厦门大学
XIAMEN UNIVERSITY

Kaldi实践

洪青阳

Kaldi介绍

Kaldi是和HTK类似的一个开源的语音识别工具箱，底层基于C++编写，可以在Windows和Linux平台上编译。

[Kaldi特色]

（1）在C++代码级别整合了OpenFst库；

（2）支持基于BLAS、LAPACK、OpenBLAS和MKL的线性代数运算库；

（3）包含通用的语音识别算法、脚本和工程示例；

（4）底层算法的实现更可靠，经过大量有效测试，代码规范易理解，易修改；

（5）每个底层源命令功能简单，容易理解，命令之间支持管道衔接，工作流程分工明确，整个任务由上层脚本联合众多底层命令完成；

（6）支持众多扩展工具，如SRILM，Sph2pipe等。

[Kaldi声学模型]

支持标准的机器学习训练模型：

线性变换如：LDA HLDA, MLLT/STC；

说话人自适应：fMLLR, MLLR, SAT；

支持GMM, DNN, TDNN, Chain

Kaldi的安装

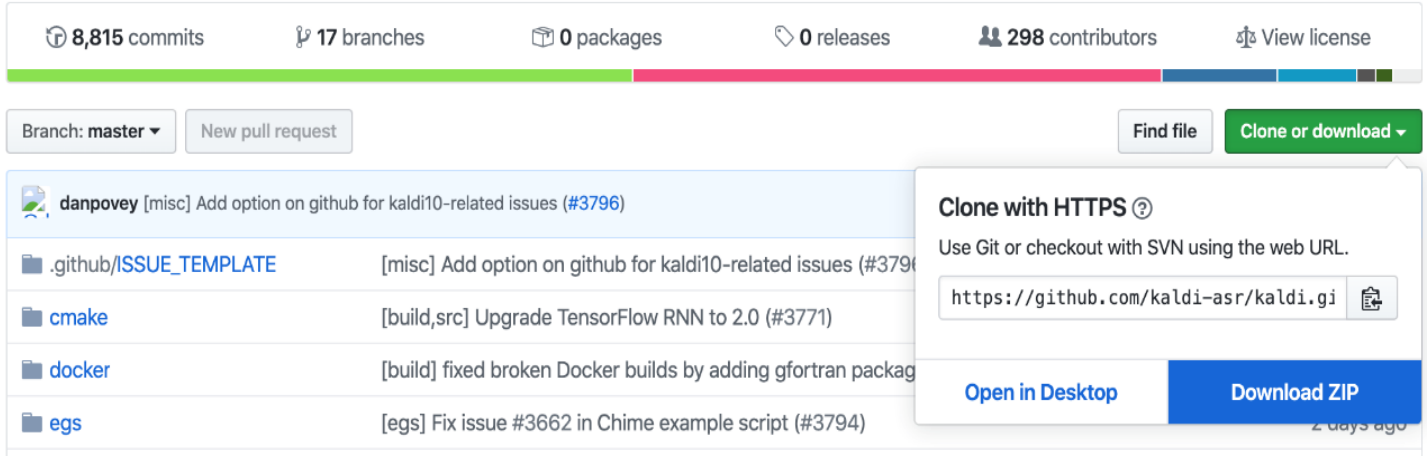
Kaldi安装在linux系统下，通过如下简单步骤：

1.获取源代码

方式一：直接在终端利用git命令从 Kaldi 的GitHub代码库克隆

[root@localhost work]# git clone https://github.com/kaldi-asr/kaldi.git kaldi --origin upstream

方式二：从Kaldi开源地址 https://github.com/kaldi-asr/kaldi 下载，获得源代码压缩包kaldi-master.zip：



Kaldi的安装

2.自主检查并安装依赖库 g++，zlib1g-dev，automake，autoconf，sox，gfortran和libtool 等。这些库是安装Kaldi的时候，Kaldi会调用到的，若无会出错。

```
[root@localhost work]# cd kaldi/tools
[root@localhost tools]# extras/check_dependencies.sh
extras/check_dependencies.sh: all OK.
```

3.编译

依赖检查通过后，使用多进程加速编译kaldi/tools：

```
[root@localhost tools]# make -j 4
```

kaldi/tools编译完成后，开始编译kaldi/src目录，在此之前，先执行配置脚本：

```
[root@localhost tools]# cd ../src/
[root@localhost src]# ./configure --shared
```

配置检查通过后，进行最后的编译：

```
[root@localhost src]# make depend -j 4
[root@localhost src]# make -j 4
```

4.通过这样的步骤后，如果没有出现错误，那么Kaldi系统就安装成功了。

Kaldi的安装

5.利用最简单的例子 yesno 进行测试

进入 [kaldi/egs/yesno/s5] 目录，[./run.sh] 以运行该例子，等待训练、解码和打分。

```
[root@localhost src]# cd ../egs/yesno/s5/[root@localhost s5]# sh run.sh......
%WER 0.00 [ 0 / 232, 0 ins, 0 del, 0 sub ] exp/mono0a/decode_test_yesno/wer_10_0.0
```

