Hands-on Speech Recognition with Kaldi/TIMIT

Demystify Automatic Speech Recognition (ASR) & Deep Neural Networks (DNN)

Yamin Ren

Leo Reny

HYPech.com

India • Japan • Korea • Singapore • United Kingdom • United States

AUTOMATIC SPEECH RECOGNITION SERIES

Hands-on Kaldi :

Leo Reny

Yamin Ren

Publisher and GM:

Mel Pearce

Associate Director:

Kris Burns

Manager of Editorial Services:

Hayden Simpson

Marketing Manager:

Lilyana Cardenas

Acquisitions Editor:

Alexandra Lewis

Project and Copy Editor:

Isabella Fletcher

Technical Reviewer:

Sergio Crawford

Interior Layout Tech:

ReApex Limited

Cover Designer:

Luke Fletcher

Indexer:

Harley Kramer

Proofreader:

Blair Salas

Printed in the US

1 2 3 4 5 6 7 12 11 10

ISBN-13: 978-1-7774122-6-5 (e-Book)

ISBN-13: 978-1-7774122-5-8 (Book)

ISBN-13: 979-8-5666334-9-7 (KDP)

Hypech.com is a leading provider of customized learning solutions to IT technology. Hypech.com learning products are represented in Canada by ReApex Corp.

Visit our corporate website at http://HYPech.com.

For You

ACKNOWLEDGMENTS

We would like to acknowledge and thank Daniel Povey for his invaluable work on Kaldi, indeed a great ASR Toolkit, and for his great help and kindness in letting us use some of the text of Kaldi's web pages describing the software. After he moved to Xiaomi, Daniel's brilliant wisdom inspired all the people working around him.

Moreover, we would like to acknowledge Karel Vesely for his great work on his very effective version of DNN.

We would like to thank those friends from Google, Xiaomi, iFlytek, and Baidu Institute, who contributed to my understanding of the ASR and AI. Without their encouragement and support, I might not start this Kaldi book project.

We would like to thank the many editors and workers at Hypech Publisher who skillfully enhanced and improved many aspects of this book as it was brought to fruition. Without Alexandra Lewis, my acquisitions editor, this book literally would not exist. Sergio Crawford, my technical editor, did an outstanding job in his review. I thank him for the numerous suggestions and corrections that he provided. Finally, Mel Pearce, my project editor was superb in pulling it all together, while adding a professional touch and coherency to the entire project.

Finally, and most especially, we would like to thank everyone in our immediate family for their encouragement and support as we've dedicated myself to this project.

THE AUTHOR

Seeing rather than listening could help hearing loss group a lot. That's why Yamin Ren has founded Seeing Voice Corporation in 2018, devoting to help hearing loss people by using the advanced Artificial Intelligence technology Automatic Speech Recognition.

As the Product Manger, Yamin led a team of 30 developers build up cloud-based ASR platform SV Cloud Speech Recognition. The SV ASR is based on Kaldi and Tensorflow. The system has been widely used in government and industry. One of the successful products is Seeing Voice AR Glass.

Yamin has been involved with AI and software developing for 20+ years. He has designed and developed multiple software products for factories and companies before starting ASR. His main area of expertise is with software development, product design, and innovation. He has served as a consultant for Canada Federal Government, Glencore, and many other industry leaders. He has worked with Accenture and HWG. He holds an MBA from the University of California, Los Angeles, with a specialization in Management Information System.

For more information on his current activities or to contact the author, please visit http://www.hypech.com.

CONTENTS

Introduction

Speech Recognition

Speech is acoustic signal containing information of idea that is formed in speaker's mind. Speech Recognition's purpose is to retrieve the acoustic information contained in speech signal. The noise or other irrelevant hurts the accuracy of speech recognition.

Speech processing has three stages. Signal processing stage studies human auditory system and process signal in form of frames.

Phoneme stage processes the speech phonemes, the basic unit of speech.

Based on signal and phoneme, the last stage is word processing. The linguistic entity of speech is the target.

Kaldi models the above three stages. It is roughly composed of two parts: traditional HMM-GMM and deep neutral network DNN.

When learning Kaldi, it's a good idea to start with HMM-GMM. Many matured models like steps/train_delta.sh, steps/train_fmllr.sh, steps/decode.sh are all based on HMM-GMM model. Lacking the fundamental knowledge of HMM-GMM model will be confused about the decision tree , alignment, lattice, etc.

To learn HMM-GMM, the best resource is not Kaldi but another

opensource tool HTK. HTK Book elaborates all the details and concepts of speech recognition from trainging to decoding.

For HMM-GMM purpose, we need only know well about decision trees(DTs), training network extension, Viterbi algorithm, EM algorithm. Trying to identify the difference among them and understanding when to use which.

Chapter 2, 8, 10, 12, 13 of HTK Book are most important. Many professionals suggest to check Kaldi script while reading the book, which could help understanding the concept while practising. For example, the build-tree script in steps/train_delta.h is a perfect match to Chapter 10 tree-based clustering of HKT Book, and GMM-EST script help understanding Chapter 8 Parameter Re-Estimation Formulae a lot.

We will understand speech recognition clearly after clarifying the HMM-GMM, following which we could start the neutral network. Kaldi supports most of the netral networks like MLP, RNN, CNN, LSTM, and CTC.

MLP might be the best entrance to the neutral network. We could treat MLP as the basic model of the neutral network.

Kaldi provides three tools for neutral network: nnet1, nnet2, and nnet3.

The nnet1 and nnet2 are using HMM-DNN framework. The difference is that the nnet2 is using NG-SGD method first introduced by Dan Povey, which support multi-threading parallel training. To learn nnet2, we could start with Dan's papers published after 2012.

Armed with nnet1 and nnet2, we could challenge nnet3 chain model and other neutral networks like RN, CNN, LSTM.

The rest is just following the papers from Microsoft, Dan Povey,

Google, and University of Toronto.

Some languages (for example Chinese) have tone. The tone itself contains meaning. We need add tone information into the feature.

All in all, Speech Recognition is still more theoritical than practical. We need learn and understand many theories before doing some valuable work. Most of the formulas need us to derive to understand.

Kaldi is a Speech Recognition platform providing many ASR models like GMM, SGMM, DNN, and HMM. We could train our own data using these platforms, and then recogine. The only thing we need do is to modify the scripts to use our own corpus. There are a couple of competitions (like HTK).

Kaldi is comparably easy to start and deliver. We have implemented several ASR projects using Kaldi. The feedback for online version is not bad. In this book, we will show every steps from start to delivery.

Our next exploration will focus on offline ASR, which is told by Dan that has been accomplished in chain-model, which is not as effective as what we expected.

Chapter 1 discussing some background of the ASR. Knowing the history and context of the new topic is the best way to understand it in my humble opinion. Always asking: Who is it? Where did it come from? Where is it going?

Once the soul questions have been answered, we get to install Kaldi. Chapter 2 Installation will explain Kaldi environment and installation process. We could have Mac, PC, Linux, Windows or any platform.

We test the Kaldi installation with some small projects like yes/

no. There are another recipe called 10 digits speech recognition good for testing purpose as well, which is not included in this book.

Chapter 3 downloads and sets up TIMIT in Kaldi with specific environment parameters.

Chapter 4 prepares the data for TIMIT. We learn about FST, dictionary and some other relevant concepts during preparation.

Chapter 5 extracts features. MFCC and CMVN will be discussed in details.

Chapter 6 runs monophone model for TIMIT. All the ASR fundamental concepts have been explained.

Chapter 7 to 9 run triphone model for tri1, tri2, tri3.

Chapter 10 runs SGMM2 model.

Chapter 11 runs MMI + SGMM2.

Chapter 12 runs Dan's DNN.

Chapter 13 covers all the stages of Karel's DNN, including store features, pre-training, frame-level cross-entropy, sequence-discriminative training, and iteration of sMBR.

Chapter 14 is the final results of the whole TIMIT output which could be used as a template for comparison.

When finishing the whole book, we will be armed with Kaldi ASR Neutral Network models running capability.

This book gives you a start point to pursue higher goals in Artificial Intelligence world.

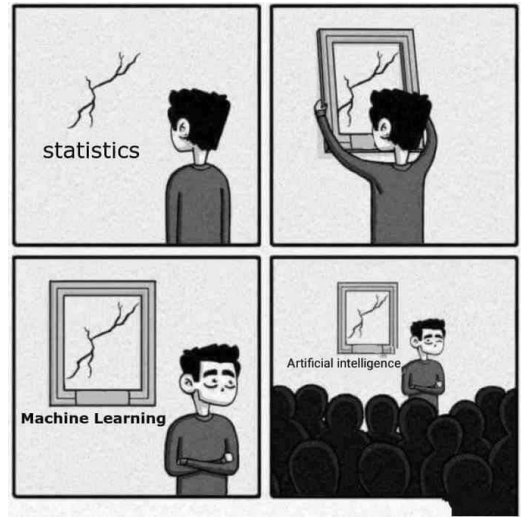
Chapter 1. ASR Review

Automatic Speech Recognition

The essence of the voice and speech recognition is pattern recognition. We compare the unknown and known speech pattern. The best matching one is the recognized result.

To put it simply, we train the voice and speech to get a statistical model, which connect the language, voice, or speech with probability to the wave based on the matching record of the speech and wave, so that once getting the wave, we could derive the actual speech.

Speech recognition is an interdisciplinary technology, including signal processing, pattern recognition, probability, information system, sounds mechanics, hearing mechanics, statistics, linguistics, and artificial intelligence.



original comic by sandserif, https://www.instagram.com/sandserifcomics/

1.1 History of ASR

History can help us understand present, predict future.

Bell Lab is considered as the pioneer of speech recognition. The first product was developed around 1950s by Davis called “Audrey”.

Even though CMU and some other famous institutes joined the speech recognition research, the whole process is very slow in 60s and 70s.

The ‘80s saw speech recognition fast growth. Two major tehcnologies have been developed: HMM and N-Gram, in which statistics started to be used for prediction. Scholars changed the research method from template model to HMM statistic model. GMM-HMM has established its main framework in speech recognition field since then.

Speech recognition was matured in 90s largely due to the personal computer. Faster processors and bigger memory made it possible for statistic models. HTK developed by Cambridge promoted GMM-HMM model, with which the speech recognition technology had achieved close to 80% accuracy.

The key breakthrough is happened in 2006, when DBN was introduced by Hinton. DBN directly led to the recovery of DNN research.

Year 2009 saw Hinton’s success with TIMIT. Scholars started using DNN-HMM instead of GMM-HMM.

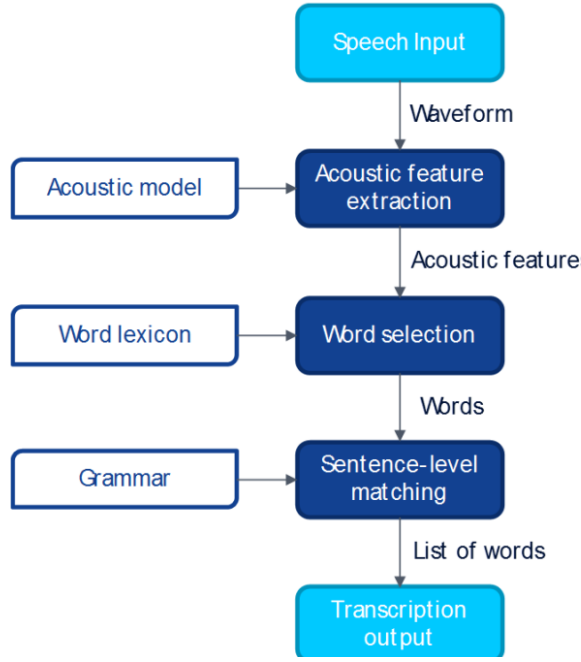
The different techniques adopted for feature vectors extraction process and speech pattern recognition have been developed since 2009, among which Dynamic Time Warping (DTW), Vector Quantization (VQ), Artificial Neural Networks(ANN), and support vector machines(SVM) are popular.

1.2 General Steps

So-called "Speech Recognition" is actually convert a piece of language signals to text. The system is mainly composed of acoustic features, acoustic model, lexicon, language model, and decoding parts.

To create more acceptable feature vectors, we usually perform pre-emphasis, windowing, Fourier Analysis, Filter-bank Analysis, framing and other pre-processing. The pre-processing of voice could help extract the original voice signal.

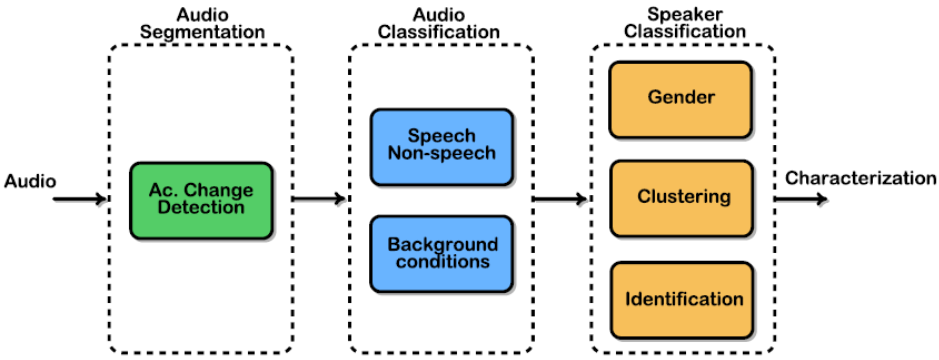
Features extraction transforms the sound signal from time domain to frequency domain, and provides appropriate features vector for acoustic model. Acoustic model calculates the mark of each feature based on acoustic analysis. The language model finds the words probability based on language theory. In the end, the words probability turns to text decoded based on lexicon.



Kaldi ASR: Extending the ASpIRE model, Mar 11, 2017 ~ KRISZTIÁN

1.2.1 Pre-processing

There are certain requirements for speech recognition model to process the raw sounds. Pre-processing could filter the noise, identify the end points, framing and pre-emphasis.



Distinguishing speech signals from other non-speech signals is always important for ASR systems. Audio and speech segmentation will always be needed to break the continuous audio stream into manageable chunks. By using ASR acoustic models trained for a particular acoustic condition such as male speaker versus female speaker or wide bandwidth (high quality microphone input) versus telephone narrow bandwidth the overall performance can be significantly improved.

The pre-processing could also be designed to provide additional interesting information such as division into speaker turns and speaker identities allowing for automatic indexing and retrieval of all occurrences of the same speaker. It can also provide end of turn information relevant for recovering punctuation marks, syntactic parsing, etc. Additionally speaker segmentation and clustering information can be used for efficient speaker adaptation which has been shown to significantly improve ASR accuracy . All these information, when combined with the text output of the ASR, results in a rich transcription easier to understand.

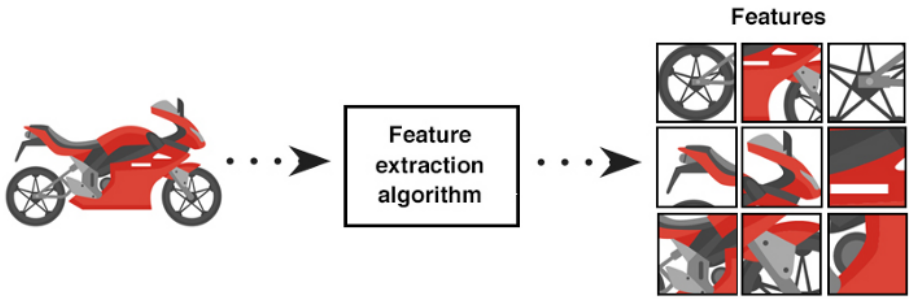
1.2.2 Feature Extraction

The speech is an analog audio signal in the form of sound waves. ASR is a math model which requires digital number input. Features are the numbers to represent the wave. The main method we use is MFCC (Mel Frequency Cepstral Coefficients).

MFCC uses FFT (Fast Fourier Transform) to convert each frame of speech sample from a time series of bounded time domain signals into a frequency spectrum. The frame that has undergone the windowing process is converted into a frequency spectrum. Humans listen to voice information based on time domain signals. At this stage Mel-spectrum will be converted into time domain by using Discrete Cosine Transform (DCT). The result is called Mel-frequency cepstrum coefficient (MFCC).

CMU Sphinx uses MFCC in different way. Starting from an audio clip, it slides windows of 25 ms width and 10 ms apart to extract MFCC features. For each window frame, 39 MFCC parameters will be extracted. The primary objective of speech recognition is to build a statistical model to infer the text sequences W (say "cat sits on a mat") from a sequence of feature vectors X. To put them in a vector, we get feature vector.

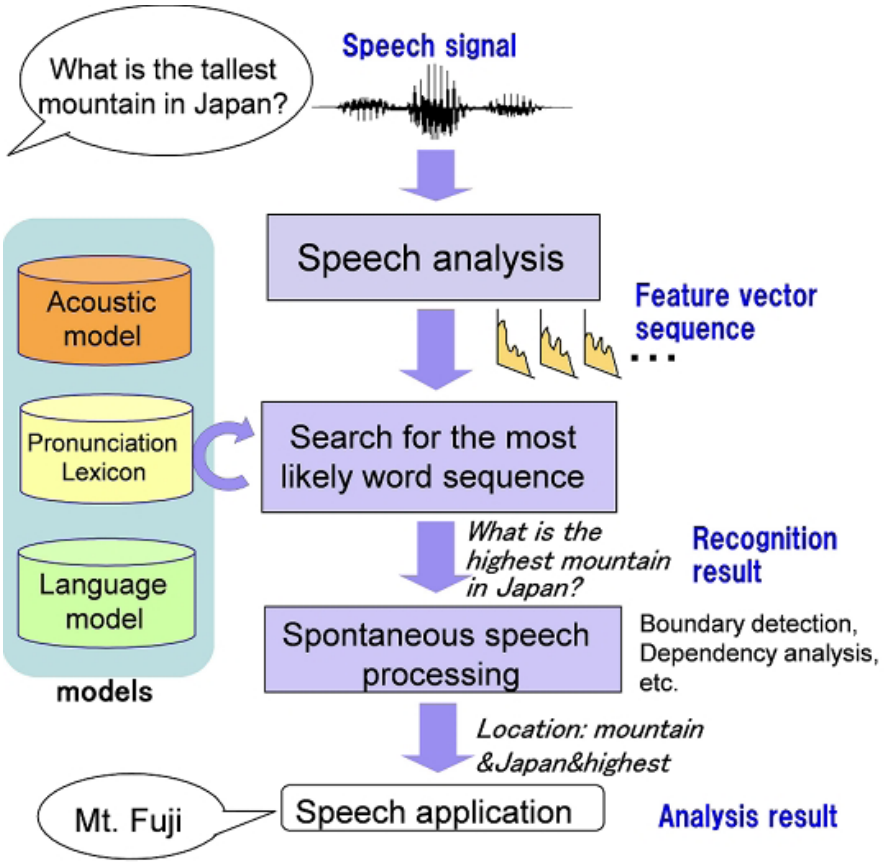
Another approach looks for all possible sequences of words (with limited maximum length) and finds one that matches the input acoustic features the best.



original from https://manningbooks.medium.com/the-computer-vision-pipeline-part-4-feature-extraction-6343ef063588

1.2.3 Acoustic Model Training

An acoustic model is used in automatic speech recognition to represent the relationship between an audio signal and the phonemes or other linguistic units that make up speech. A well trained acoustic model (often called a voice model) is critical in the performance of an ASR enabled application.



from: http://www.kecl.ntt.co.jp/icl/signal/en/topics1.html

Acoustic models are trained by taking audio recordings of speech, and their text transcriptions, and creating statistical representations of the sounds that make up each word. A simple rule is that a model trained with hundreds and thousands of hours of transcribed audio will perform better than a model trained with

limited audio. Quantity of data is important but it should also include a broad distribution of sounds.

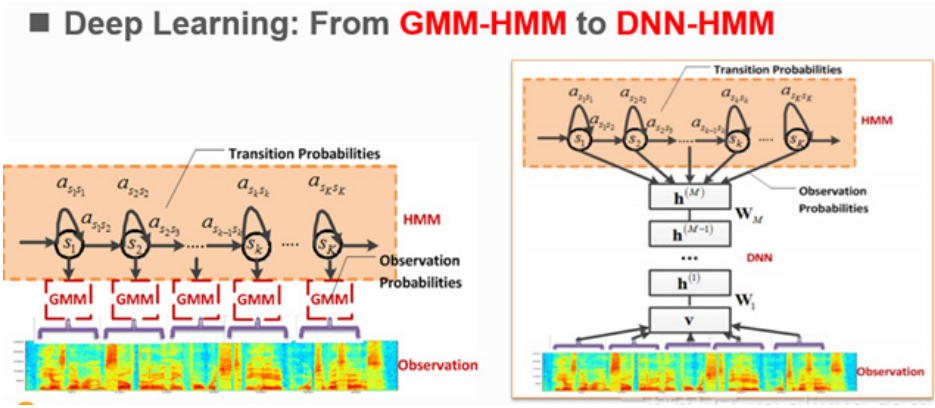
HMM (Hidden Markov Model) is widely used in the ASR systems to model acoustic. The unit of acoustic model could be phone, syllable, word or other levels of sound. For a small amount of data, syllable is good enough. For a large sample, using phone (ie. consonant, vowel) is more appropriate. The larger the sample, the smaller the acoustic model unit.

The task of automatic speech recognition has traditionally been accomplished by modeling the problem as a Hidden Markov Model (HMM). Gaussian Mixture Models (GMM) have been used to determine how well each state of each HMM fits a frame or a short window of frames of coefficients that represents the acoustic input.

In the context of speech recognition, the GMM-HMM hybrid approach has a serious shortcoming – GMMs are statistically inefficient for modeling data that lie on or near a nonlinear manifold in the data space. Because speech is typically produced by modulating a relatively small number of parameters, it has been hypothesized that the true underlying structure is much lower dimensional than is immediately apparent in a window that contains lots of coefficients.

Deep Neural Networks (DNN), on the other hand, have the potential to learn much better models of data that lie on or near a nonlinear manifold. Many studies have confirmed this hypothesis, with DNN systems outperform GMMs by approximately 6% for Individual Word Error Rates (IWER). The networks are trained to optimize a given training objective function using the standard error back propagation procedure. In a DNN-HMM hybrid system, the DNN is trained to provide posterior probability

estimates for the different HMM states. Typically, crossentropy is used as the objective and the optimization is done through stochastic gradient descent (SGD). For any given objective, the important quantity to calculate is its gradient with respect to the activations at the output layer. The gradients for all the parameters of the network can be derived from this one quantity based on the back propagation procedure.



Compared to traditional GMM-HMM acoustic models, DNN-HMM-based acoustic models perform better in terms of TIMIT database. When compared with GMM, DNN is advantageous in the following ways:

1. De-distribution hypothesis is not required for characteristic distribution when DNN models the posterior probability of the acoustic characteristics of speech.

2. GMM requires de-correlation processing for input characteristics, but DNN is capable of using various forms of input characteristics.

3. GMM can only use single-frame speech as inputs, but DNN is capable of capturing valid context information by means of splicing adjoining frames.

1.2.4 Language Model Training

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language modeling could effectively connect grammar and semantic, describe the relationship between words, so as to improve the accuracy of recognition and narrow down the search scope.

There are 3 stages for language modelling: lexicon, grammar, and syntax.

N-gram model is so far most effective to predict the language. In daily life, we could understand other's meaning before it finishes the sentence. With the first several words, we predict the rest. N-Gram means using the previous n-1 words to predict the next word. The whole sentence will be derived from the dynamic prediction. Bi-gram and tri-gram are most common used.

1.2.5 Decoding

Acoustic model and language model calculate probabilities that a particular frame of the input speech corresponds to each of a specific set of speech sounds. Decoding will find the most likely word sequence given the per-frame, per-sound probability matrix, in light of the phone models, the word pronunciations in the dictionary, and the word-sequence constraints described by the grammar.

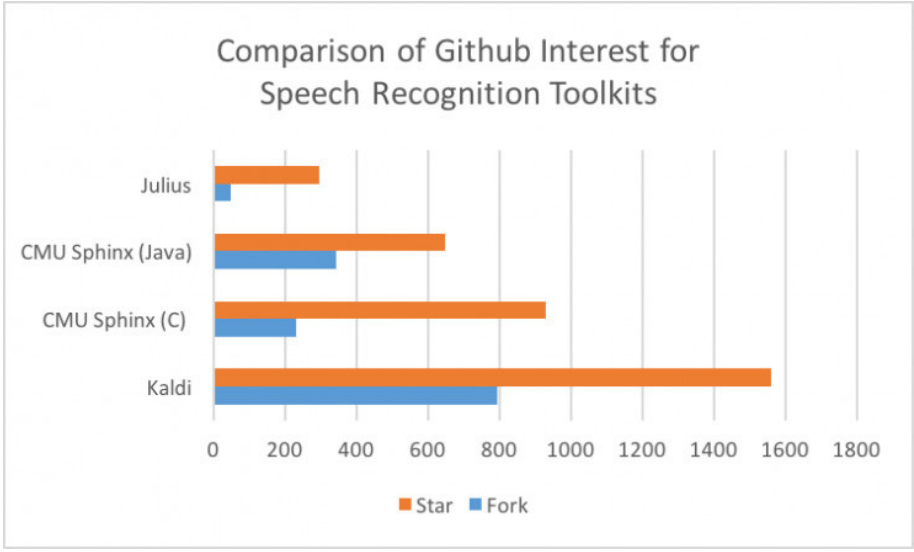
To be more accurate, decoding means searching. The decoder essentially searches over all possible alignments of all possible pronunciations of all possible word sequences to find the most likely one. Viterbi algorithm helps to limit the number of searches for the maximum path.

1.3 Popular ASR tools

Below is an old table showing the characters of Kaldi, CMU Sphinx, HTK, Julius, and ISIP. Kaldi has been cooperated with Tensorflow and Pytorch and gained dramatic progress in deep learning and machine learning.

Toolkit	Programming languages	Development activity	Tutorials and examples	Community	Trained models
CMU Sphinx	Java, C, Python, others	+++	+++	+++	English plus 10 other languages
Kaldi	C++, Python	+++	++	+++	Subset of English
HTK	C, Python	++	+++	++	None
Julius	C, Python	++	++	+	Japanese
ISIP	C++	++	++	+	Digits only

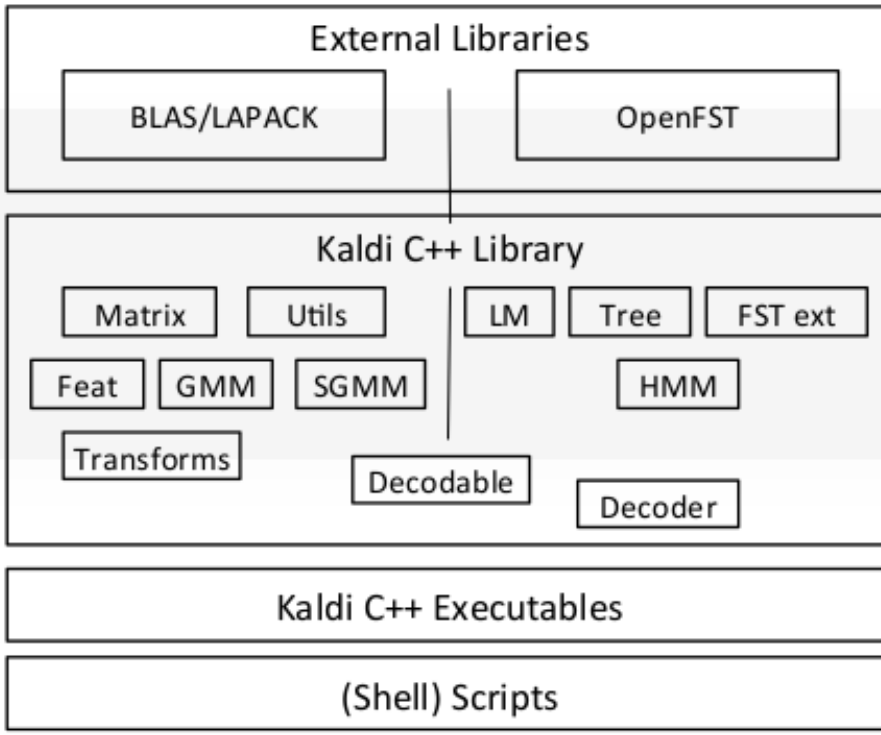
Below is the comparison from one point of view.



1.4 Kaldi

Kaldi is an open source toolkit made for dealing with speech recognition. It could handle speaker recognition and speaker separation.

Kaldi is written mainly in C++, but the toolkit is wrapped with Bash and Python scripts on top of C++. For basic usage this wrapping spares the need to get in too deep in the source code.



Daniel Povey1, The Kaldi Speech Recognition Toolkit, 2011

At its early stage, the Kaldi toolkit supported modeling of context-dependent phones of arbitrary context lengths, all commonly used techniques that can be estimated using maximum likelihood, and almost all adaptation techniques available

in the ASR literature. At present, it also supports the recently proposed Semi-supervised Gaussian Mixture Model (SGMMs) (Huang, Hasegawa-Johnson, 2010), (Povey et al., 2011), discriminative training, and the very promising DNN hybrid training and decoding (Kaldi, WEB-a; Vesely et al., 2013; Kaldi, WEB-b; Zhang et al., 2014; Povey et al., 2015). Moreover, developers are working on using 436 P. COSI, G. PACI, G. SOMMAVILLA, F. TESSER large language models in the FST framework, and the development of Kaldi is continuing.

Kaldi is considered to be "easy to use" (once you learn the basics, and assuming you understand the underlying science)

- It's "easy to extend and modify"
- It's "redistributable": unrestrictive license, community project
- If your stuff works or is interesting, the Kaldi team is open to including it and your example scripts in our central repository: more citation, as others build on it.

In particular, even if Kaldi is similar in aims and scope to HTK, and the goal is still to have modern and flexible code, written in C++, that is easy to modify and extend, the important features that represent the main reasons to use Kaldi versus other toolkits include:

- Code-level integration with Finite State Transducers (FSTs): compiling against the OpenFst toolkit (using it as a library);
- Extensive linear algebra support: including a matrix library that wraps standard BLAS and LAPACK routines
- Extensible design: providing, as far as possible, algorithms in the most generic form possible; for instance, decoders are

templated on an object that provides a score indexed by a (frame, fstinput-symbol) tuple, this meaning that the decoder could work from any suitable source of scores, such as a neural net;

- open license: the code is licensed under Apache 2.0, which is one of the least restrictive licenses available;
- complete recipes: making available complete recipes for building speech recognition systems, that work from widely available databases such as those provided by the ELRA or Linguistic Data Consortium (LDC).

It should be noted that the goal of releasing complete recipes is an important aspect of Kaldi. Since the code is publicly available under a license that permits modifications and re-release, this encourages people to release their code, along with their script directories, in a similar format to Kaldi's own example script.

Chapter 2. Kaldi Installation

Machine, OS, and System

For learning purpose, a PC with GPU will be able to run most of the Kaldi models.

