1. 人工神经网络

本实验中，根据语音信号的特征要素，采取三种形式的神经网络架构，分别是基于全连接层的深度神经网络（Deep Neural Network）、卷积神经网络（Convolutional Neural Network）和基于GRU单元的循环神经网络（Recurrent Neural Network）。三种神经网络的训练形式有所不同，效果也各异，以下详细说明。

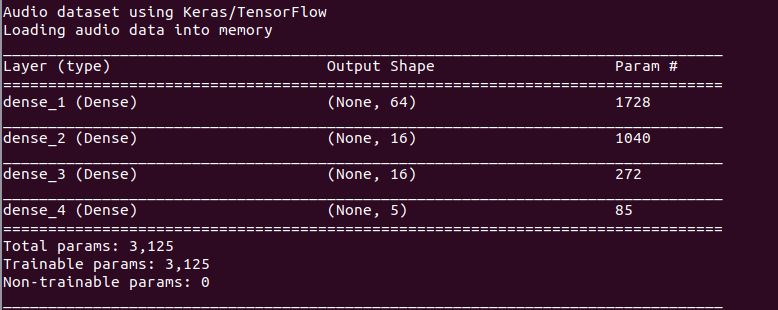
1. DNN

深度神经网络（Deep Neural Network）是一种常见的神经网络模型。其基本结构包括输入层、隐藏层和输出层。DNN的隐藏层均为全连接层，使用激活函数实现去线性化。DNN的特点是，输入层输入必为单个特征向量，这点与CNN、RNN都不同。因此，DNN网络对于序列性、结构性的特征极不敏感。

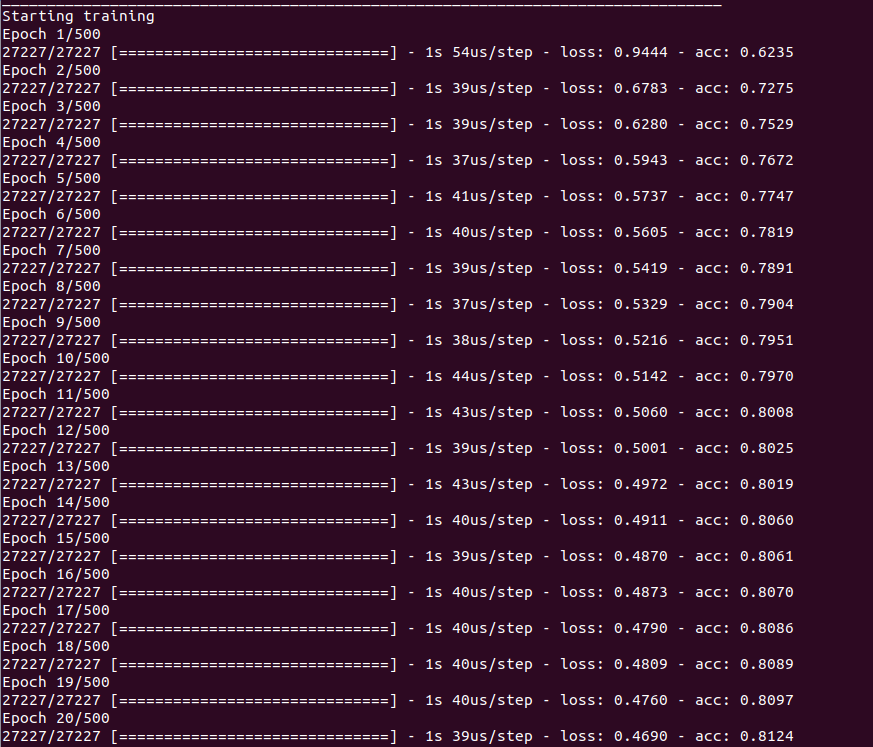
本实验使用的DNN训练代码见dnn.py。代码中主要使用TensorFlow的Keras API，其余神经网络也是基于Keras构建的，官方文档见<https://keras.io/zh/>。