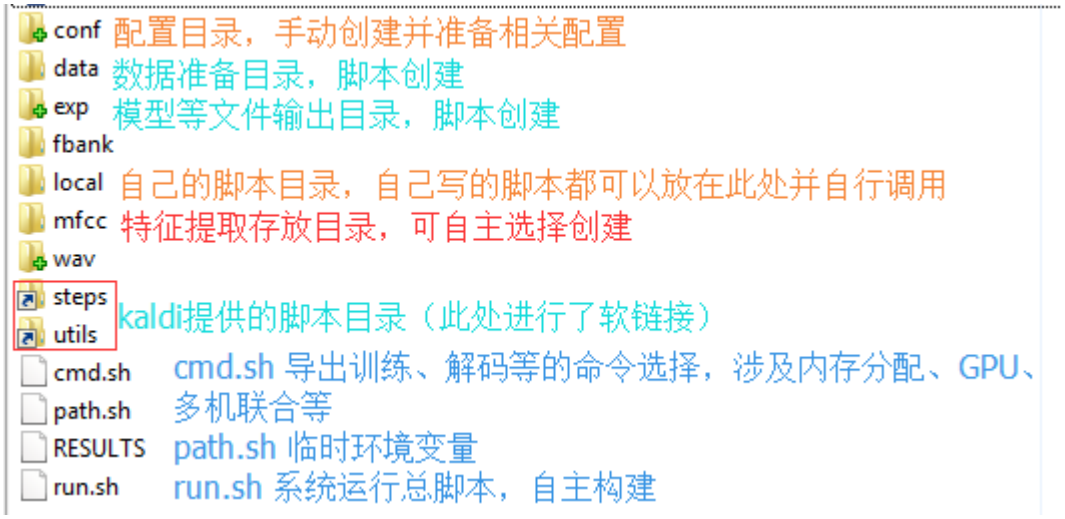
若运行成功，则最后会输出如上图所示的%WER指标。至此，Kaldi安装完毕。

Kaldi的训练系统搭建

【前言】Kaldi中模型训练解码等一整套系统框架的搭建，都是按照Kaldi中这些示例所示的标准进行的。除了需要按照标准自己构建的脚本以外，Kaldi本身提供了很多标准处理脚本。

1.目录结构 所有内容均在 kaldi/egs/myproject/s5/下，myproject 和s5命名随意，但一定需要，否则和环境变量导入代码不能相呼应。

如图所示，未提到的均为自主按需选择创建，仿照该目录结构即可搭建出最初的框架。



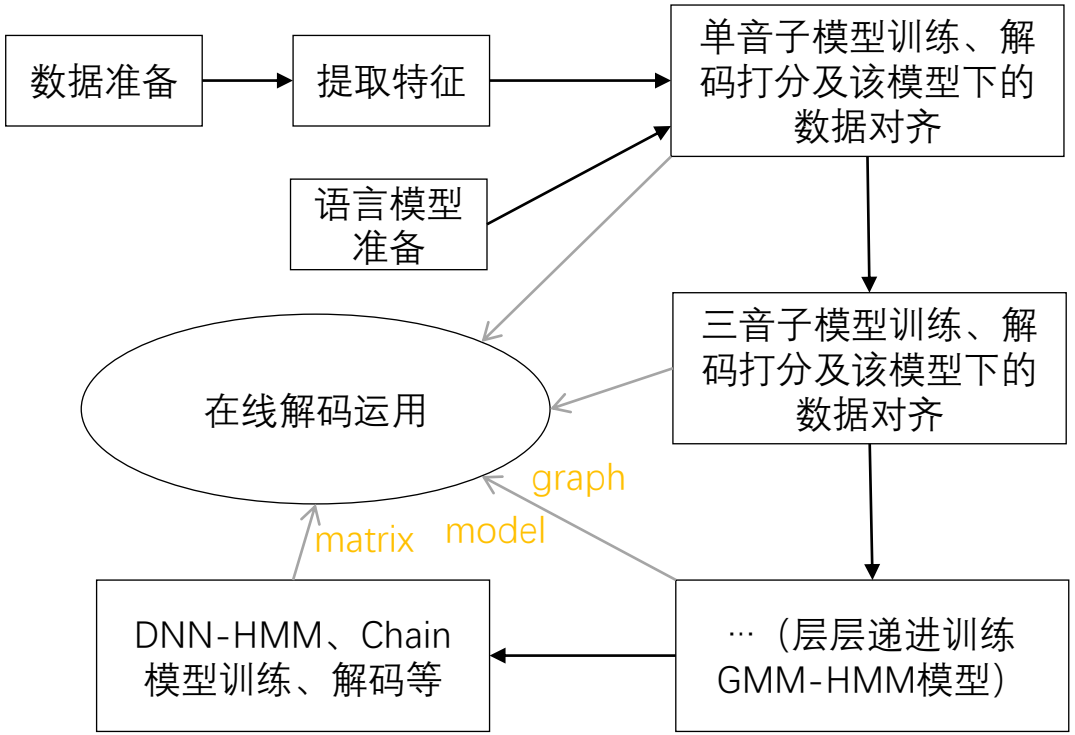
[steps][utils][cmd.sh][path.sh]可先从别的例子复制一份过来

