1、前言

本文将讲近些年来挺火的一个生成模型$\boxed{\mathbf{GAN}\text{生成对抗网络}}$，其特殊的思路解法实在让人啧啧称奇。

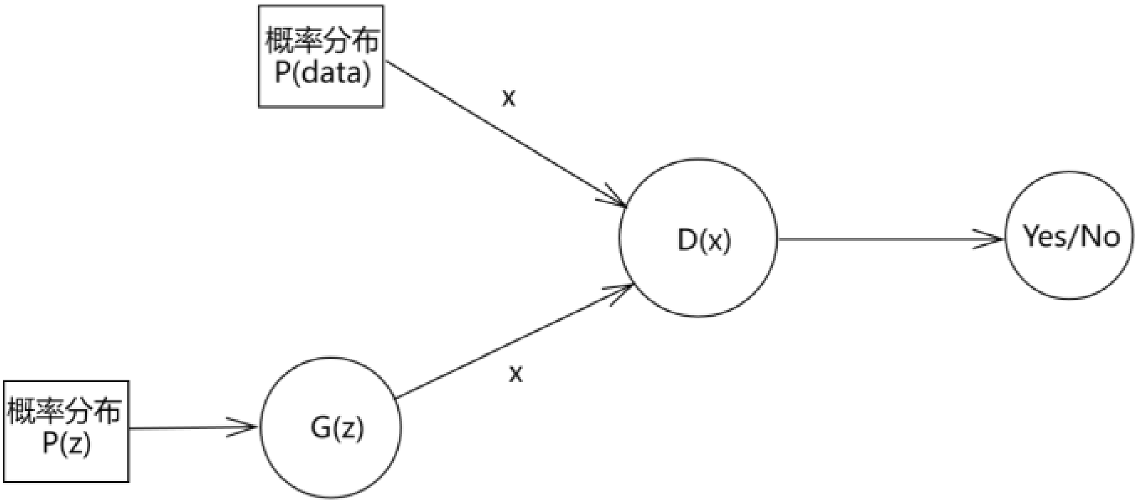
2、原理

2.1、GAN的运行机理

在传统的生成模型中，我们总是对我们的训练数据（或观测变量和隐变量）进行建模，得到概率分布，然后进行数据的生成。可GAN却不是这样，其利用神经网络这个函数逼近器，求解出了模型中概率分布的参数$\boxed{\text{在不知道概率分布是什么的情况下}}$。

其主要思想是，从一个简单的概率分布中采样，得到样本经过神经网络变换，得到一个新的样本，我们就假设这个样本就来自我们需要求解的概率分布中。然后用神经网络去辨别其是来自真实分布，还是我们要求解的概率分布。

先来看模型图



我们的训练数据x是来自真实分布$\boxed{\text{对应图中}\mathbf{P(data)}}$，我们记作$P_{data}$，训练数据都是从$P_{data}$中采样得来（图中上半部分的x）。

而我们从简单的概率分布中抽样$P(z)\boxed{\text{如正态分布}}$，让所得的样本经过一个神经网络$G(z)$，得到一个新的样本x，这个样本就来自我们的需要求解的概率分布，我们记作P_g。

然后将两个x给神经网络$D(x)$判断真伪，让它区分这个x是来自P_{data}还是P_g，其输出样本来自P_{data}的概率。依据所得信息使用梯度下降更新神经网络参数，$G(z)$也是如此。

而$G(z)$被称为生成器$\boxed{(\text{用于生成样本})}$，$D(x)$被称为判别器$\boxed{\text{用于判别样本真伪}}$。

2.2、目标函数

$\boxed{\text{损失函数来自判别器和生成器}}$

$\boxed{\text{对于判别器}}$

当样本来自P_{data}，我们要让所得的概率越大越好；当样本来自p_g，我们要让其概率越小越好，即

$$①\max_D D(x_i)$$

$$②\min_D D(G(z_i))$$

将最小化换成最大化

$$\max_D [1 - D(G(z_i))]$$

所以单个样本判别器的损失函数可以写成

$$\max_D \{D(x_i) + [1 - D(G(z_i))]\}$$

对于所有样本N，我们希望均值最大

$$\max_D \left\{ \frac{1}{N}\sum_{i=1}^{N} D(x_i) + \frac{1}{N}\sum_{i=1}^{N}[1-D(G(z_i))] \right\}$$

