生成对抗网络模型(GAN)

作者：刘帅（2020212267）、杨谨帆（2020212275）

1.问题背景

近些年来，在 CVPR 会议发表的论文中，有关 GAN 的论文占到了近 50%，GAN 从 2014 年首次提出后，在计算机视觉领域（CV）有着很高热度和大规模的发展。针对 AI 图片生成、语音生成、风格迁移等方面有着较为突出和显著的应用成果。本文将针对原生的 GAN 提出的算法及思想，并以 MNIST 手写数字数据集作为样本，进行网络模型的训练，最终实现 AI 自动生成数字的可视化结果。

2.问题描述

2.1 待解决问题的解释

针对 MNIST 数据集，我们将采取博弈论中零和博弈的思想，通过对抗的思想让 AI 自动生成手写数据。最终达到“以假乱真“的实际效果。

2.2 解决问题的形式化描述

采用生成对抗网络模型，其中生成式网络 G 生成待辨识图像，判别式网络 D 通过判断该图像，给出该图像所属类型（真实 or 虚假）。G 试图产生更接近真实值的数据，D 试图更加精确地判断出伪造图像和真实图像之间的差异。两个网络在对抗中使得 G 网络生成的数据越来越贴合真实数据的分布，进而通过 G 生成器实现以假乱真的效果。

3.系统

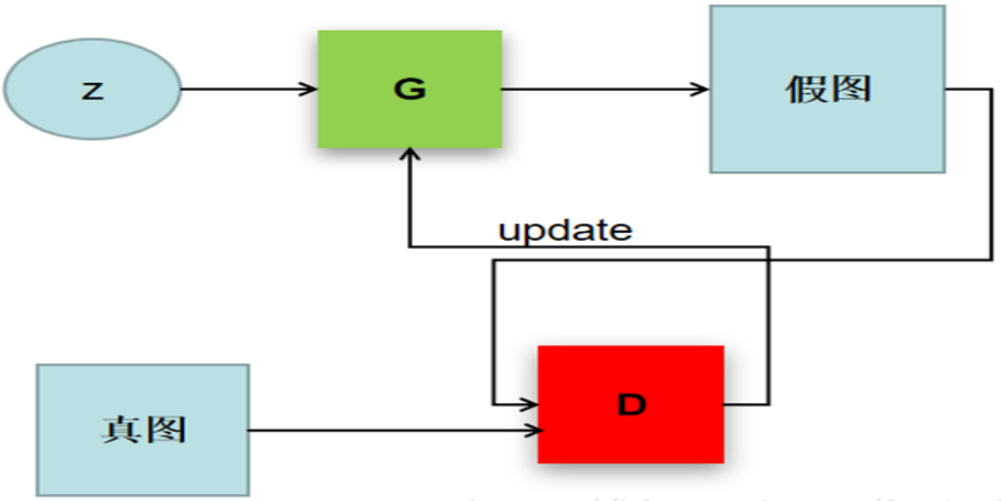
3.1 系统架构

网络模型部分，系统架构共分为生成网络 G（Generator）和判别网络 D（Discriminator）两个部分。其中，G 和 D 的主要功能为：

① G 是一个生成式的网络，它接收一个随机的噪声 z（随机数），通过这个噪声生成图像

② D 是一个判别网络，判别一张图片是不是“真实的”。它的输入参数是 x，x 代表一张图片，输出 D（x）代表 x 为真实图片的概率，如果为 1，就代表 100%是真实的图片，而输出为 0，就代表不可能是真实的图片。

该过程需要在多次迭代中实现参数更新，以流程图表示学习过程：



3.2 算法推导

3.2.1 价值函数的推导

为了学习生成器（G）关于数据 x 的分布 Pg，定义输入噪声的变量 Pz（z），符合高斯分布。用 G（z；θg）来代表数据空间的映射。同时定义判别器网络结构 D（x；θd）来输出单独标量。D（x）代表 x 来源于真实数据分布的概率。