[conf][local] [run.sh]手动创建

[其余]按需，相呼应自己写的脚本。

Kaldi系统构造过程

2.系统构造过程

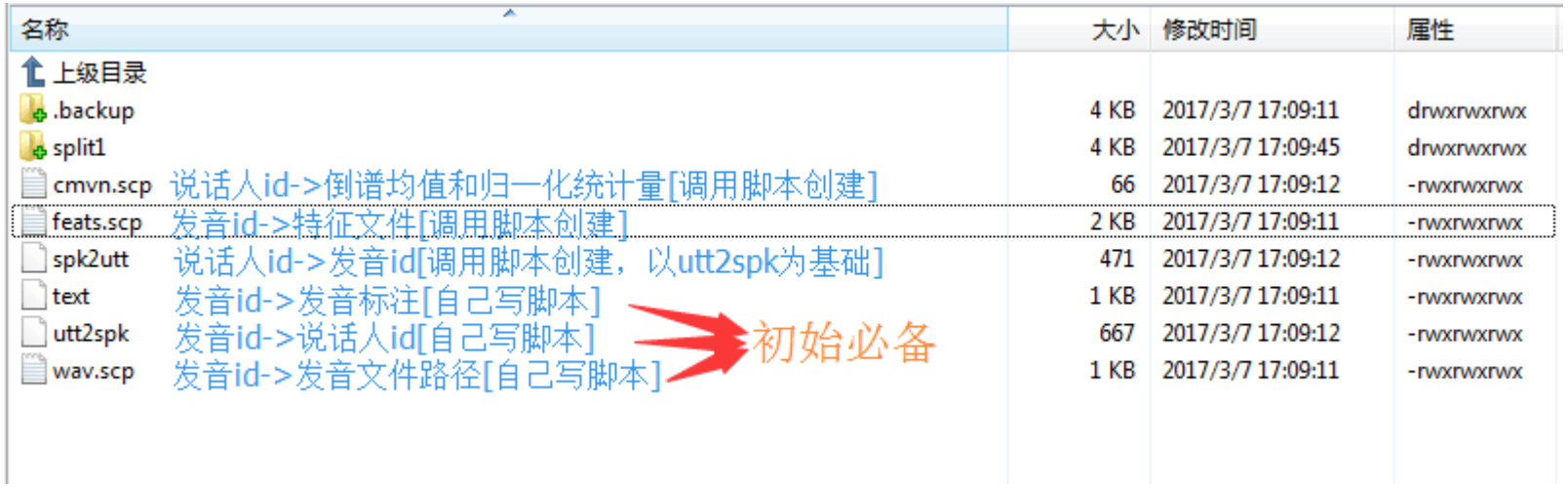


该流程与run.sh代码相呼应：准备工作需要自己构建脚本处理得到Kaldi所需的标准文件，训练解码等则调用Kaldi的标准脚本，给出输入参数即可。而这些输入参数一般都是含有信息文件的目录，这些目录放在哪无关紧要，重要的是使其管理起来方便，容易找到，可参考某些示例目录结构。几乎所有例子流程相似，代码调用及bash写法等，可参考其他示例。

Kaldi的训练系统搭建

3.数据准备

只需准备三个映射文件，如图所示：



以此获得某批数据集映射信息，该信息目录会作为提取特征脚本的参数。数据集信息目录没有真实数据，仅存放映射文件。如何处理得到自己的映射文件，需要考虑数据存放目录结构、语音段命名、标注集文件格式等方面，想尽一切办法相呼应的构造代码得到Kaldi标准格式的映射文件。也就是说，数据准备代码怎么写取决于数据集是什么样的。

Kaldi的训练系统搭建

4.语言模型准备

Kaldi的语言模型采用OpenFst标准。通常做法是用SRILM工具训练语料库得到基于ARPA的n-gram格式的语言模型，再用gzip -c 打包成*.gz文件，将此作为参数用Kaldi提供的转换脚本进行转换得到G.fst即可。

在此之前，还需要准备音素信息及lang目录。lang目录可以用标准脚本创建，但其参数dict目录需要我们自己构造脚本创建。dict下必须有这四个文件：

词典[lexicon.txt] （每行格式为：词 音素 音素 …） （如：hao h ao）

非静音音素[nonsilence_phones.txt] （每行格式为：音素 [音素 …]） （下同）

可选音素[optional_silence.txt]

静音音素[silence_phones.txt] （如：sil）

注：词典中的所有音素都要在非静音音素表或者静音音素表中找得到，且这两个表内容互斥。词典应该至少包含所有标注中的分词或分字。

Kaldi的训练系统搭建

5.模型训练

预备工作完成后，以训练数据集信息目录作参数调用训练脚本即可完成。GMM-HMM一般进行monophone,triphone,lda+mllt,lda+mllt+sat的训练，然后才训练DNN。

6.模型解码打分

解码之前需要生成graph，其先提条件就是已经含G.fst的lang目录。

在调用Kaldi的decode.sh脚本时，事先应在local目录下准备打分脚本score.sh，否则便没有打分的统计输出。这个打分脚本可以拷贝通用的脚本，在thchs30例子中即可找到。

7.利用模型对齐一批数据

在训练下一个模型时，训练集的对齐文件可由上一个模型产生，然后作为下一个模型训练时的参数。利用标准脚本（ali）即可完成。

Kaldi的训练系统搭建

训练过程描述（按序）：

GMM-HMM部分（训练类型—产生模型—相关脚本）

[1]monophone	mono	调用脚本steps/train_mono.sh
[2]triphone	tri1	调用脚本steps/train_deltas.sh
[3]lda+mllt变换	tri2b	调用脚本steps/train_lda_mllt.sh
[4]sat变换	tri3b	调用脚本steps/train_sat.sh