2.1 Machine and OS

We bought a new SSD Hard drive to install on an old HP Pavilion Gaming PC with GTX 1070TI. We installed Ubuntu 18.04.1 LTS into the new SSD and configured the machina dual-system, Ubuntu and Windows.

```
sv@HP:~$ sudo lsb_release -a
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:    18.04
Codename:   bionic

sv@HP:~$ cat /proc/cpuinfo | grep model\ name
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
sv@HP:~$ cat /proc/meminfo | grep MemTotal
MemTotal:       16321360 kB
sv@HP:~$ lspci | grep 'VGA'
01:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev
```

2.2 Install and Configure Git

We need Git to get Kaldi.

```
$sudo apt install git-all
```

Some simple configuration:

```
$git config --global user.name "Leo Reny"
$git config --global user.email leoreny@hypech.com
$git config --global alias.co checkout
$git config --global alias.br branch
$git config --global alias.st status
```

After configuration, run below to check the result:

```
$git config --list
```

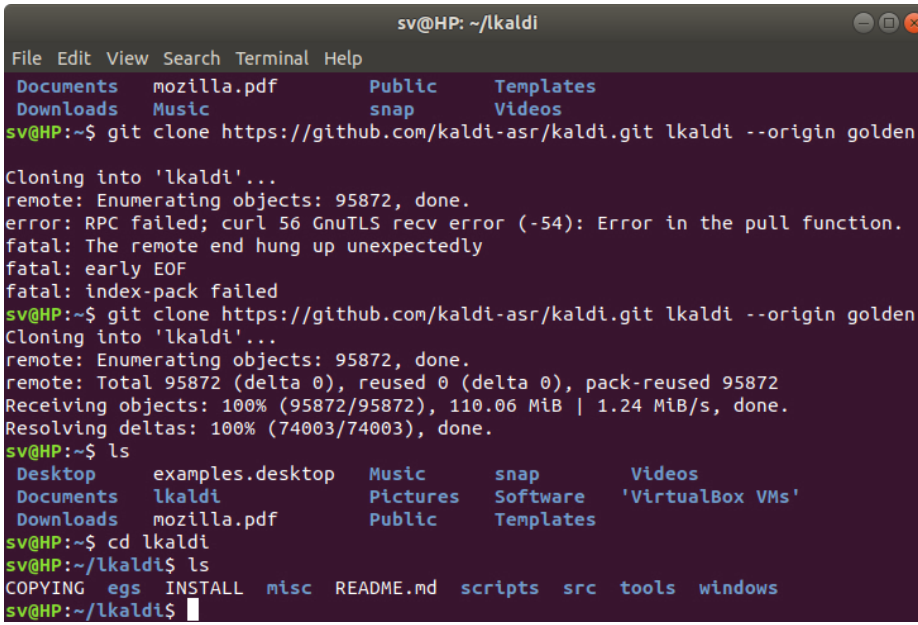
The output:

```
leo@Ubuntu:~$ git config --list
user.name=Leo Reny
user.email=leoreny@hypech.com
alias.co=checkout
alias.br=branch
alias.st=status
```

2.3 Install and Configure Kaldi

We clone Kaldi from Git to local folder .\lKaldi. We named local Kaldi as golden. Taking about 2 minutes.

```
$git clone https://github.com/Kaldi-asr/Kaldi.git lKaldi --origin golden
```



Once the cloning is finished, go inside the Kaldi folder. There, the INSTALL file explains the installation steps. Cat the file and see what's included.

Kaldi deals with many file formats, among which:

.sh for bash scripts

.py for Python scripts

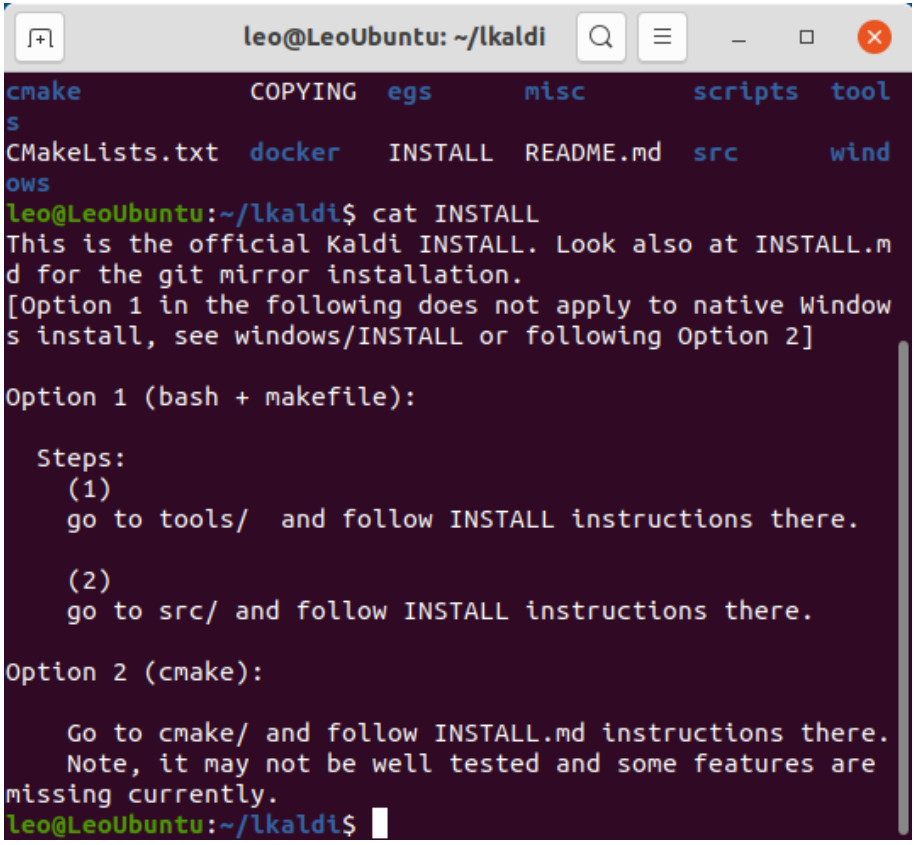
.cc for C++ code

.h for header files, containing variables, functions... used by various C++ files

.pl for Perl scripts, useful to process text files

To check the installation requirements, we run:

```
$cat INSTALL
```



Now, going to tools to check the dependencies with:

```
$extras/check_dependencies.sh
```

Following the instructions to install the following apps:

To compile the Kaldi, we need install some apps.

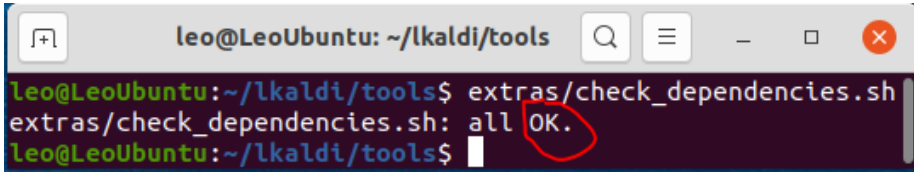
```
$sudo apt install make
$sudo apt-get install g++ automake autoconf sox gfor-
tran libtool subversion python2.7
$sudo apt-get install zlib1g-dev
$~/lKaldi/tools$ extras/install_irstlm.sh
```

```
$~/lKaldi/tools$ source env.sh
$~/lKaldi/tools$ sudo ./extras/install_mkl.sh
```

rerun:

```
$extras/check_dependencies.sh
```

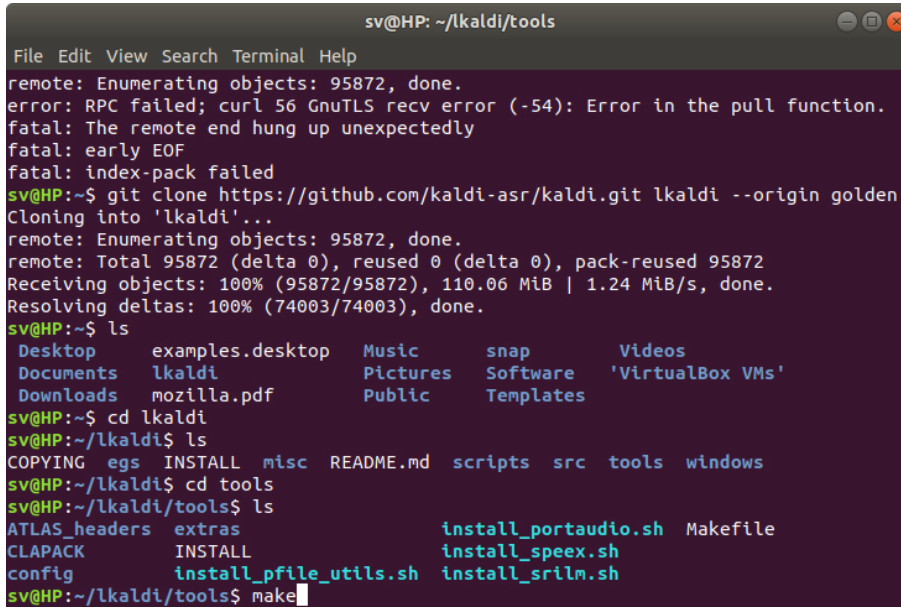
We'll get:



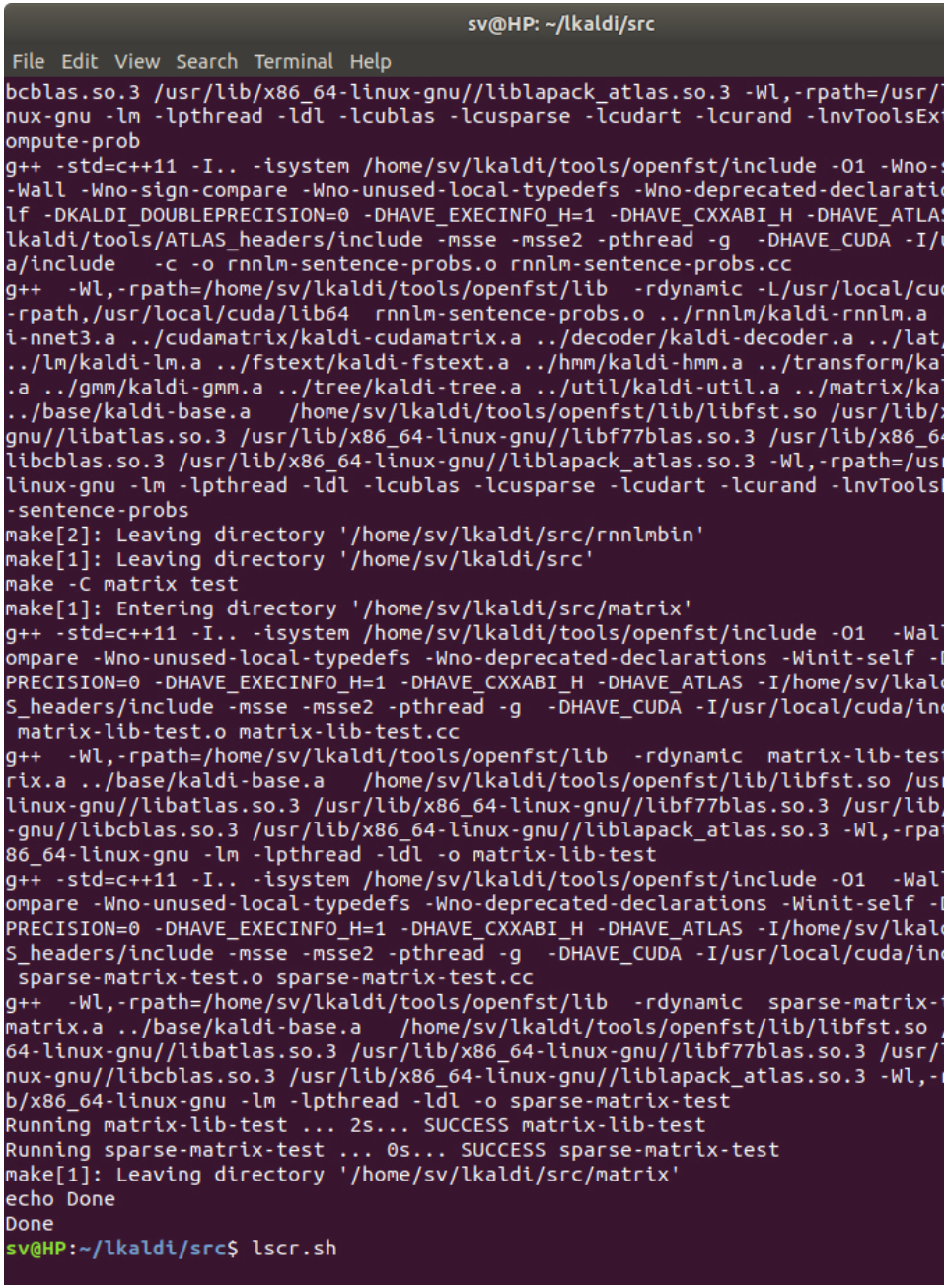
"OK" means ok.

We compile it. It might take hours.

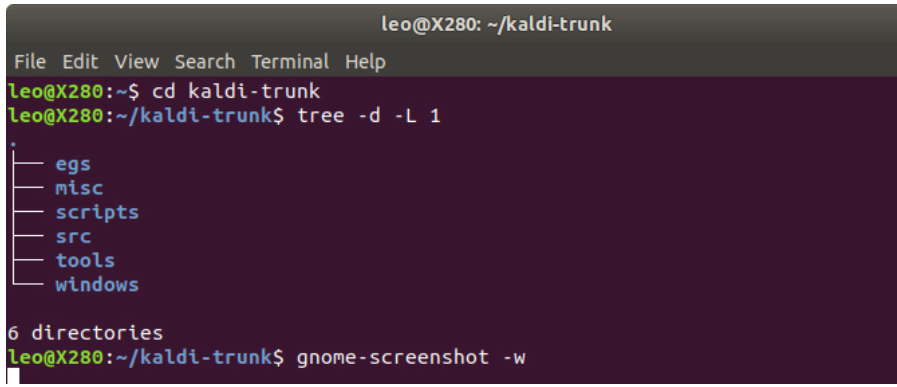
```
$cd lKaldi/tools/; make; cd ../src; ./configure; make
```



Following the instruction of what we missed, install all the components it required. After quite long time compiling, we finally make the Kaldi ready.



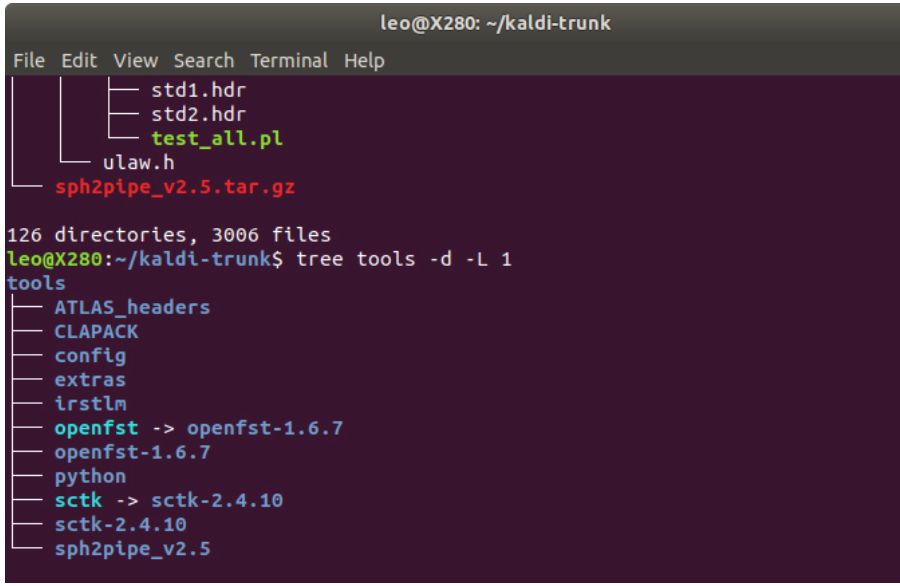
Let's check the directory structure:



The most important directories are:

- `egs`, which stands for examples
- `tools`, which contains Kaldi dependencies and setup instructions
- `src`, which contains the source code

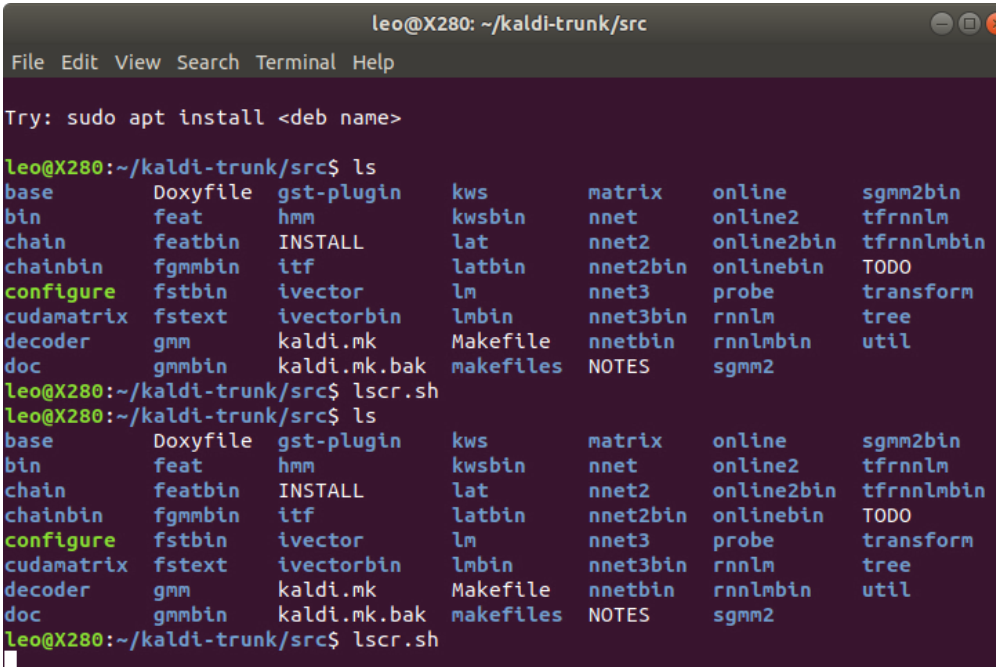
The samples were stored in the ./egs, which we will use frequently in this book. Let's check tools and src here:



WFST is popular in modeling ASR transducers. One popular open-source WFST toolkit is the OpenFst and it is heavily used by Kaldi. To use WFST in OpenFST, we need to define the input and output symbols and the FST definition. The symbol file contains the vocabulary and maps words into unique IDs used by OpenFST.

OpenFst is a library for constructing, combining, optimizing, and searching weighted finite-state transducers (FSTs). FSTs have key applications in speech recognition and synthesis.

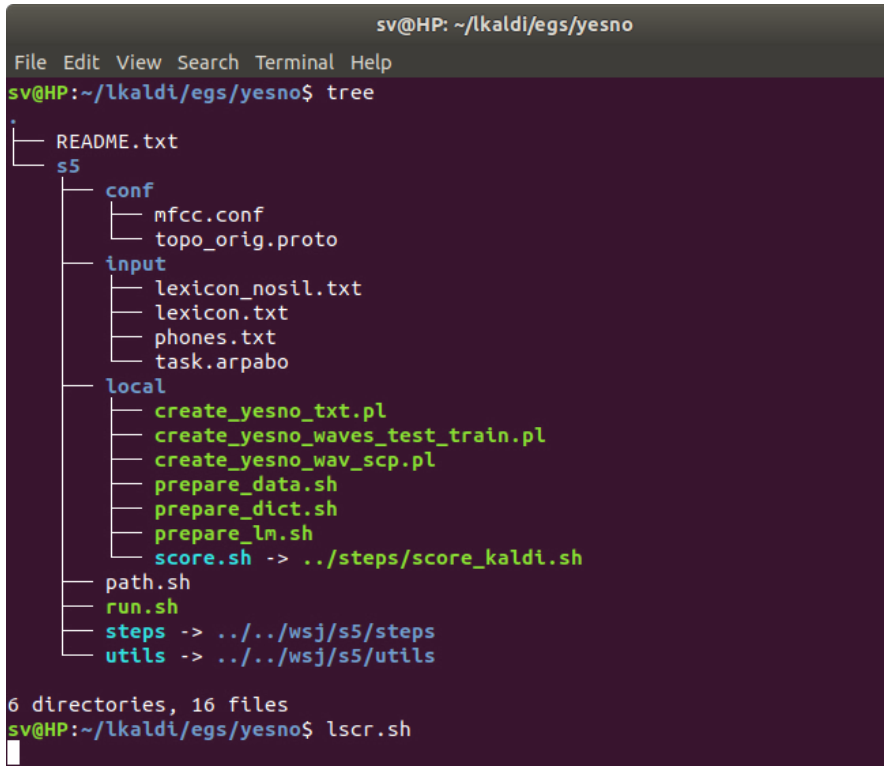
Below is src details:



With these information in hand, we are comfortable about the Kaldi installation. Let's play a test run in next section, Yes or No?

2.4 Yes or No

It's not a philosophy debate. It's a sample data provided by Kaldi. Let's check the folder.



We couldn't see the wav file. That's because Kaldi doesn't install it by default. No big deal, Kaldi will pull it when running.

```
$cd ~/lKaldi/egs/yesno/s5
$cd ~/Kaldi-trunk/egs/yesno/s5
$sudo ./run.sh
[sudo] password for sv:
--2019-01-03 09:01:50--  http://www.openslr.
org/resources/1/waves_yesno.tar.gz
Resolving www.openslr.org (www.openslr.org)... 46.101.158.64
Connecting to www.openslr.org (www.openslr.
org)|46.101.158.64|:80.. connected.
```

```
HTTP request sent, awaiting response... 302 Found
Location: http://101.96.8.160/www.openslr.org/re-
sources/1/waves_yesno.tar.gz [following]
--2019-01-03 09:01:51--http://101.96.8.160/www.
openslr.org/resources/1/waves_yesno.tar.gz
Connecting to 101.96.8.160:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4703754 (4.5M) [application/x-gzip]
Saving to: 'waves_yesno.tar.gz'
waves_yesno.tar.gz  100%[===================>]
4.49M   313KB/s    in 15s

2019-01-03 09:02:07 (304 KB/s) - 'waves_yes-
no.tar.gz' saved [4703754/4703754]
waves_yesno/
waves_yesno/1_0_0_0_0_0_1_1.wav
waves_yesno/1_1_0_0_1_0_1_0.wav
waves_yesno/1_0_1_1_1_1_0_1.wav
waves_yesno/1_1_1_1_0_1_0_0.wav
waves_yesno/0_0_1_1_1_0_0_0.wav
waves_yesno/0_1_1_1_1_1_1_1.wav
waves_yesno/0_1_0_1_1_1_0_0.wav
waves_yesno/1_0_1_1_1_0_1_0.wav
waves_yesno/1_0_0_1_0_1_1_1.wav
waves_yesno/0_0_1_0_1_0_0_0.wav
waves_yesno/0_1_0_1_1_0_1_0.wav
waves_yesno/0_0_1_1_0_1_1_0.wav
waves_yesno/1_0_0_0_1_0_0_1.wav
waves_yesno/1_1_0_1_1_1_1_0.wav
waves_yesno/0_0_1_1_1_1_0_0.wav
waves_yesno/1_1_0_0_1_1_1_0.wav
waves_yesno/0_0_1_1_0_1_1_1.wav
waves_yesno/1_1_0_1_0_1_1_0.wav
waves_yesno/0_1_0_0_0_1_1_0.wav
waves_yesno/0_0_0_1_0_0_0_1.wav
waves_yesno/0_0_1_0_1_0_1_1.wav
waves_yesno/0_0_1_0_0_0_1_0.wav
```

```
waves_yesno/1_1_0_1_1_0_0_1.wav
waves_yesno/0_1_1_1_0_1_0_1.wav
waves_yesno/0_1_1_1_0_0_0_0.wav
waves_yesno/README~
waves_yesno/0_1_0_0_0_1_0_0.wav
waves_yesno/1_0_0_0_0_0_0_1.wav
waves_yesno/1_1_0_1_1_0_1_1.wav
waves_yesno/1_1_0_0_0_0_0_1.wav
waves_yesno/1_0_0_0_0_0_0_0.wav
waves_yesno/0_1_1_1_1_0_1_0.wav
waves_yesno/0_0_1_1_0_1_0_0.wav
waves_yesno/1_1_1_0_0_0_0_1.wav
waves_yesno/1_0_1_0_1_0_0_1.wav
waves_yesno/0_1_0_0_1_0_1_1.wav
waves_yesno/0_0_1_1_1_1_1_0.wav
waves_yesno/1_1_0_0_0_1_1_1.wav
waves_yesno/0_1_1_1_0_0_1_0.wav
waves_yesno/1_1_0_1_0_1_0_0.wav
waves_yesno/1_1_1_1_1_1_1_1.wav
waves_yesno/0_0_1_0_1_0_0_1.wav
waves_yesno/1_1_1_1_0_0_1_0.wav
waves_yesno/0_0_1_1_1_0_0_1.wav
waves_yesno/0_1_0_1_0_0_0_0.wav
waves_yesno/1_1_1_1_1_0_0_0.wav
waves_yesno/README
waves_yesno/0_1_1_0_0_1_1_1.wav
waves_yesno/0_0_1_0_0_1_1_0.wav
waves_yesno/1_1_0_0_1_0_1_1.wav
waves_yesno/1_1_1_0_0_1_0_1.wav
waves_yesno/0_0_1_0_0_1_1_1.wav
waves_yesno/0_0_1_1_0_0_0_1.wav
waves_yesno/1_0_1_1_0_1_1_1.wav
waves_yesno/1_1_1_0_1_0_1_0.wav
waves_yesno/1_1_1_0_1_0_1_1.wav
waves_yesno/0_1_0_0_1_0_1_0.wav
waves_yesno/1_1_1_0_0_1_1_1.wav
```

```
waves_yesno/0_1_1_0_0_1_1_0.wav
waves_yesno/0_0_0_1_0_1_1_0.wav
waves_yesno/1_1_1_1_1_1_0_0.wav
waves_yesno/0_0_0_0_1_1_1_1.wav
Preparing train and test data
Dictionary preparation succeeded
utils/prepare_lang.sh --position-dependent-phones false
data/local/dict <SIL> data/local/lang data/lang
Checking data/local/dict/silence_phones.txt ...
--> reading data/local/dict/silence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/silence_phones.txt is OK

Checking data/local/dict/optional_silence.txt ...
--> reading data/local/dict/optional_silence.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/optional_silence.txt is OK

Checking data/local/dict/nonsilence_phones.txt ...
--> reading data/local/dict/nonsilence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/nonsilence_phones.txt is OK

Checking disjoint: silence_phones.txt, nonsilence_phones.txt
--> disjoint property is OK.

Checking data/local/dict/lexicon.txt
--> reading data/local/dict/lexicon.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/lexicon.txt is OK

Checking data/local/dict/extra_questions.txt ...
```

```
--> data/local/dict/extra_questions.txt is empty (this is OK)
--> SUCCESS [validating dictionary directory data/local/dict]

**Creating data/local/dict/lexiconp.
txt from data/local/dict/lexicon.txt
fstaddselfloops data/lang/phones/wdisambig_phones.
int data/lang/phones/wdisambig_words.int
prepare_lang.sh: validating output directory
utils/validate_lang.pl data/lang
Checking data/lang/phones.txt ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/phones.txt is OK

Checking words.txt: #0 ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/words.txt is OK

Checking disjoint: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> silence.txt and nonsilence.txt are disjoint
--> silence.txt and disambig.txt are disjoint
--> disambig.txt and nonsilence.txt are disjoint
--> disjoint property is OK

Checking sumation: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> found no unexplainable phones in phones.txt

Checking data/lang/phones/context_indep.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.int corre-
sponds to data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.csl corre-
sponds to data/lang/phones/context_indep.txt
```

```
--> data/lang/phones/context_indep.{txt, int, csl} are OK


Checking data/lang/phones/nonsilence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.int corre-
sponds to data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.csl corre-
sponds to data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.{txt, int, csl} are OK


Checking data/lang/phones/silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/silence.txt
--> data/lang/phones/silence.int corre-
sponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.csl corre-
sponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.{txt, int, csl} are OK


Checking data/lang/phones/optional_silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.int corre-
sponds to data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.csl corre-
sponds to data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.{txt, int, csl} are OK


Checking data/lang/phones/disambig.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang/phones/disambig.txt
--> data/lang/phones/disambig.int corre-
sponds to data/lang/phones/disambig.txt
```

```
--> data/lang/phones/disambig.csl corre-
sponds to data/lang/phones/disambig.txt
--> data/lang/phones/disambig.{txt, int, csl} are OK


Checking data/lang/phones/roots.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 3 entry/entries in data/lang/phones/roots.txt
--> data/lang/phones/roots.int corre-
sponds to data/lang/phones/roots.txt
--> data/lang/phones/roots.{txt, int} are OK


Checking data/lang/phones/sets.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 3 entry/entries in data/lang/phones/sets.txt
--> data/lang/phones/sets.int corre-
sponds to data/lang/phones/sets.txt
--> data/lang/phones/sets.{txt, int} are OK


Checking data/lang/phones/extra_questions.{txt, int} ...
Checking optional_silence.txt ...
--> reading data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.txt is OK


Checking disambiguation symbols: #0 and #1
--> data/lang/phones/disambig.txt has "#0" and "#1"
--> data/lang/phones/disambig.txt is OK


Checking topo ...


Checking word-level disambiguation symbols...
--> data/lang/phones/wdisambig.txt ex-
ists (newer prepare_lang.sh)
Checking data/lang/oov.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
```

```
--> 1 entry/entries in data/lang/oov.txt
--> data/lang/oov.int corresponds to data/lang/oov.txt
--> data/lang/oov.{txt, int} are OK

--> data/lang/L.fst is olabel sorted
--> data/lang/L_disambig.fst is olabel sorted
--> SUCCESS [validating lang directory data/lang]
Preparing language models for test
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/lang_
test_tg/words.txt input/task.arpabo data/lang_test_tg/G.fst
LOG (arpa2fst[5.5.164~1-9698]:Read():arpa-file-
parser.cc:94) Reading \data\ section.
LOG (arpa2fst[5.5.164~1-9698]:Read():arpa-file-
parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.164~1-9698]:RemoveRedundantStates():ar
pa-lm-compiler.cc:359) Reduced num-states from 1 to 1
fstisstochastic data/lang_test_tg/G.fst
1.20397 1.20397
Succeeded in formatting data.
steps/make_mfcc.sh --nj 1 data/train_yes-
no exp/make_mfcc/train_yesno mfcc
utils/validate_data_dir.sh: WARNING: you have only
one speaker.  This probably a bad idea.
Search for the word 'bold' in http://Kaldi-asr.
org/doc/data_prep.html for more information.
utils/validate_data_dir.sh: Successfully vali-
dated data-directory data/train_yesno
steps/make_mfcc.sh: [info]: no segments file ex-
ists: assuming wav.scp indexed by utterance.
Succeeded creating MFCC features for train_yesno
steps/compute_cmvn_stats.sh data/train_yes-
no exp/make_mfcc/train_yesno mfcc
Succeeded creating CMVN stats for train_yesno
fix_data_dir.sh: kept all 31 utterances.
fix_data_dir.sh: old files are kept in data/train_yesno/.backup
steps/make_mfcc.sh --nj 1 data/test_yes-
no exp/make_mfcc/test_yesno mfcc
utils/validate_data_dir.sh: WARNING: you have only
one speaker.  This probably a bad idea.
   Search for the word 'bold' in http://Kal-
```

```
di-asr.org/doc/data_prep.html

   for more information.

utils/validate_data_dir.sh: Successfully vali-
dated data-directory data/test_yesno

steps/make_mfcc.sh: [info]: no segments file ex-
ists: assuming wav.scp indexed by utterance.

It seems not all of the feature files were suc-
cessfully processed (29 != 31);

consider using utils/fix_data_dir.sh data/test_yesno

Less than 95% the features were successful-
ly generated.  Probably a serious error.

steps/compute_cmvn_stats.sh data/test_yes-
no exp/make_mfcc/test_yesno mfcc

Succeeded creating CMVN stats for test_yesno

fix_data_dir.sh: kept 29 utterances out of 31

fix_data_dir.sh: old files are kept in data/test_yesno/.backup

steps/train_mono.sh --nj 1 --cmd utils/run.pl --tot-
gauss 400 data/train_yesno data/lang exp/mono0a

steps/train_mono.sh: Initializing monophone system.

steps/train_mono.sh: Compiling training graphs

steps/train_mono.sh: Aligning data equally (pass 0)

steps/train_mono.sh: Pass 1

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 2

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 3

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 4

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 5

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 6

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 7

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 8

steps/train_mono.sh: Aligning data

steps/train_mono.sh: Pass 9
```

```
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 10
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 11
steps/train_mono.sh: Pass 12
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 13
steps/train_mono.sh: Pass 14
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 15
steps/train_mono.sh: Pass 16
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 17
steps/train_mono.sh: Pass 18
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 19
steps/train_mono.sh: Pass 20
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 21
steps/train_mono.sh: Pass 22
steps/train_mono.sh: Pass 23
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 24
steps/train_mono.sh: Pass 25
steps/train_mono.sh: Pass 26
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 27
steps/train_mono.sh: Pass 28
steps/train_mono.sh: Pass 29
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 30
steps/train_mono.sh: Pass 31
steps/train_mono.sh: Pass 32
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 33
steps/train_mono.sh: Pass 34
```

```
steps/train_mono.sh: Pass 35
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 36
steps/train_mono.sh: Pass 37
steps/train_mono.sh: Pass 38
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 39
steps/diagnostic/analyze_alignments.sh --cmd
utils/run.pl data/lang exp/mono0a
steps/diagnostic/analyze_alignments.sh: see stats
in exp/mono0a/log/analyze_alignments.log
1 warnings in exp/mono0a/log/update.*.log
exp/mono0a: nj=1 align prob=-81.88 over 0.05h [re-
try=0.0%, fail=0.0%] states=11 gauss=371
steps/train_mono.sh: Done training mono-
phone system in exp/mono0a
tree-info exp/mono0a/tree
tree-info exp/mono0a/tree
fsttablecompose data/lang_test_tg/L_dis-
ambig.fst data/lang_test_tg/G.fst
fstdeterminizestar --use-log=true
fstminimizeencoded
fstpushspecial
fstisstochastic data/lang_test_tg/tmp/LG.fst
0.534295 0.533859
[info]: LG not stochastic.
fstcomposecontext --context-size=1 --central-posi-
tion=0 --read-disambig-syms=data/lang_test_tg/phones/
disambig.int --write-disambig-syms=data/lang_test_tg/
tmp/disambig_ilabels_1_0.int data/lang_test_tg/tmp/
ilabels_1_0.24717 data/lang_test_tg/tmp/LG.fst
fstisstochastic data/lang_test_tg/tmp/CLG_1_0.fst
0.534295 0.533859
[info]: CLG not stochastic.
make-h-transducer --disambig-syms-out=exp/mono0a/graph_tgpr/
disambig_ tid.int --transition-scale=1.0 data/lang_test_tg/
tmp/ilabels_1_0 exp/mono0a/tree exp/mono0a/final.mdl
fsttablecompose exp/mono0a/graph_tgpr/
Ha.fst data/lang_test_tg/tmp/CLG_1_0.fst
fstdeterminizestar --use-log=true
```

```
fstminimizeencoded
fstrmsymbols exp/mono0a/graph_tgpr/disambig_tid.int
fstrmepslocal
fstisstochastic exp/mono0a/graph_tgpr/HCLGa.fst
0.5342 -0.000422432
HCLGa is not stochastic
add-self-loops --self-loop-scale=0.1 --reorder=true exp/
mono0a/final.mdl exp/mono0a/graph_tgpr/HCLGa.fst
steps/decode.sh --nj 1 --cmd utils/run.pl exp/mono0a/
graph_tgpr data/test_yesno exp/mono0a/decode_test_yesno
decode.sh: feature type is delta
steps/diagnostic/analyze_lats.sh --cmd utils/run.pl exp/
mono0a/graph_tgpr exp/mono0a/decode_test_yesno
steps/diagnostic/analyze_lats.sh: see stats in exp/
mono0a/decode_test_yesno/log/analyze_alignments.log
Overall, lattice depth (10,50,90-per-
centile)=(1,1,2) and mean=1.2
steps/diagnostic/analyze_lats.sh: see stats in exp/mono0a/
decode_test_yesno/log/analyze_lattice_depth_stats.log
local/score.sh --cmd utils/run.pl data/test_yesno exp/
mono0a/graph_tgpr exp/mono0a/decode_test_yesno
local/score.sh: scoring with word insertion penalty=0.0,0.5,1.0
%WER 0.00 [ 0 / 232, 0 ins, 0 del, 0 sub ] exp/
mono0a/decode_test_yesno/wer_10_0.0
```

The WER (word error rate) is 0, perfect. WER is widely accepted as the defacto metric for ASR. The lower the better.

```
Word Error Rate = (Substitutions + Insertions + Dele-
tions) / Number of Words Spoken
```

Substitutions are anytime a word gets replaced (for example, “twinkle” is transcribed as “crinkle”)

Insertions are anytime a word gets added that wasn’t said (for example, “trailblazers” becomes “tray all blazers”)

Deletions are anytime a word is omitted from the transcript (for example, “get it done” becomes “get done”)

Till now, we can make sure that Kaldi has been successfully in-

stalled. Let's check the folder again:

Clearly, data, exp, mfcc, and waves_yes-no have been added.