因此，GAN 的实质则是训练 D 和 G 关于价值函数 V（G,D）的 minmax 博弈问题。

定义损失函数：

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

我们假设能够得到较为理想的判别器，则对于符合$x\sim p_{data}(x)$分布的随机变量 x，$\log D(x)$=0 取到最大值，同理，对于符合$z\sim p_z(z)$分布的随机变量 z，判别器相对生成器更为聪明，因此能够分辨出样本的真假，有$D\big(G(z)\big)=0$，同理$\big[\log\big(1-D\big(G(z)\big)\big]$取到最大值，因此判别器理想化的训练过程就是$V(D,G)$对 D 取得最大值的过程。

同理，假设我们要得到较为理想的生成器，则代表着其生成的负样本不能被判别器正确识别，因此有$D\big(G(z)\big)=1$，对于$\big[\log\big(1-D\big(G(z)\big)\big]$项则取到最小值。在 D,G 参数不断更新的过程中，判别器和生成器的性能都逐渐变好，从而实现最终以假乱真的效果。

3.2.2 损失函数收敛性讨论

在已知参数训练的过程后，如何判断训练的终止则成为下一个任务，我们将损失函数按期望的形势展开，可得如下积分形式：

$$V(G,D)=\int_x p_{data}(x)\log(D(x))dx+\int_z p_z(z)\log(1-D(g(z))dz$$

$$=\int_x p_{data}(x)\log(D(x))+p_g(x)\log(1-D(x))dx$$

在此我们针对二分类问题，假设随机变量 x 满足两个分布 p(x)，q(x)，引入交叉熵的定义：$CEH(p,q)=-\sum p(x)logq(x)=-[plogq+(1-p)\log(1-q)]$

通过观察，可见最大化期望的过程可以通过求交叉熵的梯度来解决，因此，针对形如$alog(y)+(1-a)\log(1-y)$的函数，我们可以在$\frac{a}{a+b}$处取得最大值。定义最大值 $D_G^*(x)=\dfrac{p_{data}(x)}{p_{data}(x)+p_g(x)}$

将其反带入进原始公式，有：

$$C(G)=\max_{D}V(G,D)$$
$$=E_{x\sim p_{data}}[\log D_G^*(x)]+E_{z\sim p_z}[\log(1-D_G^*(G(z)))]$$
$$=E_{x\sim p_{data}}[\log D_G^*(x)]+E_{x\sim p_g}[\log(1-D_G^*(x))]$$
$$=E_{x\sim p_{data}}[\log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}]+E_{x\sim p_g}[\log\frac{p_g(x)}{p_{data}(x)+p_g(x)}]$$

此时，当判别器为真或假的概率均为$\frac{1}{2}$时认为训练过程达到收敛。

3.3 各部分介绍

在深度学习训练过程中，有四个重要的流程：数据获取，模型建立，训练（收敛）过程，调参（往往也是最为枯燥的），下面将从四个方面进行介绍：

3.3.1 数据获取

```
os.makedirs("../../data/mnist", exist_ok=True)
dataloader = torch.utils.data.DataLoader(
    datasets.MNIST(
        "../../data/mnist",
        train=True,
        download=True,
        transform=transforms.Compose(
            [transforms.Resize(opt.img_size), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]
        ),
    ),
    batch_size=opt.batch_size,
    shuffle=True,
)
```

建立mnist数据集文件夹,利用pytorch中torch.dataset引入MNIST数据集，并用DataLoader进行数据预处理，包括调整数据格式，数据类型和归一化

3.3.2 模型建立

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

        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, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            *block(1024, 2048),
            *block(2048, 4096),
            nn.Linear(4096, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.size(0), *img_shape)
        return img
```

定义生成器模块，其中每个块 block 采用线性连接，且对于每次线性映射，都经过批量归一化，从而增加模型的鲁棒性，并加快收敛速率。并利用 LeakyReLU 激活函数进行处理，从而避免梯度消失等问题。最终建立 6 层感知机模型。

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

        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape)), 4096),
            nn.ReLU(),
            nn.Linear(4096, 2048),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        img_flat = img.view(img.size(0), -1)
        validity = self.model(img_flat)

        return validity#返回一个0，1标签
```

