《人工智能导论大作业》

任务名称：Mnist 条件生成器

完成组号：

小组人员：许昊晨、张奕宁、张瀚文、刘俊杰、郑燮宇

完成时间：2023/5/13

目录

1 任务目标

基于 Mnist 数据集，训练一个 GAN 模型，当输入的条件为 0-9 的数字，输出对应条件的生成图像。在保证图像的清晰度及不可辨识性（Mnist 分类器不可辨识）的前提下，尽可能地压缩仅在 CPU 上运行的时间

2 具体内容

该项目中我们采用了 GAN 模型

2.1 实施方案

GAN 包含有两个模型，一个是生成模型（generative model），一个是判别模型 (discriminative model)。生成模型的任务是生成看起来自然真实的、和原始数据相似的实例。判别模型的任务是判断给定的实例看起来是自然真实的还是人为伪造的（真实实例来源于数据集，伪造实例来源于生成模型）GAN 网络的整体示意如下：

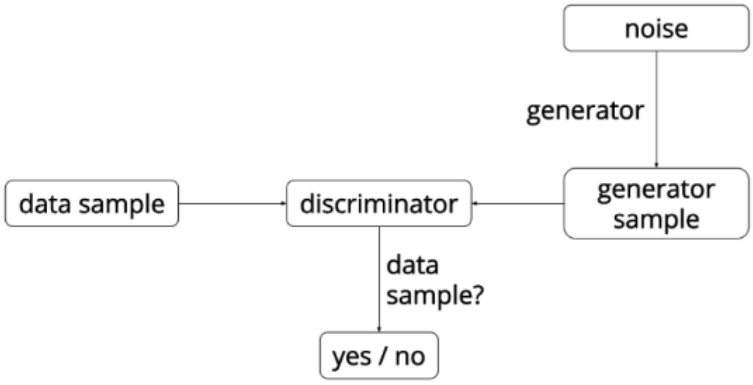

图 1: GAN 网络示意

项目主要分为 4 个部分：**数据导入 (makeData.py)、模型框架搭建 (model.py)、训练模型并保存 (train.py)、导入模型并生成图片 (aigcmn.py)**

- 数据导入部分流程比较固定，在此不多赘述
- model.py 中定义了 $make\_CNN()$ 函数用于生成训练层模板；Generator 与 Discriminator 两个类用于构建生成器与判别器的模板
- train.py 中将训练图片输入 Discriminator 并进行打分得到 $loss\_1$，将 Generator 生成的虚假图片输入 Discriminator 并进行打分得到 $loss\_2$，用 $loss\_2$ 训练 Generator，用 $loss\_1$ 与 $loss\_2$ 之和训练 Discriminator
- aigcmn.py 中将训练好的模型导入，并指定要生成的数字对应的标签，在生成随机噪音，将标签与噪音结合输入 Generator，得到输出图片，在经历了不同轮数的训练后，得到的输出图片如下

