Project 3：LVCSR 系统搭建

520030910155 邱一航

1. Baseline (70 分)

1.0. 安装环境及脚本运行说明

笔者在 Windows 提供的 Linux 子系统即 WSL（Windows Subsystem for Linux）环境下进行实验。

所安装的 WSL 版本为 2.0，笔者在该子系统中安装 Ubuntu 操作系统，Ubuntu 版本为 Ubuntu 20.04。WSL 环境下已预先安装了 CUDA 10.1。

该部分只要求训练得到 SAT+GMM-HMM。因此原脚本中涉及后续 DNN 训练的部分应当被注释，即注释以下两行（Line 142, 145）。

```
142 local/nnet3/run_tdnn.sh
145 local/chain/run_tdnn.sh
```

1.1. 数据处理及特征提取 (10 分)

官方 recipe 的数据处理和特征提取步骤如下：

1. 词典准备。

调用 `local/aishell_prepare_dict.sh`，根据提供的词库（`data/resource_aishell/lexicon.txt`）准备词典。

2. 数据准备。

调用 `local/aishell_data_prep.sh`，根据提供的文本标注文件 `data/data_aishell/transcript/aishell_transcript_v0.8.txt` 在 train，test，dev 数据集上准备文本标注及 speaker 与语音编号对应关系。

3. 词典生成。

调用 `utils/prepare_lang.sh`，根据准备好的词典文件，以标准格式构建词典的 FST 脚本。

4. 语言模型训练。

调用 `local/aishell_train_lms.sh`，进行语言模型训练。

5. 特征提取。

用 `steps/make_mfcc_pitch.sh`, `steps/compute_cmvn_stats.sh`, `utils/fix_data_dir.sh` 提取 MFCC（Mel 频率倒谱系数）和 Pitch（音高）特征。其中第二个脚本用于归一化，第三个脚本检查 `wav.scp` 文件格式的正确性。

该过程曾出现以下错误信息：

```
1 local/aishell_train_lms.sh: train_lm.sh is not
    found. That might mean it's not installed.
```

解决：根据提示使用 `tools/extras/install_kaldi_lm.sh` 安装 `train_lm.sh` 即可。

1.2. 模型训练 (50 分)

官方 recipe 中，SAT+GMM-HMM 模型训练主要分为两个阶段：

1.2.1. 单音素训练

利用 MFCC 和 pitch 特征进行单音素的 GMM-HMM 模型训练，利用最大似然法估计参数。使用 EM（期望最大化）算法，默认迭代 40 次（可以在 `steps/train_mono.sh` 脚本中修改迭代次数），每次迭代增加一定的高斯数并重新进行参数估计。

特别地，脚本通过 *realign_iters* 变量指定了一些需要执行对齐的迭代次数。

训练完成后，生成解码图并利用该模型在数据集上解码，计算 WER 和 CER。之后使用 Verrtibi 算法对解码后的特征进行对齐，以便之后的训练使用。下面简称该过程为“解码、评估、对齐”。

1.2.2. Triphone（三音素）训练 1-2: train_deltas，即一阶与二阶差分

第一阶段和第二阶段均调用 `steps/train_deltas.sh`，构建以 GMM 为节点的决策树。训练

过程中也使用了上一次训练后对齐的特征。

每一阶段的训练完成后，都进行了一次“解码、评估、对齐”。

这两个阶段训练调用的脚本 `steps/train_deltas.sh` 的主要流程如下：

- 首先对特征进行处理（CMVN，倒谱均值方差归一化方法）和差分。即对特征的倒谱值进行处理

$$\hat{x}_i = \frac{x_i - \texttt{mean}(x)}{\sqrt{\texttt{var}(x)}}$$

  后，再进行差分操作。

  这一处理能提高系统的泛化能力，提升了模型的鲁棒性。

- **stage -3:** 利用前一次训练得到的对齐结果，统计收集所有决策树的参数，结果在 treeacc 文件中。

- **stage -2:** 使用 k-means 对三音素进行聚类，根据聚类结果和上一步统计完毕的决策树，利用前一次训练得到的模型（tri1：单因素模型，tri2：tri1 得到的三音素决策树），构建基于三音素的决策树并初始化。决策树的每个节点都是一个 GMM 模型。

