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：cGAN-pyTorch		学号：201900130143
日期：11/25	班级： 智能班	姓名：吴家麒
Email：wjq_777@126.com		

实验目的：

In this assignment, you will also explore methods for visualizing the features of a pretrained model on ImageNet.

• Explore various applications of image gradients, including saliency maps, fooling images, class visualizations

实验软件和硬件环境：

Anaconda3 + Jupyter notebook

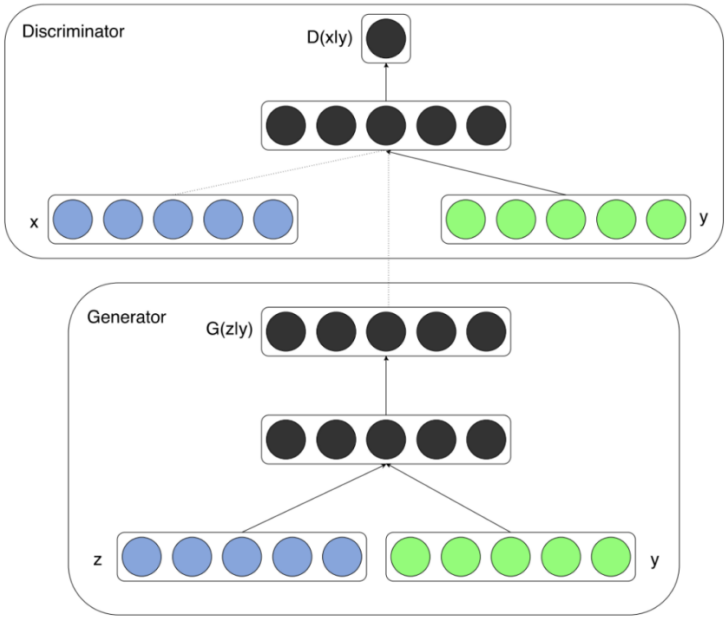
实验原理和方法：

GAN 网络由 Generator 和 Discriminator 组成。Generator 可以看作是一个输出一种分布而不是一个特定值的 Network。输入是 x 一个从已知分布中采样得到的 z，利用不同的 z,针对同一个 x 生成不同的 y ，来达到输出是一个分布的目的。

我们的目标是让生成的数据的分布和真实的数据的分布越接近越好。

这时我们利用 Divergence 来表示生成数据和真实数据之间的差异。

Discriminator 的目标就是给真实图像一个比较大的分数，给生成图像一个比较小的分数,对生成的图像和真实的图像进行分类判别。



cGAN 也就是 ConditionalGenerativeAdversarialNetwork，在基本的 GAN 上对 Generator 和 Discriminator 的输入都添加了 labels，使得我们可以针对类别训练，控制生成图片的类别，而使得结果不那么随机。

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 种不同种类的衣服数据：Dress, Coat, Shirt 等等。训练集规模有 60000 条数据。

实验步骤：（不要求罗列完整源代码）

1、Generator

输入：潜在空间的一批点（向量）和一批 label

输出：一批图片

```
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
        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.label_dim, 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
```

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

    ### START CODE HERE
    gen_input = torch.cat((self.label_embedding(labels), noise), -1) #拼接两个向量
    img = self.model(gen_input)
    img = img.view(img.size(0), *img_shape)
    return img
    ### END CODE HERE

    return img
```

2、Discriminator

输入：一批图片和图片对应的 label

输出：判别的结果

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

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

        ### START CODE HERE
        self.model = nn.Sequential(
            nn.Linear(opt.label_dim + 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
```

```
def forward(self, img, labels):

    ### 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

    return validity
```

网络结构可以看作是图像从（784+100）->512->512->512->1 的过程。

3. 训练过程

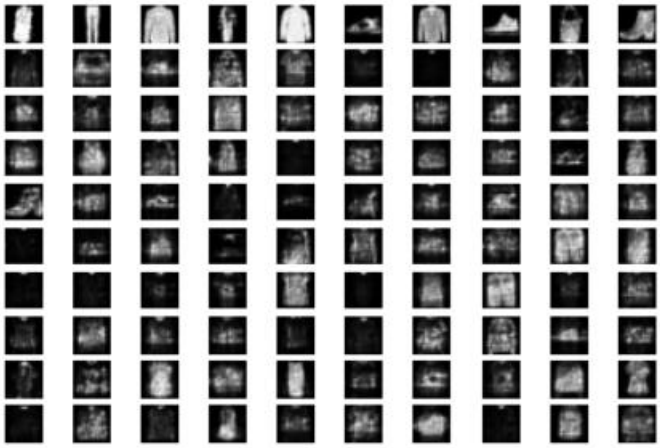
生成器的训练首先输入 z 和 label 生成一批图片，再通过判别器判断生成图片的真假，计算 generator 的 loss，然后计算反向传播更新参数。

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

### START CODE HERE
optimizer_G.zero_grad()

# Sample noise and labels as generator input
#生成一批 noise
z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size, opt.latent_dim))))
#生成一批 label
gen_labels = Variable(LongTensor(np.random.randint(0, opt.n_classes, batch_size)))

# 输入 z 和 gen_labels ，通过生成器，生成一批图片
gen_imgs = generator(z, gen_labels)

# Loss measures generator's ability to fool the discriminator
# 通过判别器，判断生成图像的真假，返回一批图像的判别结果
validity = discriminator(gen_imgs, gen_labels)
# 判别为假的产生loss，这里计算生成器的loss
g_loss = adversarial_loss(validity, valid)
# BP + 更新
g_loss.backward()
optimizer_G.step()
### END CODE HERE
```

判别器的训练：分别计算出真实图片的 loss 和生成图片的 loss，得到总的损失进行反向传播和参数更新。

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

### START CODE HERE
optimizer_D.zero_grad()

# 计算真实图片的 loss
validity_real = discriminator(real_imgs, labels)
d_real_loss = adversarial_loss(validity_real, valid)

# 计算生成图片的 loss
validity_fake = discriminator(gen_imgs.detach(), gen_labels)
d_fake_loss = adversarial_loss(validity_fake, fake)

# Total loss
d_loss = (d_real_loss + d_fake_loss) / 2

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

最后测试，生成图像：



结论分析与体会：

1、从本次实验的结果中可以发现，GAN 训练的超参数很多，一定幅度调参对结果的影响很大，也可以看出 GAN 存在一定的局限性。

2、通过本次实验对 GAN 的原理与训练过程进行了学习和运用，对对抗生成网络的实际训练过程有了更深入的理解。

3、对于神经网络在实际应用中的过程有了更为深入的理解和学习。