西北工业大学计算机学院

《多媒体技术》实验报告

学　　号：　2019302863

姓　　名：　李奇

实验时间：　2021.11.21

实验地点：　实验大楼 209-3

实验题目：　Speech command recognition

西北工业大学

2021 年 11 月

一、 实验目的及要求。

1.学习神经网络的基础知识。

2.掌握多层感知器和全连接网络的基本原理。

3.学习语音识别的流程，能够编写语音指令识别代码。

4.掌握神经网络的训练过程，得到有效的结果。

5.尝试使用 CNN 和 RNN 进行语音指令识别的实验。

二、 实验内容与操作步骤。

1.学习神经网络的基础知识，掌握多层感知器（MLP）和全连接网络（FCNN）的基本原理。

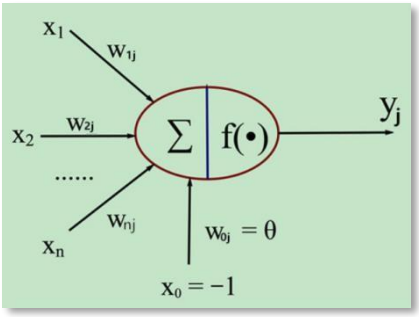
2.学习语音识别的流程，编写语音指令识别代码，并按照给定的训练集和测试集进行实验，获得指令识别准确率（错误率）。

3.[选做] 学习卷积神经网络（CNN）和回归神经网络（RNN）的基础知识，尝试使用这些网络进行语音指令识别的实验，并和 FCNN 的结果进行对比。

三、 实验结果分析。

1.学习神经网络的基础知识，掌握 MLP 和 FCNN 的基本原理。

一般来说，现在多用“M-P”神经元模型来表示神经网路中最基本的成分：神经元接收 n 个神经元的信号输入，将总输入与阈值进行比对，然后通过“激活函数”处理产生神经元输出。比如下图为一个 M-P 神经元模型：



多层感知机在单层神经网络的基础上引入了一到多个隐藏层,隐藏层位于输入层和输出层之间。感知机由两层神经元组成，感知机因为只拥有一层功能神经元，处理、学习能力有限，只能处理线性可分问题。我们引入多层神经网络，每层神经元与下层神经元相互链接，神经元之间不存在同层链接或是跨层链接。除输入、输出层外，其余层称为隐藏层。（这样的结构通常称“多层前馈神经网络”）

当 CNN 卷积核大小与输入大小相同时，其计算过程等价于 MLP，也就是说 MLP 等价于卷积核大小与每层输入大小相同的 CNN。

FCN 对图像进行像素级的分类，从而解决了语义级别的图像分割问题。与经典的 CNN 在卷积层之后使用全连接层得到固定长度的特征向量进行分类（全联接层＋softmax 输出）不同，FCN 可以接受任意尺寸的输入图像，采用反卷积层对最后一个卷积层的 feature map 进行上采样，使它恢复到输入图像相同的尺寸，从而可以对每个像素都产生了一个预测，同时保留了原始输入图像中的空间信息，最后在上采样的特征图上进行逐像素分类。

简单来说 FCNN 与 CNN 的区域在把于 CNN 最后的全连接层换成卷积层，输出的是一张已经 label 好的图片。

2.学习语音识别的流程，编写语音指令识别代码，按照给定的训练集和测试集进行实验，得到识别准确率。

语音识别的流程：

语音识别流程，就是将一段语音信号转换成相对应的文本信息的过程。转换过程中涉及到对采集的声音信号进行滤波、分帧等与处理工作，以此来将所需信号从原始信号中提取。特征提取能够将声音信号从时域转换到频域，为声学模型提供合适的特征向量。之后声学模型会根据声学特性计算每一个特征向量在声学特征上的得分，语言模型根据语言学理论计算该声音信号对应的可能的词组序列的概率值，最后根据已有词典对词组序列进行解码，得到最后可能的文本表示。指令词识别需要在训练前对音频文件做特征提取，送入模型做训练的数据就是这些提取出的特征：

我们采用全连接的方式进行训练，选择 sigmoid 作为激活函数，将 batch_size 设置为 300，训练 100 个 epoch，对应的模型代码、训练代码及注释以及训练结果如下所示：

