第二次大作业：LVCSR 系统搭建

521030910387 薛翔元

1. 系统搭建和模型训练

1.1. 数据处理及特征提取

1.1.1. 数据集下载与解压

首先，从官方网站下载数据集并解压，得到完整的 aishell 数据集，数据集目录下包含 data_aishell 与 resource_aishell 两个目录，其中 data_aishell 包含训练集、验证集和测试集的音频及其标注文件，resource_aishell 包含发音词典与音素集等资源文件。

```
local/download_and_untar.sh $data $data_url
    data_aishell || exit 1;
local/download_and_untar.sh $data $data_url
    resource_aishell || exit 1;
```

由于我们已经在本地自行准备好数据集，因此注释掉上述命令，直接使用自定义的数据集。

1.1.2. 发音词典与音素集

其次，官方 recipe 对发音词典与音素集进行归类整理，存入 data/local/dict 目录下。

```
# Lexicon Preparation,
local/aishell_prepare_dict.sh $data/
    resource_aishell || exit 1;
```

经过整理，得到的词典文件包括：

- lexicon.txt：数据集提供的发音词典
- lexiconp.txt：扩展发音词典，包含对原词典中词频信息的统计
- extra_questions.txt：包含重音音素及其声调的音素集
- nonsilence_phones.txt：非静音音素集
- silence_phones.txt：静音音素集
- optional_silence.txt：可选静音音素集

这些词典为训练流程提供了基本的发音信息，也有助于模型对重音音素的针对性训练。

1.1.3. 数据集划分与构造

然后，根据标注文本读取相应音频文件，存入 data/local 目录下，以便后续处理。

```
# Data Preparation,
local/aishell_data_prep.sh $data/data_aishell/wav
    $data/data_aishell/transcript || exit 1;
```

训练集、验证集和测试集目录下的文件包括：

- text：音频标注文本
- utt.list：音频文件列表
- wav.flist：音频路径列表
- wav.scp：音频文件与路径的对应关系
- spk2utt：说话人与音频文件的对应关系
- utt2spk：音频文件与说话人的对应关系

该步骤整理了数据集的基本信息，为后续使用提供了便利。

1.1.4. 音素词典构建

根据先前整理的发音词典与音素集，创建有限状态转录机（FST）并进行编译，存入 data/lang 目录下的 L.fst 文件中。此外，还要在部分音素串中添加消歧音素，破坏音素串之间的前缀关系，保证不存在路径歧义，相应的有限状态转录机在编译后存入 L_disambig.fst 文件中。

```
# Phone Sets, questions, L compilation
utils/prepare_lang.sh --position-dependent-phones
    false data/local/dict "<SPOKEN_NOISE>" data/
    local/lang data/lang || exit 1;
```

1.1.5. 语言模型构建

根据训练集中的音频标注文本，统计词频并构建语言模型，存入 data/local/lm 目录下。

```
# LM training
local/aishell_train_lms.sh || exit 1;
```

此处官方 recipe 主要使用一元模型（Unigram）和三元模型（Trigram）进行训练，通过统计词频，得到词典中每个词的条件概率分布。

1.1.6. 模型编译与结合

类似地，将语言模型转换为加权有限状态转录机，编译后存入 G.fst 文件中。进而将音素词典和语言模型结合，存入 LG.fst 文件中。

```
# G compilation, check LG composition
utils/format_lm.sh data/lang data/local/lm/3gram-mincount/lm_unpruned.gz data/local/dict/lexicon.txt data/lang_test || exit 1;
```

至此，音素词典和语言模型构建完成。在后续使用时，可以利用 OpenFST 工具对其进行快速解码。

1.1.7. MFCC 特征提取

最后，在训练集、验证集和测试集上分别提取梅尔频率倒谱系数（MFCC）特征，存入 mfcc 目录下，其中.ark 后缀代表二进制存档文件，.scp 后缀代表二进制存档文件的索引文件。

```
# Now make MFCC plus pitch features.
# mfccdir should be some place with a largish disk
   where you want to store MFCC features.
mfccdir=mfcc
for x in train dev test; do
  steps/make_mfcc_pitch.sh --cmd "$train_cmd" --nj 10 data/$x exp/make_mfcc/$x $mfccdir || exit 1;
  steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir || exit 1;
  utils/fix_data_dir.sh data/$x || exit 1;
done
```