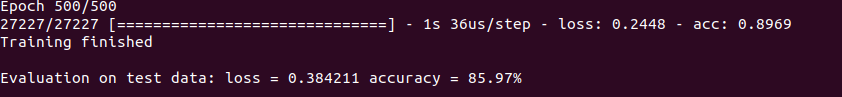
我们将总数据的60%随机设置为训练集，其余为测试集。在进行数据集分离的过程中，随机状态被指定为42（亦或者是其他值），这是为了保证每次训练的数据集一致，能有效地根据结果调试实验参数。

由于本实验为分类实验，因此target数据集应采用one-hot编码，方便使用预测概率分布计算分类结果。下图为网络结构。



训练时的批大小设置为50（根据数据集和特征规模选定，太小使得训练时长太长，太大使得收敛速度过慢），总轮次设置为300轮，收敛速度及测试效果如下。





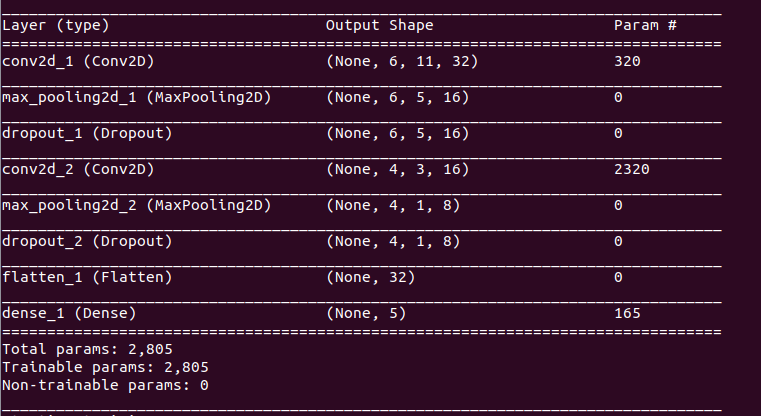
1. CNN

CNN由输入和输出层以及多个隐藏层组成，隐藏层可分为卷积层，池化层、全连接层。CNN的输入一般是二维向量，可以有高度，比如，RGB图像等。卷积层是CNN的核心，层的参数由一组可学习的滤波器（filter）或内核（kernels）组成，它们具有小的感受野，延伸到输入容积的整个深度。在前馈期间，每个滤波器对输入进行卷积，计算滤波器和输入之间的点积，并产生该滤波器的二维激活图（输入一般二维向量，但可能有高度（即RGB）。简单来说，卷积层是用来对输入层进行卷积，提取更高层次的特征。池化层又称下采样，它的作用是减小数据处理量同时保留有用信息。全连接层就是一个常规的神经网络，它的作用是对经过多次卷积层和多次池化层所得出来的高级特征进行全连接，算出最后的预测值。输出层就是对结果的预测值，会加一个softmax层生成概率分布。