写成期望形式（并取log最大$\boxed{\text{不改变最大值}}$），得到判别器的损失函数（$x \sim p_{data}$表示样本来自真实分布）

$$\boxed{\max_D \left\{ \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \right\}}$$

$\boxed{\text{对于生成器}}$

它希望生成的样本让判别器判别为真的概率越大越好，所以直接设计成（将最大写成最小）

$$\boxed{\min_G \mathbb{E}_{x\sim P_g}[\log(1-D(x))]}$$

所以最终的目标函数可以写成

$$\min_G \max_D \left\{ \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \right\}$$

3、最优解求解

得到了目标函数，我们很显然还需要证明其存在最优解。并且最优解的P_g是否和P_{data}无限接近

$\boxed{\text{先求里层（关于}D\text{求最大）}}$

$$\begin{aligned}
&\mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \\
&= \int_x \log D(x) P_{data}(x)dx + \int_x \log(1-D(x))P_g(x)dx \\
&= \int_x [\log D(x)P_{data}(x) + \log(1-D(x))P_g(x)]dx
\end{aligned}$$

要求积分最大，就是要求里面的每一个最大

$$\max_D [\log D(x)P_{data}(x) + \log(1-D(x))P_g(x)]$$

求导

$$\begin{aligned}
&\frac{\partial}{\partial D} logD(x)P_{data}(x) + \log(1-D(x))P_g(x) \\
&= \frac{1}{D(x)}P_{data}(x) - \frac{1}{1-D(x)}P_g(x)
\end{aligned}$$

整理得

$$\boxed{D(x) = \frac{P_{data}(x)}{P_g(x)+P_{data}(x)}}$$

$\boxed{\text{将其代入目标函数，并且关于外层}G\text{求最小}}$

$$\begin{aligned}
&\min_G \int_x \left[\log\frac{P_{data}(x)}{P_g(x)+P_{data}(x)}P_{data}(x) + \log\left(1-\frac{P_{data}(x)}{P_g(x)+P_{data}(x)}\right)P_g(x)\right]dx \\
&= \min_G \left[\int_x \log\left(\frac{P_{data}(x)}{\frac{P_g(x)+P_{data}(x)}{2}} * \frac{1}{2}\right)P_{data}(x)dx + \int_x \log\left(\frac{P_g(x)}{\frac{P_g(x)+P_{data}(x)}{2}} * \frac{1}{2}\right)P_g(x)dx\right] \\
&= \min_G \left[\int_x \log\left(\frac{P_{data}(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_{data}(x)dx + \int_x \log\left(\frac{P_g(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_g(x)dx + \int \log\frac{1}{2}P_{data}(x)dx + \int \log\frac{1}{2}P_g(x)dx\right] \\
&= \min_G \left[\int_x \log\left(\frac{P_{data}(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_{data}(x)dx + \int_x \log\left(\frac{P_g(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_g(x)dx + \log\frac{1}{2}\int P_{data}(x)dx + \log\frac{1}{2}\int P_g(x)dx\right] \\
&= \min_G \left[\int_x \log\left(\frac{P_{data}(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_{data}(x)dx + \int_x \log\left(\frac{P_g(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_g(x)dx + \log\frac{1}{2} + \log\frac{1}{2}\right] \\
&= \min_G \left[\int_x \log\left(\frac{P_{data}(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_{data}(x)dx + \int_x \log\left(\frac{P_g(x)}{\frac{P_g(x)+P_{data}(x)}{2}}\right)P_g(x)dx + \log\frac{1}{4}\right] \\
&= \min_G KL\left(P_{data}(x)||\frac{P_{data}(x)+P_g(x)}{2}\right) + KL\left(P_g(x)||\frac{P_{data}(x)+P_g(x)}{2}\right) - \log 4
\end{aligned}$$