此外，官方 recipe 对丢失数据进行了修复，剔除了错误的音频文件。至此，数据处理及特征提取全部完成，此后模型的训练都是基于此处提取的 MFCC 特征进行。

1.2. 模型训练

1.2.1. 用 Kaldi 训练 GMM-HMM 模型

单音子模型 官方 recipe 选择训练一个简易的单音子模型，作为流程中首个语音识别模型。

```
# Train a monophone model on delta features.
steps/train_mono.sh --cmd "$train_cmd" --nj 10 data/train data/lang exp/mono || exit 1;
```

在训练过程中，首先初始化高斯混合模型（GMM），然后使用期望最大化（EM）算法迭代优化模型参数，每隔一定迭代次数，重新进行对齐操作，直至收敛，训练得到的模型存入 exp/mono 目录下的.mdl 文件中。然后，根据模型构建单音子解码图，将音素词典、语言模型、上下文相关性和声学模型结合，存入 HCLG.fst 文件中。

```
# Decode with the monophone model.
utils/mkgraph.sh data/lang_test exp/mono exp/mono/graph || exit 1;
```

最后，在验证集和测试集上进行解码，产生词图并分析结果，得到词错误率和字错误率。

```
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 exp/mono/graph data/dev exp/mono/decode_dev
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 exp/mono/graph data/test exp/mono/decode_test
```

在下一步训练开始之前，官方 recipe 利用训练好的单音子模型对训练数据进行 Viterbi 强制对齐。

```
# Get alignments from monophone system.
steps/align_si.sh --cmd "$train_cmd" --nj 10 data/train data/lang exp/mono exp/mono_ali || exit 1;
```

至此，单音子模型的训练全部完成，后续模型的训练与此具有基本相同的流程，故不再赘述。

三音子模型 相比于单音子模型，三音子模型引入了音素的上下文相关性，可以更好地建模语音信号。在原有的 13 维 MFCC 静态特征的基础上，引入一阶差分和二阶差分，得到 39 维的动态特征，作为三音子模型的训练输入。

```
steps/train_deltas.sh --cmd "$train_cmd" 2500 20000 data/train data/lang exp/mono_ali exp/tri1 || exit 1;
```

在训练三音子模型时，官方 recipe 利用决策树对三音素组合进行聚类，以减少模型参数规模，提高模型的泛化能力，随后初始化 GMM 参数，同样使用 EM 算法迭代训练模型。在训练完成后，利用模型进行解码测试和数据对齐。

值得注意的是，官方 recipe 对基础三音子模型进行了两次训练。一方面，考虑到单音子模型

数据对齐的质量较低，利用三音子模型进行对齐操作可以有效提高模型表现。另一方面，模型参数量增长过快容易造成过拟合现象，多次训练可以有效缓解这一现象，从而提高模型的泛化能力。在第二次训练完成后，同样进行解码测试和数据对齐，以便后续模型的训练。

线性判别分析与最大似然线性变换 线性判别分析（LDA）是一种经典的有监督数据降维算法，它通过最小化类内距离、最大化类间距离的方式，将高维数据映射到低维空间。在 LDA 算法中，定义类内散度矩阵和类间散度矩阵为

$$S_{\mathrm{w}} = \sum_{i=1}^{C} \sum_{x \in X_i} (\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\top}$$

$$S_{\mathrm{b}} = \sum_{i=1}^{C} N_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^{\top}$$

其中 C 为类别个数，优化的目标函数为

$$J(\boldsymbol{W}) = \prod_{i=1}^{d} \frac{\boldsymbol{w}_i^T S_b \boldsymbol{w}_i}{\boldsymbol{w}_i^T S_w \boldsymbol{w}_i}$$