模型代码 FcNet,位于 model.py:

```
class FcNet(nn.Module):
    # DNN
    def __init__(self):
```

```
        super(FcNet, self).__init__()

        self.fc1 = nn.Linear(161 * 101 * 1, 1000)
        # 全连接层 torch.nn.Linear(in_features, out_features)
        # in_features:输入特征维度
        # out_features：输出特征维度
        self.fc2 = nn.Linear(1000, 6)
        # 输出维度 6 分类

    def forward(self, x):
        x = x.view(-1, 161 * 101 * 1)  # x = (N,161*101*1)
        x = F.sigmoid(self.fc1(x))  # x =
(N,161*101*1)*(161*101*1,1000)=(N,500)
        x = self.fc2(x)  # x = (N,1000)*(1000,6)=(N,6)
        return F.log_softmax(x, dim=1)  # 带 log 的 softmax 分类，每段音频返回 6
个概率
```

训练代码 train,位于 train.py:

```
def train(loader, model, optimizer, epoch, cuda, log_interval,
verbose=True):
    model.train() #进入训练模式
    for batch_idx, (data, target) in enumerate(loader):
        if cuda: #使用 GPU 加速
            data, target = data.cuda(), target.cuda()
        with torch.no_grad():
            data, target = Variable(data), Variable(target)
        optimizer.zero_grad() #梯度归零
        print(data.shape)
        output = model(data) #输出的维度[N,6] 这里的 data 是函数的 forward 参数 x
        train_loss = F.nll_loss(output, target) #loss 求的平均数，除以 batch
        # 单分类交叉熵损失函数，一段音频里只能有一个类别，输入 input 的需要 softmax
        train_loss.backward()
        optimizer.step()
        if verbose:
            if batch_idx % log_interval == 0:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss:
{:.6f}'.format(
                    epoch, batch_idx * len(data), len(loader.dataset),
                       100. * batch_idx / len(loader), train_loss.item()))
        return train_loss
```

训练结果：

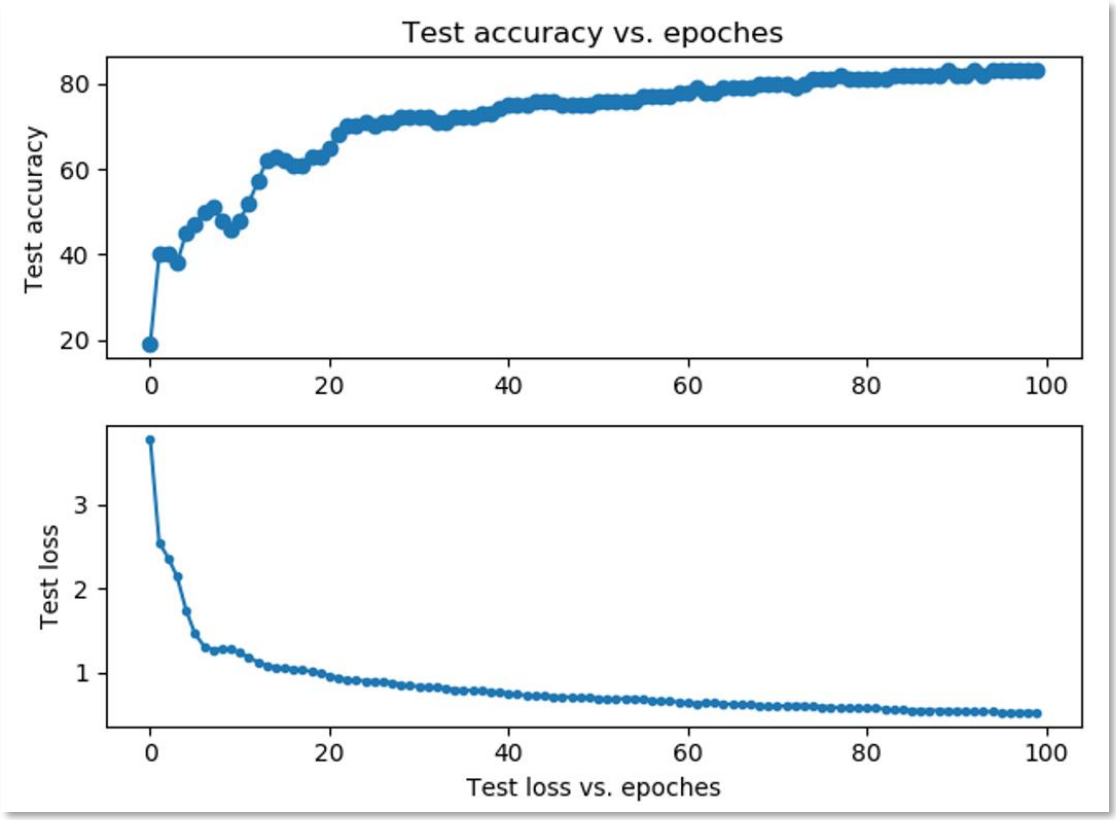
```
Test set: Average loss: 0.5257, Accuracy: 1247/1496 (83.36%)

Loss was not improved, iteration 2
torch.Size([300, 1, 161, 101])
Train Epoch: 100 [0/11155 (0%)] Loss: 0.377640

Test set: Average loss: 0.5226, Accuracy: 1246/1496 (83.29%)

Loss was not improved, iteration 3
```