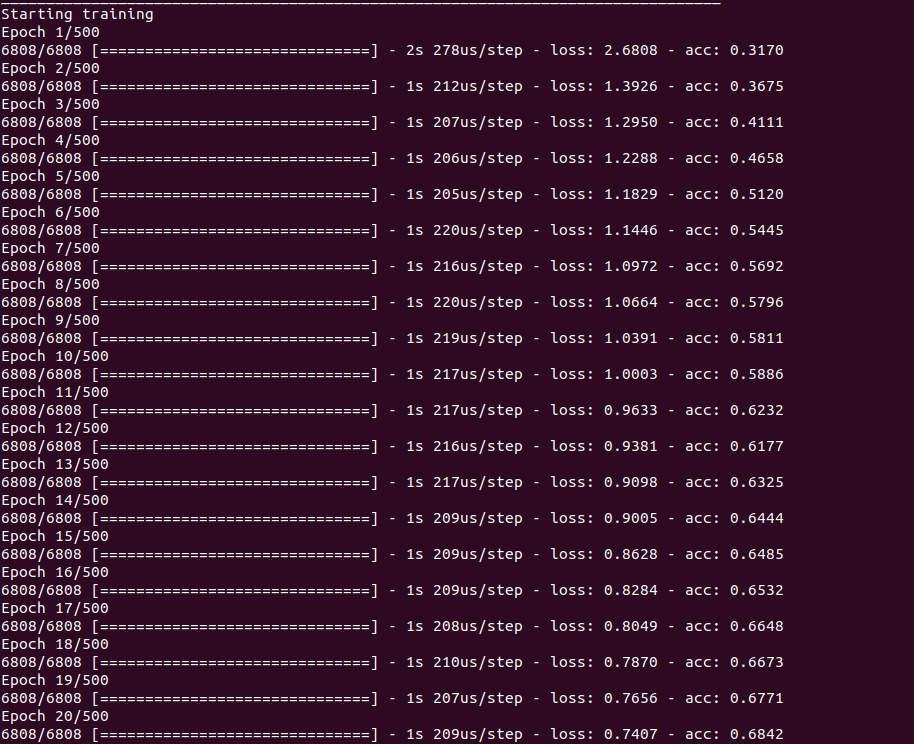
本实验使用的CNN训练代码见cnn.py。

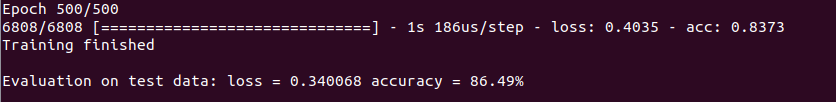
同样地，我们将总数据的60%随机设置为训练集，其余为测试集。有所不同的是，CNN网络需要输入数据维度为3，因此将语音特征按照时间序列组合reshape，否则训练会报错。

下图为网络结构。



训练时的批大小设置为50（根据数据集、特征规模和网络结构选定，太小使得训练时长太长，太大使得收敛速度过慢），总轮次设置为500轮，收敛速度及测试效果如下。





1. RNN

RNN是一种特殊的神经网络结构，它是根据“人的认知是基于过往的经验和记忆”这一观点提出的。它与DNN、CNN不同的是：它不仅考虑前一时刻的输入，而且赋予了网络对前面的内容的一种“记忆”功能。

RNN之所以称为循环神经网络，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。RNN处理时间序列的问题的效果很好, 但是仍然存在着一些问题, 其中较为严重的是容易出现梯度消失或者梯度爆炸的问题。因此, 就出现了一系列的改进的算法, 主要包括LSTM 和 GRU。

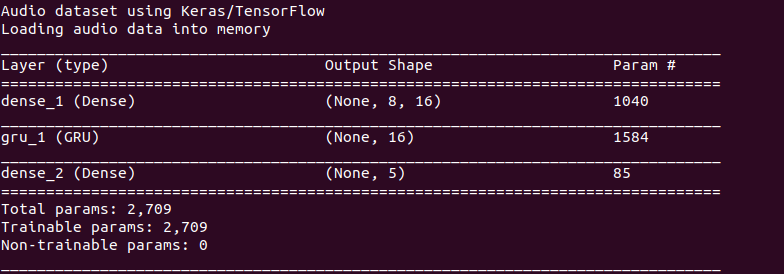
LSTM 对于梯度消失或者梯度爆炸的问题处理方法主要是：对于梯度消失，由于它们都有特殊的方式存储“记忆”，那么以前梯度比较大的“记忆”不会像简单的RNN一样马上被抹除，因此可以一定程度上克服梯度消失问题。对于梯度爆炸，用来克服梯度爆炸的问题就是gradient clipping，也就是当你计算的梯度超过阈值c或者小于阈值-c的时候，便把此时的梯度设置成c或-c。

GRU是2014年提出的一种LSTM改进算法。它将忘记门和输入门合并成为一个单一的更新门，同时合并了数据单元状态和隐藏状态，使得模型结构比之于LSTM更为简单。本实验使用的RNN网络结构就是GRU门单元。