Chapter 3. Setup TIMIT

TIMIT Structure

TIMIT is often used as a benchmark for performance, although results on TIMIT are not always transferable to other corpora. TIMIT is available as LDC corpus LDC93S1, TIMIT is one of the original clean speech databases. (http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1).

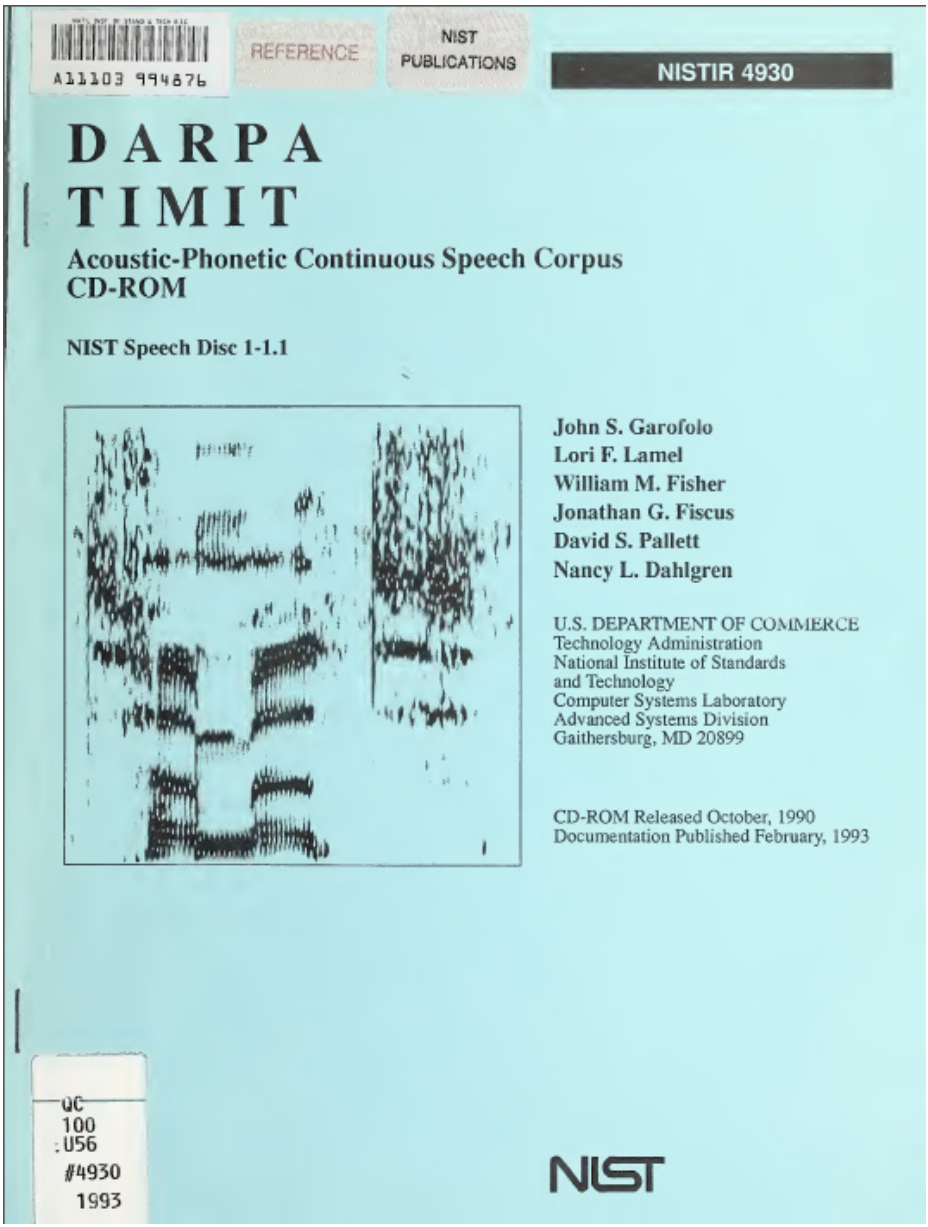
"The TIMIT corpus of read speech is designed to provide speech data for acoustic-phonetic studies and for the development and evaluation of automatic speech recognition systems. TIMIT contains broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The TIMIT corpus includes time-aligned orthographic, phonetic and word transcriptions as well as a 16-bit, 16kHz speech waveform file for each utterance."

"The TIMIT corpus transcriptions have been hand verified. Test and training subsets, balanced for phonetic and dialectal coverage, are specified. Tabular computer-searchable information is included as well as written documentation."

The zipped of TIMIT is about 442.21M, could be found at https://academictorrents.com/details/34e2b78745138186976cbc27939b1b34d18bd5b3 for free.

3.1 About TIMIT

TIMIT (TIMIT Acoustic-Phonetic Continuous Speech Corpus) has resulted from the joint efforts of several sites under sponsorship from the Defense Advanced Research Projects Agency - Information Science and Technology Office(DARPA-ISTO).



Text corpus design was a joint effort among the Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI), and Texas Instruments (TI). The speech was recorded at TI, transcribed at MIT, and has been maintained, verified, and prepared for CD-ROM production by the National Institute of Standards and Technology (NIST). Additional information including the referenced material and some relevant reprints of articles may be found in the printed documentation which is also available from NTIS (NTIS# PB91-100354).

TIMIT contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. Below table shows the number of speakers for the 8 dialect regions.

Dialect Region(dr)	#Male	#Female	Total
1	31 (63%)	18 (27%)	49 (8%)
2	71 (70%)	31 (30%)	102 (16%)
3	79 (67%)	23 (23%)	102 (16%)
4	69 (69%)	31 (31%)	100 (16%)
5	62 (63%)	36 (37%)	98 (16%)
6	30 (65%)	16 (35%)	46 (7%)
7	74 (74%)	26 (26%)	100 (16%)
8	22 (67%)	11 (33%)	33 (5%)
8	438 (70%)	192 (30%)	630 (100%)

The dialect regions are:

dr1: New England
dr2: Northern
dr3: North Midland
dr4: South Midland
dr5: Southern
dr6: New York City
dr7: Western
dr8: Army Brat (moved around)

The text material in the TIMIT prompts (found in the file "prompts.doc") consists of 2 dialect "shibboleth" sentences designed at SRI, 450 phonetically-compact sentences designed at MIT, and 1890 phonetically-diverse sentences selected at TI. The dialect sentences (the SA sentences) were meant to expose the dialectal variants of the speakers and were read by all 630 speakers. The phonetically-compact sentences were designed to provide a good coverage of pairs of phones, with extra occurrences of phonetic contexts thought to be either difficult or of particular interest. Each speaker read 5 of these sentences (the SX sentences) and each text was spoken by 7 different speakers. The phonetically-diverse sentences (the SI sentences) were selected from existing text sources - the Brown Corpus (Kuchera and Francis, 1967) and the Playwrights Dialog (Hultzen, et al., 1964) - so as to add diversity in sentence types and phonetic contexts. The selection criteria maximized the variety of allophonic contexts found in the texts. Each speaker read 3 of these sentences, with each sentence being read only by a single speaker. Below table summarizes the speech material in TIMIT.

Sentence Type	#Sentences	#Speakers	Total	#Sentences/Speaker
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Total	2342		6300	10

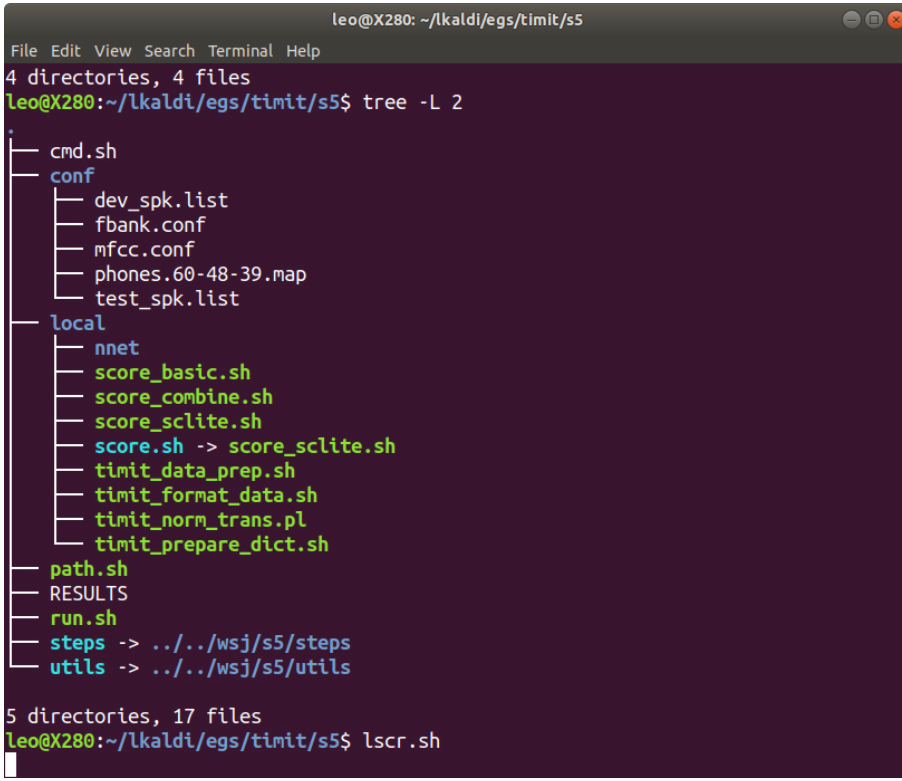
The speech material has been subdivided into portions for training and testing. The criteria for the subdivision is described in the file "testset.doc".

3.2 TIMIT Directory Structure and Files

Kaldi will install TIMIT directories and scripts. Only thing we need do is to download TIMIT database. Below screenshot shows the folder structure of once Kaldi has been successfully installed.

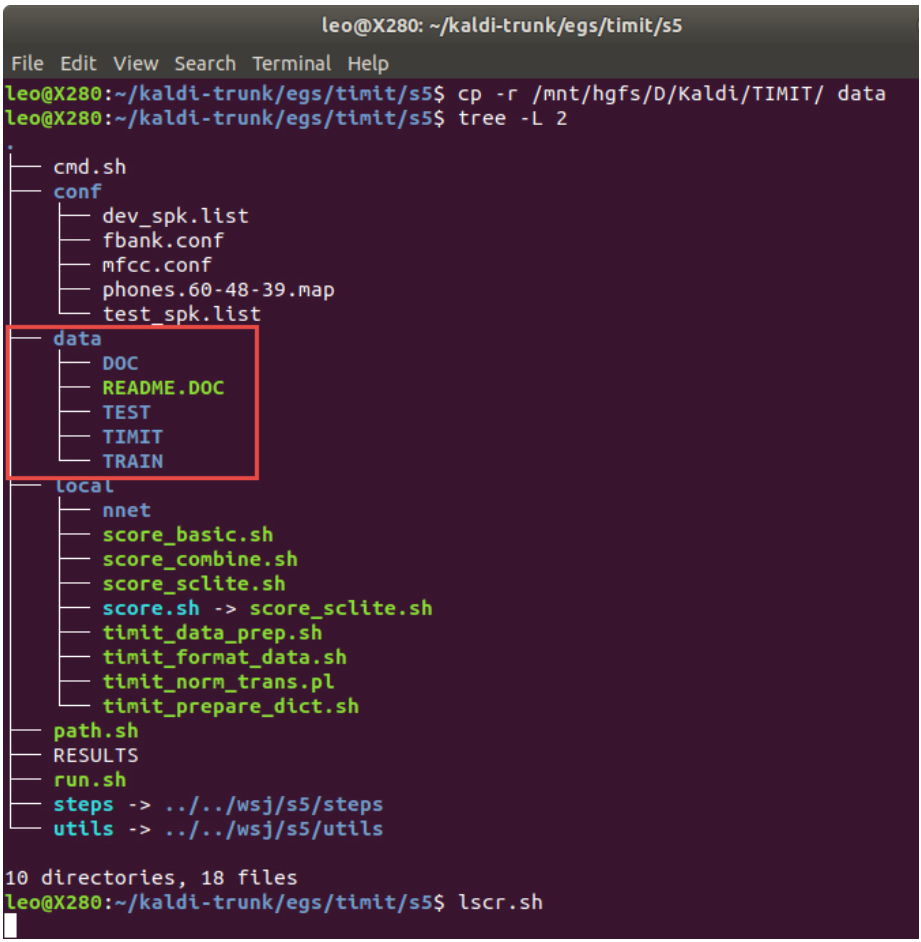
3.2.1 Top Level Directories and Files

`~/lKaldi/egs` stands for "examples", which contains example training recipes for some speech corpora like TIMIT. Under each of these folders are different versions (s3, s4, s5, etc.) The highest number, usually s5, is the most current version and should be used for any new development or training. Check the folder structure of `./egs/timit/s5`. All the folders are there except the `./data`.



Download TIMIT data (around 400M to 600M) from http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1, school library, or other sources and extract it to a local folder, rename the root folder to `data`, and put it into `./egs/timit/s5/data` as below.

We get the complete TIMIT in Kaldi now.



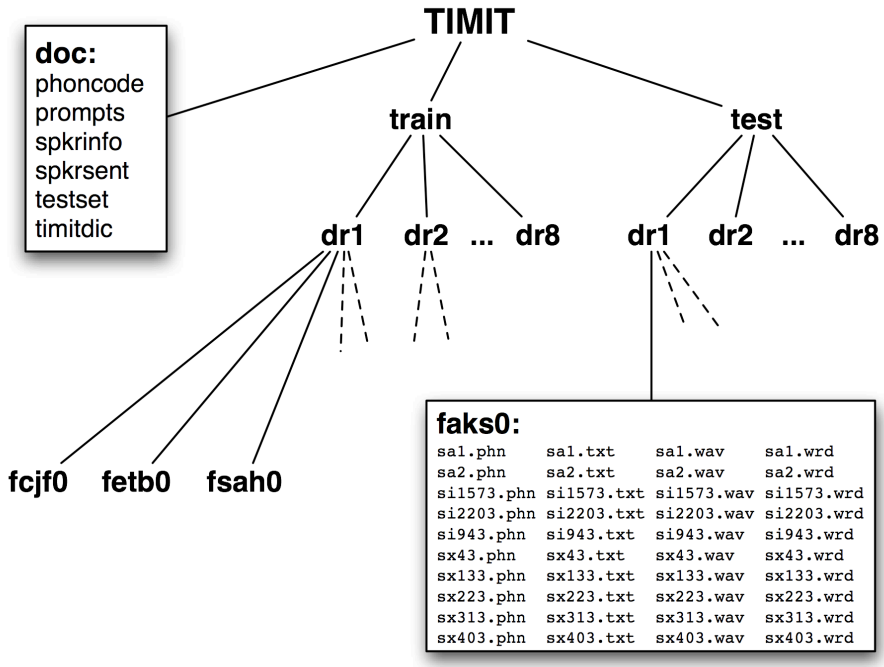
Once installation completed, read `./data/README.DOC` first, and find more information from `./data/DOC`.

The main folders and files we are going to use are:

`cmd.sh` contains all the parameters. `conf` contains configurations for certain scripts. `data` containing any data directories, such as a train and test directory for TIMIT. `local` this directory typically contains files that relate only to the corpus we're working on (e.g. TIMIT). `path.sh` contains the path to the Kaldi source directory

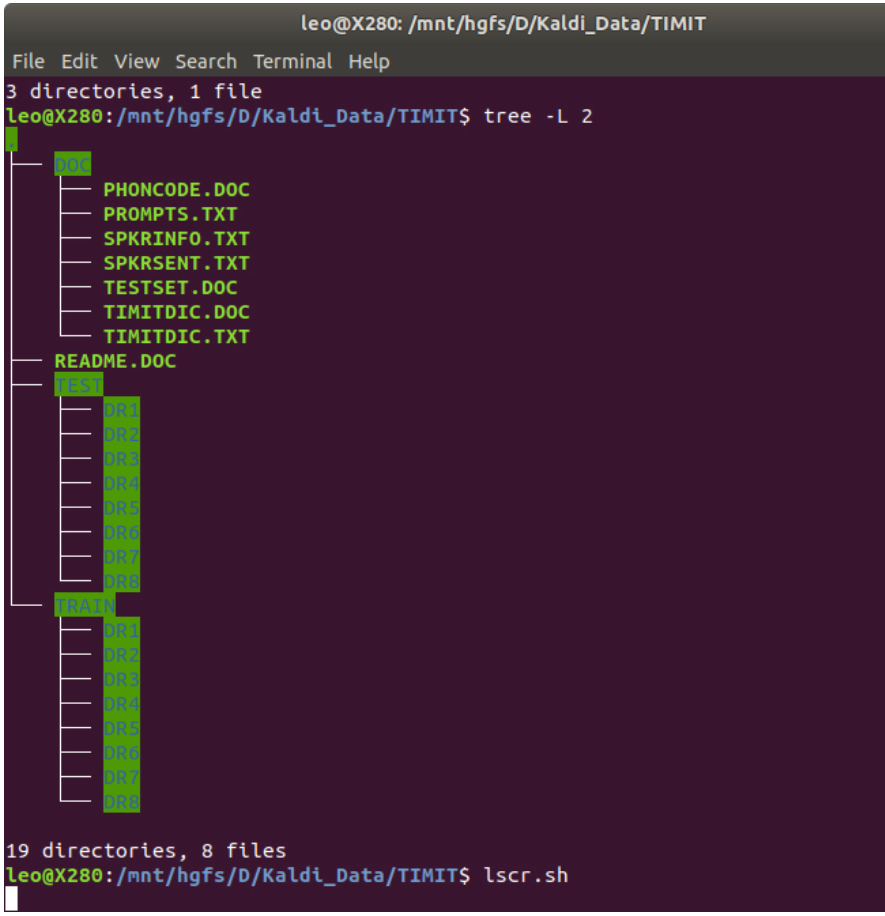
`steps` contains scripts for creating an ASR system. `utils` contains scripts to modify Kaldi files in certain ways, for example to subset data directories into smaller pieces. `exp` , which will be created later, contains the actual experiments and models, as well as logs.

`run.sh` TIMIT recipe is embodied in `run.sh` script, with supporting files in `./local`. This script will call high level scripts in `./steps` and `./utils`, which in turn call binaries in Kaldi which perform the actual computation. `steps` and `utils` are two folders linked to Kaldi.



3.2.2 TIMIT Corpus

The TIMIT corpus is located at ./Data. Below is the structure.



The folder structure is exactly same for `./TEST` and `./TRAIN`. Below them are 8 folders, `DR1-DR8`, representing 8 dialects.

Each dialect contains a couple of folders, representing different person's sound (F means female, M means male). Let's see one of them:

```
cd train/DR1/FCJF0
```

Entering one of them, we could find 10 utterances. Each has four files: .phn, .txt, .wav, and .wrd. Let's try to open them and see what's inside.

```
sv@HP: ~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0
File  Edit  View  Search  Terminal  Help
[1]+  Stopped                 less SA1.PHN
sv@HP:~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0$ head SA1.PHN
0 3050 h#
3050 4559 sh
4559 5723 ix
5723 6642 hv
6642 8772 eh
8772 9190 dcl
9190 10337 jh
10337 11517 ih
11517 12500 dcl
12500 12640 d
sv@HP:~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0$ head SA1.TXT
0 46797 She had your dark suit in greasy wash water all year.
sv@HP:~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0$ head SA1.WRD
3050 5723 she
5723 10337 had
9190 11517 your
11517 16334 dark
16334 21199 suit
21199 22560 in
22560 28064 greasy
28064 33360 wash
33754 37556 water
37556 40313 all
sv@HP:~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0$ head SA1.WAV
NIST_1A
   1024
database_id -s5 TIMIT
database_version -s3 1.0
utterance_id -s8 cjf0_sa1
channel_count -i 1
sample_count -i 46797
sample_rate -i 16000
sample_min -i -2191
sample_max -i 2790
sv@HP:~/T/Kaldi_Data/TIMIT/TRAIN/DR1/FCJF0$ lscr.sh
```

.wav is the audio file recorded with sound format (It's a little bit different. But let's assume it for now). Others are corresponding metadata like lexicon, XML metadata, and language model training text in txt format. The metadata and text files help Kaldi understand WAV and transfer the speech to feature vector and then to the character, word and language.

3.2.3 TIMIT Recipes

We have audio file .wav. We have metadata and training text files .phn, .wrd, and .txt. The next step is to put them together using a Recipe.

The recipe usually starts with `run.sh` in the root directory. Links to the Kaldi have been stored in `utils/` and `steps/` directories.

Edit the `run.sh` script to set paths to various tools and to specify the location of directories containing the WAV files, XML files, language model training text and Combilex lexicon.

Each Kaldi recipe consists of multiple stages, which simply means run the commands in this block if stage is less than or equal to number x. If choosing equal to a stage number, Kaldi will run only that stage.

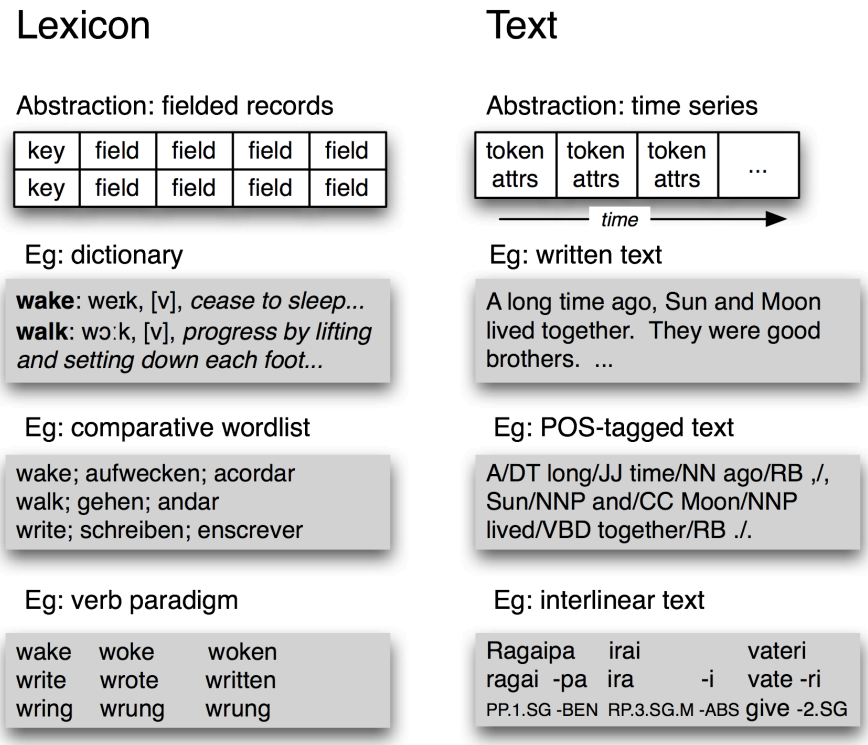
```
# Set which stage this script is on, you can set it
to the stage number that has already been executed to
avoid running the same command repeatedly.
stage=0
```

Stage is set to 0 by default, which means the recipe will run all blocks. If encountering an error, we can check which stages are successfully passes and re-run the recipe by `./run.sh --stage x`, or set the stage equal to a stage number to run that specific stage only.

Chapter 4. Kaldi Data Preparation

from beginning to FST

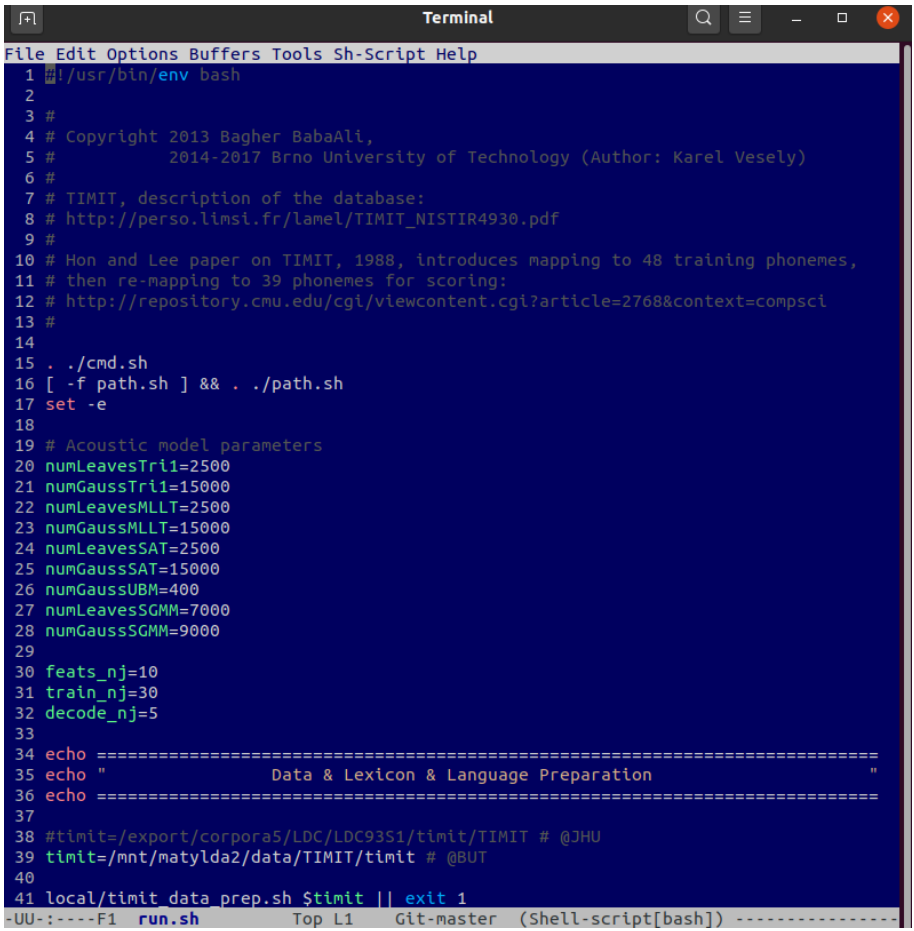
The speech data is the core data for speech recognition; The lexicon data is like a dictionary that include a word dictionary (word-pronunciation pairs), phonemes dictionary (types of phonemes); The language data is the language model (mostly the language model is trained apart from the text of speech data). Mostly, a dataset for ASR is accompanied with a language model.



For the TIMIT dataset specifically, the three kinds of data are integrated in the dataset, we can easily get them from scripts in local.

Before acting with the code, read the TIMIT description (some docs in the dataset. The data collection process is frustrating, and all the datasets are built with great efforts).

The start point of TIMIT training is `run.sh`, being located at the root of TIMIT. We will run `run.sh` step by step till DNN.

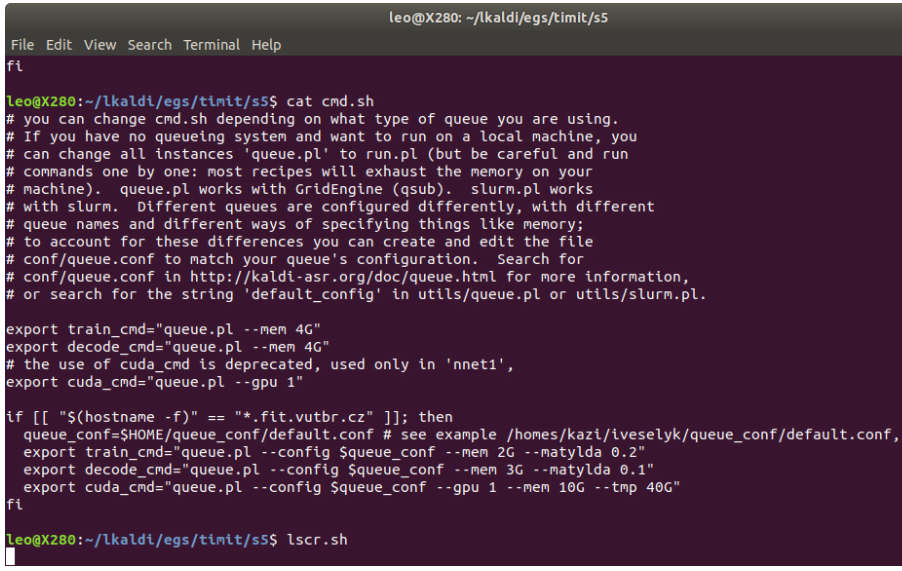


Step 1: Setup Environment (line 1-32)

The first 14 lines are comment. Line 15 source the `cmd.sh`.