第一轮训练

第五轮训练

图 2: 第一轮、第五轮产出

第十轮训练

第二十五轮训练

图 3: 第十轮、第二十五轮产出

第五十轮训练

第一百轮训练

图 4: 第五十轮、第一百轮产出

除此之外，我们记录了每轮训练中的 loss，并绘制了 loss 的变化曲线，以确定在迭代多少轮后可以使模型参数达到稳定，loss 曲线如下

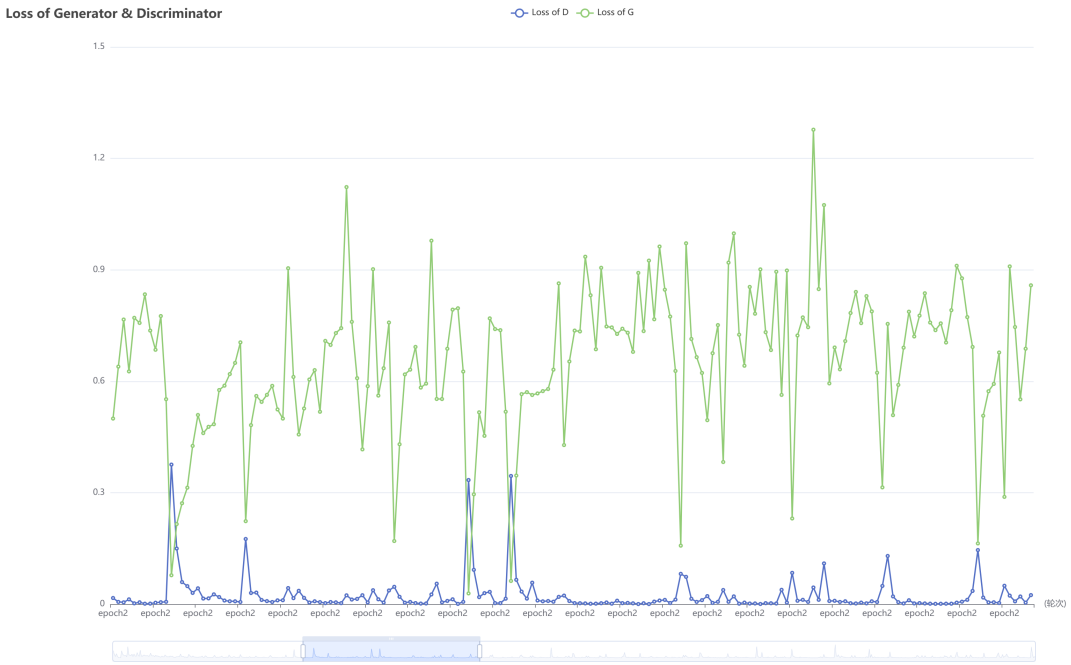

图 5: 第一轮内 loss 曲线

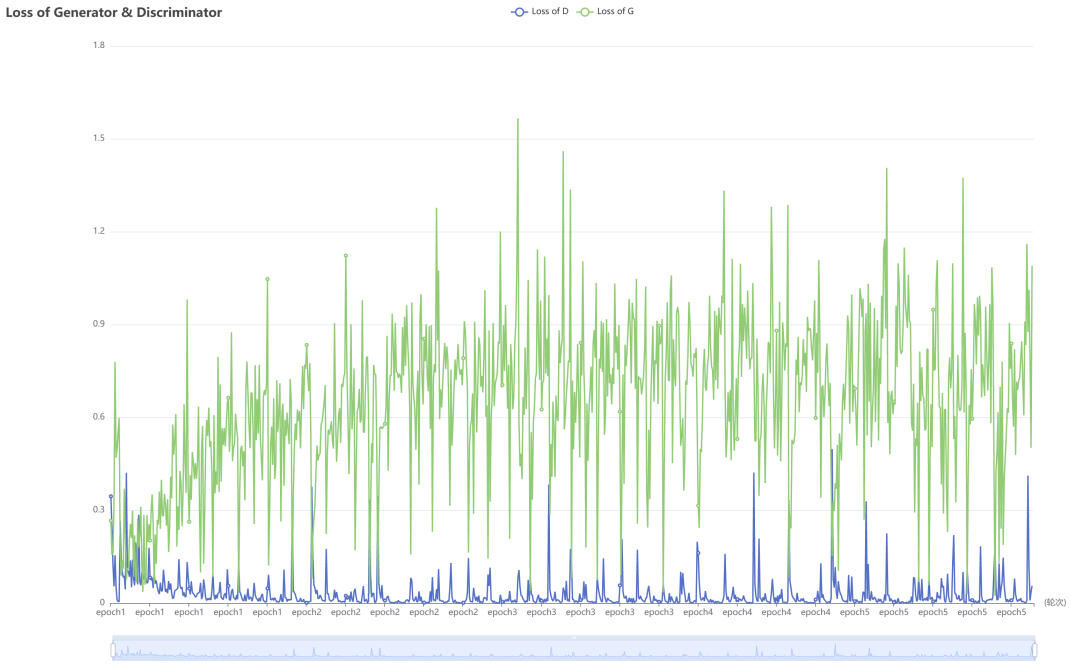

图 6: 前五轮内 loss 曲线

可以看到 Generator 与 discriminator 的 loss 值呈反相关，但由于 GAN 的特性，G 和 D 都并未收敛

2.2 核心代码分析

2.2.1 model.py

定义 $make\_CNN$ 用于搭建卷积神经网络的模板，便于实体化

```
def make_CNN(channels,max_pool,kernel_size,stride,padding,active,):
    net = []
    for i in range(len(channels)-1):
        net.append(nn.Conv2d(in_channels=channels[i], out_channels=channels[i+1],
                             kernel_size=kernel_size[i], padding=padding[i], stride=stride[i], bias=False))
        if i == 0:
            net.append(nn.LeakyReLU(0.2))
        elif active[i] == "LR":
            net.append(nn.BatchNorm2d(num_features=channels[i+1]))
            net.append(nn.LeakyReLU(0.2,inplace=True))
        elif active[i] == "sigmoid":
            net.append(nn.Sigmoid())
        elif active[i] == "tanh":
