# 《人工智能导论》大作业

任务名称：完成Mnist条件生成器

完成组号：#

小组人员：陈意博、陈嘉蔚、郑宇轩、宋云浩、曹丞铭

完成时间：2023年5月27日

**1．任务目标**

基于Mnist数据集，构建一个条件生成模型，输入的条件为0~9的数字，输出对应条件的生成图像。

要求：

- [x] 支持随机产生输出图像；
- [x] 在cpu上有合理的运行时间。

**2．具体内容**

**（1）实施方案**

## 使用cDCGAN模型训练Mnist数据集。

1）导入模块
2）全局配置（训练设备，训练参数和数据加载器）
3）权重初始化
4）构造生成器类别并实例化
5）构造判别器类别并实例化
6）定义损失函数，生成固定噪声和标签
7）创建生成器和鉴别器网络和优化器
8）开始训练并生成图像

**（2）核心代码分析**

## 神经网络

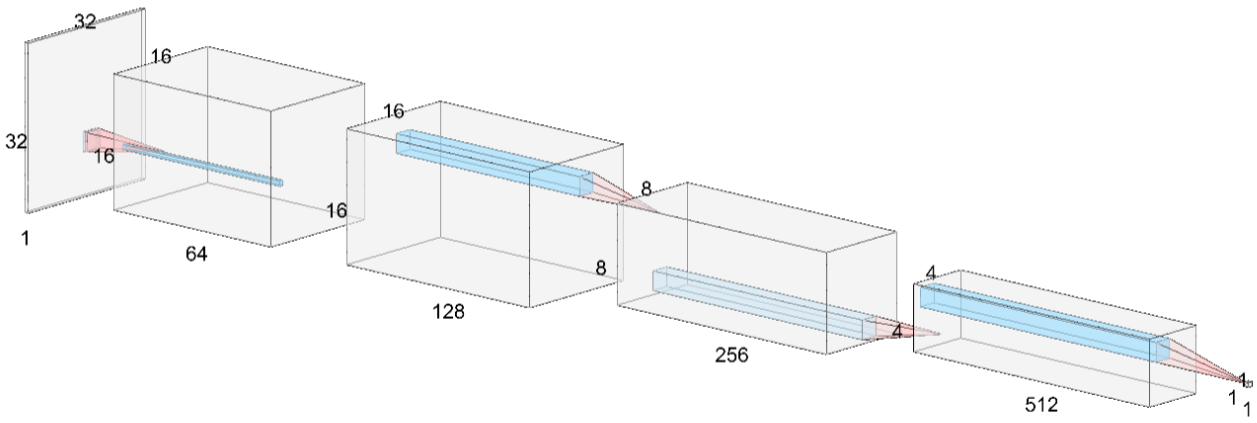
cDCGAN是生成对抗网络CGAN的改进版本，它使用了多层卷积结构以达到更好的输出效果，但是对于CPU的运算要求也更高了。其中最重要的是生成器和判别器的网络结构：

```python
class Discriminator(nn.Module):
    # initializers
    def __init__(self, d=128):
        super(Discriminator, self).__init__()
        self.conv1_1 = nn.Conv2d(1, int(d / 2), 4, 2, 1)
        self.conv1_2 = nn.Conv2d(10, int(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)

    # weight_init
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    # forward method
    def forward(self, input, label):
        x = F.leaky_relu(self.conv1_1(input), 0.2)
        y = F.leaky_relu(self.conv1_2(label), 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 = torch.sigmoid(self.conv4(x))

        return x
```




判别器结构如上图所示，其使用四层卷积完成判别工作。其中，由[64,16,16]的Tensor变为[128,64,64]的Tensor是利用 `x = torch.cat([x, y], 1)` 将label特征与图片特征结合使得通道数加倍。判别器接收一个单通道的[32,32]灰度图，输出一个浮点数，表示该图片为手写图片