TIMIT uses `cmd.sh` to setup the environment variables. "Dot space dot" format makes sure that `cmd.sh` could reserver the current parameters while return the parameters, similar with adding a source at the begining. If just use "dot" to run, after running the `cmd.sh`, the system quit the shell, the environment varaibles are not kept, which means we did nothing.

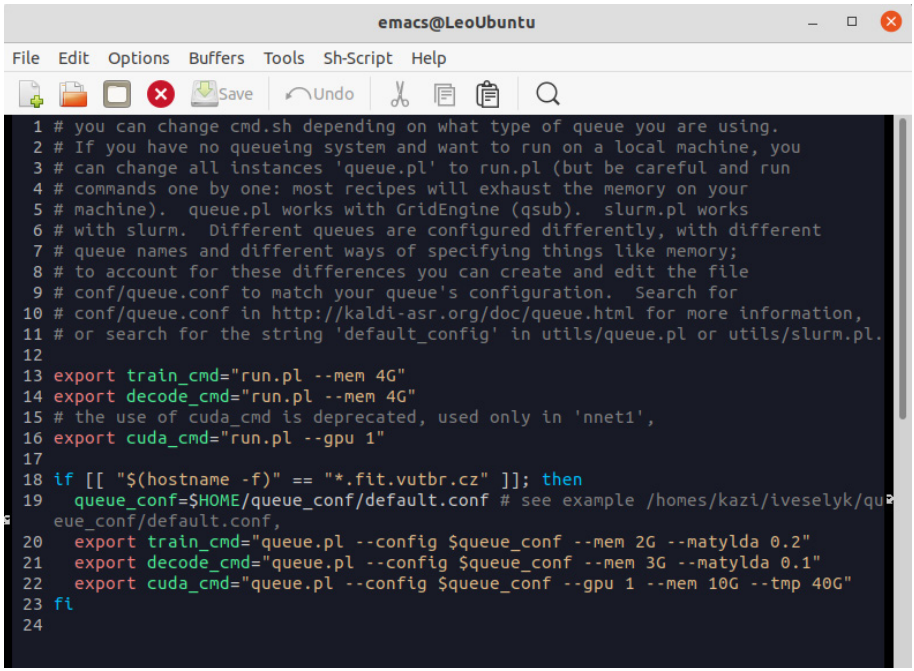
The `cmd.sh` reads:



We are running the TIMIT at our local machine, so we need change queue.pl to run.pl (queue.pl is for Oracle GridEngine, it's out of our scope).

```
export train_cmd="run.pl --mem 4G"
export decode_cmd="run.pl --mem 4G"
export cuda_cmd="run.pl --gpu 1"
```

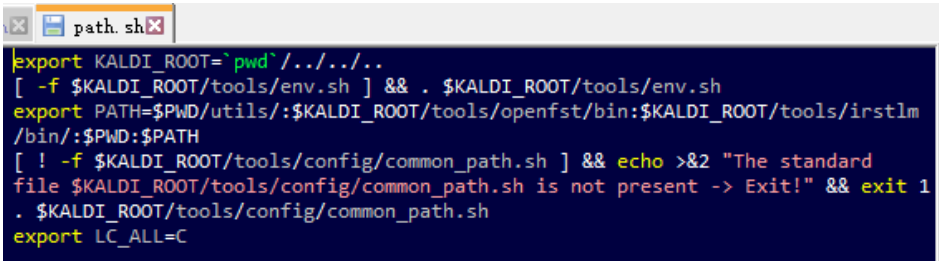
The final `cmd.sh` should look like this:



Line 16-17 of `run.sh` are:

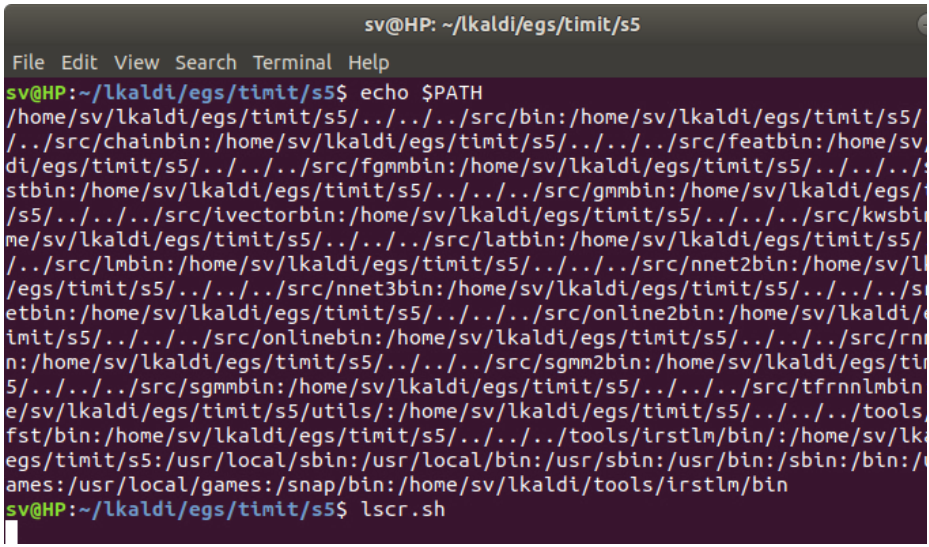
```
[ -f path.sh ] && . ./path.sh
set -e
```

The binary and executive files of Kaldi have been stored into different folders with a logic. `path.sh` has been used to make link them to the TIMIT.



Above is the `path.sh`.

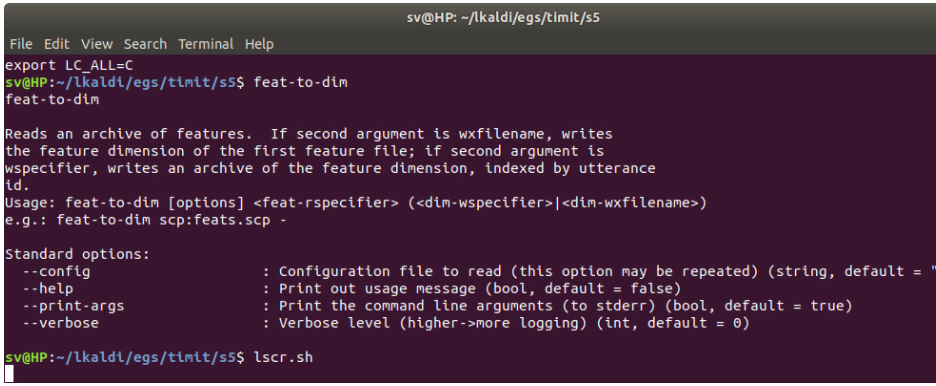
We could see Kaldi's complicated path definition below. This is why the `path.sh` file is called at the beginning of all Kaldi scripts.



set -e means stop for any error. Just run any binary file to check whether `path.sh` has been set correctly:

```
feat-to-dim
```

If like below, it's good then.



Even though we couldn't understand what it showing, we already know that the paths have been all set.

Keep the acoustic model (line 18-32) parameters as is.

Step 2: Prepare Data (line 33-50)

Line 39 points out the TIMIT data location. Change it to our own computer:

```
$ timit=/home/leo/lKaldi/egs/timit/s5/data #data path
```

```
33
34 echo ============================================================================
35 echo "                Data & Lexicon & Language Preparation                     "
36 echo ============================================================================
37 #timit=/export/corpora5/LDC/LDC93S1/timit/TIMIT # @JHU
38 #timit=/mnt/matylda2/data/TIMIT/timit # @BUT
39 timit=/home/leo/lkaldi/egs/timit/s5/data/ #LEO
40
41 local/timit_data_prep.sh $timit || exit 1
42
43 local/timit_prepare_dict.sh
44
45 # Caution below: we remove optional silence by setting "--sil-prob 0.0",
46 # in TIMIT the silence appears also as a word in the dictionary and is scored.
47 utils/prepare_lang.sh --sil-prob 0.0 --position-dependent-phones false --num-sil-states 3 \
48  data/local/dict "sil" data/local/lang_tmp data/lang
49
50 local/timit_format_data.sh
51
```

Keep running `timit_data_prep.sh` (line 41), we will get:

```
$ local/timit_data_prep.sh $timit || exit 1 #prepare data
```

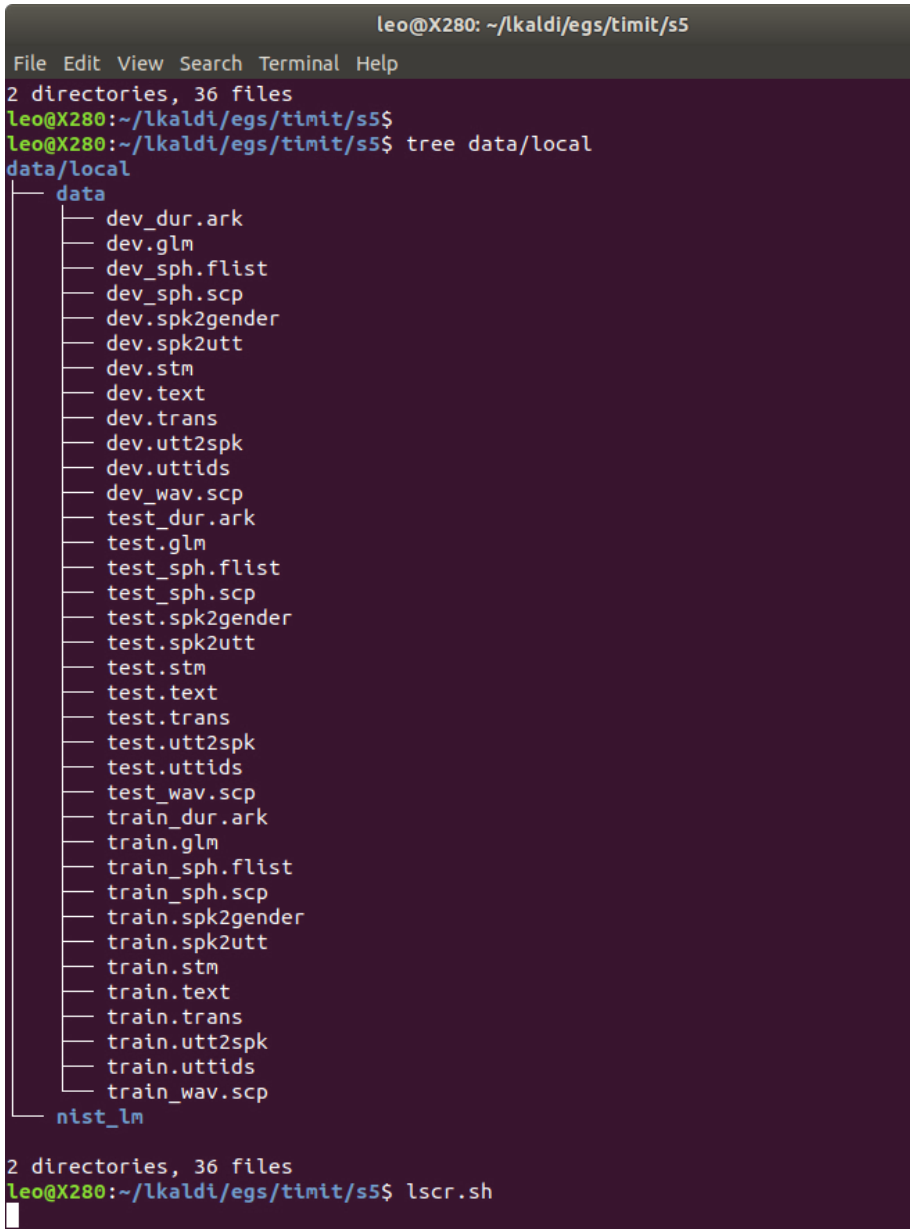
```
leo@X280: ~/lkaldi/egs/timit/s5
File  Edit  View  Search  Terminal  Help
local/timit_prepare_dict.sh

# Caution below: we remove optional silence by setting "--sil-prob 0.0",
# in TIMIT the silence appears also as a word in the dictionary and is scored.
utils/prepare_lang.sh --sil-prob 0.0 --position-dependent-phones false --num-sil-states 3 \
--More--
[1]+  Stopped                 cat run.sh | more
leo@X280:~/lkaldi/egs/timit/s5$ timit=/home/leo/lkaldi/egs/timit/s5/data
leo@X280:~/lkaldi/egs/timit/s5$ local/timit_data_prep.sh $timit || exit 1
wav-to-duration --read-entire-file=true scp:train_wav.scp ark,t:train_dur.ark
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:92) Printed duration for 154 audio files.
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:94) Mean duration was 2.9966, min and max durations
e 1.23525, 6.38081
wav-to-duration --read-entire-file=true scp:dev_wav.scp ark,t:dev_dur.ark
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:92) Printed duration for 400 audio files.
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:94) Mean duration was 3.08212, min and max duration
re 1.09444, 7.43681
wav-to-duration --read-entire-file=true scp:test_wav.scp ark,t:test_dur.ark
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:92) Printed duration for 192 audio files.
LOG (wav-to-duration[5.5.125~1-70aba]:main():wav-to-duration.cc:94) Mean duration was 3.03646, min and max duration
re 1.30562, 6.21444
Data preparation succeeded
leo@X280:~/lkaldi/egs/timit/s5$ lscr sh
```

As we could see, `timit_data_prep.sh` creates `local` folder under `data` to check and integrate the audio wav files and meta-data of the wav files, make it ready for further processing. The processing result will be stored to `./data/local/data`.

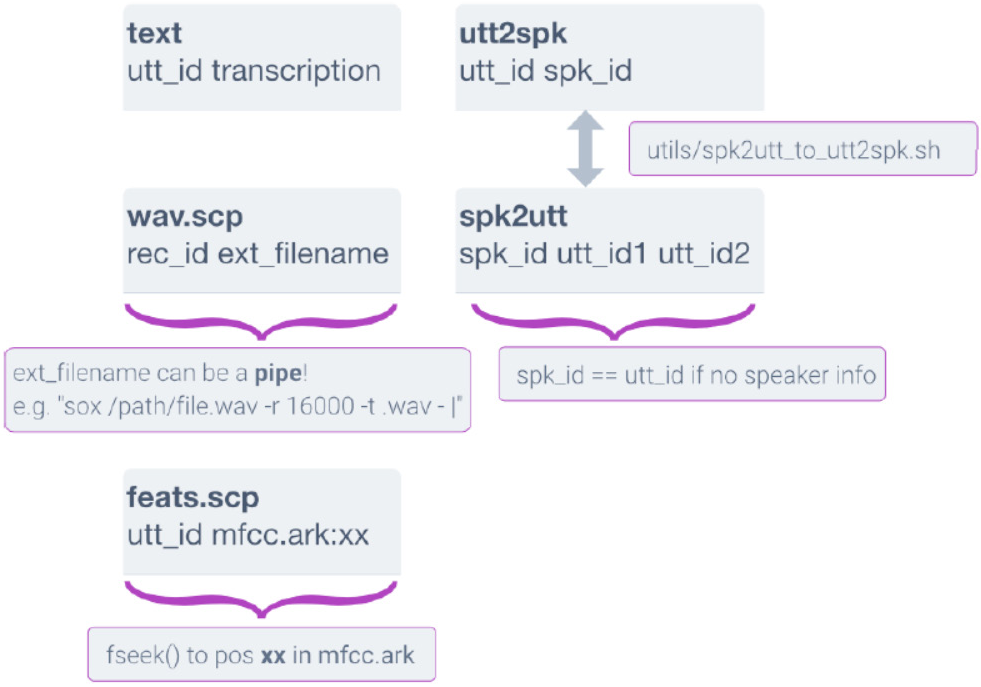
After data preparation, each data set (train, test, dev) will produce _sph.flist, _sph.scp, .uttids, .trans, .text, _wav.scp, .utt2spk,

.spk2utt, .spk2gender, .stm, and .glm.

In Kaldi, .wav is acctually .sph format. Want to really listen the wave, need run `tools/sph2pipe_v2.5/sph2pipetimit_data_prep.sh` to convert.



These files are linked together by utterance (utt_id) and speak (spk_id).



Kaldi provides two scripts to check the data preparation:

```
$ utils/validate_data_dir.sh data/train
$ utils/validate_data_dir.sh data/test
```

If everything fine, we will see:

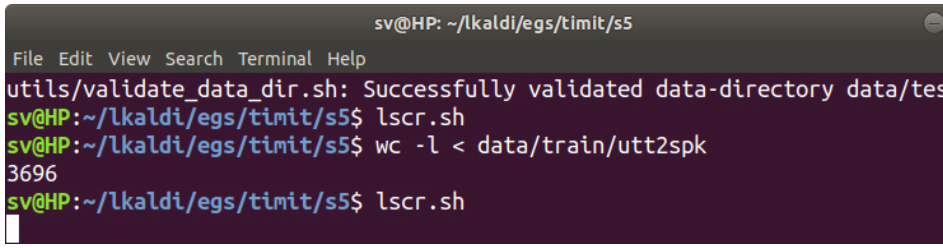
```
sv@HP: ~/lkaldi/egs/timit/s5
File  Edit  View  Search  Terminal  Help
sv@HP:~/lkaldi/egs/timit/s5$ utils/validate_data_dir.sh data/train
utils/validate_data_dir.sh: Successfully validated data-directory data/train
sv@HP:~/lkaldi/egs/timit/s5$ utils/validate_data_dir.sh data/test
utils/validate_data_dir.sh: Successfully validated data-directory data/test
sv@HP:~/lkaldi/egs/timit/s5$ lscr.sh
```

The common mistakes could be missing utt2spk. If so, correcting it by running `utils/spk2utt_to_utt2spk.pl`.

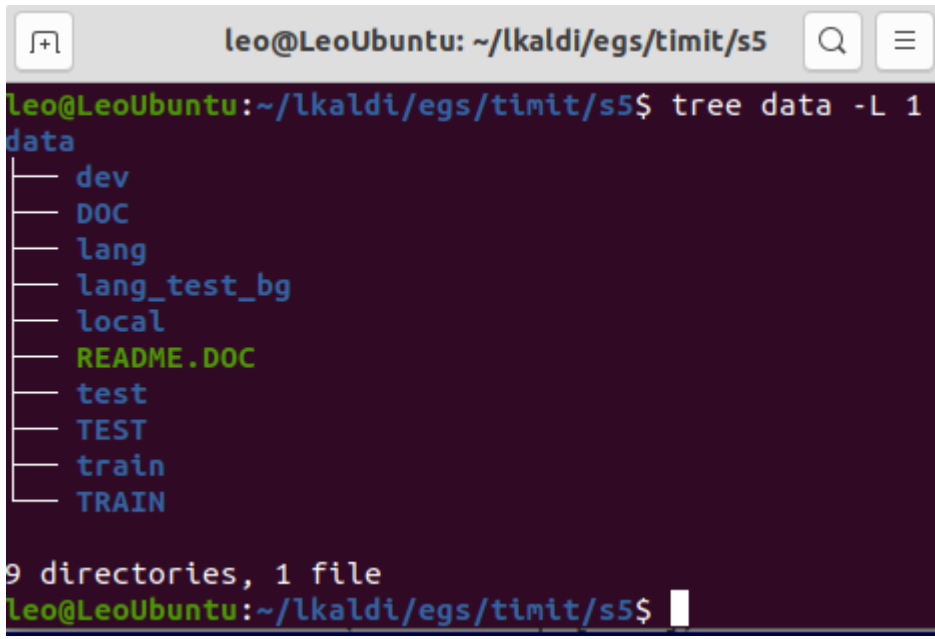
We could know the number of sentences in the system by using wc command.

```
wc -l < data/train/utt2spk
```

yes, we have 3696 sentences here.

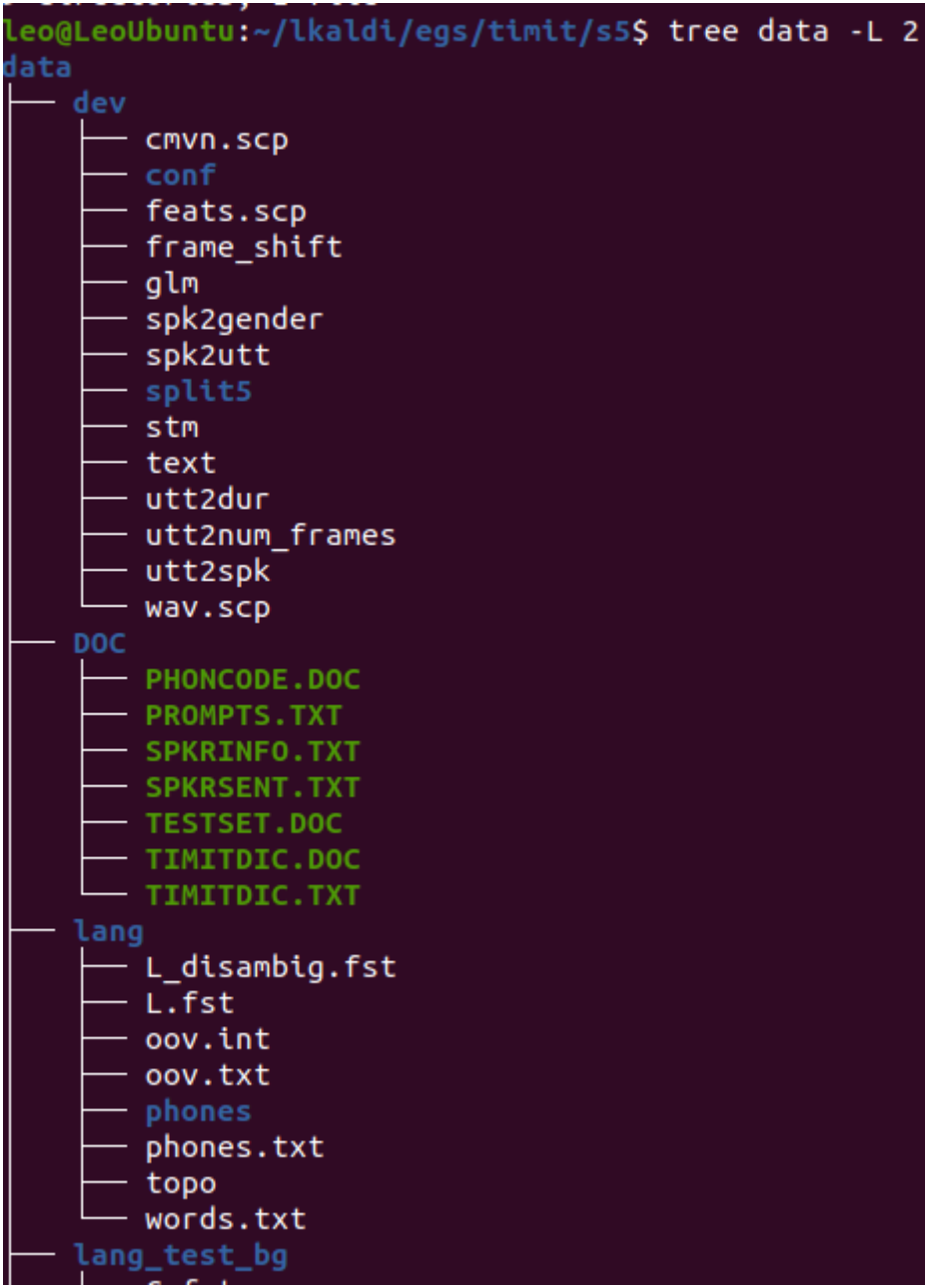


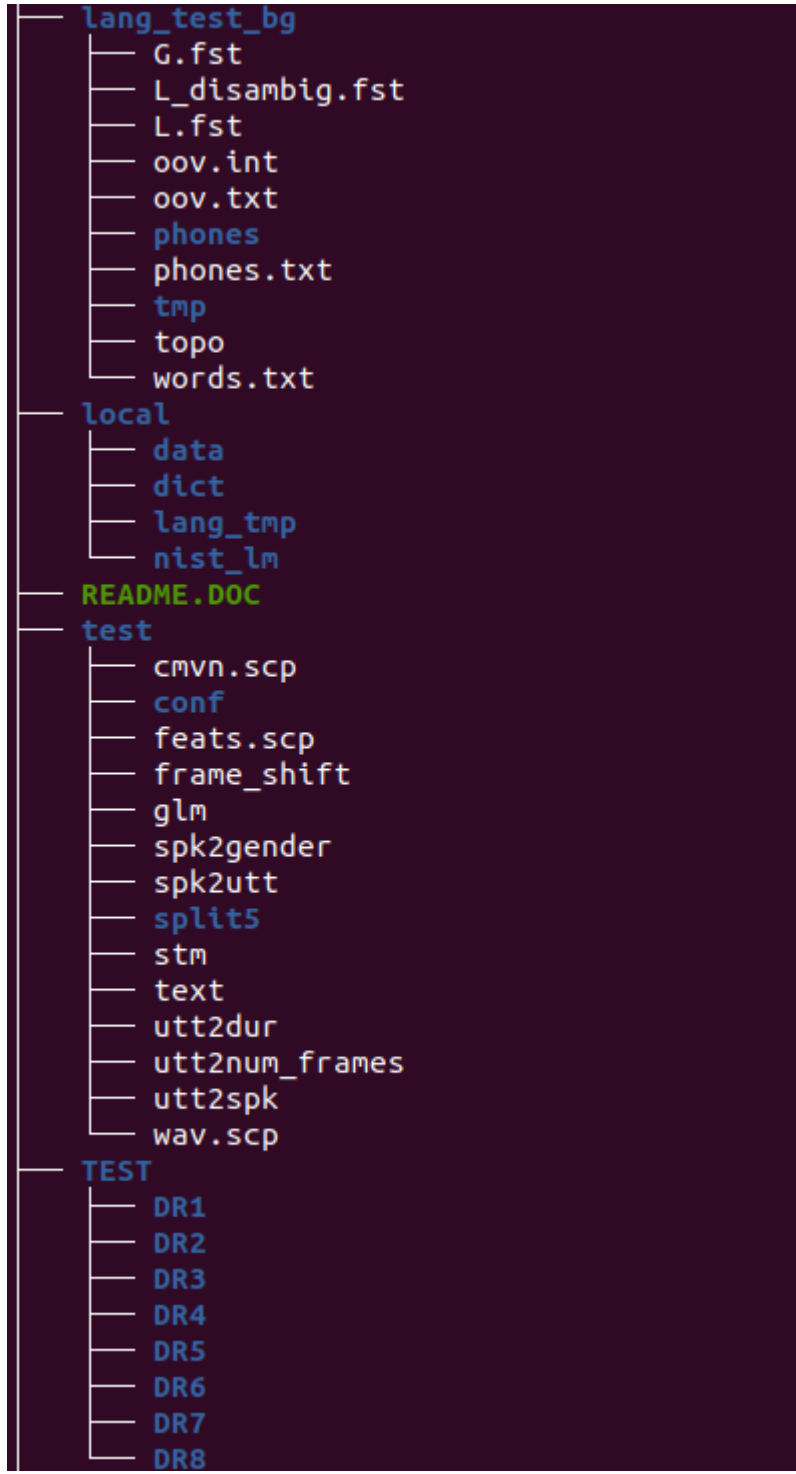
Here is the new folder structure (level 1):

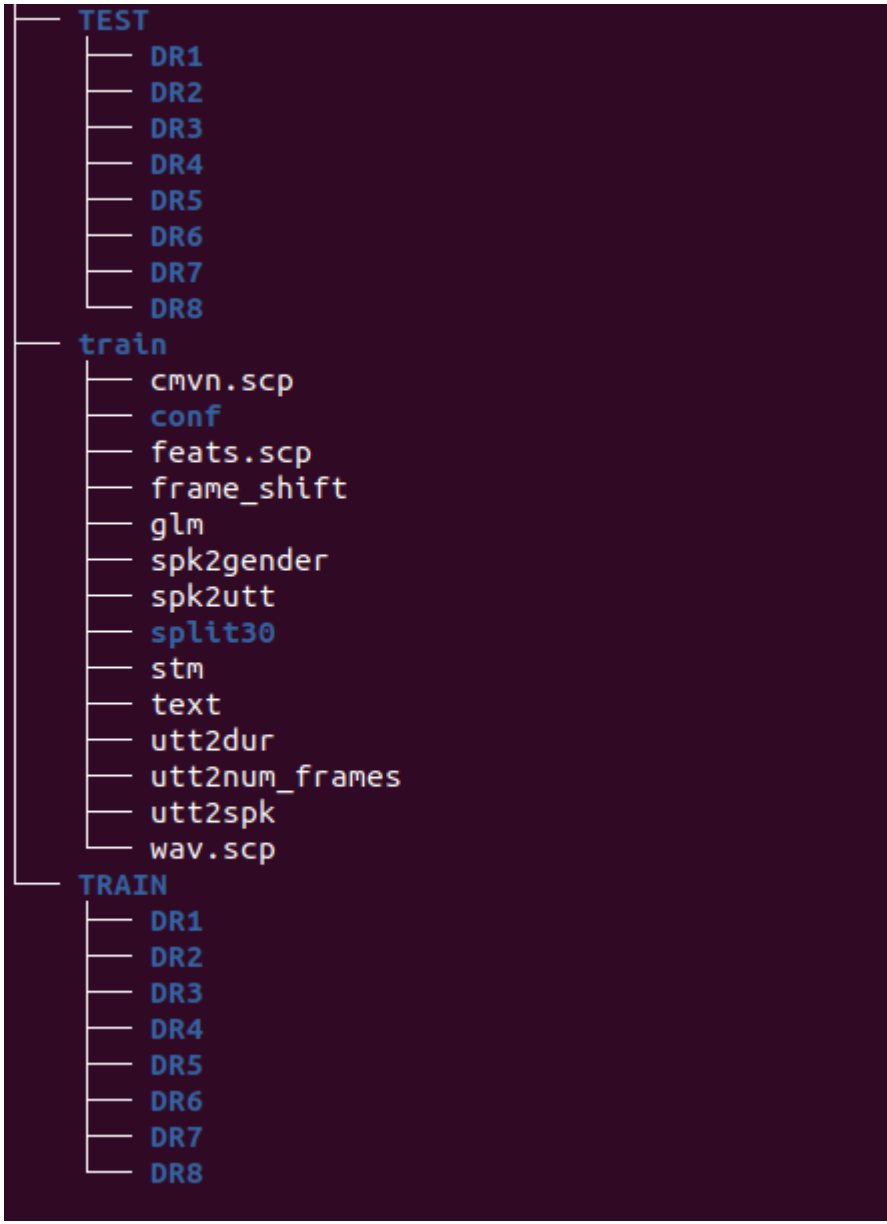


The main additions are: dev, test, train, lang, lang_test_bg, and local. Please be noticed that TRAIN and TEST come with original TIMIT installation CD/files.

The level 2 file structure of data is as below:




lang_test_bg
G.fst
L_disambig.fst
L.fst
oov.int
oov.txt
phones
phones.txt
tmp
topo
words.txt
local
data
dict
lang_tmp
nist_lm
README.DOC
test
cmvn.scp
conf
feats.scp
frame_shift
glm
spk2gender
spk2utt
split5
stm
text
utt2dur
utt2num_frames
utt2spk
wav.scp
TEST
DR1
DR2
DR3
DR4
DR5
DR6
DR7
DR8


TEST
DR1
DR2
DR3
DR4
DR5
DR6
DR7
DR8
train
cmvn.scp
conf
feats.scp
frame_shift
glm
spk2gender
spk2utt
split30
stm
text
utt2dur
utt2num_frames
utt2spk
wav.scp
TRAIN
DR1
DR2
DR3
DR4
DR5
DR6
DR7
DR8

Step 3: Dictionary

Dictionary maps words to phones. The mapping could be done through machine learning and complicated functions as well.

Line 43 `/local/timit_prepare_dict.sh` will create a diction at `./data/local/dict`. The files created are:

`timit_prepare_dict.sh` invokes some scripts:

```
leo@X280: ~/lkaldi/egs/timit/s5/data/local
File  Edit  View  Search  Terminal  Help
leo@X280:~/lkaldi/egs/timit/s5/data/local$ tree dict
dict
├── extra_questions.txt
├── lexiconp.txt
├── lexicon.txt
├── nonsilence_phones.txt
├── optional_silence.txt
├── phones.txt
└── silence_phones.txt
```

`irstlm/bin/build-lm.sh` creates phone language model `lm_tmp/lm_phone_bg.ilm.gz`.

