# 实验：利用GAN生成手写数字图片

## 一、 实验内容

使用 MNIST 手写数字来做数据集，通过生成对抗网络生成一些“以假乱真” 的手写字体

## 二、 实验目标

（1）理解 GAN 网络的原理。

（2）自编程实现一个 GAN 网络，并对数据进行生成。

（3）通过实验理解 GAN 网络对抗生成的过程。

## 三、 实验内容

#### 3.1数据集准备

继续使用MNIST数据集代码如下

|  |
| --- |
| # coding=utf-8  import torch.autograd  import torch.nn as nn  from torch.autograd import Variable  from torchvision import transforms  from torchvision import datasets  from torchvision.utils import save\_image  import os  import matplotlib.image as mpimg  from PIL import Image  import matplotlib.pyplot as plt  batch\_size = 128  num\_epoch = 200  z\_dimension = 50  # 图像预处理  img\_transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1,), (0.5,))  ])  mnist = datasets.MNIST(  root='./mnist/', train=True, transform=img\_transform, download=True  )  dataloader = torch.utils.data.DataLoader(  dataset=mnist, batch\_size=batch\_size, shuffle=True  ) |

#### 3.2判别器模型定义

定义判别器 多层全连接网络

sigmoid激活函数得到一个0到1之间的概率进行二分类。

|  |
| --- |
| class discriminator(nn.Module):  def \_\_init\_\_(self):  super(discriminator, self).\_\_init\_\_()  self.dis = nn.Sequential(  nn.Linear(784, 512), # 输入特征数为784，输出为512  nn.BatchNorm1d(512),  nn.LeakyReLU(0.2), # 进行非线性映射  nn.Linear(512, 256), # 进行一个线性映射  nn.BatchNorm1d(256),  nn.LeakyReLU(0.2),  nn.Linear(256, 1),  nn.Sigmoid()  )  def forward(self, x):  x = self.dis(x)  return x |

#### 3.3生成器模型定义

定义生成器 多层全连接网络

输入为 50维的0-1高斯分布

tanh激活函数，得到-1到1

|  |
| --- |
| class generator(nn.Module):  def \_\_init\_\_(self):  super(generator, self).\_\_init\_\_()  self.gen = nn.Sequential(  nn.Linear(50, 128),  nn.LeakyReLU(0.2),  nn.Linear(128, 256),  nn.BatchNorm1d(256),  nn.LeakyReLU(0.2),  nn.Linear(256, 512),  nn.BatchNorm1d(512),  nn.LeakyReLU(0.2),  nn.Linear(512, 1024),  nn.BatchNorm1d(1024),  nn.LeakyReLU(0.2),  nn.Linear(1024, 784),  nn.Tanh()  )  def forward(self, x):  x = self.gen(x)  return x |

#### 3.4工具函数定义

定义图片展示函数

图片转换函数

|  |
| --- |
| # 创建文件夹  if not os.path.exists('./img2'):  os.mkdir('./img2')  # 转成图片  def to\_img(x):  out=x  out = out.view(-1, 1, 28, 28)  return out  # 图片展示  def show(path,name):  print(path)  img\_array=plt.imread(path)  plt.imshow(img\_array)  plt.title(name)  plt.axis('off')  plt.show() |

#### 3.5损失函数与优化器

使用二分类交叉熵函数

使用adam优化器

|  |
| --- |
| criterion = nn.BCELoss()  criterion.to(device)  d\_optimizer = torch.optim.Adam(D.parameters(), lr=0.0003)  g\_optimizer = torch.optim.Adam(G.parameters(), lr=0.0003) |

#### 3.6训练逻辑

分别对判别器 和生成器进行响应的训练

定期保存生成器生成的图片进行展示

|  |
| --- |
| # 训练逻辑  for epoch in range(num\_epoch):  tt=True  for i, (img, \_) in enumerate(dataloader):  num\_img = img.size(0)    img = img.view(num\_img, -1)  real\_img = Variable(img).to(device)  real\_label = Variable(torch.ones(num\_img)).to(device)  fake\_label = Variable(torch.zeros(num\_img)).to(device)    ##### 训练判别器  real\_out = D(real\_img)  d\_loss\_real = criterion(real\_out.squeeze(), real\_label)  real\_scores = real\_out  # 生成噪声  z = Variable(torch.randn(num\_img, z\_dimension)).to(device)  # 生成器，生成图片  fake\_img = G(z)  # 判别器判别  fake\_out = D(fake\_img)  d\_loss\_fake = criterion(fake\_out.squeeze(), fake\_label)  fake\_scores = fake\_out  # 损失函数和优化  d\_loss = d\_loss\_real + d\_loss\_fake  d\_optimizer.zero\_grad()  d\_loss.backward()  d\_optimizer.step()  ##### 训练生成器  z = Variable(torch.randn(num\_img, z\_dimension)).to(device) # 得到随机噪声  fake\_img = G(z) # 随机噪声输入到生成器中，得到一副假的图片  #  output = D(fake\_img)  g\_loss = criterion(output.squeeze(), real\_label)  # bp and optimize  g\_optimizer.zero\_grad()  g\_loss.backward()  g\_optimizer.step()  if (i + 1) % 300 == 0:  print('Epoch[{}/{}],d\_loss:{:.6f},g\_loss:{:.6f} '  'D real: {:.6f},D fake: {:.6f}'.format(  epoch, num\_epoch, d\_loss.data.item(), g\_loss.data.item(),  real\_scores.data.mean(), fake\_scores.data.mean() # 打印的是真实图片的损失均值  ))  if epoch == 0 and i==len(dataloader)-1:  real\_images = to\_img(real\_img.to(device).data)  real\_path ='./img2/real\_images.png'  save\_image(real\_images, real\_path)  show(real\_path,"real image")    if (epoch%50==49 and tt==True) or (epoch==0 and (i == len(dataloader) - 1)):  fake\_images = to\_img(fake\_img.to(device).data)  fake\_path = './img2/fake\_images-{}.png'.format(epoch + 1);  save\_image(fake\_images, fake\_path)  show(fake\_path,"fake image")  tt=False |

#### 3.1.6运行效果

第一次打印的真是图片效果与生成的fake图片效果

可以看到，输出的图片还是完全的噪声

QR 代码

中度可信度描述已自动生成

经过50个epoch后fake的效果

可以大致看到数字轮廓

文本

描述已自动生成

经过100个epoch后fake的效果

文本

描述已自动生成

经过150个epoch后fake的效果

效果趋于明显

文本

描述已自动生成

经过200次epoch后fake的效果

文本

描述已自动生成