            net.append(nn.Tanh())
        if max_pool[i]:
            net.append(nn.MaxPool2d((2, 2)))

    return nn.Sequential(*net) #组装在一起
```

该函数用于生成训练层模板，其中参数意义如下：

- channels: 每一层的通道数
- max_pool: 是否进行最大池化
- kernel_size: 卷积核大小
- stride: 进行卷积操作时的步幅
- padding: 卷积操作时的填充

函数的主体是按照给定的参数生成一个模板，本项目中卷积网络结构模板的结构如下：

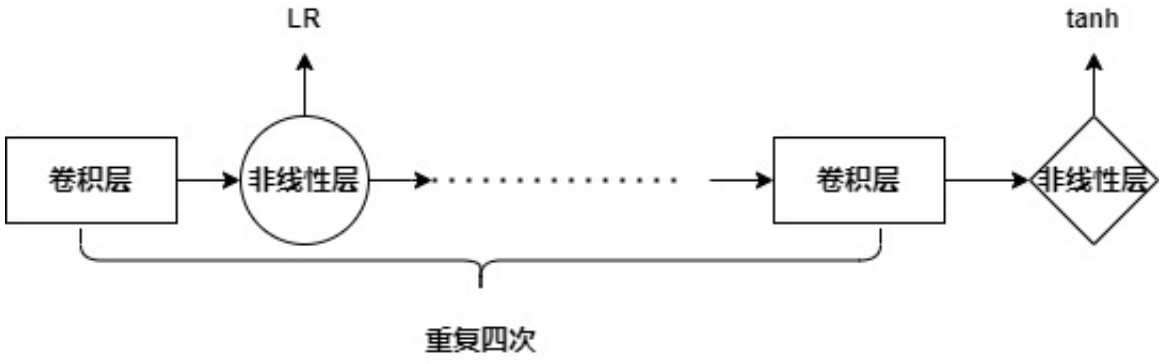

图 7: 卷积结构

最后将网络组装在一起

定义用于生成假图片的生成器 Generator，使用 CNN 和 linear 组成

```
class Generator(nn.Module):
    def __init__(self, num_feature=3136,noise_dim=100, num_classes=10,device='cpu'):
        super(Generator, self).__init__()
        self.ngpu=1
        self.noise_dim = noise_dim
        self.num_classes = num_classes
        self.device=device
        channels=[1,50,250,50,25,1]
        max_pool=[0,0,0,0,0]
        active = ["LR", "LR", "LR", "LR", "tanh"]
        stride = [1, 1, 1,1,2]
        padding=[1,1,1,1,0]
        kernel_size =[3,3,3,3,2]
        self.generator=make_CNN(channels,max_pool,kernel_size,stride,padding,active).to(device)
        self.fc = nn.Linear(noise_dim+num_classes, num_feature).to(device)
```