同理我们建立判别器模型，为 5 层感知机模型，并在每个线性层加入 ReLU 作为激活函数，并采用 dropout 方法随机丢弃掉 50%的样本，最终返回一个 bool 类型的值，代表判别数据的真假。

3.3.3 模型训练

```python
#使用Adam作为优化器
two_elements_crossentropy = torch.nn.BCELoss()#二分类交叉熵损失函数
optimizer_G = torch.optim.Adam(generator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))

```

我们在此处依赖 torch.nn 的 API，采用 BCELoss（即上文所提到的二分类交叉熵）作为损失函数，优化器选择 Adam 进行训练。

3.4 算法伪代码

for epoch do

for k steps do:

- 选取 minibatch(z1,z2……,zm)作为高斯噪声的样本
- 选取 minibatch(x1,x2……,xm)作为真实数据样本

更新$V(D,G)=E_{x\sim p_{data}(x)}[\log D(x)]+E_{z\sim p_z(z)}[\log(1-D(G(z))]$中的参数。

- 采用梯度下降的方法，求$\frac{\partial V(D,G)}{\partial D}$

End for

- 针对一个 minibatch 的高斯噪声的样本 (z1,z2……,zm)
- 利用梯度上升方法求解 G 的最大值$-\frac{\partial V(\boldsymbol{D},\boldsymbol{G})}{\partial \boldsymbol{G}}$

End for

4. 实验

4.1 实验环境

```
import argparse
import os
import numpy as np

import torchvision.transforms as transforms
from torchvision.utils import save_image

from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import torch.nn as nn
import torch
```

针对数据获取，下载了 torchvision 库中的 MNIST 作为数据集。模型训练过程，对于线性层、BN 归一化、dropout 等过程均采用了 torch。为加快训练过程，使用 GPU 进行网络训练，使用了 cuda。

4.2 数据

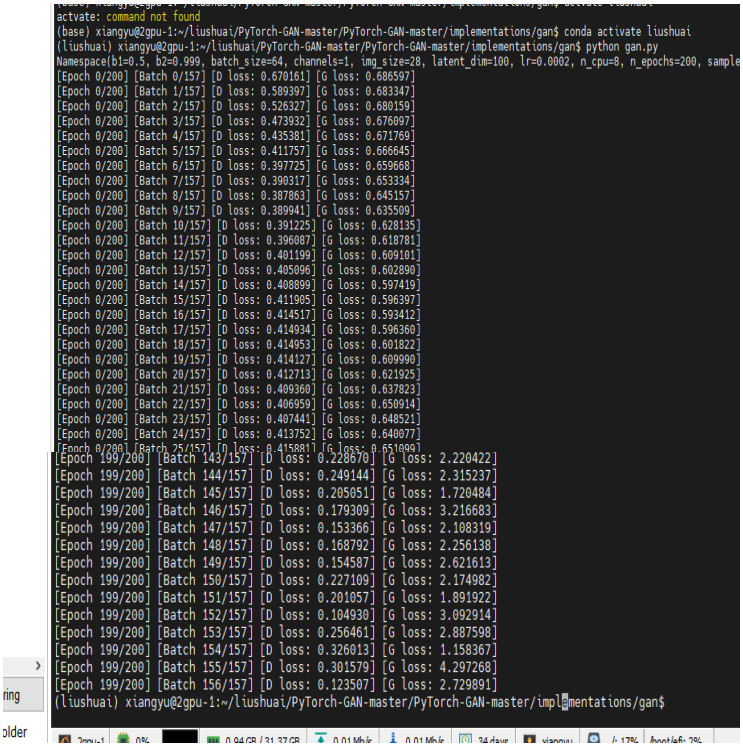
4.2.1 超参数选择

```
#定义超参数
parser = argparse.ArgumentParser()
parser.add_argument("--nums_of_epochs", type=int, default=500, help="训练轮数")
parser.add_argument("--batch_size", type=int, default=64, help="batch大小")
parser.add_argument("--lr", type=float, default=0.002, help="优化器lr")
parser.add_argument("--latent_dim", type=int, default=100, help="隐藏层维度")
parser.add_argument("--img_size", type=int, default=28, help="图片大小")
parser.add_argument("--channels", type=int, default=1, help="通道数")
parser.add_argument("--sample_interval", type=int, default=100, help="采样间隔")
parser.add_argument("--b1", type=float, default=0.5, help="第一次估计的指数衰减率")
parser.add_argument("--b2", type=float, default=0.999, help="第二次估计的指数衰减率")
opt = parser.parse_args()
print(opt)
img_shape = (opt.channels, opt.img_size, opt.img_size)
```