`irstlm/bin/compile-lm` processes the `lm_train.text` to create `nist_lm/lm_phone_bg.arpa.gz`.

```
emacs@LeoUbuntu
File  Edit  Options  Buffers  Tools  Sh-Script  Help
Save  Undo
71    echo "$0: Error: this is no longer the case." >&2
72    echo "$0: Error: To install it, go to $KALDI_ROOT/tools" >&2
73    echo "$0: Error: and run extras/install_irstlm.sh" >&2
74    exit 1
75  fi
76
77  cut -d' ' -f2- $srcdir/train.text | sed -e 's:^:<s> :' -e 's:$: </s>:' \
78    > $srcdir/lm_train.text
79
80  build-lm.sh -i $srcdir/lm_train.text -n 2 \
81    -o $tmpdir/lm_phone_bg.ilm.gz
82
83  compile-lm $tmpdir/lm_phone_bg.ilm.gz -t=yes /dev/stdout | \
84  grep -v unk | gzip -c > $lmdir/lm_phone_bg.arpa.gz
85
86  echo "Dictionary & language model preparation succeeded"
-:**-  timit_prepare_dict.sh   Bot L84   Git-master   (Shell-script[bash])
```

Step 4: Create FST Script

Line 47 & 48 create FST Script.

```
45  # Caution below: we remove optional silence by setting "--sil-prob 0.0",
46  # in TIMIT the silence appears also as a word in the dictionary and is scored.
47  utils/prepare_lang.sh --sil-prob 0.0 --position-dependent-phones false --num-sil-states 3 \
48   data/local/dict "sil" data/local/lang_tmp data/lang
49
50  local/timit_format_data.sh
```

We define the FST in a separate file. Each line in the FST file identifies an arc (except the last line). The first two columns identify the state of where it transits from and where it transits to. The third column represents the input label and the fourth column represents the output label. If it is a dash, the output is empty.

Kaldi allows users to define and categorize different types of phones, including “real phones” and silence phones. All these category information helps Kaldi to build HMM topologies and decision trees.

`prepare_lang.sh` will transform silence and non-silence phone files for Kaldi. The file `context_indep.txt` contains all the phones which are not “real phones”: i.e. silence (SIL), spoken noise (SPN), non-spoken noise (NSN), and laughter (LAU). The file may contain many variants of these phones depends on the word position. For example, SIL_B is the silence phone that occurred at the beginning of a word.

run.sh | prepare_lang.sh

```
21  # This script prepares a directory such as data/lang/, in the standard format,
22  # given a source directory containing a dictionary lexicon.txt in a form like:
23  # word phone1 phone2 ... phoneN
24  # per line (alternate prons would be separate lines), or a dictionary with probabilities
25  # called lexiconp.txt in a form:
26  # word pron-prob phone1 phone2 ... phoneN
27  # (with 0.0 < pron-prob <= 1.0); note: if lexiconp.txt exists, we use it even if
28  # lexicon.txt exists.
29  # and also files silence_phones.txt, nonsilence_phones.txt, optional_silence.txt
30  # and extra_questions.txt
31  # Here, silence_phones.txt and nonsilence_phones.txt are lists of silence and
32  # non-silence phones respectively (where silence includes various kinds of
33  # noise, laugh, cough, filled pauses etc., and nonsilence phones includes the
34  # "real" phones.)
35  # In each line of those files is a list of phones, and the phones on each line
36  # are assumed to correspond to the same "base phone", i.e. they will be
37  # different stress or tone variations of the same basic phone.
38  # The file "optional_silence.txt" contains just a single phone (typically SIL)
```

To create FST Script, we need run `utils/prepare_lang.sh`.

```
$ utils/prepare_lang.sh --sil-prob 0.0 --position-
dependent-phones false --num-sil-states 3 data/lo-
cal/dict "sil" data/local/lang_tmp data/lang
$ local/timit_format_data.sh
```

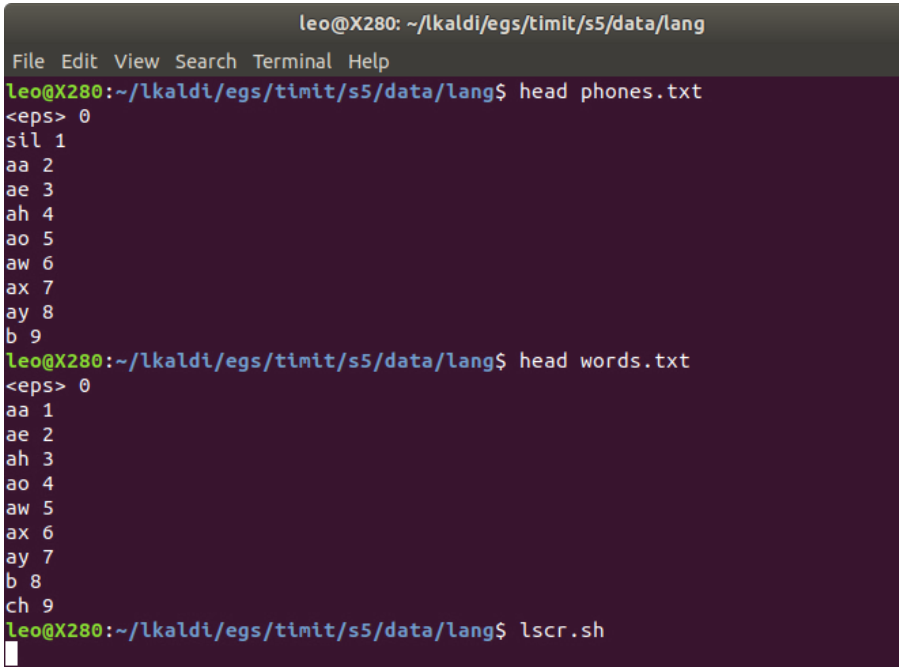
It will use files in `./data/local/dict` to create files in `data/lang` containing language information.

L.fst	fst structure of dictionary
L_disambig.fst	eliminate the ambiguity
phones.txt	phones
oov.txt/int	unknown word
topo	topology file, including transfer probability, transfer path, id, etc.
words.txt	words and id
phones	words and phones

The input files are: lexicon.txt, lexconp.txt, silence_phones.txt/nonsilence_phones.txt, optional_silence.txt, extra_questions.txt. `utils/make_lexicon_fst.pl` could change the format to fst.

Two files are most important: `words.txt` and `phones.txt`.

Both are FST format: phone to integer. Looks like same.



*.csl files including silence, non-silence and phone's id. We could use `fstprint` to check L.fst content.

```
fstprint --isymbols=data/lang/phones.txt --osymbols=data/
lang/words.txt data/lang/L.fst | head
```

The result is:

```
0 0    aa    aa
0 0    ae    ae
0 0    ah    ah
0 0    ao    ao
0 0    aw    aw
0 0    ax    ax
0 0    ay    ay
0 0    b     b
```

The topo file defines three HMM status of each phone. <Transi-

tion› defines the probability of Transition.

```
leo@X280: ~/lkaldi/egs/timit/s5/data/lang
File  Edit  View  Search  Terminal  Help
leo@X280:~/lkaldi/egs/timit/s5/data/lang$ more topo
<Topology>
<TopologyEntry>
<ForPhones>
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 3
1 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
1
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.5 <Transition> 1 0.5 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.5 <Transition> 2 0.5 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
</Topology>
```

The second script defines data format, create `data/lang_test_bg` directoy, copy files in `data/lang`, and create `G.fst` from `data/local/nist_lm` and `lm_phone_bg.arpa.gz`.

```
leo@LeoUbuntu:~/lkaldi/egs/timit/s5/data$ tree lang_test_bg
lang_test_bg
├── G.fst
├── L_disambig.fst
├── L.fst
├── oov.int
├── oov.txt
├── phones
├── phones.txt
├── tmp
├── topo
└── words.txt

2 directories, 8 files
leo@LeoUbuntu:~/lkaldi/egs/timit/s5/data$
```

The Result

Below is the final result of data preparation:

```
sv@HP:~/lKaldi/egs/timit/s5$ ./run.sh
=====================================================================
                 Data & Lexicon & Language Preparation
=====================================================================
wav-to-duration --read-entire-file=true
scp:train_wav.scp ark,t:train_dur.ark
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-to-du-
ration.cc:92) Printed duration for 3696 audio files.
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-
to-duration.cc:94) Mean duration was 3.06336, min
and max durations were 0.91525, 7.78881
wav-to-duration --read-entire-file=true
scp:dev_wav.scp ark,t:dev_dur.ark
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-to-du-
ration.cc:92) Printed duration for 400 audio files.
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-
to-duration.cc:94) Mean duration was 3.08212, min
and max durations were 1.09444, 7.43681
wav-to-duration --read-entire-file=true
scp:test_wav.scp ark,t:test_dur.ark
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-to-du-
ration.cc:92) Printed duration for 192 audio files.
LOG (wav-to-duration[5.5.164~1-9698]:main():wav-
to-duration.cc:94) Mean duration was 3.03646, min
and max durations were 1.30562, 6.21444
Data preparation succeeded
LOGFILE:/dev/null
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c
> $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dic-
tionary $additional_parameters >> $logfile 2>&1
$bin/ngt -i="$inpfile" -n=$order -gooout=y -o="$gzip -c
> $tmpdir/ngram.${sdict}.gz" -fd="$tmpdir/$sdict" $dic-
tionary $additional_parameters >> $logfile 2>&1
$scr/build-sublm.pl $verbose $prune $prune_thr_str $smooth-
ing "$additional_smoothing_parameters" --size $order
--ngrams "$gunzip -c $tmpdir/ngram.${sdict}.gz" -sublm $tmp-
dir/lm.$sdict $additional_parameters >> $logfile 2>&1
inpfile: data/local/lm_tmp/lm_phone_bg.ilm.gz
outfile: /dev/stdout
```

```
loading up to the LM level 1000 (if any)
dub: 10000000
OOV code is 50
OOV code is 50
Saving in txt format to /dev/stdout
Dictionary & language model preparation succeeded
utils/prepare_lang.sh --sil-prob 0.0 --position-de-
pendent-phones false --num-sil-states 3 data/lo-
cal/dict sil data/local/lang_tmp data/lang
Checking data/local/dict/silence_phones.txt ...
--> reading data/local/dict/silence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/silence_phones.txt is OK

Checking data/local/dict/optional_silence.txt ...
--> reading data/local/dict/optional_silence.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/optional_silence.txt is OK

Checking data/local/dict/nonsilence_phones.txt ...
--> reading data/local/dict/nonsilence_phones.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/nonsilence_phones.txt is OK

Checking disjoint: silence_phones.txt, nonsilence_phones.txt
--> disjoint property is OK.

Checking data/local/dict/lexicon.txt
--> reading data/local/dict/lexicon.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/lexicon.txt is OK

Checking data/local/dict/lexiconp.txt
```

```
--> reading data/local/dict/lexiconp.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/lexiconp.txt is OK

Checking lexicon pair data/local/dict/lexi-
con.txt and data/local/dict/lexiconp.txt
--> lexicon pair data/local/dict/lexicon.
txt and data/local/dict/lexiconp.txt match

Checking data/local/dict/extra_questions.txt ...
--> reading data/local/dict/extra_questions.txt
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/local/dict/extra_questions.txt is OK
--> SUCCESS [validating dictionary directory data/local/dict]

fstaddselfloops data/lang/phones/wdisambig_phones.
int data/lang/phones/wdisambig_words.int
prepare_lang.sh: validating output directory
utils/validate_lang.pl data/lang
Checking data/lang/phones.txt ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/phones.txt is OK

Checking words.txt: #0 ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang/words.txt is OK

Checking disjoint: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> silence.txt and nonsilence.txt are disjoint
--> silence.txt and disambig.txt are disjoint
--> disambig.txt and nonsilence.txt are disjoint
--> disjoint property is OK
```

```
Checking sumation: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> found no unexplainable phones in phones.txt

Checking data/lang/phones/context_indep.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.int corre-
sponds to data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.csl corre-
sponds to data/lang/phones/context_indep.txt
--> data/lang/phones/context_indep.{txt, int, csl} are OK

Checking data/lang/phones/nonsilence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 47 entry/entries in data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.int corre-
sponds to data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.csl corre-
sponds to data/lang/phones/nonsilence.txt
--> data/lang/phones/nonsilence.{txt, int, csl} are OK

Checking data/lang/phones/silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/phones/silence.txt
--> data/lang/phones/silence.int corre-
sponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.csl corre-
sponds to data/lang/phones/silence.txt
--> data/lang/phones/silence.{txt, int, csl} are OK

Checking data/lang/phones/optional_silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
```

```
--> 1 entry/entries in data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.int corre-
sponds to data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.csl corre-
sponds to data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.{txt, int, csl} are OK


Checking data/lang/phones/disambig.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang/phones/disambig.txt
--> data/lang/phones/disambig.int corre-
sponds to data/lang/phones/disambig.txt
--> data/lang/phones/disambig.csl corre-
sponds to data/lang/phones/disambig.txt
--> data/lang/phones/disambig.{txt, int, csl} are OK


Checking data/lang/phones/roots.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 48 entry/entries in data/lang/phones/roots.txt
--> data/lang/phones/roots.int corre-
sponds to data/lang/phones/roots.txt
--> data/lang/phones/roots.{txt, int} are OK


Checking data/lang/phones/sets.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 48 entry/entries in data/lang/phones/sets.txt
--> data/lang/phones/sets.int corre-
sponds to data/lang/phones/sets.txt
--> data/lang/phones/sets.{txt, int} are OK


Checking data/lang/phones/extra_questions.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang/phones/extra_questions.txt
--> data/lang/phones/extra_questions.int corre-
```

```
sponds to data/lang/phones/extra_questions.txt
--> data/lang/phones/extra_questions.{txt, int} are OK

Checking optional_silence.txt ...
--> reading data/lang/phones/optional_silence.txt
--> data/lang/phones/optional_silence.txt is OK

Checking disambiguation symbols: #0 and #1
--> data/lang/phones/disambig.txt has "#0" and "#1"
--> data/lang/phones/disambig.txt is OK

Checking topo ...

Checking word-level disambiguation symbols...
--> data/lang/phones/wdisambig.txt ex-
ists (newer prepare_lang.sh)
Checking data/lang/oov.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang/oov.txt
--> data/lang/oov.int corresponds to data/lang/oov.txt
--> data/lang/oov.{txt, int} are OK

--> data/lang/L.fst is olabel sorted
--> data/lang/L_disambig.fst is olabel sorted
--> SUCCESS [validating lang directory data/lang]
Preparing train, dev and test data
utils/validate_data_dir.sh: Successfully val-
idated data-directory data/train
utils/validate_data_dir.sh: Successful-
ly validated data-directory data/dev
utils/validate_data_dir.sh: Successfully val-
idated data-directory data/test
Preparing language models for test
arpa2fst --disambig-symbol=#0 --read-symbol-table=data/
lang_test_bg/words.txt - data/lang_test_bg/G.fst
LOG (arpa2fst[5.5.164~1-9698]:Read():arpa-file-
parser.cc:94) Reading \data\ section.
```

```
LOG (arpa2fst[5.5.164~1-9698]:Read():arpa-file-
parser.cc:149) Reading \1-grams: section.
LOG (arpa2fst[5.5.164~1-9698]:Read():arpa-file-
parser.cc:149) Reading \2-grams: section.
WARNING (arpa2fst[5.5.164~1-9698]:ConsumeNGram():arpa-lm-
compiler.cc:313) line 60 [-3.26717      <s> <s>]
skipped: n-gram has invalid BOS/EOS placement
LOG (arpa2fst[5.5.164~1-9698]:RemoveRedundantStates():ar
pa-lm-compiler.cc:359) Reduced num-states from 50 to 50
fstisstochastic data/lang_test_bg/G.fst
0.000510126 -0.0763018
utils/validate_lang.pl data/lang_test_bg
Checking data/lang_test_bg/phones.txt ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang_test_bg/phones.txt is OK

Checking words.txt: #0 ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> data/lang_test_bg/words.txt is OK

Checking disjoint: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> silence.txt and nonsilence.txt are disjoint
--> silence.txt and disambig.txt are disjoint
--> disambig.txt and nonsilence.txt are disjoint
--> disjoint property is OK

Checking sumation: silence.txt, nonsi-
lence.txt, disambig.txt ...
--> found no unexplainable phones in phones.txt

Checking data/lang_test_bg/phones/con-
text_indep.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang_test_
bg/phones/context_indep.txt
```

```
--> data/lang_test_bg/phones/context_indep.int corre-
sponds to data/lang_test_bg/phones/context_indep.txt
--> data/lang_test_bg/phones/context_indep.csl corre-
sponds to data/lang_test_bg/phones/context_indep.txt
--> data/lang_test_bg/phones/context_in-
dep.{txt, int, csl} are OK


Checking data/lang_test_bg/phones/non-
silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 47 entry/entries in data/lang_test_bg/phones/nonsilence.txt
--> data/lang_test_bg/phones/nonsilence.int corre-
sponds to data/lang_test_bg/phones/nonsilence.txt
--> data/lang_test_bg/phones/nonsilence.csl corre-
sponds to data/lang_test_bg/phones/nonsilence.txt
--> data/lang_test_bg/phones/nonsilence.{txt, int, csl} are OK


Checking data/lang_test_bg/phones/silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang_test_bg/phones/silence.txt
--> data/lang_test_bg/phones/silence.int corre-
sponds to data/lang_test_bg/phones/silence.txt
--> data/lang_test_bg/phones/silence.csl corre-
sponds to data/lang_test_bg/phones/silence.txt
--> data/lang_test_bg/phones/silence.{txt, int, csl} are OK


Checking data/lang_test_bg/phones/option-
al_silence.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang_test_bg/
phones/optional_silence.txt
--> data/lang_test_bg/phones/optional_silence.int corre-
sponds to data/lang_test_bg/phones/optional_silence.txt
--> data/lang_test_bg/phones/optional_silence.csl corre-
sponds to data/lang_test_bg/phones/optional_silence.txt
--> data/lang_test_bg/phones/optional_si-
lence.{txt, int, csl} are OK
```

```
Checking data/lang_test_bg/phones/disambig.{txt, int, csl} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang_test_bg/phones/disambig.txt
--> data/lang_test_bg/phones/disambig.int corre-
sponds to data/lang_test_bg/phones/disambig.txt
--> data/lang_test_bg/phones/disambig.csl corre-
sponds to data/lang_test_bg/phones/disambig.txt
--> data/lang_test_bg/phones/disambig.{txt, int, csl} are OK


Checking data/lang_test_bg/phones/roots.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 48 entry/entries in data/lang_test_bg/phones/roots.txt
--> data/lang_test_bg/phones/roots.int corre-
sponds to data/lang_test_bg/phones/roots.txt
--> data/lang_test_bg/phones/roots.{txt, int} are OK


Checking data/lang_test_bg/phones/sets.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 48 entry/entries in data/lang_test_bg/phones/sets.txt
--> data/lang_test_bg/phones/sets.int corre-
sponds to data/lang_test_bg/phones/sets.txt
--> data/lang_test_bg/phones/sets.{txt, int} are OK


Checking data/lang_test_bg/phones/ex-
tra_questions.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 2 entry/entries in data/lang_test_bg/
phones/extra_questions.txt
--> data/lang_test_bg/phones/extra_questions.int corre-
sponds to data/lang_test_bg/phones/extra_questions.txt
--> data/lang_test_bg/phones/extra_questions.{txt, int} are OK


Checking optional_silence.txt ...
```

```
--> reading data/lang_test_bg/phones/optional_silence.txt
--> data/lang_test_bg/phones/optional_silence.txt is OK

Checking disambiguation symbols: #0 and #1
--> data/lang_test_bg/phones/disambig.txt has "#0" and "#1"
--> data/lang_test_bg/phones/disambig.txt is OK

Checking topo ...

Checking word-level disambiguation symbols...
--> data/lang_test_bg/phones/wdisambig.txt ex-
ists (newer prepare_lang.sh)
Checking data/lang_test_bg/oov.{txt, int} ...
--> text seems to be UTF-8 or ASCII, checking whitespaces
--> text contains only allowed whitespaces
--> 1 entry/entries in data/lang_test_bg/oov.txt
--> data/lang_test_bg/oov.int corre-
sponds to data/lang_test_bg/oov.txt
--> data/lang_test_bg/oov.{txt, int} are OK

--> data/lang_test_bg/L.fst is olabel sorted
--> data/lang_test_bg/L_disambig.fst is olabel sorted
--> data/lang_test_bg/G.fst is ilabel sorted
--> data/lang_test_bg/G.fst has 50 states
fstdeterminizestar data/lang_test_bg/G.fst /dev/null
--> data/lang_test_bg/G.fst is determinizable
--> utils/lang/check_g_properties.pl success-
fully validated data/lang_test_bg/G.fst
--> utils/lang/check_g_properties.pl succeeded.
--> Testing determinizability of L_disambig . G
fstdeterminizestar
fsttablecompose data/lang_test_bg/L_dis-
ambig.fst data/lang_test_bg/G.fst
--> L_disambig . G is determinizable
--> SUCCESS [validating lang directory data/lang_test_bg]
Succeeded in formatting data.
```

Chapter 5. Kaldi Feature Extraction

MFCC

Voice or speech are analog signal, which is hard to be reproduce with some parameters. Scientists build up different models to make the reproduction as close as possible. They found some features are more useful than others.

One widely used model is to use frame to cut the audio wave. Each frame is about 10ms. Each frame could extract 39 digits, which representing the frame's audio feature. Put them in a vector, we get feature vector.

GMM (Gaussian Mixture Model) ignores the coefficient among all the features. The result is perfect for each unit but sometimes making no sense as a whole.

MFCC compensates GMM with the coefficient consideration and thus does a better job in feature extraction. The MFCC feature extraction technique basically includes windowing the signal, applying the DFT, taking the log of the magnitude, and then warping the frequencies on a Mel scale, followed by applying the inverse DCT.

DNN/CNN is built up to take advantage of such coefficient features extracted from MFCC. DNN/CNN implements fbank features to reduce WER with MFCC's DCT.

Step 1: Get feats.scp

Kaldi writes the feature vector into `feats.scp`. In another word, we extract those elements representing the speech, ignore those useless sounds (noises, emotions). Line 60-63 do the job.

```
52  echo ============================================================================
53  echo "         MFCC Feature Extration & CMVN for Training and Test set          "
54  echo ============================================================================
55
56  # Now make MFCC features.
57  mfccdir=mfcc
58
59
60  for x in train dev test; do
61    steps/make_mfcc.sh --cmd "$train_cmd" --nj $feats_nj data/$x exp/make_mfcc/$x $mfccdir
62    steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir
63  done
64
```

`steps/make_mfcc.sh` stores MFCC features into `feats.scp`. There are 13 dimensions for each frame. Only when MFCC features are used are deltas and deltas-deltas calculated to convert to 39 dimensions.

Take `data/train` as example, `make_mfcc.sh` reads `data/train/wav.scp`, divides it to several parts evenly, and invokes `run.pl` extracting the features. One job processes one `wav.scp`.

The scripts `compute-mfcc-feats` and `copy-feats` create `mfcc/raw_mfcc_train.JOB.ark` and corresponding `mfcc/raw_mfcc_train.JOB.scp` files. JOB is number representing how many parts are divided.

The last step is to combine those `mfcc/raw_mfcc_train.JOB.scp` files into one `data/train/feats.scp`, with which, we could calculate CMVN (Cepstral mean and variance normalization).

```
sv@HP:~/lkaldi/egs/timit/s5/data/train$ head feats.scp
FAEM0_SI1392 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:13
FAEM0_SI2022 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:6313
FAEM0_SI762 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:9349
FAEM0_SX132 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:13048
FAEM0_SX222 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:16916
FAEM0_SX312 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:20537
FAEM0_SX402 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:25549
FAEM0_SX42 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:29767
FAJW0_SI1263 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:32804
FAJW0_SI1893 /home/sv/lkaldi/egs/timit/s5/mfcc/raw_mfcc_train.1.ark:39403
sv@HP:~/lkaldi/egs/timit/s5/data/train$ lscr.sh
```

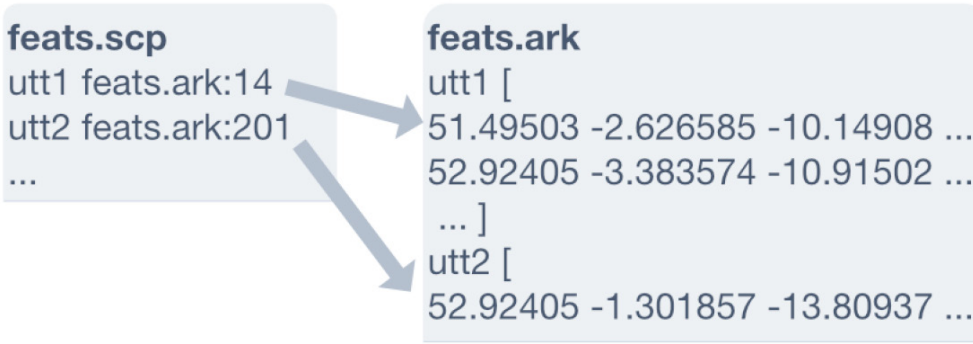
Step 2: Get cmvn.scp

We calculate CMVN for each speecher. Each dimension (13 dimensions total) should be calculated 3 groups of statistics in `count/sum/sum-squared`. Count represents the total number of frames. To sum up each dimension of one frame to get sum. Calculate sum-square. The variance and mean are calculated based on them.

There is no option to do CMVN per utterance. The idea is that if you did it per utterance it would not make sense to do per-speaker fMLLR on top of that (since you'd be doing fMLLR on top of different offsets). Therefore what would be the use of the speaker information? In this case you should probably make the speaker-ids identical to the utterance-ids. The speaker information does not have to correspond to actual speakers, it's just the level you want to adapt at.

To check `cmvn.scp` or `cmvn.ark`, we could use `copy-matrix`. Each speecher has a matrix with 28 dimensions. The first 13 dimensions are sum, then count, and last 13 are sum-squared. The result is stored in `cmvn.scp`.

MFCC folder contains .ark and .scp files. The utterances and features pair are stored in .scp. the real features are stored in .ark. Kaldi use these two files together.



Let's check the files:

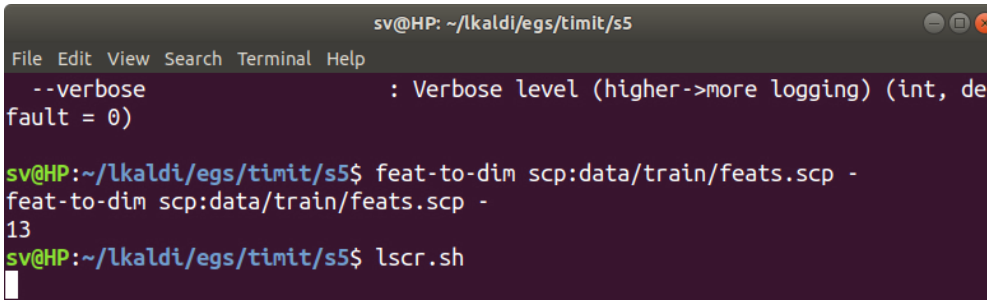
```
$ head raw_mfcc_train.1.scp
faem0_si1392 ./timit/s5/mfcc/raw_mfcc_train.1.ark:13
faem0_si2022 ./timit/s5/mfcc/raw_mfcc_train.1.ark:6313
faem0_si762 ./timit/s5/mfcc/raw_mfcc_train.1.ark:9349
faem0_sx132 ./timit/s5/mfcc/raw_mfcc_train.1.ark:13048
faem0_sx222 ./timit/s5/mfcc/raw_mfcc_train.1.ark:16916
faem0_sx312 ./timit/s5/mfcc/raw_mfcc_train.1.ark:20537
faem0_sx402 ./timit/s5/mfcc/raw_mfcc_train.1.ark:25549
faem0_sx42 ./timit/s5/mfcc/raw_mfcc_train.1.ark:29767
fajw0_si1263 ./timit/s5/mfcc/raw_mfcc_train.1.ark:32804
fajw0_si1893 ./timit/s5/mfcc/raw_mfcc_train.1.ark:39403
```

Scripts and Archives are oganzied as the table. One table has an index (like utt_id). .scp is text file, each line has a key and a corresponding file name, which telling Kaldi the data location.

Archive file could be text or binary. The format is key (like utt_id), space, and the object.

We could check MFCC's dimension like below:

```
feat-to-dim scp:data/train/feats.scp -
feat-to-dim ark:mfcc/raw_mfcc_train.1.ark -
```



The result is 13, as we expected.

MFCC & CMVN Output

```
=====================================================================
     MFCC Feature Extration & CMVN for Training and Test set
=====================================================================

steps/make_mfcc.sh --cmd run.pl --max-jobs-run 10
--nj 10 data/train exp/make_mfcc/train mfcc

steps/make_mfcc.sh: moving data/train/
feats.scp to data/train/.backup

utils/validate_data_dir.sh: Successfully val-
idated data-directory data/train

steps/make_mfcc.sh: [info]: no segments file ex-
ists: assuming wav.scp indexed by utterance.

Succeeded creating MFCC features for train

steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train mfcc

Succeeded creating CMVN stats for train

steps/make_mfcc.sh --cmd run.pl --max-jobs-run
10 --nj 10 data/dev exp/make_mfcc/dev mfcc

steps/make_mfcc.sh: moving data/dev/
feats.scp to data/dev/.backup

utils/validate_data_dir.sh: Successful-
ly validated data-directory data/dev

steps/make_mfcc.sh: [info]: no segments file ex-
ists: assuming wav.scp indexed by utterance.

Succeeded creating MFCC features for dev

steps/compute_cmvn_stats.sh data/dev exp/make_mfcc/dev mfcc

Succeeded creating CMVN stats for dev

steps/make_mfcc.sh --cmd run.pl --max-jobs-run 10
--nj 10 data/test exp/make_mfcc/test mfcc

steps/make_mfcc.sh: moving data/test/
feats.scp to data/test/.backup

utils/validate_data_dir.sh: Successfully val-
idated data-directory data/test

steps/make_mfcc.sh: [info]: no segments file ex-
ists: assuming wav.scp indexed by utterance.

Succeeded creating MFCC features for test

steps/compute_cmvn_stats.sh data/test exp/make_mfcc/test mfcc

Succeeded creating CMVN stats for test
```

Chapter 6. MonoPhone

Viterbi

Features Vector is the last step for us to retrieve from the audio wave. The rest work will rely on our modeling and analyzing ability. First thought is to use phone recognition, or monophone as ASR professional named.

Monophone is the easiest model to train for two reasons: first, English has only around 40 phones. Second, mono also means it is context independent.

The drawback is also clear: it does not has much ability to identify the variation.

The following classes are used in monophone training.

- DiagGmm: reserving GMM parameters.
- DiagGmmNormal: GMM original value.
- AmDiagGmm: all the GMMs.
- AccumDiagGmm: accumulation of GMM
- AccumAmDiagGmm: keeping AccumDiagGmm

TransitionalModel: keeping HMM Topo, logo transitional probability, transition-id, transition-state, triplets (phone, hmm-state, forward-pdf mappings.

6.1 Monophone Training Method

Kaldi uses Viterbi instead of Baum-Welch for monophone GMM training. We just need modify the three parameters in EM method.

Updating Baum-Welch method is time consuming because it requires to calculate forward and backward. Monophone is context independent, so Viterbi is better. Baum-Welch will use all the paths accumulated, while Viterbi just uses Viterbi path.

Viterbi aligns the model feature extractions based on last round running results to get the HMM status of each frame's feature (i.e. transition-id), which is called forced alignment. The forced alignment could get the status series of the featured vector. For example:

A representation of the sequence of HMM states taken by the Viterbi (best-path) alignment of an utterance. In Kaldi an alignment is synonymous with a sequence of transition-ids. Most of the time an alignment is derived from aligning the reference transcript of an utterance, in which case it is called a forced alignment. lattices also contain alignment information as sequences of transition-ids for each word sequence in the lattice. The program show-alignments shows alignments in a human-readable format.