$KL(p||q)=\int_x p\log\frac{p}{q}dx$，KL散度是衡量概率分布$p$和$q$的相似程度，其大于等于0，当其相似程度一样时，则散度为0，也就是我们要求的最小值。

小补充

$$\boxed{\mathbf{2JS}\left(\mathbf{P}_{\mathbf{data}}(\mathbf{x})||\mathbf{P}_{\mathbf{g}}(\mathbf{x})\right)=\mathbf{KL}\left(\mathbf{P}_{\mathbf{data}}(\mathbf{x})||\frac{\mathbf{P}_{\mathbf{data}}(\mathbf{x})+\mathbf{P}_{\mathbf{g}}(\mathbf{x})}{\mathbf{2}}\right)+\mathbf{KL}\left(\mathbf{P}_{\mathbf{g}}(\mathbf{x})||\frac{\mathbf{P}_{\mathbf{data}}(\mathbf{x})+\mathbf{P}_{\mathbf{g}}(\mathbf{x})}{\mathbf{2}}\right)}$$

$JS(p||q)$被称为JS散度，其仍然是大于等于0的。所以是一样的。

所以

$$P_{data}(x)=\frac{P_g(x)+P_{data}}{2}\rightarrow P_{data}=P_g(x)$$

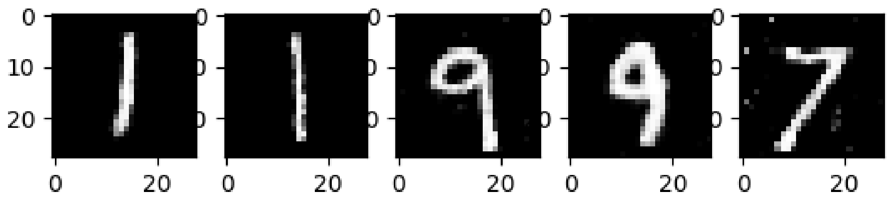
由此可见，目标函数最优值能够让$\mathbb{P}_{\mathrm{g}}$逼近$\mathbb{P}_{\mathrm{data}}$，并且当其相等时，有

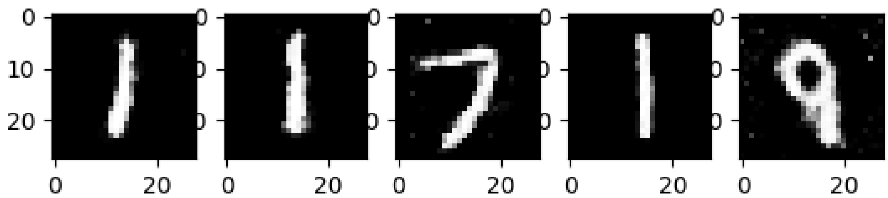
$$\boxed{D(x)=\frac{P_{data}(x)}{P_g(x)+P_{data}(x)}}=\frac{1}{2}$$