- **stage -1:** 将上一步的对齐（tri1：单因素表示的对齐，tri2：tri1 得到的三音素决策树节点表示的对齐）转换为用上一步构建的三音素决策树的节点表示的对齐。

- **stage 0:** 构建三音素决策树的训练图。

- 此后类似 **1.2.1** 中单音素 GMM 的训练，使用 EM 算法，通过最大化似然来估计每个节点处 GMM 的具体参数。

  同样，也通过 *realign_iters* 变量指定了需要对齐的迭代次数。

第一阶段和第二阶段分别针对 MFCC 特征经 CMVN 后一阶和二阶差分特征进行建模，完成了两个模型（三音素决策树）的建构和训练。

1.2.3. Triphone（三音素）训练 *3*: *LDA+MLLT*

第三阶段生成并训练的模型为：对 splice（帧链接）处理后的 MFCC 和 pitch 特征进行 LDA 降维，降维后再做 MLLT 变换，通过以 GMM-HMM 为节点的决策树得到结果。

该阶段调用的脚本 `steps/train_lda_mllt.sh` 的主要流程如下：

- 对 MFCC 和 pitch 特征进行 CMVN 处理，然后使用 splice 进行对处理后的特征进行帧链接。拼接后的特征相当于三音素的“特征”。

- **stage -5:** 使用 LDA（线性判别分析）对拼接后的特征进行降维和去相关（默认降维至 40 维）。LDA 的具体流程在机器学习课程 [1] 中已有详细讲解，此处只给出方法的核心。

  1) 构建类内散射矩阵：

$$S_w = \sum_c \sum_{\mathbf{x}\in\mathbf{H_c}} (\mathbf{x}-\boldsymbol{\mu}_\mathbf{c})(\mathbf{x}-\boldsymbol{\mu}_\mathbf{c})^\mathbf{T}$$

  其中 c 为类别，H_c 包括了第 c 类的全部数据点，$\boldsymbol{\mu}_c$ 为第 c 类全部数据点均值（中心）。

  2) 通过计算全局散度矩阵

$$S_T = \sum_{\mathbf{x}} (\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\mathbf{T} = \mathbf{S_b} + \mathbf{S_w}$$

  计算类间散度矩阵 S_b。其中 $\boldsymbol{\mu}$ 为所有数据点的均值。

  3) LDA 的目标是最大化类间距离同时最小化类内距离，即

$$\max J = \frac{\boldsymbol{w}^T S_w \boldsymbol{w}}{\boldsymbol{w}^T S_b \boldsymbol{w}} \implies S_w^{-1} S_b \boldsymbol{w} = J\boldsymbol{w}$$

  其中 $\boldsymbol{w}$ 为降维后的超平面的一个基，即将数据点 $\boldsymbol{x}$ 投影到 $\boldsymbol{w}^T\mathbf{x}$。该目标可转化为上式右侧的特征值分解问题。多个 $\boldsymbol{w}$ 可以组成一个降维变换矩阵 $\mathbf{W}$，很显然各个 $\boldsymbol{w}$ 都是特征向量，彼此不相关。

- **stage -4 - stage 0** 对三音素聚类，统计决策树参数，构建以 GMM 为节点的决策树并

初始化。

- 用 LDA 降维和去相关后，类似 `train_deltas.sh`，通过 EM 算法多次迭代最大似然估计得到“最优”的 MLLT（最大似然线性变换）矩阵 [3] 和以 GMM-HMM 为节点的三音素决策树模型。这一训练过程类似 **1.2.2** 中的训练，不再赘述。

  由于特征维度有 40，GMM 所需要的协方差矩阵参数量太大，因此使用 MLLT 矩阵，即通过相对较小的对角阵 $\boldsymbol{\Sigma}_{diag}$ 和 semi-tied covariance matrices $\boldsymbol{H}$ 来近似估计 GMM 中的协方差矩阵。数学表达式如下：

  $$\boldsymbol{\Sigma} = \boldsymbol{H}\boldsymbol{\Sigma}_{diag}\boldsymbol{H}^T$$

  为了方便 EM 算法的使用（每一次迭代都会使用上一次的迭代结果），同时对特征和 GMM 做这一处理，即

  $$\begin{cases} \hat{\boldsymbol{\mu}} = \boldsymbol{H}\boldsymbol{\mu} \\ \hat{\boldsymbol{\Sigma}} = \boldsymbol{H}\boldsymbol{\Sigma}\boldsymbol{H}^T \end{cases}$$