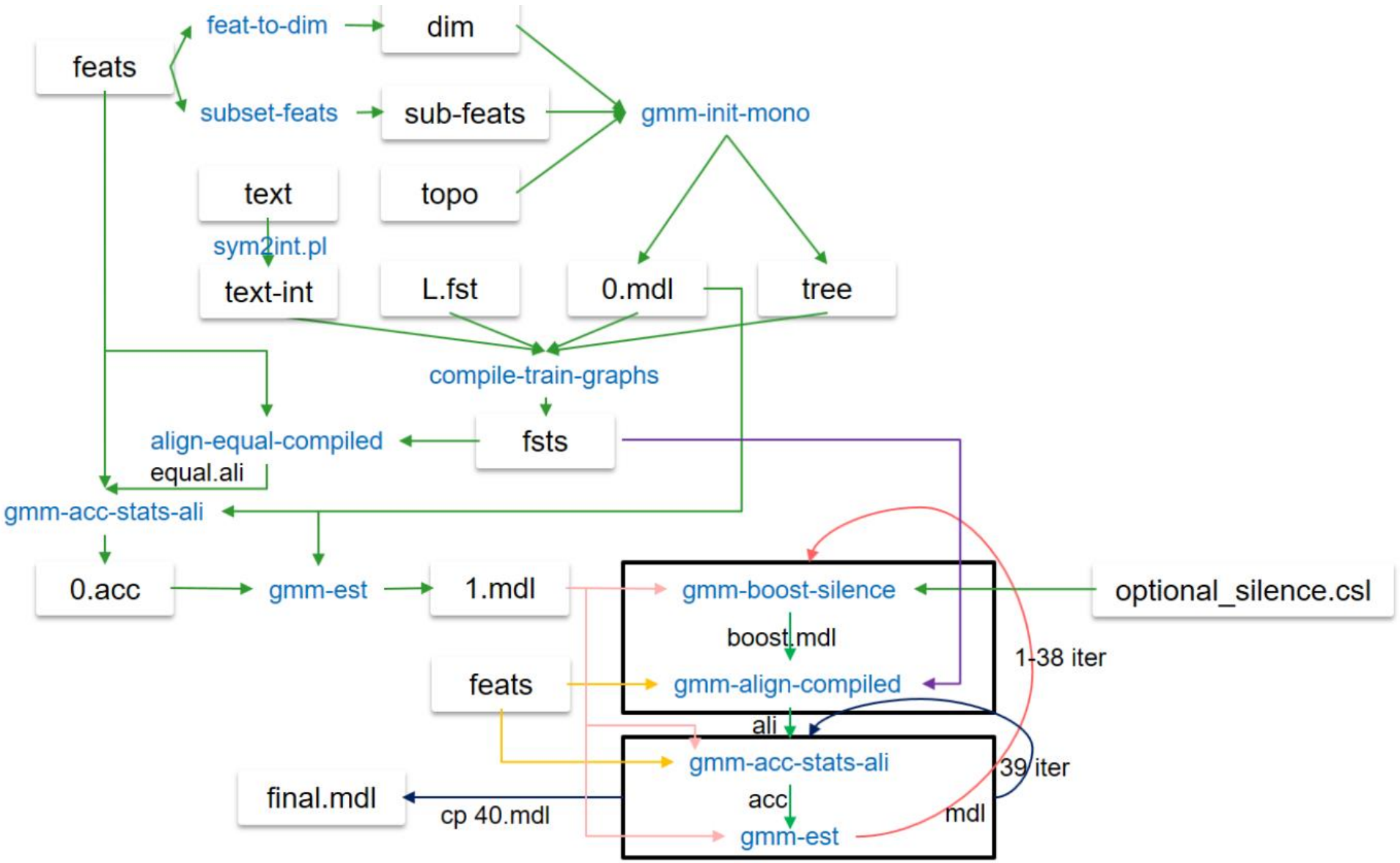
……

DNN部分

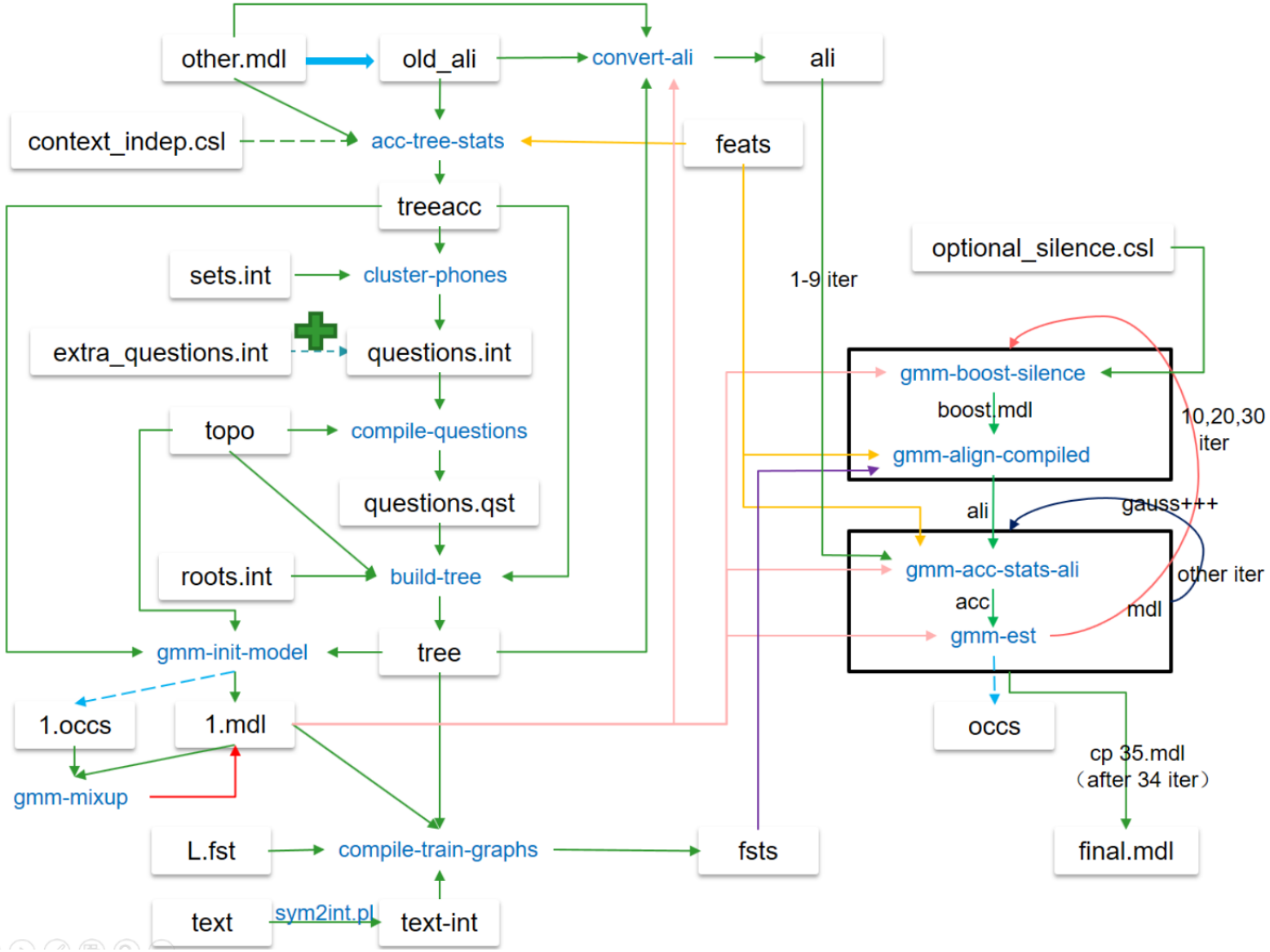
[5]在基于GMM-HMM模型上做nnet2或nnet3的训练

以上过程中，按序训练，后面模型会需要基于前面模型的对齐文件。

单音子(monophone)训练流程图



三音子(triphone)训练流程图



Kaldi的示例—AISHELL-1

AISHELL-1是希尔贝壳开源的178小时中文普通话数据，采样率16kHz。包括400位来自中国不同口音区域的发音人，语料内容涵盖财经、科技、体育、娱乐、时事新闻。

官方网址:
http://www.aishelltech.com/kysjcp

相关论文:
https://arxiv.org/abs/1709.05522

数据介绍、下载

- 数据下载和解压

```
# 原始数据保存目录
data=./data_aishell

# 数据下载网址
data_url=www.openslr.org/resources/33

# 下载语音数据
local/download_and_untar.sh $data $data_url data_aishell || exit 1;

# 下载词典
local/download_and_untar.sh $data $data_url resource_aishell || exit 1;
```

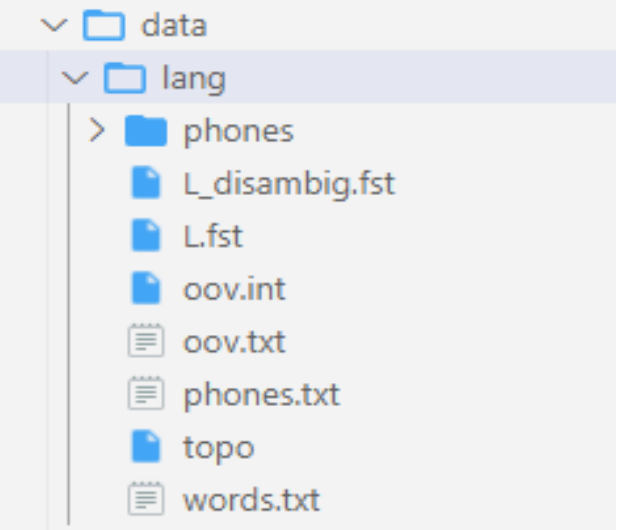
Dict准备

```