该函数用于定义 Generator，其中参数意义如下：

- channels: 各层网络的通道参数，第一层的输入输出是【1，50】，第二层的输入输出是【50，250】以此类推
- ngpu: 使用的设备的数量
- $noise\_dim$: 使用的噪声的维数，本实验使用完全随机噪声

```
self.generator=make_CNN(channels,max_pool,kernel_size,stride,padding,active).to(device)
self.fc = nn.Linear(noise_dim+num_classes, num_feature).to(device)
```

在实例化的过程中，先调用上述的 $make\_CNN$ 创建卷积神经网络，之后创建线形层，接收来自噪声向量和类别标签的输入

```
def forward(self, x, label):  # x: [batch_size, 1, 28, 28]
        label=self.label(label)
        x=self.image(x).to(self.device)
        data = torch.cat(tensors=(x, label), dim=1).to(self.device)
        out = self.discriminator(data).to(self.device)
        out = out.view(out.size(0), -1)
        out=self.fc(out)
        return out  # [batch_size, 10]
```

forward 部分实现了前向传播过程，在此方法中，首先通过全连接层将输入的 batch_size,noise_dim,num_classes 拼接起来，将其放入到 linear 层中进行计算，将得到的结果重构成 batch_size,1,28,28 的矩阵，最终将其传入卷积神经网络进行最终的图片生成计算。

之后我们定义用于判别图片真假的判别器 Discriminator，它的功能是识别出哪些图片是真实图片，哪些图片是由 Generator 生成的假图片

其中具体的参数意义如下：

- channels: 代表每一层的卷积核数量
- max_pool: 是否采用 pooling 类操作进行特征抽样（0 表示不使用，1 表示使用）
- kernel_size: 卷积核的大小
- stride: 卷积核的滑动步长
- padding：零填充边界区域的大小
- active：每一层卷积的激活函数类型

在如下代码段中，self.image 和 self.label 分别表示处理输入图片和标签的卷积部分，他们都通过 LR 激活函数来增强神经网络的非线性表达能力

```
self . discriminator =make_CNN(channels,max_pool,kernel_size,stride,padding,active)
#利用上述定义的make_CNN来创建卷积模型
self .image = nn.Sequential(
# input is (nc) x 32 x 32
nn.Conv2d(1, 16, 4, 2, 1, bias=False),#用于处理输入图片的卷积层
nn.LeakyReLU(0.2, inplace=True)
 # state size . (ndf) x 16 x 16
        )
self . label = nn.Sequential(
nn.Conv2d(10, 16, 4, 2, 1, bias=False),#用于处理标签的卷积层
nn.LeakyReLU(0.2, inplace=True)
 )
```

接下来，定义了一个全连接层 self.fc，是具有三个全连接层的神经网络

```
    self . fc = nn.Sequential(
            nn.Linear(3* 3 * 1024, 2048), # 第一个线性层
            nn.LeakyReLU(0.2, True),
            nn.Dropout(0.02),
            nn.Linear(2048, 512), #第二个线性层
            nn.LeakyReLU(0.2, True),
            nn.Dropout(0.02),
            nn.Linear(512, 10), # 第三个线性层
            nn.Sigmoid()
        )
```

nn.Linear 是第一个线性层，用于将输入的数据变换成为指定维度，其输入为一个 3*3*1024 维的数据，输出为一个 2048 的向量

nn.LeakyReLU 即为上述提到过的 LR 函数，其为修正线性单元函数，也是该网络的激活函数，可以使得神经网络中一部分神经元的输出为负数，从而增强模型学习的非线性表达能力