  因此，MLLT 可视为特征空间的变换矩阵。

训练完成后，需要为后续 SAT 训练准备特征。这一步的目的是将原本数据集中的特征（是与说话人“无关”的特征）根据特定的说话人进行变换（对特征的均值进行一定变换处理，即

$$\boldsymbol{\mu}_{SA} = \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}$$

（$\boldsymbol{\mu}$ 是原本的均值，$\boldsymbol{b}$ 是偏移量，$\boldsymbol{A}$ 是特征空间中的变换即一个矩阵。）

这一变换生成了说话人自适应的音频。对变换 $\boldsymbol{A}$ 的求解使用了 fMLLR，该方法能在训练得到的模型上能取得最大似然。[3]

`steps/align_fmllr.sh` 使用 EM 算法迭代求解 fMLLR，默认迭代次数是两次。

1.2.4. Triphone（三音素）训练 4-5: SAT

tri4a 和 tri5a 都训练了 SAT（说话人自适应训练）。该部分主要是根据说话人声音的特征（如性别，平均音高等）对音频特征进行一定处理，根据不同的“说话人”进行不同的处理。

某种意义上这一操作可以被视为针对“说话人”的归一化，将原本不同的声线根据“说话人”自适应得到的变换，全部转化为“标准声线”下的音频进行后续识别处理。tri4a 和 tri5a 所训练就是说话人自适应，即不同的声线该如何转化为“标准声线”。

tri4a 训练的 SAT 模型相对较小，tri5a 训练的 SAT 模型参数量更大，高斯数也更多。

其训练流程类似于前两阶段的训练，也是训练一个节点为 GMM 的三音素决策树。唯一的区别是输入的特征都经过了 fMLLR 处理。因此这里不再赘述。

两次训练完成后都需要经过 fMLLR 变换，以便进行下一次的 SAT 训练。

1.3. 实验结果 (10 分)

不同阶段训练得到的模型在 test 测试集和 dev 开发集上的解码结果保存在 `exp/[1]/decode_[2]/log/decode.[3].log` 文件中。其中，[1] 是训练阶段的名称（mono, tri1, tri2, tri3a, tri4a, tri5a），[2] 是数据集名称（test, dev），[3] 是数值（1-6）.

不同阶段训练得到的系统对应的评分结果相对目录如表1：（均为 test 数据集上的结果）

阶段	相对路径
mono	`exp/mono/decode_test/scoring_kaldi`
tri1	`exp/tri1/decode_test/scoring_kaldi`
tri2	`exp/tri2/decode_test/scoring_kaldi`
tri3a	`exp/tri3a/decode_test/scoring_kaldi`
tri4a	`exp/tri4a/decode_test/scoring_kaldi`
tri5a	`exp/tri5a/decode_test/scoring_kaldi`
tri4a-si	`exp/tri4a/decode_test.si/scoring_kaldi`
tri5a-si	`exp/tri5a/decode_test.si/scoring_kaldi`

表 1: *Baseline* 不同阶段评分结果相对路径

评分结果为上述路径下的 `best_wer` 和 `best_cer` 文件，分别为 WER 和 CER。

不同阶段训练结果对应的性能对比如表4：
（以下评分均为 test 测试集上的评分结果）

阶段	WER(%)	CER(%)
mono	57.98	45.30
tri1	43.64	28.50
tri2	43.77	28.56
tri3a	41.23	25.81
tri4a	**36.33**	**20.72**
tri5a	38.01	22.38
tri4a-si	42.80	27.56
tri5a-si	44.20	28.81

表 2: *Baseline* 不同阶段训练结果对比

显然，tri4a 阶段训练后的系统性能最佳。

tri5a 的评分结果不如 tri4a 的可能原因是经过 tri5a 训练后 SAT 模型过拟合，导致泛化能力不如 tri4a 训练后的系统，性能下降。

特别地，tri4a-si 和 tri5a-si 可以看成消融实验，证明对不同说话人进行特定的 fMMLR 变换对降低 WER 和 CER、提升系统性能有重要作用。

2. 语料数量对模型性能的影响 (15 分)

2.1. 修改方法

为使用更大的训练集语料（原本的 AIShell-1 训练集语料），需对语言模型训练脚本做一定修改。修改 `local/aishell_train_lms.sh` 的第 7 行

```
7 text=data/local/train/text
```