Though Viterbi training yields a good performance in most cases, sometimes it leads to suboptimal models, specially when using discrete HMMs to model spontaneous speech. In these cases, Baum-Welch shows more robust than both Viterbi training and the combined approach, compensating for its high computational cost. When using continuous HMMs, Viterbi training reveals as good as Baum-Welch at a much lower cost.

6.2 Training Process

There are three main steps to train monophone:

1. Train the mono acoustic model.
2. Make a decoding graph with current configuration for test.
3. Decode the development and test dataset with the graph.

The three steps match Kaldi's three scrpits: train_mono.sh, mkgraph.sh, and decode.sh.

```
65  echo ============================================================================
66  echo "                    MonoPhone Training & Decoding                         "
67  echo ============================================================================
68
69  steps/train_mono.sh  --nj "$train_nj" --cmd "$train_cmd" data/train data/lang exp/mono
70
71  utils/mkgraph.sh data/lang_test_bg exp/mono exp/mono/graph
72
73  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
74   exp/mono/graph data/dev exp/mono/decode_dev
75
76  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
77   exp/mono/graph data/test exp/mono/decode_test
78
```

train_nj has been given 30. --nj 30 instructs Kaldi to split the wav file into four even parts to process, which are stored in data/train/split30 labeled with number 1 to 30. Each block contains three scp files and other supporting files.

```
sv@HP: ~/lkaldi/egs/timit/s5/data/train/split30
File  Edit  View  Search  Terminal  Help
1   11  13  15  17  19  20  22  24  26  28  3   4  6  8
10  12  14  16  18  2   21  23  25  27  29  30  5  7  9
sv@HP:~/lkaldi/egs/timit/s5/data/train/split30$ ls 1
cmvn.scp   spk2gender  text      wav.scp
feats.scp  spk2utt     utt2spk
sv@HP:~/lkaldi/egs/timit/s5/data/train/split30$ lscr.sh
```

train-mono.sh sets up the variable $stage, which could resume from the interrupt point to avoid re-training from the start point. For example, if $stage = 10, the iteration will start from the 11.

6.3 Train Acoustic Model

Line 69 of run.sh is `train_mono.sh`.

run.sh | train_mono.sh

```
67  echo "$0: Initializing monophone system."
68
69  [ ! -f $lang/phones/sets.int ] && exit 1;
70  shared_phones_opt="--shared-phones=$lang/phones/sets.int"
71
72  if [ $stage -le -3 ]; then
73    # Note: JOB=1 just uses the 1st part of the features-- we
      only need a subset anyway.
74    if ! feat_dim=`feat-to-dim "$example_feats" - 2>/dev/null`
      || [ -z $feat_dim ]; then
75      feat-to-dim "$example_feats" -
76      echo "error getting feature dimension"
77      exit 1;
78    fi
79    $cmd JOB=1 $dir/log/init.log \
80      gmm-init-mono $shared_phones_opt "--train-feats=$feats
      subset-feats --n=10 ark:- ark:-|" $lang/topo $feat_dim \
81      $dir/0.mdl $dir/tree || exit 1;
82  fi
83
84  numgauss=`gmm-info --print-args=false $dir/0.mdl | grep
    gaussians | awk '{print $NF}'`
85  incgauss=$[($totgauss-$numgauss)/$max_iter_inc] # per-iter
    increment for #Gauss
86
87  if [ $stage -le -2 ]; then
88    echo "$0: Compiling training graphs"
89    $cmd JOB=1:$nj $dir/log/compile_graphs.JOB.log \
90      compile-train-graphs --read-disambig-syms=$lang/phones/
      disambig.int $dir/tree $dir/0.mdl  $lang/L.fst \
91      "ark:sym2int.pl --map-oov $oov_sym -f 2- $lang/words.txt
      < $sdata/JOB/text|" \
92      "ark:|gzip -c >$dir/fsts.JOB.gz" || exit 1;
93  fi
```

Line 80 `gmm-init-mono` initializes the monophone model, which creates `0.mdl` and tree under `exp/mono/`, calculates the global variance and mean of each feature of every dimension,

reading topo file and create shared phone list based on `$lang/phones/sets.init`, and creates ctx_dep (tree) based on shared phone list.

To be more specific, Kaldi creates one component GMM, whose mean and variance initialize as the global mean and global variance. The Gauss' class is DiagGMM. It stores the inverse of variance and mean and the product of mean and varaince inverse instead of the original data. It also stores the constant (the ‹GCONSTS› of GMM part in .mdl.

```
86
87  if [ $stage -le -2 ]; then
88    echo "$0: Compiling training graphs"
89    $cmd JOB=1:$nj $dir/log/compile_graphs.JOB.log \
90      compile-train-graphs --read-disambig-syms=$lang/phones/
      disambig.int $dir/tree $dir/0.mdl  $lang/L.fst \
91      "ark:sym2int.pl --map-oov $oov_sym -f 2- $lang/words.txt
      < $sdata/JOB/text|" \
92      "ark:|gzip -c >$dir/fsts.JOB.gz" || exit 1;
93  fi
```

The loop from 87 to 93 compile-train-graphs will compile graphs of transcripts to exp/mono/fsts.JOB.gz（total 30 JOBs）in ark format including each utt-id in the `train.tra`. It complies the FST for each sentence, which is composed of non-transferable HCLG.

run.sh | train_mono.sh

```
94
95  if [ $stage -le -1 ]; then
96    echo "$0: Aligning data equally (pass 0)"
97    $cmd JOB=1:$nj $dir/log/align.0.JOB.log \
98      align-equal-compiled "ark:gunzip -c $dir/fsts.JOB.gz|"
      "$feats" ark,t:-  \| \
99      gmm-acc-stats-ali --binary=true $dir/0.mdl "$feats" ark
      :- \
100     $dir/0.JOB.acc || exit 1;
101 fi
102
```

The loop from 95 to 101 aligns data equally, creating alignment

```
103  # In the following steps, the --min-gaussian-occupancy=3
     option is important, otherwise
104  # we fail to est "rare" phones and later on, they never
     align properly.
105
106  if [ $stage -le 0 ]; then
107    gmm-est --min-gaussian-occupancy=3  --mix-up=$numgauss --
       power=$power \
108      $dir/0.mdl "gmm-sum-accs - $dir/0.*.acc|" $dir/1.mdl 2>
         $dir/log/update.0.log || exit 1;
109    rm $dir/0.*.acc
110  fi
```

Line 107 gmm-est will re-calculate MLE of 0.mdl based on GMM acoustic model. The new MLE will be stored to `exp/mono/1.mdl`. The old `exp/mono/0.*.acc` will be removed.

```
112  beam=$initial_beam # will change to regular_beam below after 1st pass
113  # note: using slightly wider beams for WSJ vs. RM.
114  x=1
115  while [ $x -lt $num_iters ]; do
116    echo "$0: Pass $x"
117    if [ $stage -le $x ]; then
118      if echo $realign_iters | grep -w $x >/dev/null; then
119        echo "$0: Aligning data"
120        mdl="gmm-boost-silence --boost=$boost_silence `cat $lang/phones/optional_silence.csl`
           $dir/$x.mdl - |"
121        $cmd JOB=1:$nj $dir/log/align.$x.JOB.log \
122          gmm-align-compiled $scale_opts --beam=$beam --retry-beam=$retry_beam --careful=
           $careful "$mdl" \
123          "ark:gunzip -c $dir/fsts.JOB.gz|" "$feats" "ark,t:|gzip -c >$dir/ali.JOB.gz" \
124          || exit 1;
125      fi
126      $cmd JOB=1:$nj $dir/log/acc.$x.JOB.log \
127        gmm-acc-stats-ali  $dir/$x.mdl "$feats" "ark:gunzip -c $dir/ali.JOB.gz|" \
128        $dir/$x.JOB.acc || exit 1;
129
130      $cmd $dir/log/update.$x.log \
131        gmm-est --write-occs=$dir/$[$x+1].occs --mix-up=$numgauss --power=$power $dir/$x.mdl \
132        "gmm-sum-accs - $dir/$x.*.acc|" $dir/$[$x+1].mdl || exit 1;
133      rm $dir/$x.mdl $dir/$x.*.acc $dir/$x.occs 2>/dev/null
134    fi
135    if [ $x -le $max_iter_inc ]; then
136       numgauss=$[$numgauss+$incgauss];
137    fi
138    beam=$regular_beam
139    x=$[$x+1]
140  done
141
142  ( cd $dir; rm final.{mdl,occs} 2>/dev/null; ln -s $x.mdl final.mdl; ln -s $x.occs final.occs )
143
144
145  steps/diagnostic/analyze_alignments.sh --cmd "$cmd" $lang $dir
146  utils/summarize_warnings.pl $dir/log
147
148  steps/info/gmm_dir_info.pl $dir
149
150  echo "$0: Done training monophone system in $dir"
151
152  exit 0
153
154  # example of showing the alignments:
155  # show-alignments data/lang/phones.txt $dir/30.mdl "ark:gunzip -c $dir/ali.0.gz|" | head -4
```

Line 112 - 149 iterates the result. The process will update Transitional Model and GMM.

The output of this `train_mono.sh` will be:

```
steps/train_mono.sh: Initializing monophone system.
steps/train_mono.sh: Compiling training graphs
steps/train_mono.sh: Aligning data equally (pass 0)
steps/train_mono.sh: Pass 1
steps/train_mono.sh: Aligning data
    .....................
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 36
steps/train_mono.sh: Pass 37
steps/train_mono.sh: Pass 38
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 39
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/mono
steps/diagnostic/analyze_alignments.sh: see stats in
exp/mono/log/analyze_alignments.log
    2 warnings in exp/mono/log/align.*.*.log
exp/mono: nj=4 align prob=-99.15 over 3.12h [re-
try=0.0%, fail=0.0%] states=144 gauss=985
```

The mono folder will be created under `exp`, in which the model parameters are stored in `.mdl` files. We could check the contents like below:

```
$ gmm-copy --binary=false exp/mono/0.mdl - | less
```

The out put will be:

```
<TransitionModel>
<Topology>
<TopologyEntry>
      <ForPhones>
```

```
              2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23

              24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42                    43 44 45 46 47 48

       </ForPhones>

  <State> 0 <PdfClass> 0 <Transition> 0
0.75 <Transition> 1 0.25 </State>

  <State> 1 <PdfClass> 1 <Transition> 1
0.75 <Transition> 2 0.25 </State>

  <State> 2 <PdfClass> 2 <Transition> 2
0.75 <Transition> 3 0.25 </State>

  <State> 3 </State>

</TopologyEntry>


<TopologyEntry>

    <ForPhones>

       1

    </ForPhones>

  <State> 0 <PdfClass> 0 <Transition> 0
0.5 <Transition> 1 0.5 </State>

  <State> 1 <PdfClass> 1 <Transition> 1
0.5 <Transition> 2 0.5 </State>

  <State> 2 <PdfClass> 2 <Transition> 2
0.75 <Transition> 3 0.25 </State>

  <State> 3 </State>

</TopologyEntry>

</Topology>

    <Triples> 144

    1 0 0

    1 1 1

    1 2 2

    2 0 3

    2 1 4

    2 2 5

    3 0 6

    3 1 7

    3 2 8

    4 0 9
```

```
    ............
```

The top is `topo` information. There are 49 phones, and phone 1 is sitting in a separated section. Checking `phones.txt` we know that it is sil (silence). topo file is compiled as starting with initial status (100% probability) and the last status is ending (100% probability). In our case, obviously, -1 is the starting point, 0,1,2 are transient status of HMM, and 3 is the ending status.

Underneath the topology there is a `<Triples>` tag. This maps each phone and one of its three states to a unique number. That is, there are `num phones x num states = 144` triples.

It's easy to conclude that 0.mdl is the initialized model, in which all the parameters are initial value. We train this model 40 times. The final probability will be stored in 40.mdl.

Kaldi label p.d.f with number starting with 0 (it's called pdf-ids), while there is no name in HTK model.

.mdl also telling us that there is no data to link context-dependent phones and pdf-ids. Let's check the tree file:

```
$ copy-tree --binary=false exp/mono/tree - | less
```

The output is:

```
copy-tree --binary=false exp/mono/tree -
LOG (copy-tree:main():copy-tree.cc:55) Copied tree
ContextDependency 1 0 ToPdf TE 0 49 ( NULL TE -1 3 (
CE 0 CE 1 CE 2 )
    TE -1 3 ( CE 3 CE 4 CE 5 )
    TE -1 3 ( CE 6 CE 7 CE 8 )
    TE -1 3 ( CE 9 CE 10 CE 11 )
    TE -1 3 ( CE 12 CE 13 CE 14 )
    TE -1 3 ( CE 15 CE 16 CE 17 )
    TE -1 3 ( CE 18 CE 19 CE 20 )
```

```
    TE -1 3 ( CE 21 CE 22 CE 23 )
    TE -1 3 ( CE 24 CE 25 CE 26 )
    TE -1 3 ( CE 27 CE 28 CE 29 )
    TE -1 3 ( CE 30 CE 31 CE 32 )
    TE -1 3 ( CE 33 CE 34 CE 35 )
    ..................
    EndContextDependency
```

This is the tree of monophone. It's very trivial since there are no branches. CE means constant eventmap, representing the leaves of the tree.

TE means table eventmap, representing searching table.

There is no SE (split eventmap, representing branches of the tree) since it's monophone.

TE 0 29 is a table eventmap from key 0 to 29. In the parentheses sitting 29 event maps. The first one is Null representing a pointer pointing to the 0 since phone-id0 is reserved for epsilon.

TE -1 3 (CE 75 CE 76 CE 77), means one table eventmap is divided from key-1. The key is the pdfclass in topo file, representing the HMM status. There are 3 status for this phone, so the key will be 0, 1, 2. Three constant eventmap are in parentheses, representing 3 leaves of the tree.

Below we will check Viterbi alignment of the training. Each line of data. ali.*.gz is linked with a training file.

```
$ copy-int-vector "ark:gunzip -c exp/mono/ali.1.gz|"
ark,t:- | head -n 2
```

The output is:

```
    faem0_si1392 2 4 3 3 3 3 3 3 6 5 5 5 5 5 38 37 37
40 42 218 217 217 217 217 217 217 217 217 217 217 217
220 219 222 221 221 221 248 247 247 247 247 247 250
```

```
252 176 175 178 177 177 177 177 177 180 122 121 121
121 121 121 121 121 124 123 123 126 125 26 28 27 30
29 29 212 211 214 216 215 146 148 150 260 262 264 263
263 278 277 277 277 280 282 281 281 281 14 13 13 13
13 16 15 18 17 176 175 178 180 62 64 66 206 208 210
242 241 244 246 170 169 169 169 169 172 171 174 173
173 173 173 173 38 37 37 37 40 42 41 218 217 217 217
217 217 217 217 217 217 220 219 222 221 146 145 148
150 62 64 66 56 55 55 55 55 55 55 58 60 59 59 248 250
249 252 251 251 251 251 251 116 115 115 115 115 118
117 117 120 119 224 223 223 223 223 223 223 226 225
225 228 98 97 97 100 99 102 101 266 265 265 265 265
268 267 267 267 270 269 269 86 88 90 110 109 112 111
114 122 121 121 121 121 121 121 121 121 121 124 123
126 125 8 7 7 7 7 7 7 7 7 10 9 9 12 212 214 216 176
175 175 178 177 180 134 133 133 133 136 135 138 137
86 88 90 89 278 277 277 277 280 282 281 281 38 37 40
42 62 64 66 65 206 208 207 207 207 207 207 210 209 14
13 13 16 15 15 18 17 17 17 62 61 64 66 164 166 168
167 167 152 154 156 188 190 189 189 192 191 191 224
223 223 223 223 223 223 223 223 223 226 225 225 228
227 86 85 85 85 85 85 85 85 85 85 85 88 87 90 260 259
262 264 263 68 70 72 71 71 71 2 4 3 3 6 5 5 5 5 5 5
14 13 13 16 15 15 15 15 15 18 17 17 182 181 184 186
260 262 264 68 70 72 122 121 121 121 121 121 121 124
123 123 126 152 151 151 151 151 154 153 153 153 156
155 155 155 155 155 155 170 169 169 169 169 169 169
172 171 174 173 260 259 262 261 264 263 263 263 218
217 217 217 217 217 217 217 217 217 220 219 222 221
221 2 1 4 3 3 3 3 3 3 3 6

    faem0_si2022 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 4 3 3 3 3 3 6 5 5 5 266 265 265 265 265 265 265
268 267 270 269 269 20 22 24 80 82 81 84 83 32 31 34
33 33 33 33 36 35 35 35 35 35 35 122 121 121 121 121
121 121 121 121 124 123 126 125 146 145 148 147 150
62 64 63 66 65 65 65 65 65 65 68 70 72 71 242 241 244
246 245 224 223 223 223 223 223 223 223 223 223 223
223 226 225 225 228 152 154 153 156 260 262 264 263
68 70 72 71 212 211 211 211 214 213 213 216 215 215
215 215 44 43 46 45 45 45 48 47 47 254 256 258
```

```
122 121 121 121 121 121 121 121 121 121 121 124 123
126 125 125 26 25 25 28 27 27 27 27 27 30 29 29 29 2
1 1 1 1 4 3 3 3 3 3 3 6 5
```

This data is the result of Viterbi alignment. Each speech is represented by one line. When checking the above exp/mono/tree file, the maximum p.d.f-id is only 83, but the data here is far more than 83. The reason is that alignment doesn't use p.d.f-id, instead, it uses more sliced identifier (transition-id). These ids combinedthe phone and the transition probability in topo file. To know more about the transition-id, we could:

```
$show-transitions data/lang/phones.txt exp/mono/0.mdl
```

The output is:

```
 Transition-id = 1 p = 0.5 [self-loop]
 Transition-id = 2 p = 0.5 [0 -> 1]
Transition-state 2: phone = sil hmm-state = 1 pdf = 1
 Transition-id = 3 p = 0.5 [self-loop]
 Transition-id = 4 p = 0.5 [1 -> 2]
Transition-state 3: phone = sil hmm-state = 2 pdf = 2
 Transition-id = 5 p = 0.75 [self-loop]
 Transition-id = 6 p = 0.25 [2 -> 3]
Transition-state 4: phone = a hmm-state = 0 pdf = 3
 Transition-id = 7 p = 0.75 [self-loop]
 Transition-id = 8 p = 0.25 [0 -> 1]
Transition-state 5: phone = a hmm-state = 1 pdf = 4
 Transition-id = 9 p = 0.75 [self-loop]
 Transition-id = 10 p = 0.25 [1 -> 2]
Transition-state 6: phone = a hmm-state = 2 pdf = 5
 Transition-id = 11 p = 0.75 [self-loop]
 Transition-id = 12 p = 0.25 [2 -> 3]
Transition-state 7: phone = b hmm-state = 0 pdf = 6
 Transition-id = 13 p = 0.75 [self-loop]
```

```
 Transition-id = 14 p = 0.25 [0 -> 1]
Transition-state 8: phone = b hmm-state = 1 pdf = 7
 Transition-id = 15 p = 0.75 [self-loop]
 Transition-id = 16 p = 0.25 [1 -> 2]
Transition-state 9: phone = b hmm-state = 2 pdf = 8
 Transition-id = 17 p = 0.75 [self-loop]
 Transition-id = 18 p = 0.25 [2 -> 3]
Transition-state 10: phone = ch hmm-state = 0 pdf = 9
 Transition-id = 19 p = 0.75 [self-loop]
 Transition-id = 20 p = 0.25 [0 -> 1]
Transition-state 11: phone = ch hmm-state = 1 pdf =
10
 Transition-id = 21 p = 0.75 [self-loop]
 Transition-id = 22 p = 0.25 [1 -> 2]
Transition-state 12: phone = ch hmm-state = 2 pdf =
11
 Transition-id = 23 p = 0.75 [self-loop]
 Transition-id = 24 p = 0.25 [2 -> 3]
.....
Transition-state 82: phone = z hmm-state = 0 pdf = 81
 Transition-id = 163 p = 0.75 [self-loop]
 Transition-id = 164 p = 0.25 [0 -> 1]
Transition-state 83: phone = z hmm-state = 1 pdf = 82
 Transition-id = 165 p = 0.75 [self-loop]
 Transition-id = 166 p = 0.25 [1 -> 2]
Transition-state 84: phone = z hmm-state = 2 pdf = 83
 Transition-id = 167 p = 0.75 [self-loop]
 Transition-id = 168 p = 0.25 [2 -> 3]
```

Obviously, it's the initial status before training. To make it more clear, let's try this way:

```
show-alignments data/lang/phones.txt exp/mono/40.mdl
```

```
exp/mono/40.occs | less
# (.occs means occupation counts)
```

The output is:

```
Transition-state 1: phone = sil hmm-state = 0 pdf = 0
 Transition-id = 1 p = 0.934807 count of pdf = 1.13866e+06 [self-loop]
 Transition-id = 2 p = 0.0651934 count of pdf = 1.13866e+06 [0 -> 1]
Transition-state 2: phone = sil hmm-state = 1 pdf = 1
 Transition-id = 3 p = 0.889584 count of pdf = 672302 [self-loop]
 Transition-id = 4 p = 0.110416 count of pdf = 672302 [1 -> 2]
Transition-state 3: phone = sil hmm-state = 2 pdf = 2
 Transition-id = 5 p = 0.7137 count of pdf = 259284 [self-loop]
 Transition-id = 6 p = 0.2863 count of pdf = 259284 [2 -> 3]
Transition-state 4: phone = a hmm-state = 0 pdf = 3
 Transition-id = 7 p = 0.713307 count of pdf = 390711 [self-loop]
 Transition-id = 8 p = 0.286693 count of pdf = 390711 [0 -> 1]
Transition-state 5: phone = a hmm-state = 1 pdf = 4
 Transition-id = 9 p = 0.594051 count of pdf = 275931 [self-loop]
 Transition-id = 10 p = 0.405949 count of pdf = 275931 [1 -> 2]
Transition-state 6: phone = a hmm-state = 2 pdf = 5
 Transition-id = 11 p = 0.594987 count of pdf = 276569 [self-loop]
 Transition-id = 12 p = 0.405013 count of pdf = 276569 [2 -> 3]
Transition-state 7: phone = b hmm-state = 0 pdf = 6
```

```
 Transition-id = 13 p = 0.590539 count of pdf = 19660
[self-loop]
 Transition-id = 14 p = 0.409461 count of pdf = 19660
[0 -> 1]
Transition-state 8: phone = b hmm-state = 1 pdf = 7
 Transition-id = 15 p = 0.417553 count of pdf = 13821
[self-loop]
```

We are using 40.mdl. The result is the final transition probability the model get.

Let's check what's the next step for training.

```
grep Overall exp/mono/log/acc.{?.? ,?.??,??.?,??.??}.log
```

#Below is the last part of the output:

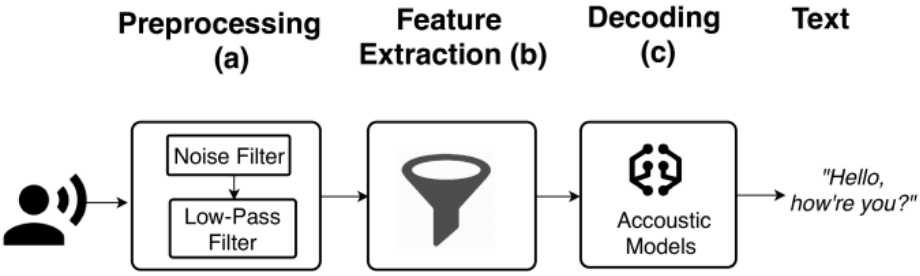
```
exp/mono/log/acc.35.10.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115)
exp/mono/log/acc.37.12.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -99.1242 over 595815 frames.
exp/mono/log/acc.38.10.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -99.0045 over 666753 frames.
exp/mono/log/acc.38.11.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -95.769 over 793715 frames.
exp/mono/log/acc.38.12.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -99.0953 over 595815 frames.
exp/mono/log/acc.39.10.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -98.9901 over 666753 frames.
exp/mono/log/acc.39.11.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -95.7472 over 793715 frames.
exp/mono/log/acc.39.12.log:LOG (gmm-acc-stats-
ali:main():gmm-acc-stats-ali.cc:115) Overall avg like per
frame (Gaussian only) = -99.0786 over 595815 frames.
```

We could see the Acoustic likelihood of each iteration.

6.4 Decoding

With the acoustic, pronunciation lexicon and language model built, we are ready to decode (transcript) the audio clips into words. Conceptually, our objective is to search for the most likely sequence of words according to these models.

However, searching for all the possible sequences sounds astonishingly inefficient.



The overall picture for decoding-graph creation of Kaldi is that it is constructing the graph HCLG.fst, which is composed of 4 files derived from 4 fst files, i.e. H.fst, C.fst, L.fst, and G.fst.

- G is for Grammar Model, or Language Model. Technically, it is an acceptor (i.e. its input and output symbols are the same) that encodes the grammar or language model. To make things easier to cooperate with the other three WFST, treat it as an input and output same WFST.
- L is for Lexicon Dictionary. Its output symbols are words and its input symbols are phones. It's the source of G.
- C represents the Context-dependency: its output symbols are phones and its input symbols represent context-dependent phones, i.e. windows of N phones.
- H contains the HMM definitions; its output symbols represent context-dependent phones and its input symbols are transition-ids, encoding the pdf-id and other information.

If we were to summarize our approach on one line (and one line can't capture all the details, obviously), the line would probably as follows, where asl == "add-self-loops" and rds == "remove-disambiguation-symbols", and H' is H without the self-loops:

HCLG = asl(min(rds(det(H' o min(det(C o min(det(L o G))))))))

The training sequence is: G -> L -> C -> H.

	transducer	input sequence	output sequence
G	word-level grammar	words	words
L	pronunciation lexicon	phones	words
C	context-dependency	CD phones	phones
H	HMM	HMM states	CD phones

Lattice is to be used to store the candidate sequence during decoding process, which goes through all the branches to get the best score, N-best to output the text.

The essence of the Lattice is a directed acyclic graph. Each node of the graph represents the ending time of a word, each line represents a possible word and the word's acoustic score and language mode score.

Computing the denominator involves summing over all possible word sequences: estimate by generating lattices, and summing over all words in the lattice. In practice also compute numerator statistics using lattices (useful for summing multiple pronunciations).

Generate numerator and denominator lattices for every training utterance. Denominator lattice uses recognition setup (with a weaker language model). Each word in the lattice is decoded to give a phone segmentation, and forward-backward is then used to compute the state occupation probabilities.

6.3.1 Build up Decoding Graph

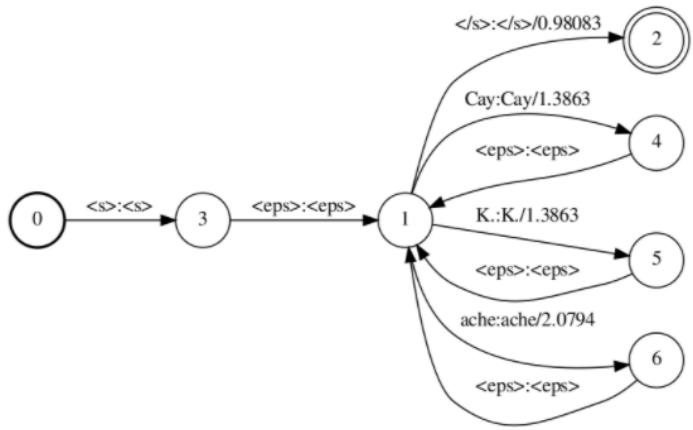
We need build up decoding graph before decoding. Line 71 telling us that it will be done by:

```
$ utils/mkgraph.sh --mono data/lang_test_bg exp/mono
exp/mono/graph
```

`utils/mkgraph.sh` creates an extendable decoding graph HCLG, which contains language model, phone dictionary, context, and HMM structure. With HCLG, `steps/decode.sh` could decode the speech to FST.