nn.Dropout 层会在训练时随机地将输入张量的某些元素清零，用于控制模型的过拟合

nn.linear(2048,512) 是第二个线性层。输入为 2048 维，输出为 512 维

nn.linear(512,10) 是最后一个线性层，而且此层使用了 Sigmoid 激活函数，使得此模型具有了处理二分类、多分类、回

归问题的能力

之后，Discriminator 定义了与 Generator 相似的 forward 函数，由于功能相似，在此仅列出代码，具体功能不再赘诉

```
        def forward(self, x, label):   # x: [batch_size, 1, 28, 28]

        label=self.label(label)
        x=self.image(x).to(self.device)

        data = torch.cat(tensors=(x, label), dim=1).to(self.device)
        out = self.discriminator(data).to(self.device)

        out = out.view(out.size(0), -1)
        out=self.fc(out)
        return out  # [batch_size, 10]
```

最后我们定义了一个用于生成图片的函数，用于将训练生成的图片统一存放到 pictures_outputs 文件夹中，便于项目的管理

```
    def show_images(images): # 定义画图工具
    print('images: ', images.shape)
    images = np.reshape(images, [images.shape[0], -1])
    sqrtn = int(np.ceil(np.sqrt(images.shape[0])))
    sqrtimg = int(np.ceil(np.sqrt(images.shape[1])))

    fig = plt.figure(figsize=(sqrtn, sqrtn))
    gs = gridspec.GridSpec(sqrtn, sqrtn)
    gs.update(wspace=0.05, hspace=0.05)

    for i, img in enumerate(images):
        ax = plt.subplot(gs[i])
        plt.axis('off')
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        ax.set_aspect('equal')
        plt.imshow(img.reshape([sqrtimg, sqrtimg]))
    return
```

2.2.2 makeData.py

```
def load_data(if_download=False,batch_size=64):
    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5],std=[0.5])])
    train_data = datasets.MNIST(root = "./data/",transform=transform, train = True,download =if_download)
    test_data = datasets.MNIST(root="./data/", transform = transform, train = False)

    dataset=train_data+test_data

```

```
    loader = torch.utils.data.DataLoader(dataset,
                                         batch_size=batch_size,
                                         shuffle=True,
                                         num_workers=2)
    return dataset, loader
```

最后将数据集返回

2.2.3 train.py

首先定义了一些训练过程中的参数

```
lr = 0.0002          # 学习率
noise_dim=100        # 生成器输入通道
beta1 = 0.5          # 优化器Adam用的beta1
num_epochs=100       # 训练轮次
image_size = 28      #图像大小
b_size =64           #批大小
num_feature=3136
ngpu = 1             #gpu数量
```

生成器、判别器、优化器的初始化:

```
#生成器和判别器
netG=Generator(num_feature=num_feature,noise_dim=noise_dim,num_classes=10,device=device).to(device)
netD=Discriminator(device=device).to(device)
#优化器
optimizerD = torch.optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = torch.optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
```

判别器的训练:

```
# 生成 lable 的 one-hot 向量，且设置对应类别位置是 1
labels_onehot = np.zeros((real_image.shape[0], 10))
labels_onehot[np.arange(real_image.shape[0]), Ten_real_label.numpy()] = 1
# 真实数据标签和虚假数据标签，
real_label = Variable(torch.from_numpy(labels_onehot).float()).to(device)  # 真实label对应类别是为1
fake_label = Variable(torch.zeros(real_image.shape[0], 10,device=device)).to(device) # 假的label全是为0
# Forward pass real batch through D
real_output = netD(real_image, D_label)
real_output = real_output.view(-1)
# 计算判别器的loss，我们希望它更接近正确答案，所以去和真实标签做交叉熵
errD_real = criterion(real_output, real_label.view(-1))
# 把这个loss反向传播回去
errD_real.backward()
# 生成随机向量，也就是噪声z，带有标签信息给生成器使用
z = Variable(torch.randn(real_image.shape[0], noise_dim))
z = np.concatenate((z.numpy(), labels_onehot), axis=1) # 噪音和one-hot 向量加在一起
z = Variable(torch.from_numpy(z).float())
# 用刚刚的随机噪音生成假图像fake
```

```
fake_image = netG(z)
# 用我们自己的判别器Discriminator去判别一下我们自己的Generator
fake_output = netD(fake_image.detach(), D_label).view(−1)
# 计算一下假的图像和全零的交叉熵，我们现在在训练D，
所以我们希望判别器认为G生成的图片越假越好，和全零越贴合越好，这里于是算一下和全零的交叉熵
errD_fake = criterion(fake_output, fake_label.view(−1))
# 计算梯度反向传播
errD_fake.backward()
# 把看见假数据和真数据的误差相加当作loss
 errD = errD_real + errD_fake
# 更新D的optimizer
optimizerD.step()
```