利用 LDA 算法对 MFCC 特征进行降维，可以有效提高模型的泛化能力。考虑到对角化协方差矩阵会造成似然度损失，官方 `recipe` 进一步使用最大似然线性变换（MLLT）算法得到更好的特征表示，从而提高模型的性能。

```
steps/train_lda_mllt.sh --cmd "$train_cmd" 2500
    20000 data/train data/lang exp/tri2_ali exp/
    tri3a || exit 1;
```

特征变换完成后，用同样的方式训练增强三音子模型，并利用该模型进行解码测试。

说话人自适应训练 由于语料库中的音频来源于不同的说话人，其语速、语调、音色、口音等特征细节均存在差异，这为语音的建模带来了一定的困难。语音信号总是包含说话人和语义两部分信息，我们可以认为这两部分信息相互独立，因此将说话人信息从语音信号中分离出来，使得特征表示只与语义相关，可以有效提高模型的泛化能力。官方 `recipe` 采用特征空间最大似然线性变换（fMLLR）算法，实现说话人自适应训练（SAT）。

```
# align tri3a with fMLLR
steps/align_fmllr.sh --cmd "$train_cmd" --nj 10
    data/train data/lang exp/tri3a exp/tri3a_ali
    || exit 1;
# Train the third triphone pass model tri4a on LDA
    +MLLT+SAT features.
# From now on, we start building a more serious
    system with Speaker Adaptive Training (SAT).
steps/train_sat.sh --cmd "$train_cmd" 2500 20000
    data/train data/lang exp/tri3a_ali exp/tri4a
    || exit 1;
```

与前文中的方法类似，在特征变换后训练增强三音子模型。值得注意的是，利用 SAT 模型进行解码测试时，需要首先使用未经变换的特征进行解码，随后利用对齐结果对特征进行 fMLLR 变换，最后重新进行解码测试。

```
steps/align_fmllr.sh --cmd "$train_cmd" --nj 10
    data/train data/lang exp/tri4a exp/tri4a_ali
```

参数规模提升 官方 `recipe` 在上述模型的基础上进一步提高参数规模。具体而言，结合 LDA、MLLT、fMLLR 三种算法，构建相同的 SAT 模型，但决策树叶节点数量从 2500 增加到 3500，GMM 高斯分量个数从 20000 增加到 100000，大幅度增加了模型的参数量。理论上说，更大的参数规模意味着更强的拟合能力，可以有效提高模型表现，但同时也会带来更多的计算开销。

1.2.2. 【可选 *A*】基于深度神经网络的模型

我们使用官方 `recipe` 提供的脚本训练基于深度神经网络（DNN）的语音识别模型。相比于前文中使用的模型，我们不再采用 GMM 建模状态的发射概率，而是直接使用 DNN 建模发射概率，形成 DNN-HMM 模型，因此原先的音素词典、语言模型、上下文相关性和声学模型仍然可以使用。

```
# nnet3
local/nnet3/run_tdnn.sh
```

在 `nnet3` 的训练过程中，首先对已有音频进行数据增强，例如随机改变语音的语速和音量，从而增强训练数据的多样性，更好地服务于 DNN 的训练。接着，对增强后的音频重新进行归一化和对齐操作，提取 40 维的高分辨率 MFCC 特征，并利用 LDA 算法去除数据之间的相关性。此外，官方 `recipe` 使用四分之一的数据训练通用背景模

型（UBM），进而训练说话人矢量因子（I-Vector）提取器，提取所有音频的 I-Vector 特征，与 MFCC 特征拼接后共同作为模型的输入。然后，构建一个包含 6 个隐藏层的全连接网络进行训练。最后，利用训练好的模型进行解码测试。

值得注意的是，DNN-HMM 模型的训练计算量大，需要使用 GPU 进行加速。实验表明，相比使用 CPU 训练，使用 GPU 训练可以带来 10 倍以上的速度提升，而且具有更高的运算精度和稳定的训练过程。此外，由于我们只能使用单张 GPU，必须将训练进程数全部设置为 1，否则会因为 GPU 数量不足导致训练失败。