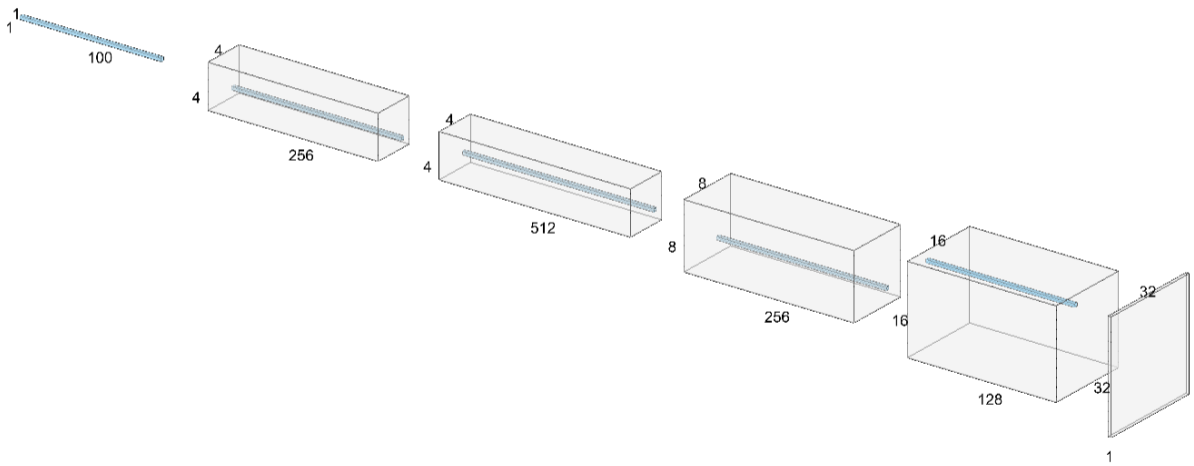
的概率值。

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

    # weight_init
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    # forward method
    def forward(self, input, label):
        x = F.relu(self.deconv1_1_bn(self.deconv1_1(input)))
        y = F.relu(self.deconv1_2_bn(self.deconv1_2(label)))
        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 = torch.tanh(self.deconv4(x))

        return x
```




生成器的结构如上图所示，其使用四层反卷积(deConv)来输出生成的图片。其接受两个输入：[100,1,1]的随机正态分布噪声，和[10,1,1]的label标志。他们都通过反卷积变为[256,4,4]，而后仍然利用 `torch.cat()` 函数变为一个Tensor，通道数加倍。经过多次反卷积得到[1,32,32]的Tensor后，利用激活函数Tanh将其每个值控制在[-1,1]之间，生成一张灰度图。

## 独热编码

条件生成器与一般的生成器不同的点在于，其对于不同输入的Label，应该在生成过程中使用不同的参数。因此必须要将Label的信息与图片一起放在神经网络中训练。而0至9的十个数字不可能直接与图片相拼接，这是因为尽管数字2相比于数字1距离数字9更远，但也未必就是说数字2相比于数字1与数字9更类似。因此要进行独热编码。在训练过程中对于生成器和判别器使用不同的独热编码方案：

对于生成器，使用一个[10]的Tensor，其中当且仅当该数字为$k$时，该Tensor的第$k$位为1，否则为零。而后使用 `view()` 将其转化为[10,1,1]的Tensor作为输入。

对于判别器，使用一个[10,32,32]的Tensor，当且仅当该数字为$k$时，该Tensor的第$k$个通道全为1，否则全为0.

将上述的Label独热编码后的Tensor与输入利用 `torch.cat()` 函数拼接，得到总的特征图。独热编码的实现如下：

```
# 1hot
onehot = torch.zeros(10, 10)
onehot = onehot.scatter_(1, torch.LongTensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).view(10, 1), 1).view(
10, 10, 1, 1).to(device)

fill = torch.zeros([10, 10, img_size, img_size])
for i in range(10):
    fill[i, i, :, :] = 1
fill = fill.to(device)
```

## 训练过程

根据GAN的论文，损失函数选取交叉熵损失函数。训练过程采取在一次训练中先后训练判别器和生成器各一次。

判别器的损失分为两部分：mnist数据集的图片和generator生成的图片。其需要向如下方向优化：对于mnist数据集的图片，其需要输出1；对于generator生成的图片，其需要输出0。下面的代码完成了这个过程：