从结果可以看出，模型最终在 83.29%收敛，达到了预期效果。训练过程的准确率和 loss 随 epoch 的增加发生的变化如下图所示：



3.[选做]学习 CNN 和 RNN 的基础知识，并使用这些网络进行语音指令识别的实验，和 FCNN 的结果进行对比，可以只做其中一个。

本题我选用卷积神经网络 CNN 来完成，卷积神经网络是一类包含卷积计算并且具有深度结构的前馈神经网络，是深度学习的代表算法之一，他的基本结构为输入层->隐藏层->输出层。其中隐藏层是卷积神经网络的关键，它包括了卷积层、池化层、激活函数和全连接层等组成成分。对于这四个部分，个人的理解是：卷积层用来提取特征，池化层用来降采样，激活函数用来增加非线性组合，全链接层应当是将卷积（矩阵）转化为向量，进行特征整合，方便交给分类器。要注意的是池化过程中深度保留且没有参数。

为了实现该实验，我设计了 4 个卷积层，2 个池化层，激活函数采用 relu 函数。具体结构为：第一层卷积核大小为 22*22，共 10 层，步长为 1，之后经过 1 层 2*2 的池化层；第二层卷积核大小为 1*1，共 20 层，步长为 1；第三层卷积核大小为 11*11，共 40 层，步长为 1，之后经过 10*10 的池化层；第四层卷积核大小为 2*2，共 80 层，步长为 1；最后经过全连接层进行分类。对应的模型代码、训练代码及注释以及训练结果如下所示：

模型代码 ConvNet，位于 model.py：

```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 22, 1)
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1)
        # in_channels：输入音频通道数
        # out_channels：输出通道数，这个等于卷积核的数量
        # kernel_size：卷积核大小
        # stride：步长
        self.conv2 = nn.Conv2d(10, 20, 1, 1)
        # 上个卷积网络的 out_channels 就是下一个网络的 in_channels，所以这里是 30
        # out_channels：卷积核数量 60
        # 通过一个 1*1 的卷积 步长为 1
        self.conv3 = nn.Conv2d(20, 40, 11, 1)
        self.conv4 = nn.Conv2d(40, 80, 2, 1)
        # 同上
        self.fc1 = nn.Linear(5 * 2 * 80, 800)
        # 全连接层 torch.nn.Linear(in_features, out_features)
        # in_features:输入特征维度
        # out_features：输出特征维度
        self.fc2 = nn.Linear(800, 6)
        # 输出维度 6 分类
    def forward(self, x):
        # print(x.shape)  #输入维度，(N,1,161,101) N 为一个 batch 大小
        x = F.relu(self.conv1(x))  # x = (N,10,140,80)
        x = F.max_pool2d(x, 2, 2)  # x = (N,10,70,40) 池化 降采样
        x = F.relu(self.conv2(x))  # x = (N,20,70,40)
        x = F.relu(self.conv3(x))  # x = (N,40,60,30)
        x = F.max_pool2d(x, 10, 10)  # x=(N,40,6,3)
        x = F.relu(self.conv4(x))  # x = (N,80,5,2)
        x = x.view(-1, 5 * 2 * 80)  # x = (N,10*4*50)
        x = F.sigmoid(self.fc1(x))
        x = self.fc2(x)
```

```
        return F.log_softmax(x, dim=1)  # 带 log 的 softmax 分类，每段音频返回 6
个概率
```