由于时间有限且防止过拟合，此处选择训练进行 500 轮，间隔 100 个样本取一个值。

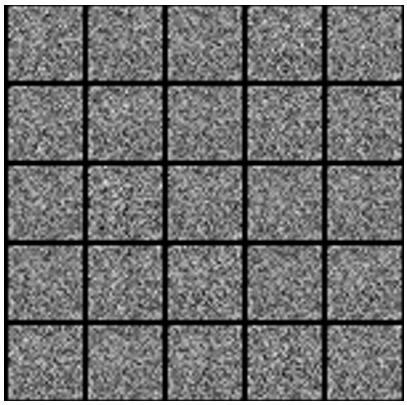
4.2.2 训练过程损失函数变化

可见，该过程中 D loss 在逐渐下降，证明其与真实标签的相似度越来越高，说明生成器成功的将噪声生成为手写数字。

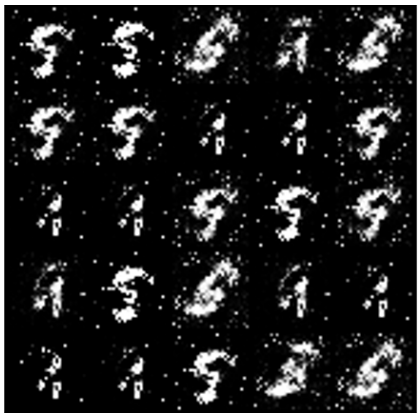


4.2.3 实验结果

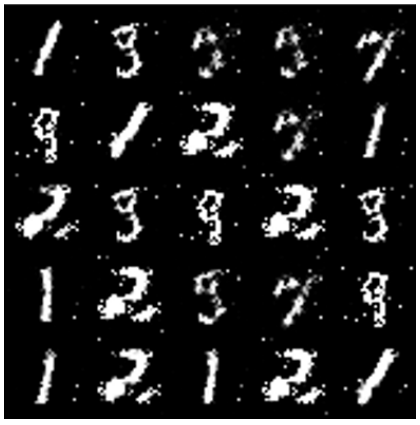
实验开始前生成 Z 的结果:



经 10 个 epoch 后训练的结果：



经 100 个 epoch 后训练的结果:



经 300 个 epoch 后训练的结果:

经 400 个 epoch 后训练的结果:

经 500 个 epoch 后训练的结果:

5. 总结

在智能科学导论课上，我们接触了多层感知机算法，其在模型训练上有着有效而广泛的应用。在本次实验之前，也曾了解过 GAN 算法，并以此为模型基础跑过 cyclegan、infogan 以及近期较为热门的CUT(contrastive unpaired translation)的风格迁移模型，本次实验能够回头再次回顾最基本最为奠基性的 GAN 算法，从概率和信息论的角度再次看待 GAN 的原理，将是我本次实验的最大收获。同时，本次实验中仍有所不足，比如超参数的选择还没有达到最优，对于第一轮 D（x）相对 G（x）较大因此产生梯度消失问题并没有有效求解，以及对于 D_loss 的合理性解释，都将是我在后续学习中需要重温和不断学习中寻求答案的。在后续的学习中，我也将利用通过 GAN 所学习到的基础继续解决 CV 领域有关风格迁移的问题，并能够在数学推导方面有所收获。