Check to make sure `L_disambig`.fst and `G.fst` existed.

```
53  # If $lang/tmp/LG.fst does not exist or is older than its sources, make it...
54  # (note: the [[ ]] brackets make the || type operators work (inside [ ], we
55  # would have to use -o instead),  -f means file exists, and -ot means older than).
56
57  required="$lang/L_disambig.fst $lang/G.fst $lang/phones.txt $lang/words.txt
    $lang/phones/silence.csl $lang/phones/disambig.int $model $tree"
58  for f in $required; do
59    [ ! -f $f ] && echo "mkgraph.sh: expected $f to exist" && exit 1;
60  done
61
62  if [ -f $dir/HCLG.fst ]; then
63    # detect when the result already exists, and avoid overwriting it.
64    must_rebuild=false
65    for f in $required; do
66      [ $f -nt $dir/HCLG.fst ] && must_rebuild=true
67    done
68    if ! $must_rebuild; then
69      echo "$0: $dir/HCLG.fst is up to date."
70      exit 0
71    fi
72  fi
```



from http://vpanayotov.blogspot.com/2012/06/kaldi-decoding-graph-construction.html

Creates new language model: $lang/tmp/LG.fst based on L_disambig.fst and G.fst.

```
75    N=$(tree-info $tree | grep "context-width" | cut -d' ' -f2) || { echo "Error when
      getting context-width"; exit 1; }
76    P=$(tree-info $tree | grep "central-position" | cut -d' ' -f2) || { echo "Error
      when getting central-position"; exit 1; }
77
78    [[ -f $2/frame_subsampling_factor && "$loopscale" == "0.1" ]] && \
79      echo "$0: WARNING: chain models need '--self-loop-scale 1.0'";
80
81    if [ -f $lang/phones/nonterm_phones_offset.int ]; then
82      if [[ $N != 2  || $P != 1 ]]; then
83        echo "$0: when doing grammar decoding, you can only build graphs for
          left-biphone trees."
84        exit 1
85      fi
86      nonterm_phones_offset=$(cat $lang/phones/nonterm_phones_offset.int)
87      nonterm_opt="--nonterm-phones-offset=$nonterm_phones_offset"
88      prepare_grammar_command="make-grammar-fst
        --nonterm-phones-offset=$nonterm_phones_offset - -"
89    else
90      prepare_grammar_command="cat"
91      nonterm_opt=
92    fi
93
94    mkdir -p $lang/tmp
95    trap "rm -f $lang/tmp/LG.fst.$$" EXIT HUP INT PIPE TERM
96    # Note: [[ ]] is like [ ] but enables certain extra constructs, e.g. || in
97    # place of -o
98    if [[ ! -s $lang/tmp/LG.fst || $lang/tmp/LG.fst -ot $lang/G.fst || \
99          $lang/tmp/LG.fst -ot $lang/L_disambig.fst ]]; then
100     fsttablecompose $lang/L_disambig.fst $lang/G.fst | fstdeterminizestar --use-log=
        true | \
101       fstminimizeencoded | fstpushspecial > $lang/tmp/LG.fst.$$ || exit 1;
102     mv $lang/tmp/LG.fst.$$ $lang/tmp/LG.fst
103     fstisstochastic $lang/tmp/LG.fst || echo "[info]: LG not stochastic."
104   fi
105
```

With new LG.fst and disambig.int, create new $lang/tmp/CLG_1_0.fst , ilabels_1_0, and disambig_ilabels_1_0.int.

```
106   clg=$lang/tmp/CLG_${N}_${P}.fst
107   clg_tmp=$clg.$$
108   ilabels=$lang/tmp/ilabels_${N}_${P}
109   ilabels_tmp=$ilabels.$$
110   trap "rm -f $clg_tmp $ilabels_tmp" EXIT HUP INT PIPE TERM
111   if [[ ! -s $clg || $clg -ot $lang/tmp/LG.fst \
112         || ! -s $ilabels || $ilabels -ot $lang/tmp/LG.fst ]]; then
113     fstcomposecontext $nonterm_opt --context-size=$N --central-position=$P \
114      --read-disambig-syms=$lang/phones/disambig.int \
115      --write-disambig-syms=$lang/tmp/disambig_ilabels_${N}_${P}.int \
116       $ilabels_tmp $lang/tmp/LG.fst |\
117       fstarcsort --sort_type=ilabel > $clg_tmp
118     mv $clg_tmp $clg
119     mv $ilabels_tmp $ilabels
120     fstisstochastic $clg || echo "[info]: CLG not stochastic."
121   fi
```

Then create new `exp/mono/graph/HCLGa.fst` with $tree and $model.

run. sh mkgraph. sh

```
122
123     trap "rm -f $dir/Ha.fst.$$" EXIT HUP INT PIPE TERM
124   if [[ ! -s $dir/Ha.fst || $dir/Ha.fst -ot $model  \
125         || $dir/Ha.fst -ot $lang/tmp/ilabels_${N}_${P} ]]; then
126       make-h-transducer $nonterm_opt --disambig-syms-out=$dir/disambig_tid.int \
127         --transition-scale=$tscale $lang/tmp/ilabels_${N}_${P} $tree $model \
128         > $dir/Ha.fst.$$  || exit 1;
129       mv $dir/Ha.fst.$$ $dir/Ha.fst
130     fi
131
132     trap "rm -f $dir/HCLGa.fst.$$" EXIT HUP INT PIPE TERM
133   if [[ ! -s $dir/HCLGa.fst || $dir/HCLGa.fst -ot $dir/Ha.fst || \
134           $dir/HCLGa.fst -ot $clg ]]; then
135     if $remove_oov; then
136       [ ! -f $lang/oov.int ] && \
137         echo "$0: --remove-oov option: no file $lang/oov.int" && exit 1;
138       clg="fstrmsymbols --remove-arcs=true --apply-to-output=true $lang/oov.int
          $clg|"
139     fi
140       fsttablecompose $dir/Ha.fst "$clg" | fstdeterminizestar --use-log=true \
141         | fstrmsymbols $dir/disambig_tid.int | fstrmepslocal | \
142         fstminimizeencoded > $dir/HCLGa.fst.$$ || exit 1;
143       mv $dir/HCLGa.fst.$$ $dir/HCLGa.fst
144       fstisstochastic $dir/HCLGa.fst || echo "HCLGa is not stochastic"
145     fi
146
147     trap "rm -f $dir/HCLG.fst.$$" EXIT HUP INT PIPE TERM
148   if [[ ! -s $dir/HCLG.fst || $dir/HCLG.fst -ot $dir/HCLGa.fst ]]; then
149       add-self-loops --self-loop-scale=$loopscale --reorder=true $model $dir/HCLGa.fst
          | \
150         $prepare_grammar_command | \
151         fstconvert --fst_type=const > $dir/HCLG.fst.$$ || exit 1;
152       mv $dir/HCLG.fst.$$ $dir/HCLG.fst
153     if [ $tscale == 1.0 -a $loopscale == 1.0 ]; then
154       # No point doing this test if transition-scale not 1, as it is bound to fail.
155       fstisstochastic $dir/HCLG.fst || echo "[info]: final HCLG is not stochastic."
156     fi
157     fi
158
159   # note: the empty FST has 66 bytes.  this check is for whether the final FST
160     # is the empty file or is the empty FST.
161   if ! [ $(head -c 67 $dir/HCLG.fst | wc -c) -eq 67 ]; then
162       echo "$0: it looks like the result in $dir/HCLG.fst is empty"
163       exit 1
164     fi
165
```

```
leo@LeoUbuntu:~/lkaldi/egs/timit/s5/exp/mono/graph$
.
├── disambig_tid.int
├── HCLG.fst
├── num_pdfs
├── phones
├── phones.txt
└── words.txt

1 directory, 5 files
```

The final result is:

```
tree-info exp/mono/tree
tree-info exp/mono/tree
fsttablecompose     data/lang_test_bg/L_disambig.fst
data/lang_test_bg/G.fst
fstdeterminizestar --use-log=true
fstpushspecial
fstminimizeencoded
fstisstochastic data/lang_test_bg/tmp/LG.fst
    -0.00841335 -0.00928529
fstcomposecontext   --context-size=1   --central-po-
sition=0     --read-disambig-syms=data/lang_test_bg/
phones/disambig.int --write-disambig-syms=data/lang_
test_bg/tmp/disambig_ilabels_1_0.int data/lang_test_
bg/tmp/ilabels_1_0
     fstisstochastic data/lang_test_bg/tmp/CLG_1_0.
fst
    -0.00841335 -0.00928515
make-h-transducer      --disambig-syms-out=exp/mono/
graph/disambig_tid.int  --transition-scale=1.0  data/
lang_test_bg/tmp/ilabels_1_0 exp/mono/tree exp/mono/
final.mdl
    fstminimizeencoded
    fstdeterminizestar --use-log=true
fsttablecompose   exp/mono/graph/Ha.fst   data/lang_
test_bg/tmp/CLG_1_0.fst
    fstrmsymbols exp/mono/graph/disambig_tid.int
    fstrmepslocal
    fstisstochastic exp/mono/graph/HCLGa.fst
    0.000381767 -0.00951546
add-self-loops  --self-loop-scale=0.1  --reorder=true
exp/mono/final.mdl
```

final.mdl is what we need.

6.3.2 Decoding

We decode based on developing and testing separately. Line 73 & 74:

```
steps/decode.sh --nj "$decode_nj" --cmd "$decode_
cmd" exp/mono/graph data/dev exp/mono/decode_dev

steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd"
exp/mono/graph data/test exp/mono/decode_test
```

The decoding log will be stored at `exp/mono/decode_dev/log` and `exp/mono/decode_test/log`. Below is the decoding output of developing section.

```
steps/decode.sh --nj 1 --cmd run.pl --max-jobs-run 10 exp/mono/
graph data/dev exp/mono/decode_dev

decode.sh: feature type is delta

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-jobs-run 10
exp/mono/graph exp/mono/decode_dev

steps/diagnostic/analyze_lats.sh: see stats in exp/mono/decode_
dev/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percentile)=(5,26,120) and
mean=61.6

steps/diagnostic/analyze_lats.sh: see stats in exp/mono/decode_
dev/log/analyze_lattice_depth_stats.log
```

Following, we could finish other training and decoding.

`final.mdl`, along with `words.txt` (dictionary) and `HCLG.fst` below `./graph_word`, we could recognize the speech.

Decoding process results in a huge HMM. In theory, we can use it with the Viterbi decoding to find the most likely sequence of words.

In reality, this involves a whole lot of works and simplifications. We need to optimize this huge graph for faster decoding. We apply determinations to make it easier to search deterministically and minimization to reduce the number of redundancy in states and transitions.

6.5 Final Results and Output

Here we go:

```
#!/bin/bash

for x in exp/{mono,tri,sgmm,dnn,combine}*/decode*; do [
-d $x ] && echo $x | grep "${1:-.*}" >/dev/null && grep
WER $x/wer_* 2>/dev/null | utils/best_wer.sh; done

for x in exp/{mono,tri,sgmm,dnn,combine}*/decode*; do [ -d
$x ] && echo $x | grep "${1:-.*}" >/dev/null && grep Sum
$x/score_*/*.sys 2>/dev/null | utils/best_wer.sh; done
```

See my computer's result:

```
%WER 31.8 | 400 15057 | 71.7 19.4 8.9 3.5 31.8 100.0 |
-0.496 | exp/mono/decode_dev/score_5/ctm_39phn.filt.sys

%WER 32.2 | 192 7215 | 70.5 19.4 10.0 2.8 32.2 100.0 |
-0.211 | exp/mono/decode_test/score_6/ctm_39phn.filt.sys
```

WER is 31.8% and 32.2% for developing set and testing set separately. Not bad.

The most popular metric in ASR is WER. Consider a corpse with N words, WER (Word error rate) is the ratio on how many insertions, deletions, and substitutions (a.k.a. edit distance) are needed to convert our predictions to the ground truth.

$$WER = (I + D + S) / N$$

edit distance
(number of insert, delete and substitute)

word count in the referencec (ground truth)

Another metric is perplexity, which we use to measure how well a probability distribution in making predictions. For a good LVCSR and language model, the prediction should have low entropy and perplexity.

Intuitively, we can view perplexity being the average number of choices a random variable has. In a language model, perplexity is a measure of on average how many probable words can follow a sequence of words. For a good language model, the choices should be small. If any word is equally likely, the perplexity will be high and equals the number of words in the vocabulary.

```
=====================================================
            MonoPhone Training & Decoding
=====================================================
steps/train_mono.sh --nj 30 --cmd run.pl --max-jobs-
run 10 data/train data/lang exp/mono
steps/train_mono.sh: Initializing monophone system.
steps/train_mono.sh: Compiling training graphs
steps/train_mono.sh: Aligning data equally (pass 0)
steps/train_mono.sh: Pass 1
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 2
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 3
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 4
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 5
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 6
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 7
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 8
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 9
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 10
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 11
steps/train_mono.sh: Pass 12
steps/train_mono.sh: Aligning data
```

```
steps/train_mono.sh: Pass 13
steps/train_mono.sh: Pass 14
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 15
steps/train_mono.sh: Pass 16
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 17
steps/train_mono.sh: Pass 18
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 19
steps/train_mono.sh: Pass 20
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 21
steps/train_mono.sh: Pass 22
steps/train_mono.sh: Pass 23
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 24
steps/train_mono.sh: Pass 25
steps/train_mono.sh: Pass 26
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 27
steps/train_mono.sh: Pass 28
steps/train_mono.sh: Pass 29
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 30
steps/train_mono.sh: Pass 31
steps/train_mono.sh: Pass 32
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 33
steps/train_mono.sh: Pass 34
steps/train_mono.sh: Pass 35
```

```
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 36
steps/train_mono.sh: Pass 37
steps/train_mono.sh: Pass 38
steps/train_mono.sh: Aligning data
steps/train_mono.sh: Pass 39
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/mono
steps/diagnostic/analyze_alignments.sh: see stats in
exp/mono/log/analyze_alignments.log
2 warnings in exp/mono/log/align.*.*.log
exp/mono: nj=30 align prob=-99.15 over 3.12h [re-
try=0.0%, fail=0.0%] states=144 gauss=986
steps/train_mono.sh: Done training monophone system
in exp/mono
tree-info exp/mono/tree
tree-info exp/mono/tree
fsttablecompose    data/lang_test_bg/L_disambig.fst
data/lang_test_bg/G.fst
fstminimizeencoded
fstpushspecial
fstdeterminizestar --use-log=true
fstisstochastic data/lang_test_bg/tmp/LG.fst
-0.00841336 -0.00928521
fstcomposecontext   --context-size=1   --central-po-
sition=0    --read-disambig-syms=data/lang_test_bg/
phones/disambig.int      --write-disambig-syms=data/
lang_test_bg/tmp/disambig_ilabels_1_0.int data/lang_
test_bg/tmp/ilabels_1_0.11459 data/lang_test_bg/tmp/
LG.fst
fstisstochastic data/lang_test_bg/tmp/CLG_1_0.fst
-0.00841336 -0.00928521
make-h-transducer     --disambig-syms-out=exp/mono/
graph/disambig_tid.int --transition-scale=1.0 data/
```

```
lang_test_bg/tmp/ilabels_1_0 exp/mono/tree exp/mono/
final.mdl

fstrmepslocal

fsttablecompose   exp/mono/graph/Ha.fst   data/lang_
test_bg/tmp/CLG_1_0.fst

fstdeterminizestar --use-log=true

fstrmsymbols exp/mono/graph/disambig_tid.int

fstminimizeencoded

fstisstochastic exp/mono/graph/HCLGa.fst

0.000381709 -0.00951555

add-self-loops  --self-loop-scale=0.1  --reorder=true
exp/mono/final.mdl exp/mono/graph/HCLGa.fst

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run 10
exp/mono/graph data/dev exp/mono/decode_dev

decode.sh: feature type is delta

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/mono/graph exp/mono/decode_dev

steps/diagnostic/analyze_lats.sh: see stats in exp/
mono/decode_dev/log/analyze_alignments.log

Overall,     lattice     depth     (10,50,90-percen-
tile)=(5,25,121) and mean=55.2

steps/diagnostic/analyze_lats.sh: see stats in exp/
mono/decode_dev/log/analyze_lattice_depth_stats.log

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run 10
exp/mono/graph data/test exp/mono/decode_test

decode.sh: feature type is delta

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/mono/graph exp/mono/decode_test

steps/diagnostic/analyze_lats.sh: see stats in exp/
mono/decode_test/log/analyze_alignments.log

Overall,     lattice     depth     (10,50,90-percen-
tile)=(6,27,142) and mean=76.0

steps/diagnostic/analyze_lats.sh: see stats in exp/
mono/decode_test/log/analyze_lattice_depth_stats.
log
```

Chapter 7. Triphone: Deltas

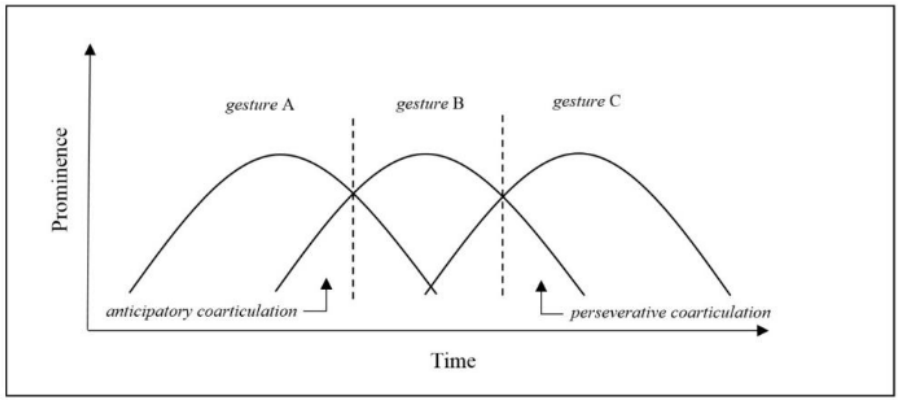
Tri1: Deltas + Delta-Deltas

Three steps for ASR: phone, word, sentence. Monophone brings us the phones. We use triphone model to get word.

The key issue for Speech Recognition (not only Kaldi, but the whole field) is the speech variation, which occur everywhere: channel/microphone type, environmental noise, speaking style, vocal anatomy, gender, accent, health, etc.

Speech is a continuous acoustic flow. It is composed of partially stable status and partially dynamic status. A word's phonetic (or wave) in reality is affected by many issues, not just phone. For example, context of the phone, speaker, style, dialect, etc.

Coarticulation leads to the difference between feeling and standard of the phone. We have to rely on context to identify a phone in daily life.



7.1 General

To solve coarticulation problem, we divided the phone into several sub-phone units. For example, the word “speech”, the first part of the phone is connected with the previous word, the middle part is stable, the third part is connected with the following word. This is the reason that we use Tri-Phone model when using HMM model to recognize the speech. If just considering the previous word, it’s called Bi-Phone.

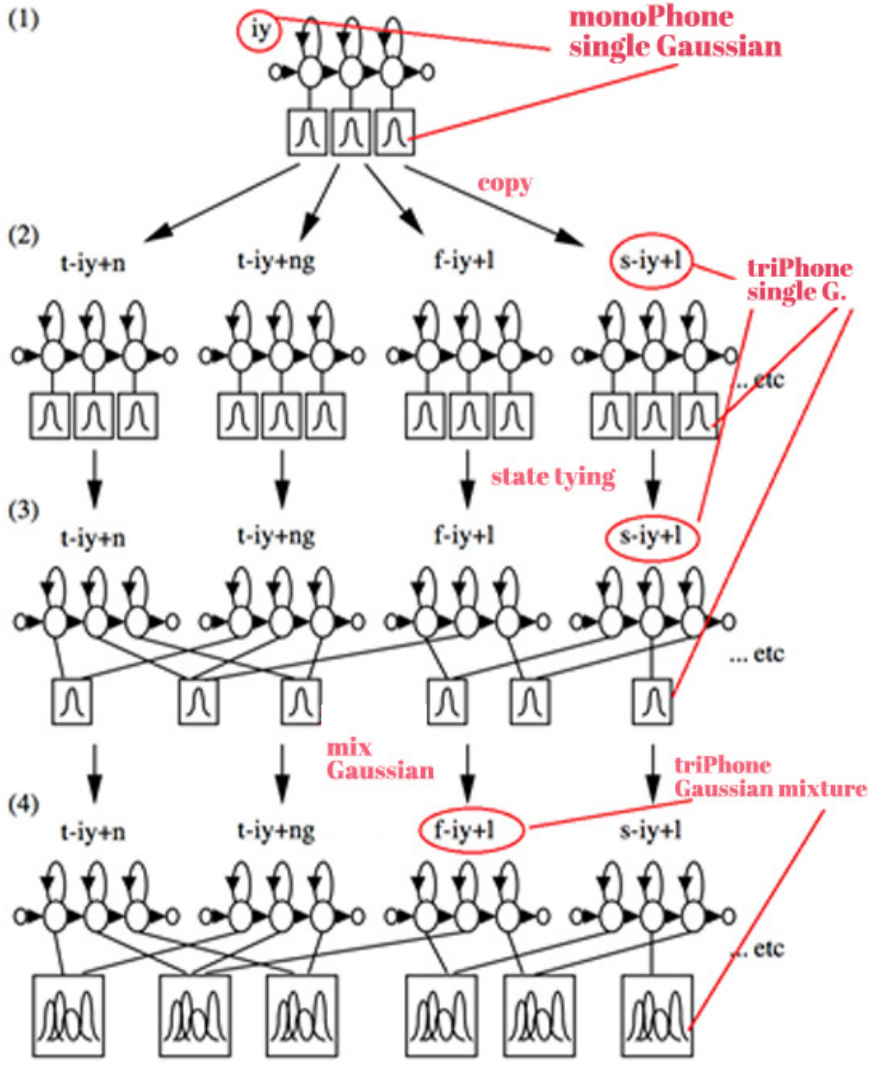
Putting the phone into context, we get tri-phone or senone models. Senone is not just two or three more words, but connected with decision tree and some other more complicated functions.

Triphone is one of phones. Comparing with monophone (i.e. t, iy, n), triphone is represented as t-iy+n, three monophones. It’s like monophone iy, but put the context into the consideration, which means the phone before iy is t, and after iy is n.

Triphone is a useful sub-word unit as it models phone-in-context, and thus captures the most important coarticulation effect. As both left and right contexts are involved, the number of such triphones would be very large.

How many triphones are there? People believe English has about 40 phones. In this case, there will be 40 x 40 x 40 = 64k possible triphones. In a cross-word system, 60k three-state HMMs, with 10 component Gaussian mixtures per state: 1.8M Gaussians 39-dimension feature vectors (12 MFCCs + energy), deltas and accelerations. Assuming diagonal Gaussians: about 790 parameters/state. Total will be well above 1 billion parameters. We would need a very large amount of training data to train such a system.

The general steps for triphone training:



The main difference between triPhone and monoPhone is decision tree state tying. The principle, class, and program are the same. TriPhone and monoPhone both are HMM.

7.2 tri1 : Deltas + Delta-Deltas Analysis

Here is the script.

```
79  echo ============================================================================
80  echo "           tri1 : Deltas + Delta-Deltas Training & Decoding               "
81  echo ============================================================================
82
83  steps/align_si.sh --boost-silence 1.25 --nj "$train_nj" --cmd "$train_cmd" \
84   data/train data/lang exp/mono exp/mono_ali
85
86  # Train tri1, which is deltas + delta-deltas, on train data.
87  steps/train_deltas.sh --cmd "$train_cmd" \
88   $numLeavesTri1 $numGaussTri1 data/train data/lang exp/mono_ali exp/tri1
89
90  utils/mkgraph.sh data/lang_test_bg exp/tri1 exp/tri1/graph
91
92  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
93   exp/tri1/graph data/dev exp/tri1/decode_dev
94
95  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
96   exp/tri1/graph data/test exp/tri1/decode_test
97
```

Line 83 `steps/align_si.sh` aligns data based on mono_ali.

```
if [ $# != 4 ]; then
   echo "usage: steps/align_si.sh <data-dir> <lang-dir> <src-dir> <align-dir>"
   echo "e.g.:  steps/align_si.sh data/train data/lang exp/tri1 exp/tri1_ali"
   echo "main options (for others, see top of script file)"
   echo "  --config <config-file>                          # config containing options"
   echo "  --nj <nj>                                       # number of parallel jobs"
   echo "  --use-graphs true                               # use graphs in src-dir"
   echo "  --cmd (utils/run.pl|utils/queue.pl <queue opts>) # how to run jobs."
   exit 1;
fi
```

In ali, each utt will be related with a corresponding transition id. Each transition-id will have transition-state and transition-index. Transition-state is one-to-one related to phone-id, hmm-state, pdf-state. Where phone-id and hmm-state are (0,1,2). pdf-id is the leave's number of the tree, corresponding to an independent phone type.

To a specific hmm-state, there are two conversions: forward pdf and self-look pdf. They are same in most cases.

Transition-index marks the source: forward or self-loop.

We could conclude that the main purpose of ali is to link the featured vector with the specific phone status.

Line 87 `steps/train_deltas.sh`, the main part is:

run.sh train_deltas.sh

```
77
78  if [ $stage -le -3 ]; then
79    echo "$0: accumulating tree stats"
80    $cmd JOB=1:$nj $dir/log/acc_tree.JOB.log \
81      acc-tree-stats $context_opts \
82      --ci-phones=$ciphonelist $alidir/final.mdl "$feats" \
83      "ark:gunzip -c $alidir/ali.JOB.gz|" $dir/JOB.treeacc || exit 1;
84    sum-tree-stats $dir/treeacc $dir/*.treeacc 2>$dir/log/sum_tree_acc.log || exit 1;
85    rm $dir/*.treeacc
86  fi
87
88  if [ $stage -le -2 ]; then
89    echo "$0: getting questions for tree-building, via clustering"
90    # preparing questions, roots file...
91    cluster-phones $context_opts $dir/treeacc $lang/phones/sets.int \
92      $dir/questions.int 2> $dir/log/questions.log || exit 1;
93    cat $lang/phones/extra_questions.int >> $dir/questions.int
94    compile-questions $context_opts $lang/topo $dir/questions.int \
95      $dir/questions.qst 2>$dir/log/compile_questions.log || exit 1;
96
97    echo "$0: building the tree"
98    $cmd $dir/log/build_tree.log \
99      build-tree $context_opts --verbose=1 --max-leaves=$numleaves \
100     --cluster-thresh=$cluster_thresh $dir/treeacc $lang/phones/roots.int \
101     $dir/questions.qst $lang/topo $dir/tree || exit 1;
102
103   $cmd $dir/log/init_model.log \
104     gmm-init-model  --write-occs=$dir/1.occs  \
105       $dir/tree $dir/treeacc $lang/topo $dir/1.mdl || exit 1;
106   if grep 'no stats' $dir/log/init_model.log; then
107     echo "** The warnings above about 'no stats' generally mean you have phones **"
108     echo "** (or groups of phones) in your phone set that had no corresponding data. **"
109     echo "** You should probably figure out whether something went wrong, **"
110     echo "** or whether your data just doesn't happen to have examples of those **"
111     echo "** phones. **"
112   fi
113
114   gmm-mixup --mix-up=$numgauss $dir/1.mdl $dir/1.occs $dir/1.mdl 2>$dir/log/mixup.log || exit 1;
115   rm $dir/treeacc
116 fi
117
118 if [ $stage -le -1 ]; then
119   # Convert the alignments.
120   echo "$0: converting alignments from $alidir to use current tree"
121   $cmd JOB=1:$nj $dir/log/convert.JOB.log \
122     convert-ali $alidir/final.mdl $dir/1.mdl $dir/tree \
123     "ark:gunzip -c $alidir/ali.JOB.gz|" "ark:|gzip -c >$dir/ali.JOB.gz" || exit 1;
```

Line 81 is to accumulate tree stats. Using acc-tree-stats and sum-tree-stats, based on alignment information, accumulate the context features of each monophone.

Line 91 cluster-phones and line 94 compile-questions get questions for tree-building.

Based on above, line 99 `build-tree` build up the decision tree.

Line 104 `gmm-init-model` and line 114 `gmm-mixup` create the primary model based on the alignment previous done. It will calculate the mean and variance of each leaf of the decision tree, and serve as the primary model for each status.

run.sh train_deltas.sh

```
118 if [ $stage -le -1 ]; then
119   # Convert the alignments.
120   echo "$0: converting alignments from $alidir to use current tree"
121   $cmd JOB=1:$nj $dir/log/convert.JOB.log \
122     convert-ali $alidir/final.mdl $dir/1.mdl $dir/tree \
123     "ark:gunzip -c $alidir/ali.JOB.gz|" "ark:|gzip -c >$dir/ali.JOB.gz" || exit 1;
124 fi
125
126 if [ $stage -le 0 ]; then
127   echo "$0: compiling graphs of transcripts"
128   $cmd JOB=1:$nj $dir/log/compile_graphs.JOB.log \
129     compile-train-graphs --read-disambig-syms=$lang/phones/disambig.int $dir/tree $dir/1.mdl  $lang/L.fst  \
130       "ark:utils/sym2int.pl --map-oov $oov -f 2- $lang/words.txt < $sdata/JOB/text |" \
131       "ark:|gzip -c >$dir/fsts.JOB.gz" || exit 1;
132 fi
133
134 x=1
135 while [ $x -lt $num_iters ]; do
136   echo "$0: training pass $x"
137   if [ $stage -le $x ]; then
138     if echo $realign_iters | grep -w $x >/dev/null; then
139       echo "$0: aligning data"
140       mdl="gmm-boost-silence --boost=$boost_silence `cat $lang/phones/optional_silence.csl` $dir/$x.mdl - |"
141       $cmd JOB=1:$nj $dir/log/align.$x.JOB.log \
142         gmm-align-compiled $scale_opts --beam=$beam --retry-beam=$retry_beam --careful=$careful "$mdl" \
143           "ark:gunzip -c $dir/fsts.JOB.gz|" "$feats" \
144           "ark:|gzip -c >$dir/ali.JOB.gz" || exit 1;
145     fi
146     $cmd JOB=1:$nj $dir/log/acc.$x.JOB.log \
147       gmm-acc-stats-ali  $dir/$x.mdl "$feats" \
148       "ark,s,cs:gunzip -c $dir/ali.JOB.gz|" $dir/$x.JOB.acc || exit 1;
149     $cmd $dir/log/update.$x.log \
150       gmm-est --mix-up=$numgauss --power=$power \
151         --write-occs=$dir/$[$x+1].occs $dir/$x.mdl \
152       "gmm-sum-accs - $dir/$x.*.acc |" $dir/$[$x+1].mdl || exit 1;
153     rm $dir/$x.mdl $dir/$x.*.acc
154     rm $dir/$x.occs
155   fi
156   [ $x -le $max_iter_inc ] && numgauss=$[$numgauss+$incgauss];
157   x=$[$x+1];
158 done
159
160 rm $dir/final.mdl $dir/final.occs 2>/dev/null
161 ln -s $x.mdl $dir/final.mdl
162 ln -s $x.occs $dir/final.occs
163
164 steps/diagnostic/analyze_alignments.sh --cmd "$cmd" $lang $dir
```

Line 122 convert-ali converting alignments from $alidir to use current tree by invoking ali/final.mdl, dir/1.mdl, dir/tree to convert ali/ali.gz to dir/ali.gz. which match the original speech to the decision tree.

Line 129 compile-train-graphs compiling graphs of transcripts based on the fst created in previous parts.

compile-train-graphs map the utt to lexicon and then create the new state fst combining context ,topo ,and tree. It creates dir/

fsts.JOB.gz. The iteration will be based on the above alignment.

The rest is just keeping iteration for better result. The default is 35 iterations. When reaching 10, 20, and 30 iteration, Kaldi will re-ali to create new ali.gz for further processing.

The `ali.gz` won't change in other iterations (excluding 10, 20, and 30). Kaldi only make state alignment and calculate expectation and maximize it. Line 147 gmm-acc-stats-ali and line 150 gmm-est train GMM for each state.

`incgauss=$[($totgauss-$numgauss)/$max_iter_inc]` and the max_iter_inc = 25, the maximum Gauss is 25, 25 iterations. numgauss = numleaves. It implies that each leaf has one Gauss at the beginging. Adding 1 to incgauss for each iteration, and till 25 maximum.

Now, we could understand why `train_deltas` is working better than `train_mono`.

Each monophone will have different pronunciation in different context environment. The `train_mono` could only provide one GMM to fit the situation. The work is far from complete.

While train_deltas will use different GMM to fit the monophone based on different context. For example, one GMM to fit w_a_n and another GMM fitting b_a_i. Both GMM outputs will map to the mono-phone a. We use different model to identify the same monophone's different situation, so that the same monophone's multiple status and changing problem.

The features of "a" will be easier to be aligned to "a". From this we could see that ali file actually maps each feature vector to the phone state.