训练代码 train,位于 train.py:

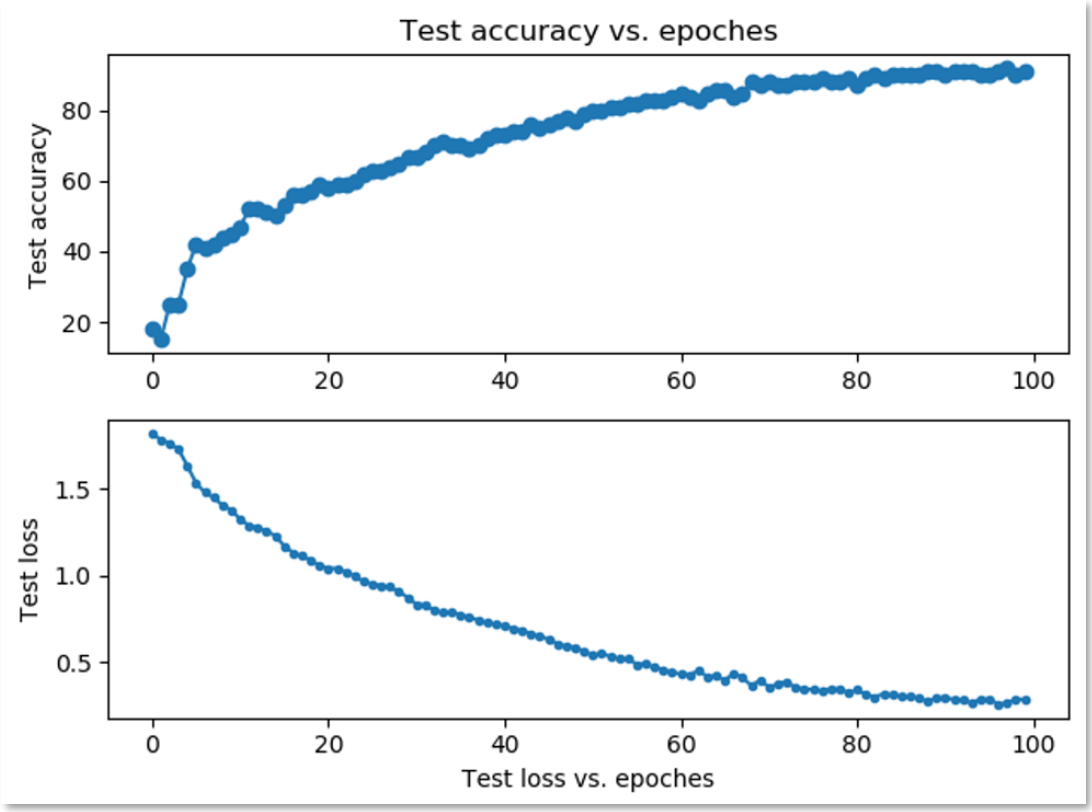
```
def train(loader, model, optimizer, epoch, cuda, log_interval,
verbose=True):
    model.train() #进入训练模式
    for batch_idx, (data, target) in enumerate(loader):
        if cuda: #使用 GPU 加速
            data, target = data.cuda(), target.cuda()
        with torch.no_grad():
            data, target = Variable(data), Variable(target)
        optimizer.zero_grad() #梯度归零
        print(data.shape)
        output = model(data) #输出的维度[N,6] 这里的 data 是函数的 forward 参数 x
        train_loss = F.nll_loss(output, target) #loss 求的平均数，除以 batch
        # 单分类交叉熵损失函数，一段音频里只能有一个类别，输入 input 的需要 softmax
        train_loss.backward()
        optimizer.step()
        if verbose:
            if batch_idx % log_interval == 0:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss:
{:.6f}'.format(
                    epoch, batch_idx * len(data), len(loader.dataset),
                       100. * batch_idx / len(loader), train_loss.item()))
        return train_loss
```

训练结果：

```
Train Epoch: 100 [0/11155 (0%)] Loss: 0.235929

Test set: Average loss: 0.2824, Accuracy: 1370/1496 (91.58%)
```

同样训练了 100 个 epoch，从训练结果可以看出模型最终在 91.58%收敛，达到了预期效果,对比全连接 acc 有着显著提升。训练过程的准确率和 loss 随 epoch 的增加发生的变化如下图所示：



四、实验中存在问题及改进。