# 《人工智能》实验五报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 俞家宝 | 学号 | 2021117338 |
| 实验地点 | 实验室321 | 实验日期 | 2023/11/13 |

**一、实验内容**

GANs have rapidly emerged as the state-of-the-art technique in realistic image generation. Its applications span realistic image editing that is omnipresent in popular app filters, enabling tumor classification under low data schemes in medicine, and visualizing realistic scenarios of climate change destruction. Building a simple Generative Adversarial Network (GAN) using pytorch. Visualizing the Training losses.（搭建GAN网络，生成MNIST手写体。Minist 手写字体数据库在课程群里下载）

1. **实验原理**

**GAN对抗网络是一种深度学习模型，包含两部分：**

**生成模型（Generative Model，G）和判别模型（Discriminative Model，D），在二者的互相博弈学习中产生预期的输出。**

**在实现中使用神经网络的形式作为 G 和 D ：**

**神经网络G，向其输入一个随机噪声（一维向量），G就会有一个对应的产出，依据我们的需求，分为不同的类型**

**神经网络D，用来评判G产生的图片好坏，标准就是真正的图片，而将G产生的图片叫做虚假的图片**

1. **实验过程以及结果分析**

import time  
import matplotlib.pyplot as plt  
import torch  
import torch.nn as nn  
import torchvision  
from torchvision import transforms  
  
device = torch.device("cpu")  
  
batch\_size = 32  
  
# Compose定义了一系列transform，此操作相当于将多个transform一并执行  
transform = transforms.Compose([  
 transforms.ToTensor(),  
 # mnist是灰度图，此处只将一个通道标准化  
 transforms.Normalize(mean=0.5,  
 std=0.5)  
])  
  
# 设定数据集 第一次使用时`download=True`进行MNIST的数据集下载,若根目录有则设置为False  
mnist\_data = torchvision.datasets.MNIST("./mnist\_data", train=True, download=False, transform=transform)  
  
# 加载数据集，按照上述要求，shuffle本意为洗牌，这里指打乱顺序，很形象  
dataloader = torch.utils.data.DataLoader(dataset=mnist\_data,  
 batch\_size=batch\_size,  
 shuffle=True)  
# MNIST的数据集是28\*28的  
image\_size = 784  
hidden\_size = 256  
  
# Discriminator 判别器  
D = nn.Sequential(  
 nn.Linear(image\_size, hidden\_size),  
 nn.LeakyReLU(0.2),  
 nn.Linear(hidden\_size, hidden\_size),  
 nn.LeakyReLU(0.2),  
 nn.Linear(hidden\_size, 1),  
 nn.Sigmoid() # sigmoid结果为（0，1）  
)  
  
# Generator 生成器  
latent\_size = 64 # latent\_size，相当于初始噪声的维数  
G = nn.Sequential(  
 nn.Linear(latent\_size, hidden\_size),  
 nn.ReLU(),  
 nn.Linear(hidden\_size, hidden\_size),  
 nn.ReLU(),  
 nn.Linear(hidden\_size, image\_size),  
 nn.Tanh() # 转换至（-1，1）  
)  
# 定义损失函数、优化器、学习率  
loss\_fn = nn.BCELoss()  
d\_optimizer = torch.optim.Adam(D.parameters(), lr=0.0002)  
g\_optimizer = torch.optim.Adam(G.parameters(), lr=0.0002)  
  
  
# 开始训练  
# 先定义一个梯度清零的函数，方便后续使用  
def reset\_grad():  
 d\_optimizer.zero\_grad()  
 g\_optimizer.zero\_grad()  
  
  
# 迭代次数与计时  
total\_step = len(dataloader)  
num\_epochs = 300  
start = time.perf\_counter() # 开始时间  
  
# 开始训练  
for epoch in range(num\_epochs):  
 for i, (images, \_) in enumerate(dataloader): # 当前step  
 batch\_size = images.size(0) # 变成一维向量  
 images = images.reshape(batch\_size, image\_size).to(device)  
  
 # 定义真假label，用作评分  
 real\_labels = torch.ones(batch\_size, 1).to(device)  
 fake\_labels = torch.zeros(batch\_size, 1).to(device)  
  
 # 对D进行训练，D的损失函数包含两部分  
 # 第一部分，D对真图的判断能力  
 outputs = D(images) # 将真图送入D，输出（0，1），应该是越接近1越好  
 d\_loss\_real = loss\_fn(outputs, real\_labels)  
 real\_score = outputs # 真图的分数，越大越好  
  
 # 第二部分，D对假图的判断能力  
 z = torch.randn(batch\_size, latent\_size).to(device) # 开始生成一组fake images即32\*784的噪声经过G的假图  
 fake\_images = G(z)  
 outputs = D(fake\_images.detach()) # 将假图片给D，detach表示不作用于求grad  
 d\_loss\_fake = loss\_fn(outputs, fake\_labels)  
 fake\_score = outputs # 假图的分数，越小越好  
  
 # 开始优化discriminator  
 d\_loss = d\_loss\_real + d\_loss\_fake # 总的损失就是以上两部分相加，越小越好  
 reset\_grad()  
 d\_loss.backward()  
 d\_optimizer.step()  
  
 # 对G进行训练，G的损失函数包含一部分  
 # 可以用前面的z，也可以新生成，因为模型没有改变，事实上是一样的  
 z = torch.randn(batch\_size, latent\_size).to(device)  
 fake\_images = G(z)  
 outputs = D(fake\_images)  
 g\_loss = loss\_fn(outputs, real\_labels) # G想骗过D，故让其越接近1越好  
  
 # 开始优化generator  
 reset\_grad()  
 g\_loss.backward()  
 g\_optimizer.step()  
  
 # 优化完成，下面进行一些反馈，展示学习进度  
 if i % 500 == 0:  
 print("Epoch [{}/{}], Step [{}/{}], d\_loss: {:.4f}, g\_loss: {:.4f}, D(x): {:.2f}, D(G(z)): {:.2f}"  
 .format(epoch, num\_epochs, i, total\_step, d\_loss.item(), g\_loss.item(), real\_score.mean().item(),  
 fake\_score.mean().item()))  
  
# 训练结束，跳出循环，检验成果  
end = time.perf\_counter() # 结束时间  
total = end - start  
minutes = total // 60  
seconds = total - minutes \* 60  
print("利用GPU总用时：{:.2f}分钟{:.2f}秒".format(minutes, seconds))  
  
# 保存判别器模型  
torch.save(D.state\_dict(), 'discriminator.pth')  
# 保存生成器模型  
torch.save(G.state\_dict(), 'generator.pth')  
  
# 向G输入一个噪声，观察生成的图片  
z = torch.randn(1, latent\_size).to(device)  
fake\_images = G(z).view(28, 28).data.cpu().numpy()  
plt.imshow(fake\_images, cmap=plt.cm.gray)  
plt.show()  
plt.imshow(next(iter(dataloader))[0][0][0], cmap=plt.cm.gray)  
plt.show()

**运行截图：**

文本

描述已自动生成

**生成1**

**QR 代码

描述已自动生成**

**原图1**

**QR 代码

描述已自动生成**

**生成2**

**QR 代码

描述已自动生成**

**原图2**

**QR 代码

描述已自动生成**

1. **实验总结**

**深入理解了GAN对抗网络，了解它的原理，结合神经网络的知识进行了实现，进一步加深了印象，并熟悉了人工智能深入学习的训练风格。**