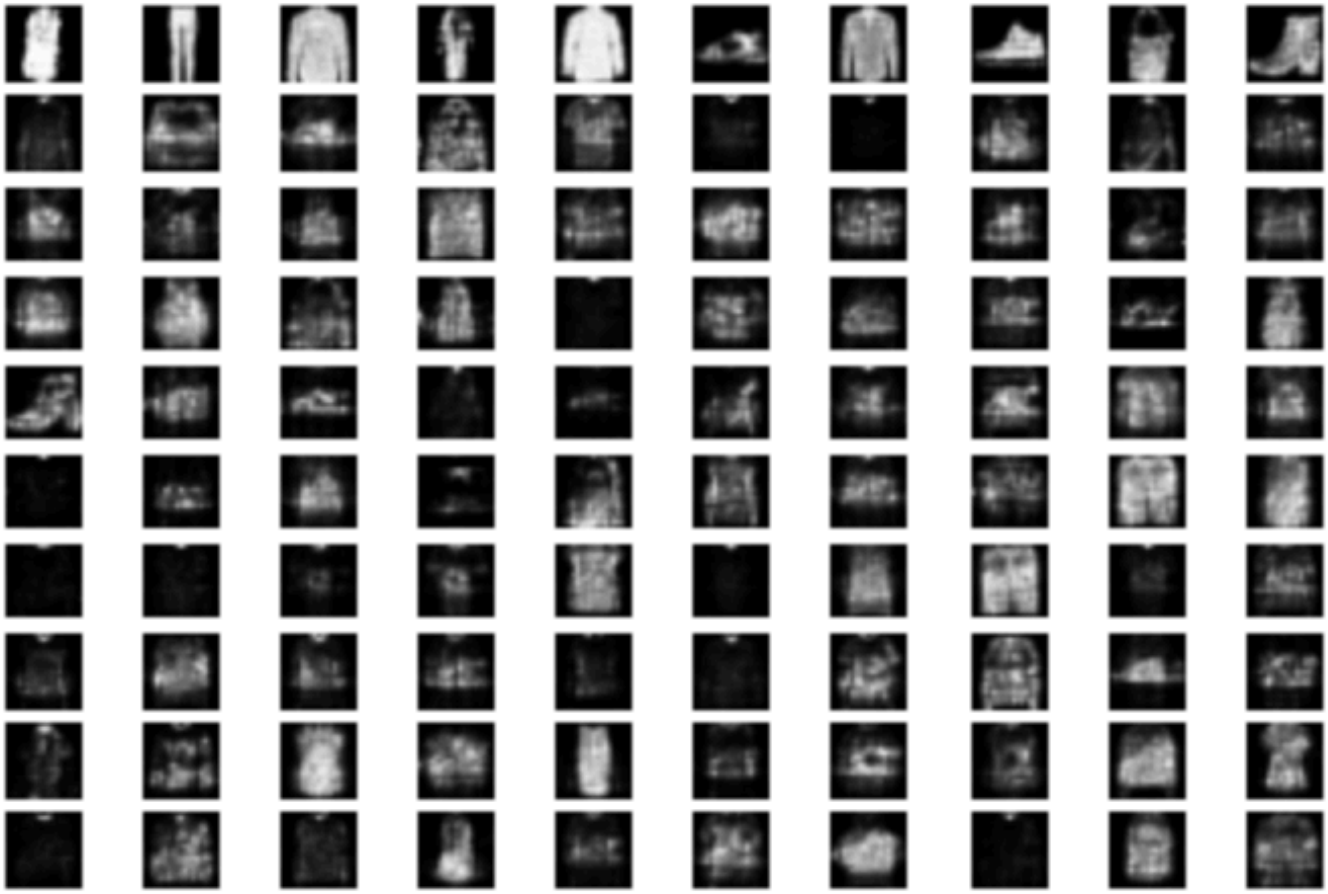


最后加载我们训练得到的结果：



（感觉效果不是很好）

相比之下训练的epoch轮次变大之后，效果有变好：



实验心得体会：

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。