生成器的训练

```
netG.zero_grad()
# 用我们制作好的D去检测我们刚刚用G制作的假图片
fake_output = netD(fake_image, D_label).view(−1)
fake_output=fake_output.to(device)
# 计算loss，因为我们在训练生成器G，所以我们希望这个生成器的结果越接近真实越好，
这样说明我们的G足够优秀骗过了D
errG = criterion(fake_output, real_label.view(−1))
# 计算梯度反向传播
errG.backward()
D_G_z2 = fake_output.mean().item()
# 更新D的optimizer
optimizerG.step()
```

保存最佳网络结果

```
if errG < loss_tep:
torch.save(netG.state_dict(), "model/model_"+str(epoch+1)+"_"+str(iters)+".pt" )
loss_tep = errG
```

生成训练过程中的图片

```
if( iters %250==0):
    i = vutils.make_grid(fake_image, padding=2, normalize=True)
    fig = plt.figure( figsize =(4, 4))
    plt.imshow(np.transpose(i.to('cpu'), (1, 2, 0)))
    plt.axis('off')  # 关闭坐标轴
    plt.savefig("picures_output/%d_%d.png" % (epoch+1, iters))
    plt.close( fig )
iters +=1
```

2.2.4 aigcmn.py

该文件中实现了接口类 AiGcMn()：

```
class AiGcMn():
```

```
    def __init__(self):
        self.G=Generator()
        self.G.load_state_dict(torch.load("modelG/model_0_0.pt"))

    def generator(self,Labels):

        # self.Labels = Labels
        labels_onehot = np.zeros((64, 10))
        labels_onehot[np.arange(64), Labels.numpy()] = 1
        z = Variable(torch.randn(64, 100))
        z = np.concatenate((z.numpy(), labels_onehot), axis=1) # 噪音和one-hot 向量加在一起
        z = Variable(torch.from_numpy(z).float())
        output=self.G(z)
        return output
```

构造函数中初始化一个 Generator 的对象，并加载已经训练好的模型。接口函数 generator() 中生成随机噪音然后将噪音与提供的标签加在一起，通过 self.G() 产生结果，**output** 即为所需的 64*1*28*28 的 tensor.
除了实现接口函数外，我们还产生了一个实例，即调用 generator() 函数产生图片，代码如下

```
# 示例代码：输出20张 的前64位
def EXAMPLE_PI_20():
    Label='31415926535897932384626433832795028841971693993751058209749445 92'
    Labels=[]
    for c in Label:
        Labels.append(int(c))
    Labels=torch.tensor(Labels)
    filename = r'/home/tim/workspace/test.txt'


    if not(os.path.exists('EXAMPLE_PI_OUTPUT')):
        os.mkdir('EXAMPLE_PI_OUTPUT')

    for j in range(20):
        a=AiGcMn()
        output=a.generator(Labels)
        i = vutils.make_grid(output, padding=2, normalize=True)
        fig = plt.figure(figsize=(5, 5))
        plt.imshow(np.transpose(i.to('cpu'), (1, 2, 0)))
        plt.axis('off')  # 关闭坐标轴
        plt.savefig("EXAMPLE_PI_OUTPUT/test_pic_%d.png" % (j))
        plt.close(fig)
```

该实例生成的是以圆周率为例的数字图像，此处的 **Labels** 即为上述所提及的与噪音相加的标签，若要生成其他的手写数字，将 **Labels** 改为 1*64 的一维数组即可.