# Dict目录 准备
./local/aishell_prepare_dict.sh $data/resource_aishell || exit 1;

# Dict目录 校验
./utils/validate_dict_dir.pl data/local/dict/
```

lexicon.txt	定义词到音素之间的映射关系
nonsilence_phones.txt	定义所有非静音的音素
silence_phone.txt	定义表示无效语音的音素
optional_silence.txt	定义词间静音的音素
extra_questions.txt	定义构建上下文决策树的基本问题

lang目录准备

```
# 语言目录 准备
utils/prepare_lang.sh --position-dependent-phones false data/local/dict \
    "<SPOKEN_NOISE>" data/local/lang data/lang || exit 1;

# 语言目录 校验
./utils/validate_lang.pl data/lang
```



语言模型准备

- 语言模型训练

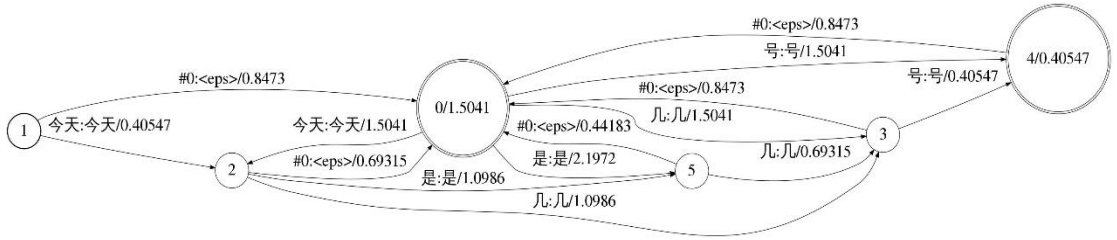
```
# 语言模型训练
local/aishell_train_lms.sh

# 如何使用自己的数据训练
1、语料的收集(爬虫)
2、文本的清洗(https://github.com/speechio/chinese_text_normalization)
3、分词(jieba工具)
4、ngram-count得到ARPA格式语言模型
5、测试困惑度
6、剪枝
```