1.3. 实验结果

1.3.1. GMM-HMM 模型结果

利用官方 `recipe` 中的命令可以快速获取不同阶段 GMM-HMM 模型的评分结果。

```
# getting results (see RESULTS file)
for x in exp/*/decode_test; do [ -d $x ] && grep
    WER $x/cer_* | utils/best_wer.sh; done 2>/dev
    /null
```

评分结果对应的相对路径如下表所示。各个阶段

表 1: *GMM-HMM* 模型评分结果路径

Stage	Path
mono	exp/mono/decode_test/wer_9_0.0 exp/mono/decode_test/cer_9_0.0
tri1	exp/tri1/decode_test/wer_15_0.5 exp/tri1/decode_test/cer_13_0.5
tri2	exp/tri2/decode_test/wer_14_0.5 exp/tri2/decode_test/cer_14_0.5
tri3a	exp/tri3a/decode_test/wer_14_0.5 exp/tri3a/decode_test/cer_14_0.5
tri4a	exp/tri4a/decode_test/wer_15_0.5 exp/tri4a/decode_test/cer_15_0.5
tri5a	exp/tri5a/decode_test/wer_17_1.0 exp/tri5a/decode_test/cer_17_1.0

GMM-HMM 模型的评分结果对比如下表所示。可以看到，模型的评分结果随着阶段不断提升，在 `tri4a` 阶段达到最佳效果。其中，使用三音子模型

表 2: *GMM-HMM* 模型评分结果对比

Stage	%WER	%CER
mono	58.31	45.40
tri1	44.05	28.71
tri2	44.01	28.69
tri3a	41.13	25.70
tri4a	**36.22**	**20.66**
tri5a	37.82	22.22

以及引入 SAT 方法对模型性能的提升尤为明显，可见特征提取与特征变换在语音识别系统中的重要性。此外我们注意到，在提高三音子模型的参数规模后，评分结果反而有所下降，推测训练数据过少导致了过拟合的发生，因此训练大型语音识别系统时，语料的数量也至关重要。

1.3.2. 【可选 A】基于深度神经网络模型的结果

同样利用官方 `recipe` 中的命令获取 DNN-HMM 模型的评分结果，相对路径如下表所示。与

表 3: *DNN-HMM* 模型评分结果路径

Stage	Path
nnet3	exp/nnet3/tdnn_sp/decode_test/wer_17_0.5 exp/nnet3/tdnn_sp/decode_test/cer_16_1.0

GMM-HMM 模型的评分结果对比如下表所示。为了简洁起见，我们只将 `nnet3` 与 GMM-HMM 模型中表现最好的 `tri4a` 阶段的评分结果进行对比。可以看到，基于深度神经网络的模型相比于

表 4: *DNN-HMM* 模型评分结果对比

Stage	%WER	%CER
tri4a	36.22	20.66
nnet3	**33.20**	**17.53**

GMM-HMM 模型在 `tri4a` 阶段的评分又有明显的提升。DNN 的使用为语音识别系统带来了更大的参数量，却并不像 `tri5a` 阶段一样造成模型的过拟合。可见 DNN-HMM 具有更强的特征挖掘和表征能力，在语音识别任务中具有天然的优越性。

2. 【可选 B】语料数量对模型性能的影响

2.1. 修改方法

我们可以发现，原先 data/local 目录下的 text 文件仅包含了训练集中少部分的语料，为了使用数据集中全部的语料训练语音识别系统，我们需要在 recipe 完成数据集划分和构造以后，使用 transcript 目录下的 train_large.txt 文件替换原有 text 文件，为全部语料建立索引。

```
$ rm data/local/text
$ mv train_large.txt data/local/text
```