3 工作总结

3.1 收获与心得

- 了解生成模型：通过实验，我们更深入的了解了生成模型。生成模型是一种从潜在空间到真实样本分布的映射，其目标是生成与真实数据样本相似的新样本。
- 理解 GAN 网络结构：GANs 已被证明是一种非常有效的生成模型，它们可以生成高品质的图像、音频和文本数据等，并在众多任务中刷新记录；而在制作生成器的过程中，我们研究深度学习框架如何实现 GAN，并对 GAN 网络的结构进行了更深入的理解。这有助于我们更好地发现并解决 GAN 实验过程中出现的问题。
- 超参数调节非常重要：在实现 MNIST 条件生成器模型时，调整超参数是至关重要的。例如，更改网络架构、损失函数或优化器的设置可能会对模型的性能产生巨大影响。
- 训练时间长：训练 MNIST 条件生成器模型需要耗费一定的时间，并且算力也较为消耗，需要具备一定的运算设备和技术水平；在制作过程中，我们对于已有的生成器有了一定的了解，深刻认识到了我们当下设备与正规设备算力的差距以及算力的重要性。

3.2 遇到问题及解决思路

- Q：在确定卷积层参数时，稍微改动一点参数就会导致训练结果出现很大的波动，甚至无法进行训练
  解决：手动调整卷积层参数，多次试验直至产生较好的训练效果
- Q: 在进行 cnn 参数调节时，我们花费了不少时间，参数的些微变动便会导致结果出现很大的波动，甚至使训练无法正常进行，如何得到一个优秀的结果并且将时间控制在一个可接受的范围内，是一个十分重要的问题.
  解决: 超参数的选择通常需要根据经验和试错来进行。常见的调参方法包括网格搜索、随机搜索和贝叶斯优化等。这些方法可以帮助我们在一定范围内快速找到最优的超参数组合。同时，一些经验法则例如学习率初始值的选择，也可以帮助我们快速找到一个不错的超参数组合。对于模型参数的调整，可以使用随机初始化的方法来找到一个较好的起始点。在训练过程中，可以使用一些优化器，例如随机梯度下降 (SGD)、动量 (momentum)、Adam 等来进行参数优化。而在实际操作中，我们进行了手动的多次尝试，寻找到了一组较为合适与平衡的参数。

4 课程建议

- 通过本门课程的学习，我们对人工智能的历史、现状、未来发展趋势等有了更为清晰的认知，更好地理解了本课程所探究的问题。本门课程对于人工智能的基础概念也有着系统的教学，讲授了一些人工智能技术的基础概念与方法，如机器学习、知识表示、自然语言处理等，并且解释了它们的原理以及如何运用。通过实战项目，我们深入了解实际使用不同类型的人工智能技术，提出问题并解决问题。这有助于我们将所学的概念应用到实际场景中，并锻炼了我们的创新思维和动手实践能力。
- 而课程的改进与补充之处可以考虑加强对于学生基础参差不齐的考虑与引导，培养学生查阅文档解决问题的能力，提供一些专业性更强的论坛与网站并鼓励学生积极参与；
  引入伦理和社会问题：向学生介绍人工智能经常查询的伦理及社会问题，如人工智能对就业、隐私、安全等问题的影响，并鼓励学生探讨并分享自己对于这些问题的看法；
  加强课堂的互动，鼓励学生对于自己所了解的人工智能的基础概念以及当下的一些模型、技术进行分享；
  进一步提供一个针对人工智能领域的进一步阅读材料建议，例如专业期刊文章，期刊论文或者其他有用的书籍。这样可以帮助学生加深所学的概念，并在发展中不断了解更多的行业状况。