为

```
7 text=data/data_aishell/transcript/train_large.txt
```

之前的词典准备、数据准备、词典构建的结果依然可以继续沿用（没有发生变化），不必重新执行 `local/aishell_prepare_dict.sh`，`local/aishell_data_prep.sh`，`utils/prepare_lang.sh`。数据准备和特征提取部分只需要重新执行语言模型的训练即可。

更换语料后，只需要重新执行**语言模型训练**步骤及模型训练部分的代码即可。

2.2. 性能对比

换用大语料前后，SAT+GMM-HMM 系统的性能对比如表3。

（仅给出 test 测试集上的最优结果）

阶段	baseline(小语料)		大语料	
	WER%	CER%	WER%	CER%
mono	57.98	45.30	45.37	36.01
tri1	43.64	28.50	32.18	21.98
tri2	43.77	28.56	32.07	21.98
tri3a	41.23	25.81	29.62	19.50
tri4a	**36.33**	**20.72**	**25.13**	**15.18**
tri5a	38.01	22.38	26.98	16.81

表 3: 使用新旧语言模型的最佳系统评分结果对比

由表3可见，使用大语料后模型的 WER 和 CER 均有明显的下降，系统的性能进一步提高。相对而言，使用大语料后 WER 的下降比 CER 更加明显。

实际上，基于大语料训练得到的语言模型在训练过程 tri3a 阶段的 WER 和 CER（分别为 29.62% 和 19.50%）就已经优于 baseline 的最佳性能 (tri4a，36.33% 和 20.72%)。

由此可见，使用大语料训练语言模型能很大程度地提升系统的性能，而且对 WER 影响相对更大。这一点也符合直觉。

进一步对比两者最佳性能系统的解码结果。第一个代码块显示的是 baseline（小语料）的结果，第二个代码块显示的是大语料训练后的结果，第三个代码块为标准的文本。

```
1 BAC009S0904W0121 为了 解救 额度 化 问题
2 BAC009S0904W0124 公积金 的 身影 贷款 直接 的 的 细
    察 王
3 BAC009S0904W0128 广州 和 深圳 分 并未 五万 元 和
    七万 元
```

```
1 BAC009S0904W0121 为了 解决 入 额度 化 问题
2 BAC009S0904W0124 公积金 的 商业 贷款 直接 的 弟媳
    岔路
3 BAC009S0904W0128 广州 和 深圳 分 别 为 五万 元 和
    七万 元
```

```
1 BAC009S0904W0121 为了 解决 额度 荒 的 问题
2 BAC009S0904W0124 公积金 贷款 与 商业 贷款 之间 的
    利息 差额
3 BAC009S0904W0128 广州 和 深圳 分别 为 五万 元和 七
    万 元
```

大语料训练的系统在解码结果中相对更细致，如第一行的结果将“救”字错误地拆解为两个字

("决入")；但同时在准确率方面也更高，如第二行中"商业"的正确识别（baseline 识别为"身影"）和第三行中"分别为"的正确识别（baseline 识别为"分并未"）。

3. 训练 DNN-HMM 系统 (15 分)

3.0. 遇到的问题及解决

调用 `local/nnet3/run_tdnn.sh` 脚本时，出现以下错误信息：

```
Command 'python' not found, but can be installed with:
apt install python3
apt install python
apt install python-minimal
```

实际上，python 已经成功安装，但 python 指令默认调用 python 2.x 版本。

解决方案：笔者找到了以下两种解决方案。

方案 1. 为 python3 创建化名 python，即输入以下指令。

```
alias python=python3
```

方案 2. 安装 python-is-python3。

3.1. 训练流程

DNN-HMM 系统基于 tri5a 训练完毕后经 fMMLR 对齐的特征继续训练。主要流程分为 ivector 的准备和提取，及 DNN 部分的构建与训练。

3.1.1. ivector 的准备和提取

1. 使用不同的速度，基于训练集数据生成 low-resolution speed-perturbed data；
2. 生成 3-way speed-perturbed 数据集 train_sp；
3. 在 train_sp 数据上提取 MFCC 和 pitch 特征，并使用 fMLLR 进行对齐；
4. 整合 tri5a 对齐后的特征，对原 train 数据集进行音量扰动，计算 MFCC 和 pitch 特征后，分别对同时具有 MFCC 与 pitch 的特征和只有 MFCC 的特征使用 CMVN 归一化；
5. 对开发集（dev）和测试集（test）也进行 4 中的处理；
6. 对扰动后的数据集进行 PCA 降维和去相关，得到一组数据子集用于训练 diagonal UBM（通用背景模型）；
7. 训练 ivector 提取器；
8. 在 train、dev、test 数据集上提取 ivector。