此时训练集语料范围已经发生改变，我们需要从构建音素词典和语言模型开始，重新提取 MFCC 特征并训练 GMM-HMM 模型。

2.2. 性能对比

增加语料数量前后，不同阶段 GMM-HMM 模型的评分结果对比如下表所示。可以看到，增加

表 5: 不同语言模型评分结果对比

Stage	Small		Large	
	%WER	%CER	%WER	%CER
mono	58.31	45.40	45.83	36.44
tri1	44.05	28.71	31.94	21.91
tri2	44.01	28.69	32.11	22.06
tri3a	41.13	25.70	29.30	19.20
tri4a	36.22	20.66	**25.09**	**15.13**
tri5a	37.82	22.22	26.96	16.71

语料数量后，GMM-HMM 模型评分结果在各个阶段的性能都有明显的提升。评分结果随着阶段具有相似的变化趋势，且同样在 tri4a 阶段取得了最佳性能，在 tri5a 阶段评分结果出现了轻微的下降，可见增加语料数量不能解决模型的过拟合问题，在传统分类算法中参数量也并非越大越好。

3. 【可选 C】优化系统

3.1. 方法

在官方 recipe 中还有 chain 模型未被使用，它是目前 Kaldi 中效果最好的基于深度神经网络的语音识别模型，我们尝试使用 chain 模型替换 nnet3 模型，以获得更好的性能表现。

```
# chain
local/chain/run_tdnn.sh
```

相比于 nnet 模型，chain 模型使用双音子（bi-phone）声学建模单元和经过简化的 HMM 拓扑结构（不对转移概率进行学习），从头开始训练神经网络，而不依赖基于交叉熵的预训练神经网络。

从算法思想上而言，chain 模型摒弃了传统的最大似然目标函数，使用性能更好的区分性目标函数，目标函数大致可以写成

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\log \frac{p(\boldsymbol{x};\theta)}{\sum\limits_{\boldsymbol{x}\in S} p(\boldsymbol{x};\theta)}\right]$$

为了优化这一目标，一方面要提高分子的得分，另一方面要压制分母的得分，从而使得正确路径在状态空间中的分数优势更加突出。

阅读官方 recipe 中的 run_tdnn.sh 脚本，我们可以看到 chain 模型的训练过程。首先与 nnet3 一样，训练 I-Vector 提取模型并提取所有音频的 I-Vector 特征，实验表明使用 I-Vector 特征可以有效提高 DNN-HMM 模型的识别效果。接着生成 chain 模型所需的语言模型，对数据进行强制对齐，并针对简化的 HMM 拓扑结构重新构建决策树。然后利用 nnet3 构建类似的全连接神经网络，调用训练脚本进行训练。最后使用训练完成的模型进行解码测试。在运行训练脚本前，我们需要删除原有的 feats.scp 文件，强制要求脚本重新训练 DNN-HMM 模型。

3.2. 性能对比

表 6: 语音识别系统评分结果对比

Stage	Small		Large	
	%WER	%CER	%WER	%CER
tri4a	36.22	20.66	25.09	15.13
nnet3	33.20	17.53	**22.47**	**12.66**
chain	33.45	17.99	22.65	13.04

我们在大小两种语料数量下分别训练 tri4a、

nnet3、chain 三个模型，评分结果如上表所示。

可以看到，使用 chain 模型代替 nnet3 模型并没有对评分结果带来提升，反而出现了轻微的下降。而官方给出的参考结果中，chain 模型的评分显著高于 nnet3 模型，可见由于 chain 模型具有更大的网络模型和参数规模，我们使用 10 小时的语料训练 chain 模型并不能充分发挥其优势，反而容易出现过拟合现象。但是我们必须指出，chain 模型支持在线解码，而且相比于 nnet3 模型具有更快的解码速度，符合实际生产需求，这是 nnet3 模型无法比拟的优势。

4. 最佳性能

CER=12.66%，WER=22.47%

5. 参考文献

[1] 鸟哥. 鸟哥的 Linux 私房菜：基础学习篇（第四版）. 人民邮电出版社, 2012.

[2] 周志华. 机器学习. 清华大学出版社, 2016.

[3] 黄艳, 陈明. 语音识别基础. 清华大学出版社, 2019.

[4] 陈果果, 都家宇, 那兴宇, 张俊博. Kaldi 语音识别实战. 电子工业出版社, 2020.