语言模型准备

- ARPA-format转换为FST

```
# ARPA-format语言模型转换为G.fst
utils/format_lm.sh data/lang data/local/lm/3gram-mincount/lm_unpruned.gz \
    data/local/dict/lexicon.txt data/lang_test || exit 1;
```

```
\data\
ngram 1=6
ngram 2=6

\1-grams:
-0.6532125	</s>
-99	<s>	-
0.3679768
-0.6532125	今天	-0.30103
-0.6532125	几	-
0.3679768
-0.6532125	号	-
0.3679768
-0.9542425	是	-
0.1918855

\2-grams:
-0.1760913	<s> 今天
-0.4771213	今天 几
-0.4771213	今天 是
-0.1760913	几 号
-0.1760913	号 </s>
-0.30103	是 几
\end\
```



数据预处理

```
# 数据预处理
local/aishell_data_prep.sh $data/data_aishell/wav \
       $data/data_aishell/transcript || exit 1;

# 数据格式校验
utils/validate_data_dir.sh data/train/
```

（注意中文文本会报错，注释掉相应的exit语句）

wav.scp	音频ID+ 音频路径（绝对路径）
utt2spk	音频ID + 说话人ID
spk2utt	说话人ID + 此人所有音频ID（utils/utt2spk_to_spk2utt.pl）
text	text：音频ID+分好词的文本（jeiba分词）

声学特征提取

- 修改cmd.sh文件

使用集群训练

```
# 默认cmd.sh内容
export train_cmd="queue.pl --mem 2G"
export decode_cmd="queue.pl --mem 4G"
export mkgraph_cmd="queue.pl --mem 8G"
```

使用单机训练

```
# 修改cmd.sh内容
export train_cmd="run.pl"
export decode_cmd="run.pl"
export mkgraph_cmd="run.pl"
```

声学特征提取

- 提取声学特征（mfcc+pitch）

```
# 声学特征提取
steps/make_mfcc_pitch.sh --cmd “$train_cmd” --nj 10 data/$x \
        exp/make_mfcc/$x $mfccdir
```

```
BAC009S0764W0202  [
  44.11316 -9.00462 15.23577 -1.442114 -3.072395 -6.846048 -3.19953 -3.941009 9.384831 13.3998 13.70805 16.21714 6.23234 -0.6750021 0.3381608 0.01375642
  43.83405 -7.09164 9.592178 -2.973187 -3.580292 -11.19833 0.4905128 4.844214 21.41292 19.43657 13.70805 6.890126 0.06613922 -0.7070547 0.3430049 0.08456525
  43.83405 -9.534603 7.205004 0.08895969 -4.850031 -11.19833 -2.848097 7.636418 10.98858 9.427057 9.333549 6.890126 3.3375 -0.5677292 0.3478491 0.07040349
```

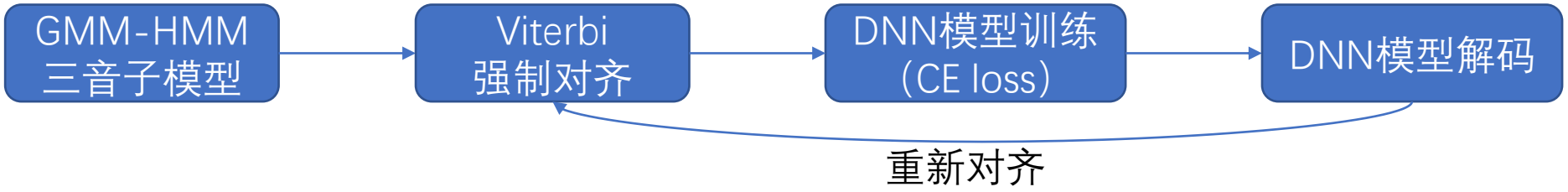
声学特征提取

- 计算倒谱均值方差归一化系数（CMVN）

```
# 计算倒谱均值方差归一化系数
steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir
```

```
S0764  [
  1.148539e+07 -572461.1 -297434.4 -315284.8 -1842024 -1927375 -1852338 -2152830 -3227037 938890.7 -1672140 -530654.4 -954705.4 -89236.11 1208.553 -230.496 171896
  7.920888e+08 3.211006e+07 4.424268e+07 4.187042e+07 8.125914e+07 7.212415e+07 5.788484e+07 6.26528e+07 1.116311e+08 3.30395e+07 5.259662e+07 2.567214e+07 2.89689e+07 58689.89 17486.03 4903
.233 0 ]
```

帧级别交叉熵（CE）训练



帧级别交叉熵（CE）训练

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

- local/nnet3/run_ivector_common.sh
  - 数据扩充语速、音量
  - 数据提特征
  - 数据对齐
  - ivector模型训练、提取
- steps/nnet3/xconfig_to_configs.py
- steps/nnet3/train_dnn.py
- steps/nnet3/decode.sh

帧级别交叉熵（CE）训练

Kaldi nnet3网络结构:

1）xconfig: 基于层的方式定义网络结构，xconfig覆盖了大部分常用的神经网络layer。

2）config: Kaldi编译网络结构实际使用的配置， 基于图节点来定义网络结构，如果xconfig无法满足需求，可以在config基于图的方式搭建网络。

3）C++: 如果一些网络不能利用现有的组件构建，或者想提高运行效率，则可以在C++层实现。

帧级别交叉熵（CE）训练

xconfig文件:

```
input dim=40 name=input
relu-batchnorm-layer name=tdnn dim=128 input=Append(-1,0,1)
output-layer name=output input=tdnn dim=26 max-change=1.5
```

基于层的描述语言来转化成图描述的配置文件:

```
steps/nnet3/xconfig_to_configs.py --xconfig-file tdnn.xconfig --config-dir ./config
```

帧级别交叉熵（CE）训练

XIAMEN UNIVERSITY

config文件:

```
input-node name=input dim=40
component name=tdnn.affine type=NaturalGradientAffineComponent input-dim=120 output-dim=128  max-change=0.75
component-node name=tdnn.affine component=tdnn.affine input=Append(Offset(input, -1), input, Offset(input, 1))
component name=tdnn.relu type=RectifiedLinearComponent dim=128 self-repair-scale=1e-05
component-node name=tdnn.relu component=tdnn.relu input=tdnn.affine
component name=tdnn.batchnorm type=BatchNormComponent dim=128 target-rms=1.0
component-node name=tdnn.batchnorm component=tdnn.batchnorm input=tdnn.relu
component name=output.affine type=NaturalGradientAffineComponent input-dim=128 output-dim=26  max-change=1.5 param-stddev=0.0 bias-stddev=0.0
component-node name=output.affine component=output.affine input=tdnn.batchnorm
component name=output.log-softmax type=LogSoftmaxComponent dim=26
component-node name=output.log-softmax component=output.log-softmax input=output.affine
output-node name=output input=output.log-softmax objective=linear
```

帧级别交叉熵（CE）训练

```
steps/nnet3/train_dnn.py --stage=$train_stage \
    --cmd="$decode_cmd" \
    --feat.online-ivector-dir exp/nnet3/ivectors_${train_set} \
    --feat.cmvn-opts="--norm-means=false --norm-vars=false" \
    --trainer.num-epochs $num_epochs \
    --trainer.optimization.num-jobs-initial $num_jobs_initial \
    --trainer.optimization.num-jobs-final $num_jobs_final \
    --trainer.optimization.initial-effective-lrate $initial_effective_lrate \
    --trainer.optimization.final-effective-lrate $final_effective_lrate \
    --egs.dir "$common_egs_dir" \
    --cleanup.remove-egs $remove_egs \
    --cleanup.preserve-model-interval 500 \
    --use-gpu true \
    --feat-dir=data/${train_set}_hires \
    --ali-dir $ali_dir \
    --lang data/lang \
    --reporting.email="$reporting_email" \
    --dir=$dir  || exit 1;
```

帧级别交叉熵（CE）训练

```
--stage:   控制神经网络训练的阶段，可以断点续上接着训练模型
--trainer.num-epochs    设置总的训练轮次
--trainer.optimization.num-jobs-initial     设置开始时并行训练的个数
--trainer.optimization.num-jobs-final       设置结束时并行训练的个数
--trainer.optimization.initial-effective-lrate    设置开始时的学习率
--trainer.optimization.final-effective-lrate      设置结束时的学习率
```

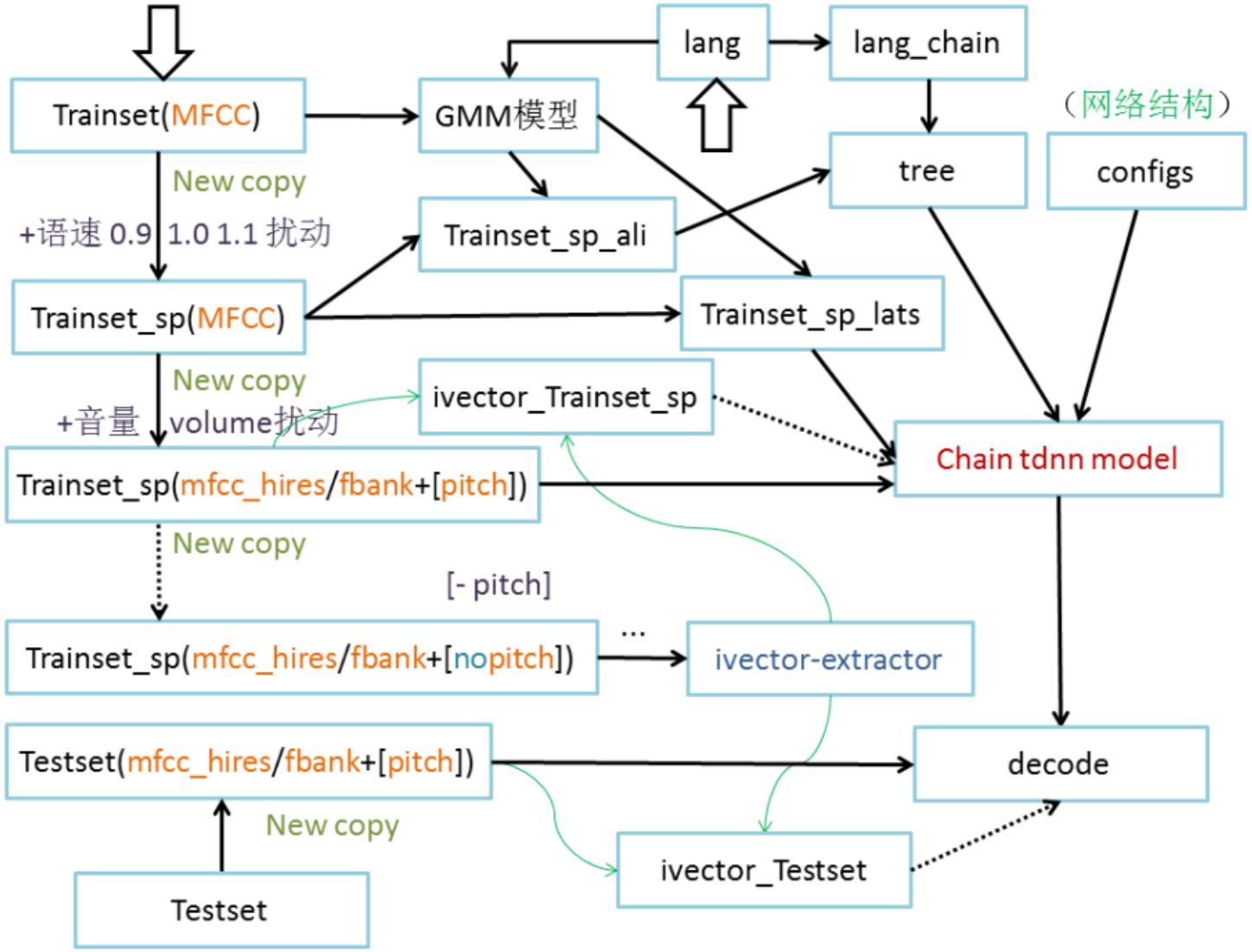
```
CUDA_VISIVABLE=0,1  steps/nnet3/train_dnn.py   指定GPU调用
```