3.1.2. DNN 部分的构建与训练

1. 生成 DNN 的结构文件 network.xconfig，初始化神经网络。

   网络结构为 8 层，第一层为 fixed affine layer，第二至七层为 256 个神经元的全连接层，每一层后有 batch norm 层和 ReLU 层，第八层为输出层。

2. 训练神经网络。默认迭代训练 42 次，学习率自动调整。
3. 使用训练得到的 DNN-HMM 系统对测试集 test 进行解码和评估。

3.2. 性能对比

笔者基于大语料的语言模型继续训练 DNN-HMM 系统。因此只对比基于大语料的 GMM-HMM 系统和 DNN-HMM 系统的评分结果。

系统	WER(%)	CER(%)
GMM-HMM	25.13	15.18
DNN-HMM	22.89	13.42

表 4: *GMM-HMM 与 DNN-HMM 评分结果对比*

由上表可见，同样基于大语料训练得到的语言模型，使用 DNN-HMM 系统后的 WER 和 CER 均略低于 GMM-HMM 系统，性能相对更好。

进一步对比两者最佳性能系统的解码结果。其中 DNN-HMM 系统的解码结果位于 `exp/nnet3/new_dnn_sp/decode_test/log` 目录下。

下面第一个代码块显示的是大语料 GMM-HMM 的结果，第二个代码块显示的是大语料 DNN-HMM 的结果，第三个代码块是标准文本。

```
BAC009S0904W0121 为了 解决 入 额度 化 问题
BAC009S0904W0124 公积金 的 商业 贷款 直接 的 弟媳 岔路
BAC009S0904W0134 包括 仔仔 三 部委 发布 弘基 金 新政
```

```
BAC009S0904W0121 为了 给 就 额度 方 的 问题
BAC009S0904W0124 公积金 贷款 商业 贷款 之间 的 弟媳 差额
BAC009S0904W0134 包括 次 三 部委 发布 公积金 新政
```

```
BAC009S0904W0121 为了 解决 额度 荒 的 问题
BAC009S0904W0124 公积金 贷款 与 商业 贷款 之间 的 利息 差额
BAC009S0904W0134 包括 此次 三 部委 发布 公积金 新政
```

可以发现 DNN-HMM 系统识别字词相对而言更加准确，如成功识别了第二行的“之间”和“差额”，第三行的“次”和“公积金”。同时，对语速不恒定的音频（如第一行的音频 BAC009S0904W0121），DNN-HMM 能捕捉时间较短的字词，如第一行中成功识别出了在音频中持续时间很短的“的”字，第二行中成功识别持续时间较短的“贷款”（在 GMM-HMM 系统中识别为“的”）。

4. 优化系统 (20 分)

以下只是一些初步的想法，由于时间限制和资源限制，这些优化想法的具体实现并未完成。

1. RNN 更适合序列的识别问题，相对 DNN 而言应该更适合语音转文本问题，而且本身也一定程度实现了 HMM 的功能。

   因此，如果使用 RNN 替代 DNN（即搭建 RNN-HMM 系统），应该能进一步取得比较好的效果。

2. 语言模型的改进。

   实际上，语言模型也需要考虑上下文语境。因此如果使用 RNN 来建模语言模型，或许也能获得更准确的效果。

5. 最佳性能

CER=13.42%，WER=22.89%

6. 特别鸣谢

王崇华同学和**季弋琨**同学。

通过与上述两位同学的讨论，笔者完成了本文中对脚本代码的理解和解读，以及对 SAT 部分具体流程、功能的解释。

7. References

[1] AI2611，机器学习

[2] Phoneme 的相关概念以及 Triphone, https://zhuanlan.zhihu.com/p/322194937

[3] Feature and model-space transforms in Kaldi, http://kaldi-asr.org/doc/transform.html

[4] Context and chunk-size in the ”nnet3” setup, http://kaldi-asr.org/doc/dnn3_scripts_context.html