```
D_result = D(x_, y_fill_).squeeze()
        # mnist数据集的loss，y_real_为全tensor
        D_real_loss = BCE_loss(D_result, y_real_)

        z_ = torch.randn((mini_batch, 100)).view(-1, 100, 1, 1)
        y_ = (torch.rand(mini_batch, 1) * 10).type(torch.LongTensor).squeeze()
        y_label_ = onehot[y_]
        y_fill_ = fill[y_]
        z_, y_label_, y_fill_ = z_.to(device), y_label_.to(device), y_fill_.to(device)

        G_result = G(z_, y_label_)
        D_result = D(G_result, y_fill_).squeeze()
        # generator生成图片的loss，y_fake_为全0tensor
        D_fake_loss = BCE_loss(D_result, y_fake_)

        D_train_loss = D_real_loss + D_fake_loss

        D_train_loss.backward()
        D_optimizer.step()
```

对于生成器，其优化方向为使得判别器认为自己生成的图像是手写的，下面的代码完成了这个过程：

```
_result = G(z_, y_label_)
D_result = D(G_result, y_fill_).squeeze()

G_train_loss = BCE_loss(D_result, y_real_)

G_train_loss.backward()
G_optimizer.step()
```

### 3．工作总结

#### （1）收获、心得

1. 通过这门课的学习，我第一次比之前更深入的了解到了关于人工智能的一些内容及知识，例如机器学习一些原理，人工神经网络的搭建，自然语言的处理等等，尤其是课后作业和大作业的完成，让我对python这个工具的应用和对遗传算法的使用领域有了更多的了解。让我对人工智能的学习有了更多的兴趣，对生活中遇到的一些关于人工智能的问题也有了更科学的解释。
2. 通过参与本次大作业，我学习了基本的python语法和深度学习框架pytorch，能够编写简单的python程序，会调用pytorch的各种相关库实现神经网络模型。通过网上查阅资料，我了解了生成对抗网络GAN的原理，并且能够简单实现GAN、利用GAN生成Mnist数据集。本次大作业锻炼了代码能力，受益匪浅。

#### （2）遇到问题及解决思路

## 如何完成条件生成

GAN本身并不具有条件生成的能力，因此需要考虑CGAN。如何传递标签信息是第一个困难点。最后选择使用独热编码的方式解决。在核心代码中提到了这一点，这里不多赘述。

### 收敛速度不理想

一开始的训练结果不是很好，原因是当学习率(Learning rate)太高时，模型生成的图片会忽而质量高，忽而质量低。原因是过高的学习率很难让模型的参数稳定在一个范围内。但是太低的学习率会导致收敛速度很慢，并且增加学习次数来缓解这个问题时，又会出现过拟合的情况发生。后来选择使用学习率衰减来解决这个问题：

```
for epoch in range(train_epoch):
    # learning rate decay
    if (epoch + 1) == 11:
        G_optimizer.param_groups[0]['lr'] /= 10
        D_optimizer.param_groups[0]['lr'] /= 10
        print("learning rate change!")

    if (epoch + 1) == 16:
        G_optimizer.param_groups[0]['lr'] /= 10
        D_optimizer.param_groups[0]['lr'] /= 10
        print("learning rate change!")
```

一共训练20代时，前10次循环基于模型充分的机会快速逼近一个可能的最优点，而后在后几次循环中逐渐减小学习率，也就是不期待模型找到更好的收敛点了，而选择在当前区间内逐步优化。这样经过20次迭代，模型未必在全局最优上，但一定已经达到了一个很靠近局部最优的收敛点，使得训练效果上升了。

### 4．课程建议

1. 老师讲课生动形象，与实际结合很紧密，希望之后可以更多的通过解释现有的一些人工智能模型来进行课程的讲解。
2. 注意到大作业内容和课程内容相关性不大，而且最后几周复习任务重，事情较多，因此希望能将大作业布置时间提前，比如在第九周就将大作业布置下去，给予学生充分的时间。