LF-MMI区分性训练（Chain）

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

- local/nnet3/run_ivector_common.sh
  - 数据扩充语速、音量
  - 数据提特征
  - 数据对齐
  - ivector模型训练、提取
- steps/align_fmllr_lats.sh
- steps/nnet3/chain/gen_topo.py
- steps/nnet3/chain/build_tree.sh
- steps/nnet3/xconfig_to_configs.py
- steps/nnet3/chain/train_dnn.py
- utils/mkgraph.sh
- steps/nnet3/decode.sh

Chain模型训练流程图



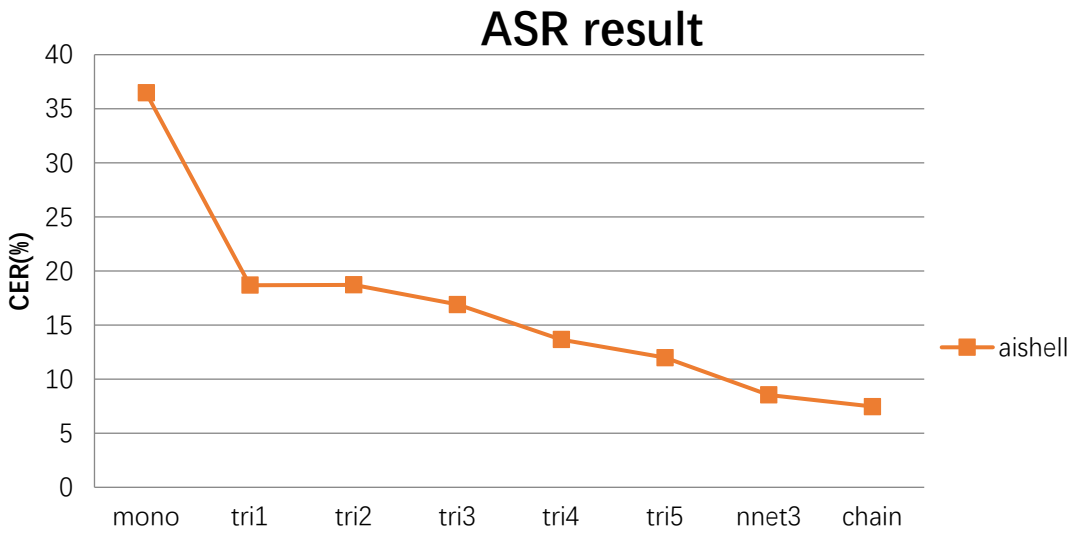
Chain模型训练流程

- Chain model也是HMM，使用nnet3结构，与传统模型有很多不同点。
- 它的训练过程是LF-MMI的无Lattice的版本，通过在解码图中进行一个完整的前向后向过程实现，这个解码图来源于音素的n-gram语言模型。
- 神经网络的输出的帧率缩小三倍，明显的缩小了测试时的计算量，使实时解码更加容易，使用非传统的HMM拓扑结构（允许在单一状态下遍历HMM）。
- 在传统的DNN-HMM模型中，HMM模型状态之间是有转移概率的，并且会在训练过程中对转移概率进行评估；Chain模型的HMM模型转移概率设置为常数0.5，使用固定的转移概率，并且不训练转移概率。
- 目前比传统DNN-HMM的结果要稍微好一点（大概提升了5%），但是解码速度比以前快了三倍。

测试结果

[aishell-1]

以下给出了aishell-1的实验结果，对比了不同声学模型的CER%指标。



致谢

- 感谢赵淼、夏仕鹏对Kaldi实践过程做了深入细致的整理。
- 感谢厦门大学智能语音实验室其他同学的贡献。