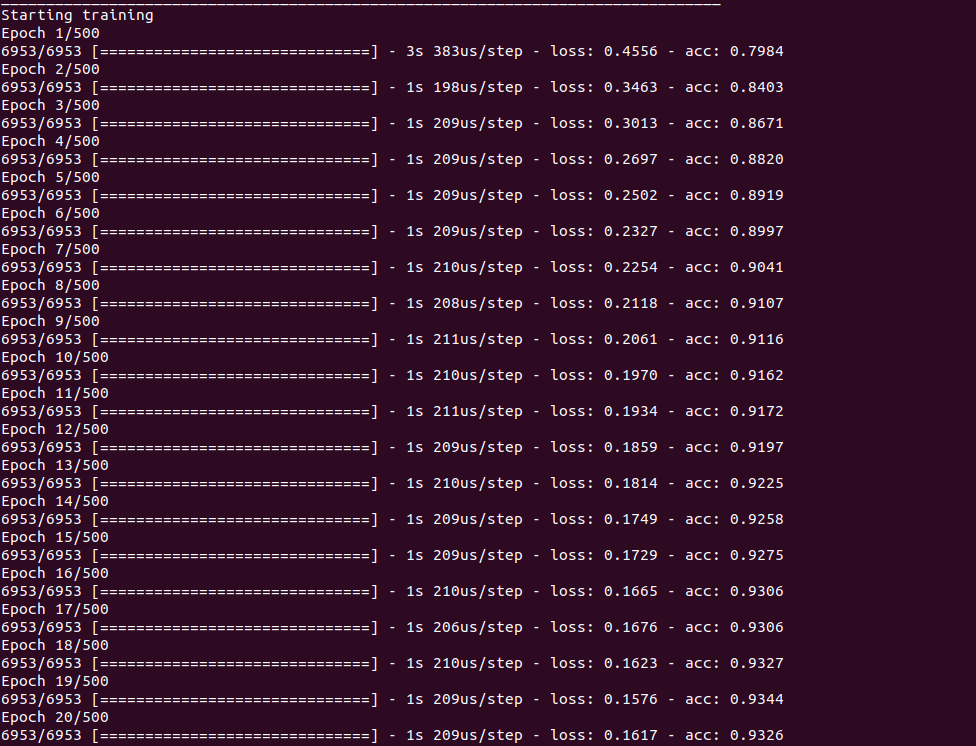
本实验使用的RNN训练代码见rnn.py。

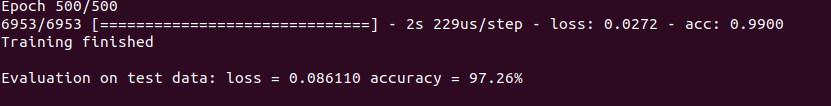
同样地，我们将总数据的60%随机设置为训练集，其余为测试集。值得注意的是，RNN网络需要输入数据维度为2，代表time\_steps和input\_dim，因此将语音特征按照时间序列组合reshape，否则训练会报错。

下图为网络结构。



训练时的批大小设置为50（根据数据集和特征规模选定，太小使得训练时长太长，太大使得收敛速度过慢），总轮次设置为500轮，收敛速度及测试效果如下。





1. RNN神经网络部署

根据上述实验可以看出，在同等参数规模下，RNN网络的收敛速度和训练准确度都是最优的，这是因为RNN网络和GRU门单元充分考虑了音频信号的时序信息，采用序列建模是最科学的音频建模方法。

网络模型训练完成之后，使用Keras中的模型保存API save，将RNN网络模型以hdf5格式文件保存下来，名为rnn\_model.hdf5，其中包括各层权值、偏置、激活函数等等。使用C语言部署此网络模型，目的是为了提高计算速度和适应实时性需求（python代码不适合实时场景）。

gen\_c\_rnn.py的功能是将model模型文件转为C语言代码。转化后自动生成的代码为rnn\_data.h和rnn\_data.c，相关的头文件还包括rnn.h，几类结构体和函数的声明都在其中。rnn\_data.h中定义了三层神经网络，分别对应实验中RNN网络里的全连接层、GRU层和softmax层。rnn\_data.c里列出了三层的具体网络参数。在rnn.c源文件里实现了样本数据的前向传播过程，最终调用compute\_rnn()函数即可。

rnn\_test为示例demo（包括fft-mfcc特征的完整实现和64fb-mfcc特征的示例展示，由宏编译命令区分），目前采用64fb-mfcc特征，使用方法为

usage: <audio feature path> <sampling begin point,如：

./rnn\_test 16 ./features\_64fb/4.txt

输出为该取样点开始一个语音序列的分类结果。

1. 其余功能性代码

path.c：根据.WAV后缀列出所有音频文件路径。

test.c：提取所有音频fft-mfcc特征并保存为txt。

data\_gen.py：将txt文件内特征集转化为numpy文件，供神经网络训练使用。