也就是判别器再也无法判断出样本是来自P_{data}还是P_g

4、代码实现

结果如下





效果一般，在其他变种优化有很多比这个好的，感兴趣的读者自行查阅。

```python
import torch
from torchvision.datasets import MNIST
from torchvision import transforms
from torch.utils.data import DataLoader
from tqdm  import tqdm
import matplotlib.pyplot as plt

class Generate_Model(torch.nn.Module):
    '''
    生成器
    '''
```

```python
    def __init__(self):
        super().__init__()
        self.fc=torch.nn.Sequential(
            torch.nn.Linear(in_features=128,out_features=256),
            torch.nn.Tanh(),
            torch.nn.Linear(in_features=256,out_features=512),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=512,out_features=784),
            torch.nn.Tanh()
        )
    def forward(self,x):
        x=self.fc(x)
        return x

class Distinguish_Model(torch.nn.Module):
    '''
    判别器
    '''
    def __init__(self):
        super().__init__()
        self.fc=torch.nn.Sequential(
            torch.nn.Linear(in_features=784,out_features=512),
            torch.nn.Tanh(),
            torch.nn.Linear(in_features=512,out_features=256),
            torch.nn.Tanh(),
            torch.nn.Linear(in_features=256,out_features=128),
            torch.nn.Tanh(),
            torch.nn.Linear(in_features=128,out_features=1),
            torch.nn.Sigmoid()
        )
    def forward(self,x):
        x=self.fc(x)
        return x
def train():
    device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu") #判断是否存在可用GPU
    transformer = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.5, std=0.5)
    ]) #图片标准化
    train_data = MNIST("./data", transform=transformer,download=True) #载入图片
    dataloader = DataLoader(train_data, batch_size=64,num_workers=4, shuffle=True) #将图片放入数据加载器

    D = Distinguish_Model().to(device) #实例化判别器
    G = Generate_Model().to(device) #实例化生成器

    D_optim = torch.optim.Adam(D.parameters(), lr=1e-4) #为判别器设置优化器
    G_optim = torch.optim.Adam(G.parameters(), lr=1e-4) #为生成器设置优化器

    loss_fn = torch.nn.BCELoss() #损失函数

    epochs = 100 #迭代100次
    for epoch in range(epochs):
        dis_loss_all=0 #记录判别器损失损失
        gen_loss_all=0 #记录生成器损失
        loader_len=len(dataloader) #数据加载器长度
        for step,data in tqdm(enumerate(dataloader), desc="第{}轮".format(epoch),total=loader_len):
            # 先计算判别器损失
            sample,label=data #获取样本，舍弃标签
            sample = sample.reshape(-1, 784).to(device) #重塑图片
            sample_shape = sample.shape[0] #获取批次数量
            #从正态分布中抽样
            sample_z = torch.normal(0, 1, size=(sample_shape, 128),device=device)

            Dis_true = D(sample) #判别器判别真样本

            true_loss = loss_fn(Dis_true, torch.ones_like(Dis_true)) #计算损失
```

```
            fake_sample = G(sample_z) #生成器通过正态分布抽样生成数据
            Dis_fake = D(fake_sample.detach()) #判别器判别伪样本
            fake_loss = loss_fn(Dis_fake, torch.zeros_like(Dis_fake)) #计算损失

            Dis_loss = true_loss + fake_loss #真假加起来
            D_optim.zero_grad()
            Dis_loss.backward() #反向传播
            D_optim.step()

            # 生成器损失
            Dis_G = D(fake_sample) #判别器判别
            G_loss = loss_fn(Dis_G, torch.ones_like(Dis_G)) #计算损失
            G_optim.zero_grad()
            G_loss.backward() #反向传播
            G_optim.step()
            with torch.no_grad():
                dis_loss_all+=Dis_loss #判别器累加损失
                gen_loss_all+=G_loss #生成器累加损失
        with torch.no_grad():
            dis_loss_all=dis_loss_all/loader_len
            gen_loss_all=gen_loss_all/loader_len
            print("判别器损失为: {}".format(dis_loss_all))
            print("生成器损失为: {}".format(gen_loss_all))
        torch.save(G, "./G.pth") #保存模型
        torch.save(D, "./D.pth") #保存模型
if __name__ == '__main__':
    train() #训练模型
    model_G=torch.load("./G.pth",map_location=torch.device("cpu")) #载入模型
    fake_z=torch.normal(0,1,size=(10,128))  #抽样数据
    result=model_G(fake_z).reshape(-1,28,28)  #生成数据
    result=result.detach().numpy()

    #绘制
    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(result[i])
        plt.gray()
    plt.show()
```

5、结束

以上，就是GAN生成对抗网络的全部内容了，如有问题，还望指出。阿里嘎多