7.3 Output

```
====================================================================
           tri1 : Deltas + Delta-Deltas Training & Decoding
====================================================================

steps/align_si.sh --boost-silence 1.25 --nj 30 --cmd run.pl
--max-jobs-run 10 data/train data/lang exp/mono exp/mono_ali

steps/align_si.sh: feature type is delta

steps/align_si.sh: aligning data in data/train using mod-
el from exp/mono, putting alignments in exp/mono_ali

steps/diagnostic/analyze_alignments.sh --cmd run.
pl --max-jobs-run 10 data/lang exp/mono_ali

steps/diagnostic/analyze_alignments.sh: see stats
in exp/mono_ali/log/analyze_alignments.log

steps/align_si.sh: done aligning data.

steps/train_deltas.sh --cmd run.pl --max-jobs-run 10
2500 15000 data/train data/lang exp/mono_ali exp/tri1

steps/train_deltas.sh: accumulating tree stats

steps/train_deltas.sh: getting ques-
tions for tree-building, via clustering

steps/train_deltas.sh: building the tree

steps/train_deltas.sh: converting align-
ments from exp/mono_ali to use current tree

steps/train_deltas.sh: compiling graphs of transcripts

steps/train_deltas.sh: training pass 1

steps/train_deltas.sh: training pass 2

steps/train_deltas.sh: training pass 3

steps/train_deltas.sh: training pass 4

steps/train_deltas.sh: training pass 5

steps/train_deltas.sh: training pass 6

steps/train_deltas.sh: training pass 7

steps/train_deltas.sh: training pass 8

steps/train_deltas.sh: training pass 9

steps/train_deltas.sh: training pass 10

steps/train_deltas.sh: aligning data

steps/train_deltas.sh: training pass 11

steps/train_deltas.sh: training pass 12

steps/train_deltas.sh: training pass 13

steps/train_deltas.sh: training pass 14
```

```
steps/train_deltas.sh: training pass 15
steps/train_deltas.sh: training pass 16
steps/train_deltas.sh: training pass 17
steps/train_deltas.sh: training pass 18
steps/train_deltas.sh: training pass 19
steps/train_deltas.sh: training pass 20
steps/train_deltas.sh: aligning data
steps/train_deltas.sh: training pass 21
steps/train_deltas.sh: training pass 22
steps/train_deltas.sh: training pass 23
steps/train_deltas.sh: training pass 24
steps/train_deltas.sh: training pass 25
steps/train_deltas.sh: training pass 26
steps/train_deltas.sh: training pass 27
steps/train_deltas.sh: training pass 28
steps/train_deltas.sh: training pass 29
steps/train_deltas.sh: training pass 30
steps/train_deltas.sh: aligning data
steps/train_deltas.sh: training pass 31
steps/train_deltas.sh: training pass 32
steps/train_deltas.sh: training pass 33
steps/train_deltas.sh: training pass 34
steps/diagnostic/analyze_alignments.sh --cmd run.
pl --max-jobs-run 10 data/lang exp/tri1
steps/diagnostic/analyze_alignments.sh: see stats
in exp/tri1/log/analyze_alignments.log
63 warnings in exp/tri1/log/init_model.log
44 warnings in exp/tri1/log/update.*.log
1 warnings in exp/tri1/log/compile_questions.log
exp/tri1: nj=30 align prob=-95.27 over 3.12h [retry=0.0%,
fail=0.0%] states=1872 gauss=15037 tree-impr=5.40
steps/train_deltas.sh: Done training sys-
tem with delta+delta-delta features in exp/tri1
tree-info exp/tri1/tree
tree-info exp/tri1/tree
fstcomposecontext --context-size=3 --central-posi-
tion=1 --read-disambig-syms=data/lang_test_bg/phones/
disambig.int --write-disambig-syms=data/lang_test_bg/
```

```
tmp/disambig_ilabels_3_1.int data/lang_test_bg/tmp/
ilabels_3_1.8577 data/lang_test_bg/tmp/LG.fst

fstisstochastic data/lang_test_bg/tmp/CLG_3_1.fst

0 -0.00928518

make-h-transducer --disambig-syms-out=exp/tri1/graph/dis-
ambig_tid.int --transition-scale=1.0 data/lang_test_bg/
tmp/ilabels_3_1 exp/tri1/tree exp/tri1/final.mdl

fsttablecompose exp/tri1/graph/Ha.fst data/
lang_test_bg/tmp/CLG_3_1.fst

fstrmepslocal

fstminimizeencoded

fstrmsymbols exp/tri1/graph/disambig_tid.int

fstdeterminizestar --use-log=true

fstisstochastic exp/tri1/graph/HCLGa.fst

0.000443876 -0.0175772

HCLGa is not stochastic

add-self-loops --self-loop-scale=0.1 --reorder=true
exp/tri1/final.mdl exp/tri1/graph/HCLGa.fst

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run
10 exp/tri1/graph data/dev exp/tri1/decode_dev

decode.sh: feature type is delta

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri1/graph exp/tri1/decode_dev

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri1/decode_dev/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percen-
tile)=(3,11,42) and mean=18.9

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri1/decode_dev/log/analyze_lattice_depth_stats.log

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run
10 exp/tri1/graph data/test exp/tri1/decode_test

decode.sh: feature type is delta

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri1/graph exp/tri1/decode_test

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri1/decode_test/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percen-
tile)=(3,12,47) and mean=21.7

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri1/decode_test/log/analyze_lattice_depth_stats.log
```

Chapter 8. tri2: LDA + MLLT

Discriminant Analysis

LDA + MLLT refers to the way we transform the features after computing the MFCCs: we splice across several frames, reduce the dimension (to 40 by default) using Linear Discriminant Analysis), and then later estimate, over multiple iterations, a diagonalizing transform known as MLLT or STC. *(from http://Kaldi-asr.org/doc/transform.html)*

8.1 General

Here is the script:

```
98  echo ============================================================================
99  echo "                 tri2 : LDA + MLLT Training & Decoding                    "
100 echo ============================================================================
101
102 steps/align_si.sh --nj "$train_nj" --cmd "$train_cmd" \
103   data/train data/lang exp/tri1 exp/tri1_ali
104
105 steps/train_lda_mllt.sh --cmd "$train_cmd" \
106  --splice-opts "--left-context=3 --right-context=3" \
107  $numLeavesMLLT $numGaussMLLT data/train data/lang exp/tri1_ali exp/tri2
108
109 utils/mkgraph.sh data/lang_test_bg exp/tri2 exp/tri2/graph
110
111 steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
112  exp/tri2/graph data/dev exp/tri2/decode_dev
113
114 steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
115  exp/tri2/graph data/test exp/tri2/decode_test
116
```

Let's examine the output and see the difference:

```
steps/train_lda_mllt.sh --cmd slurm.pl --mem 4G 2500
20000 data/train data/lang exp/tri2_ali exp/tri3a
```

```
steps/train_lda_mllt.sh: Accumulating LDA statis-
tics.
steps/train_lda_mllt.sh: Accumulating tree stats
steps/train_lda_mllt.sh: Getting questions for tree
clustering.
steps/train_lda_mllt.sh: Building the tree
steps/train_lda_mllt.sh: Initializing the model
steps/train_lda_mllt.sh: Converting alignments from
exp/tri2_ali to use current tree
steps/train_lda_mllt.sh: Compiling graphs of tran-
scripts
Training pass 1
Training pass 2
steps/train_lda_mllt.sh: Estimating MLLT
Training pass 3
...
steps/diagnostic/analyze_alignments.sh --cmd slurm.
pl --mem 4G data/lang exp/tri3a
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri3a/log/analyze_alignments.log
333 warnings in exp/tri3a/log/acc.*.*.log
1422 warnings in exp/tri3a/log/align.*.*.log
7 warnings in exp/tri3a/log/lda_acc.*.log
1 warnings in exp/tri3a/log/build_tree.log
1 warnings in exp/tri3a/log/compile_questions.log
exp/tri3a: nj=10 align prob=-48.75 over 150.18h [re-
try=0.3%, fail=0.0%] states=2136 gauss=20035 tree-
impr=5.07 lda-sum=24.62 mllt:impr,logdet=0.96,1.40
steps/train_lda_mllt.sh: Done training system with
LDA+MLLT features in exp/tri3a
```

Comparing with train_delta, there are one more extra step after reading original features: features conversion. The training is based on the converted features instead of original features.

8.2 tri2 Output

```
====================================================
      tri2 : LDA + MLLT Training & Decoding
====================================================
steps/align_si.sh --nj 30 --cmd run.pl --max-jobs-
run 10 data/train data/lang exp/tri1 exp/tri1_ali
steps/align_si.sh: feature type is delta
steps/align_si.sh: aligning data in data/train us-
ing model from exp/tri1, putting alignments in exp/
tri1_ali
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/tri1_ali
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri1_ali/log/analyze_alignments.log
steps/align_si.sh: done aligning data.
steps/train_lda_mllt.sh --cmd run.pl --max-jobs-run
10 --splice-opts --left-context=3 --right-context=3
2500 15000 data/train data/lang exp/tri1_ali exp/
tri2
steps/train_lda_mllt.sh: Accumulating LDA statis-
tics.
steps/train_lda_mllt.sh: Accumulating tree stats
steps/train_lda_mllt.sh: Getting questions for tree
clustering.
steps/train_lda_mllt.sh: Building the tree
steps/train_lda_mllt.sh: Initializing the model
steps/train_lda_mllt.sh: Converting alignments from
exp/tri1_ali to use current tree
steps/train_lda_mllt.sh: Compiling graphs of tran-
scripts
Training pass 1
Training pass 2
steps/train_lda_mllt.sh: Estimating MLLT
Training pass 3
```

```
Training pass 4
steps/train_lda_mllt.sh: Estimating MLLT
Training pass 5
Training pass 6
steps/train_lda_mllt.sh: Estimating MLLT
Training pass 7
Training pass 8
Training pass 9
Training pass 10
Aligning data
Training pass 11
Training pass 12
steps/train_lda_mllt.sh: Estimating MLLT
Training pass 13
Training pass 14
Training pass 15
Training pass 16
Training pass 17
Training pass 18
Training pass 19
Training pass 20
Aligning data
Training pass 21
Training pass 22
Training pass 23
Training pass 24
Training pass 25
Training pass 26
Training pass 27
Training pass 28
Training pass 29
```

```
Training pass 30
Aligning data
Training pass 31
Training pass 32
Training pass 33
Training pass 34
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/tri2
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri2/log/analyze_alignments.log
215 warnings in exp/tri2/log/update.*.log
92 warnings in exp/tri2/log/init_model.log
1 warnings in exp/tri2/log/compile_questions.log
exp/tri2: nj=30 align prob=-47.97 over 3.12h [re-
try=0.0%, fail=0.0%] states=2016 gauss=15030 tree-
impr=5.58 lda-sum=28.42 mllt:impr,logdet=1.62,2.22
steps/train_lda_mllt.sh: Done training system with
LDA+MLLT features in exp/tri2
tree-info exp/tri2/tree
tree-info exp/tri2/tree
make-h-transducer      --disambig-syms-out=exp/tri2/
graph/disambig_tid.int --transition-scale=1.0 data/
lang_test_bg/tmp/ilabels_3_1 exp/tri2/tree exp/tri2/
final.mdl
fstrmepslocal
fsttablecompose   exp/tri2/graph/Ha.fst   data/lang_
test_bg/tmp/CLG_3_1.fst
fstminimizeencoded
fstrmsymbols exp/tri2/graph/disambig_tid.int
fstdeterminizestar --use-log=true
fstisstochastic exp/tri2/graph/HCLGa.fst
0.000445813 -0.0175771
HCLGa is not stochastic
```

```
add-self-loops --self-loop-scale=0.1 --reorder=true
exp/tri2/final.mdl exp/tri2/graph/HCLGa.fst

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run 10
exp/tri2/graph data/dev exp/tri2/decode_dev

decode.sh: feature type is lda

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri2/graph exp/tri2/decode_dev

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri2/decode_dev/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percentile)=(2,8,30)
and mean=13.6

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri2/decode_dev/log/analyze_lattice_depth_stats.log

steps/decode.sh --nj 5 --cmd run.pl --max-jobs-run 10
exp/tri2/graph data/test exp/tri2/decode_test

decode.sh: feature type is lda

steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri2/graph exp/tri2/decode_test

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri2/decode_test/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percentile)=(2,9,34)
and mean=15.4

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri2/decode_test/log/analyze_lattice_depth_stats.
log
```

Chapter 9. tri3: LDA+MLLT+SAT

Adaptive

SAT refers to the Speaker Adapted Training (SAT), i.e. train on fMLLR-adapted features. It can be done on top of either LDA+MLLT, or delta and delta-delta features. If there are no transforms supplied in the alignment directory, it will estimate transforms itself before building the tree (and in any case, it estimates transforms a number of times during training).

```
117  echo ============================================================================
118  echo "               tri3 : LDA + MLLT + SAT Training & Decoding"
119  echo ============================================================================
120
121  # Align tri2 system with train data.
122  steps/align_si.sh --nj "$train_nj" --cmd "$train_cmd" \
123   --use-graphs true data/train data/lang exp/tri2 exp/tri2_ali
124
125  # From tri2 system, train tri3 which is LDA + MLLT + SAT.
126  steps/train_sat.sh --cmd "$train_cmd" \
127   $numLeavesSAT $numGaussSAT data/train data/lang exp/tri2_ali exp/tri3
128
129  utils/mkgraph.sh data/lang_test_bg exp/tri3 exp/tri3/graph
130
131  steps/decode_fmllr.sh --nj "$decode_nj" --cmd "$decode_cmd" \
132   exp/tri3/graph data/dev exp/tri3/decode_dev
133
134  steps/decode_fmllr.sh --nj "$decode_nj" --cmd "$decode_cmd" \
135   exp/tri3/graph data/test exp/tri3/decode_test
```

SAT adds features transformation to LDA+MLLT. The model could accept either fMLLR aligned features or the original MFCC features based on whether transform files existed in ali folder. If existed, it will use fMLLR, otherwise, it will convert the original one to fMLLR trans.

SAT uses this trans to run acc_tree_stats to accumulate the decision tree. Build_tree uses the transformed features, too.

tri3 Output

```
==================================================
     tri3 : LDA + MLLT + SAT Training & Decoding
==================================================
steps/align_si.sh --nj 30 --cmd run.pl --max-jobs-
run 10 --use-graphs true data/train data/lang exp/
tri2 exp/tri2_ali
steps/align_si.sh: feature type is lda
steps/align_si.sh: aligning data in data/train us-
ing model from exp/tri2, putting alignments in exp/
tri2_ali
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/tri2_ali
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri2_ali/log/analyze_alignments.log
steps/align_si.sh: done aligning data.
steps/train_sat.sh --cmd run.pl --max-jobs-run 10
2500 15000 data/train data/lang exp/tri2_ali exp/
tri3
steps/train_sat.sh: feature type is lda
steps/train_sat.sh: obtaining initial fMLLR trans-
forms since not present in exp/tri2_ali
steps/train_sat.sh: Accumulating tree stats
steps/train_sat.sh: Getting questions for tree clus-
tering.
steps/train_sat.sh: Building the tree
steps/train_sat.sh: Initializing the model
steps/train_sat.sh: Converting alignments from exp/
tri2_ali to use current tree
steps/train_sat.sh: Compiling graphs of transcripts
Pass 1
Pass 2
Estimating fMLLR transforms
Pass 3
```

```
Pass 4
Estimating fMLLR transforms
Pass 5
Pass 6
Estimating fMLLR transforms
Pass 7
Pass 8
Pass 9
Pass 10
Aligning data
Pass 11
Pass 12
Estimating fMLLR transforms
Pass 13
Pass 14
Pass 15
Pass 16
Pass 17
Pass 18
Pass 19
Pass 20
Aligning data
Pass 21
Pass 22
Pass 23
Pass 24
Pass 25
Pass 26
Pass 27
Pass 28
Pass 29
Pass 30
Aligning data
Pass 31
Pass 32
```

```
Pass 33
Pass 34
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/tri3
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri3/log/analyze_alignments.log
11 warnings in exp/tri3/log/update.*.log
1 warnings in exp/tri3/log/compile_questions.log
40 warnings in exp/tri3/log/init_model.log
steps/train_sat.sh: Likelihood evolution:
-50.308 -49.4272 -49.2236 -49.0161 -48.2868 -47.5841
-47.1577 -46.8853 -46.6384 -46.1028 -45.8385 -45.5103
-45.3276 -45.1911 -45.0677 -44.9549 -44.8449 -44.7357
-44.6329 -44.4723 -44.3375 -44.2495 -44.1644 -44.0833
-44.0051 -43.9287 -43.8543 -43.7816 -43.7107 -43.6167
-43.542 -43.5163 -43.5002 -43.4878
exp/tri3: nj=30 align prob=-47.11 over 3.12h [re-
try=0.0%, fail=0.0%] states=1904 gauss=15020 fmllr-
impr=4.07 over 2.78h tree-impr=8.78
steps/train_sat.sh: done training SAT system in exp/
tri3
tree-info exp/tri3/tree
tree-info exp/tri3/tree
make-h-transducer      --disambig-syms-out=exp/tri3/
graph/disambig_tid.int --transition-scale=1.0 data/
lang_test_bg/tmp/ilabels_3_1 exp/tri3/tree exp/tri3/
final.mdl
fstrmepslocal
fsttablecompose   exp/tri3/graph/Ha.fst   data/lang_
test_bg/tmp/CLG_3_1.fst
fstminimizeencoded
fstrmsymbols exp/tri3/graph/disambig_tid.int
fstdeterminizestar --use-log=true
fstisstochastic exp/tri3/graph/HCLGa.fst
0.000445813 -0.0175772
```

```
HCLGa is not stochastic
add-self-loops --self-loop-scale=0.1 --reorder=true
exp/tri3/final.mdl exp/tri3/graph/HCLGa.fst
steps/decode_fmllr.sh --nj 5 --cmd run.pl --max-jobs-
run 10 exp/tri3/graph data/dev exp/tri3/decode_dev
steps/decode.sh --scoring-opts --num-threads 1
--skip-scoring false --acwt 0.083333 --nj 5 --cmd
run.pl --max-jobs-run 10 --beam 10.0 --model exp/
tri3/final.alimdl --max-active 2000 exp/tri3/graph
data/dev exp/tri3/decode_dev.si
decode.sh: feature type is lda
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri3/graph exp/tri3/decode_dev.si
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_dev.si/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(2,9,34)
and mean=15.3
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_dev.si/log/analyze_lattice_depth_stats.
log
steps/decode_fmllr.sh: feature type is lda
steps/decode_fmllr.sh: getting first-pass fMLLR
transforms.
steps/decode_fmllr.sh: doing main lattice generation
phase
steps/decode_fmllr.sh: estimating fMLLR transforms a
second time.
steps/decode_fmllr.sh: doing a final pass of acoustic
rescoring.
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri3/graph exp/tri3/decode_dev
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(1,5,16)
and mean=7.6
steps/diagnostic/analyze_lats.sh: see stats in exp/
```

```
tri3/decode_dev/log/analyze_lattice_depth_stats.log
steps/decode_fmllr.sh --nj 5 --cmd run.pl --max-jobs-
run 10 exp/tri3/graph data/test exp/tri3/decode_test
steps/decode.sh --scoring-opts --num-threads 1
--skip-scoring false --acwt 0.083333 --nj 5 --cmd
run.pl --max-jobs-run 10 --beam 10.0 --model exp/
tri3/final.alimdl --max-active 2000 exp/tri3/graph
data/test exp/tri3/decode_test.si
decode.sh: feature type is lda
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri3/graph exp/tri3/decode_test.si
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_test.si/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(2,10,36)
and mean=16.5
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_test.si/log/analyze_lattice_depth_
stats.log
steps/decode_fmllr.sh: feature type is lda
steps/decode_fmllr.sh: getting first-pass fMLLR
transforms.
steps/decode_fmllr.sh: doing main lattice generation
phase
steps/decode_fmllr.sh: estimating fMLLR transforms a
second time.
steps/decode_fmllr.sh: doing a final pass of acoustic
rescoring.
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/tri3/graph exp/tri3/decode_test
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_test/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(1,5,18)
and mean=8.4
steps/diagnostic/analyze_lats.sh: see stats in exp/
tri3/decode_test/log/analyze_lattice_depth_stats.
log
```

Chapter 10. SGMM2

SGMM on top of fMLLR

10.1 SGMM2 General

SGMM2 is introduced by Karel, referring instead to SGMM (the subspace Gaussian mixture model) training with speaker vectors. This training would normally be called on top of fMLLR features obtained from a conventional system, but it also works on top of any type of speaker-independent features (based on deltas+delta-deltas or LDA+MLLT). Here is the script:

```
echo ============================================================================
echo "                        SGMM2 Training & Decoding                         "
echo ============================================================================

steps/align_fmllr.sh --nj "$train_nj" --cmd "$train_cmd" \
 data/train data/lang exp/tri3 exp/tri3_ali

exit 0 # From this point you can run Karel's DNN : local/nnet/run_dnn.sh

steps/train_ubm.sh --cmd "$train_cmd" \
 $numGaussUBM data/train data/lang exp/tri3_ali exp/ubm4

steps/train_sgmm2.sh --cmd "$train_cmd" $numLeavesSGMM $numGaussSGMM \
 data/train data/lang exp/tri3_ali exp/ubm4/final.ubm exp/sgmm2_4

utils/mkgraph.sh data/lang_test_bg exp/sgmm2_4 exp/sgmm2_4/graph

steps/decode_sgmm2.sh --nj "$decode_nj" --cmd "$decode_cmd"\
 --transform-dir exp/tri3/decode_dev exp/sgmm2_4/graph data/dev \
 exp/sgmm2_4/decode_dev

steps/decode_sgmm2.sh --nj "$decode_nj" --cmd "$decode_cmd"\
 --transform-dir exp/tri3/decode_test exp/sgmm2_4/graph data/test \
 exp/sgmm2_4/decode_test
```

After alignment in train_delta, three more stages have been implemented: steps/align_fmllr.sh.

```
steps/align_fmllr.sh --cmd slurm.pl --mem 4G --nj 10
data/train data/lang exp/tri3a exp/tri3a_ali

steps/align_fmllr.sh: feature type is lda

steps/align_fmllr.sh: compiling training graphs

steps/align_fmllr.sh: aligning data in data/train
using exp/tri3a/final.mdl and speaker-independent
features.

steps/align_fmllr.sh: computing fMLLR transforms

steps/align_fmllr.sh: doing final alignment.

steps/align_fmllr.sh: done aligning data.

steps/diagnostic/analyze_alignments.sh --cmd slurm.
pl --mem 4G data/lang exp/tri3a_ali

steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri3a_ali/log/analyze_alignments.log

283 warnings in exp/tri3a_ali/log/align_pass1.*.log

4 warnings in exp/tri3a_ali/log/fmllr.*.log

305 warnings in exp/tri3a_ali/log/align_pass2.*.log
```

Align first as the pre_ali.gz, calculate fmllr transforms based on pre_ali.gz, and then create trans.job.

The memory required by SGMM2 is slightly larger, preferably with more than 8G, preferably with a GPU, which can be faster. From the beginning to MMI + SGMM2, it took a total of 1 hour and 40 minutes, then in the DNN phase took 2 hours, DNN + SGMM took 20 minutes, a total of about 4 hours, if there is a GPU it should be much faster. The recognition results obtained at each stage are posted below.

10.2 SGMM2 Output

```
=====================================================
              SGMM2 Training & Decoding
=====================================================
steps/align_fmllr.sh --nj 30 --cmd run.pl --max-jobs-
run 10 data/train data/lang exp/tri3 exp/tri3_ali
steps/align_fmllr.sh: feature type is lda
steps/align_fmllr.sh: compiling training graphs
steps/align_fmllr.sh: aligning data in data/train
using exp/tri3/final.alimdl and speaker-independent
features.
steps/align_fmllr.sh: computing fMLLR transforms
steps/align_fmllr.sh: doing final alignment.
steps/align_fmllr.sh: done aligning data.
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/tri3_ali
steps/diagnostic/analyze_alignments.sh: see stats in
exp/tri3_ali/log/analyze_alignments.log
steps/train_ubm.sh --cmd run.pl --max-jobs-run 10
400 data/train data/lang exp/tri3_ali exp/ubm4
steps/train_ubm.sh: feature type is lda
steps/train_ubm.sh: using transforms from exp/tri3_ali
steps/train_ubm.sh: clustering model exp/tri3_ali/
final.mdl to get initial UBM
steps/train_ubm.sh: doing Gaussian selection
Pass 0
Pass 1
Pass 2
steps/train_sgmm2.sh --cmd run.pl --max-jobs-run 10
7000 9000 data/train data/lang exp/tri3_ali exp/ubm4/
final.ubm exp/sgmm2_4
steps/train_sgmm2.sh: feature type is lda
steps/train_sgmm2.sh: using transforms from exp/
```

```
tri3_ali
steps/train_sgmm2.sh: accumulating tree stats
steps/train_sgmm2.sh: Getting questions for tree
clustering.
steps/train_sgmm2.sh: Building the tree
steps/train_sgmm2.sh: Initializing the model
steps/train_sgmm2.sh: doing Gaussian selection
steps/train_sgmm2.sh: compiling training graphs
steps/train_sgmm2.sh: converting alignments
steps/train_sgmm2.sh: training pass 0 ...
steps/train_sgmm2.sh: training pass 1 ...
steps/train_sgmm2.sh: training pass 2 ...
steps/train_sgmm2.sh: training pass 3 ...
steps/train_sgmm2.sh: training pass 4 ...
steps/train_sgmm2.sh: training pass 5 ...
steps/train_sgmm2.sh: re-aligning data
steps/train_sgmm2.sh: training pass 6 ...
steps/train_sgmm2.sh: training pass 7 ...
steps/train_sgmm2.sh: training pass 8 ...
steps/train_sgmm2.sh: training pass 9 ...
steps/train_sgmm2.sh: training pass 10 ...
steps/train_sgmm2.sh: re-aligning data
steps/train_sgmm2.sh: training pass 11 ...
steps/train_sgmm2.sh: training pass 12 ...
steps/train_sgmm2.sh: training pass 13 ...
steps/train_sgmm2.sh: training pass 14 ...
steps/train_sgmm2.sh: training pass 15 ...
steps/train_sgmm2.sh: re-aligning data
steps/train_sgmm2.sh: training pass 16 ...
steps/train_sgmm2.sh: training pass 17 ...
steps/train_sgmm2.sh: training pass 18 ...
steps/train_sgmm2.sh: training pass 19 ...
```

```
steps/train_sgmm2.sh: training pass 20 ...
steps/train_sgmm2.sh: training pass 21 ...
steps/train_sgmm2.sh: training pass 22 ...
steps/train_sgmm2.sh: training pass 23 ...
steps/train_sgmm2.sh: training pass 24 ...
steps/train_sgmm2.sh: building alignment model (pass 25)
steps/train_sgmm2.sh: building alignment model (pass 26)
steps/train_sgmm2.sh: building alignment model (pass 27)
213 warnings in exp/sgmm2_4/log/update_ali.*.log
1875 warnings in exp/sgmm2_4/log/update.*.log
1 warnings in exp/sgmm2_4/log/compile_questions.log
Done
tree-info exp/sgmm2_4/tree
tree-info exp/sgmm2_4/tree
make-h-transducer  --disambig-syms-out=exp/sgmm2_4/
graph/disambig_tid.int --transition-scale=1.0 data/
lang_test_bg/tmp/ilabels_3_1 exp/sgmm2_4/tree exp/
sgmm2_4/final.mdl
fstrmepslocal
fsttablecompose exp/sgmm2_4/graph/Ha.fst data/lang_
test_bg/tmp/CLG_3_1.fst
fstrmsymbols exp/sgmm2_4/graph/disambig_tid.int
fstminimizeencoded
fstdeterminizestar --use-log=true
fstisstochastic exp/sgmm2_4/graph/HCLGa.fst
0.000485195 -0.0175772
HCLGa is not stochastic
add-self-loops --self-loop-scale=0.1 --reorder=true
exp/sgmm2_4/final.mdl exp/sgmm2_4/graph/HCLGa.fst
steps/decode_sgmm2.sh --nj 5 --cmd run.pl --max-
jobs-run 10 --transform-dir exp/tri3/decode_dev exp/
sgmm2_4/graph data/dev exp/sgmm2_4/decode_dev
steps/decode_sgmm2.sh: feature type is lda
```

```
steps/decode_sgmm2.sh: using transforms from exp/
tri3/decode_dev
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/sgmm2_4/graph exp/sgmm2_4/decode_dev
steps/diagnostic/analyze_lats.sh: see stats in exp/
sgmm2_4/decode_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(2,6,20)
and mean=9.4
steps/diagnostic/analyze_lats.sh: see stats in exp/
sgmm2_4/decode_dev/log/analyze_lattice_depth_stats.
log
steps/decode_sgmm2.sh --nj 5 --cmd run.pl --max-jobs-
run 10 --transform-dir exp/tri3/decode_test exp/
sgmm2_4/graph data/test exp/sgmm2_4/decode_test
steps/decode_sgmm2.sh: feature type is lda
steps/decode_sgmm2.sh: using transforms from exp/
tri3/decode_test
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 exp/sgmm2_4/graph exp/sgmm2_4/decode_
test
steps/diagnostic/analyze_lats.sh: see stats in exp/
sgmm2_4/decode_test/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(2,6,23)
and mean=10.4
steps/diagnostic/analyze_lats.sh: see stats in exp/
sgmm2_4/decode_test/log/analyze_lattice_depth_
stats.log
```

Even though SGMM is a good approach, Dan Povey points out that "But I recommend that you ignore SGMMs. Right now neural nets are where it's at, and SGMMs were, at best, the most highly evolved state of a dying approach. MMI is still somewhat relevant as it's an example of a sequence-level objective (maximum posterior of the correct transcript) which is equivalent to the objective function used in CTC and other popular modern approaches, and it's still used when training neural nets."

Chapter 11. MMI + SGMM2

Maximum Mutual Information

11.1 Why MMI?

Until last chapter, most of the models are based on MLE. Maximum likelihood provides a consistent approach to parameter estimation problems. This means that maximum likelihood estimates can be developed for a large variety of estimation situations. For example, they can be applied in reliability analysis to censored data under various censoring models. However, some disadvantages are clear, too:

- The likelihood equations need to be specifically worked out for a given distribution and estimation problem. The mathematics is often non-trivial, particularly if confidence intervals for the parameters are desired.
- The numerical estimation is usually non-trivial. Except for a few cases where the maximum likelihood formulas are in fact simple, it is generally best to rely on high quality statistical software to obtain maximum likelihood estimates. Fortunately, high quality maximum likelihood software is becoming increasingly common.
- Maximum likelihood estimates can be heavily biased for small samples. The optimality properties may not apply for small samples.

- Maximum likelihood can be sensitive to the choice of starting values.

To solve the first problem, we use GMM to mimic the living environment. However, GMM might not be trained thoroughly, which is a problem.

The other disadvantages just have no effective way to overcome. That's why MMI (Maximum Mutual Information) is introduced. Rather than some well-known methods of deep learning, this task presents a more specific framework adopted by speech processing (i.e. discriminative training). As for the DNN-HMM system above, Cross Entropy (CE) loss function is often employed to minimize the phonemes prediction error rate. The CE criterion evaluates each speech frame independently. Hence, the training process ignores the context information among phonemes series. To address the error, discriminative training methods for DNN were proposed. The discriminative criteria for DNN include Maximum Mutual Information (MMI),

MMI Formula:

$$F_{MMIE}(\lambda) = \sum_{r=1}^{R} \log \frac{P_\lambda(O_r \mid M_{w_r}) P(w_r)}{\sum_{\hat{w}} P_\lambda(O_r \mid M_{\hat{w}}) P(\hat{w})}$$

Let's convert it a little bit:

$$\sum \log(P(O_r|w_r)P(w_r)/\sum wP(O_r|w)P(w))$$

$$= \sum \log P(O_r|w_r)P(w_r)/P(O_r) = P(W|O)$$

11.2 MMI General

It's easier to see now that the target function is P(W|O), in another word, MMI model put the language model into consideration. Comparing with MLE, the advantage is that even if the assumed distribution is not ideal, the output will not be that bad.

We could think it the ration of right paths and wrong paths. When right paths have been increased, the wrong paths will be decreased, which means "discriminative training"

Here is the script from Kaldi.

```
162  echo ============================================================================
163  echo "                    MMI + SGMM2 Training & Decoding                       "
164  echo ============================================================================
165
166  steps/align_sgmm2.sh --nj "$train_nj" --cmd "$train_cmd" \
167   --transform-dir exp/tri3_ali --use-graphs true --use-gselect true \
168   data/train data/lang exp/sgmm2_4 exp/sgmm2_4_ali
169
170  steps/make_denlats_sgmm2.sh --nj "$train_nj" --sub-split "$train_nj" \
171   --acwt 0.2 --lattice-beam 10.0 --beam 18.0 \
172   --cmd "$decode_cmd" --transform-dir exp/tri3_ali \
173   data/train data/lang exp/sgmm2_4_ali exp/sgmm2_4_denlats
174
175  steps/train_mmi_sgmm2.sh --acwt 0.2 --cmd "$decode_cmd" \
176   --transform-dir exp/tri3_ali --boost 0.1 --drop-frames true \
177   data/train data/lang exp/sgmm2_4_ali exp/sgmm2_4_denlats exp/sgmm2_4_mmi_b0.1
178
179  for iter in 1 2 3 4; do
180    steps/decode_sgmm2_rescore.sh --cmd "$decode_cmd" --iter $iter \
181     --transform-dir exp/tri3/decode_dev data/lang_test_bg data/dev \
182     exp/sgmm2_4/decode_dev exp/sgmm2_4_mmi_b0.1/decode_dev_it$iter
183
184    steps/decode_sgmm2_rescore.sh --cmd "$decode_cmd" --iter $iter \
185     --transform-dir exp/tri3/decode_test data/lang_test_bg data/test \
186     exp/sgmm2_4/decode_test exp/sgmm2_4_mmi_b0.1/decode_test_it$iter
187  done
188
```

Line 175 steps/train_mmi_sgmm2.sh uses data/train_si84 as the input, and output the exp/tri2b_mmi

train_mm_sgmm2 script:

run.sh train_mmi_sgmm2.sh

```
37  data=$1
38  lang=$2
39  alidir=$3
40  denlatdir=$4
41  dir=$5
42  mkdir -p $dir/log
43
44  utils/lang/check_phones_compatible.sh $lang/phones.txt $alidir/phones.txt || exit 1;
45  cp $lang/phones.txt $dir || exit 1;
46
47  for f in $data/feats.scp $alidir/{tree,final.mdl,ali.1.gz} $denlatdir/lat.1.gz; do
48    [ ! -f $f ] && echo "$0: no such file $f" && exit 1;
49  done
50  nj=`cat $alidir/num_jobs` || exit 1;
51  [ "$nj" -ne "`cat $denlatdir/num_jobs`" ] && \
52    echo "$alidir and $denlatdir have different num-jobs" && exit 1;
53
54  sdata=$data/split$nj
55  splice_opts=`cat $alidir/splice_opts 2>/dev/null`
56  cmvn_opts=`cat $alidir/cmvn_opts 2>/dev/null`
57  mkdir -p $dir/log
58  cp $alidir/splice_opts $dir 2>/dev/null
59  cp $alidir/cmvn_opts $dir 2>/dev/null # cmn/cmvn option.
60  [[ -d $sdata && $data/feats.scp -ot $sdata ]] || split_data.sh $data $nj || exit 1;
61  echo $nj > $dir/num_jobs
62
63  cp $alidir/tree $dir
64  cp $alidir/final.mdl $dir/0.mdl
65  cp $alidir/final.alimdl $dir
66
67  silphonelist=`cat $lang/phones/silence.csl` || exit 1;
68
69  # Set up features
70
71  if [ -f $alidir/final.mat ]; then feat_type=lda; else feat_type=delta; fi
72  echo "$0: feature type is $feat_type"
73
```

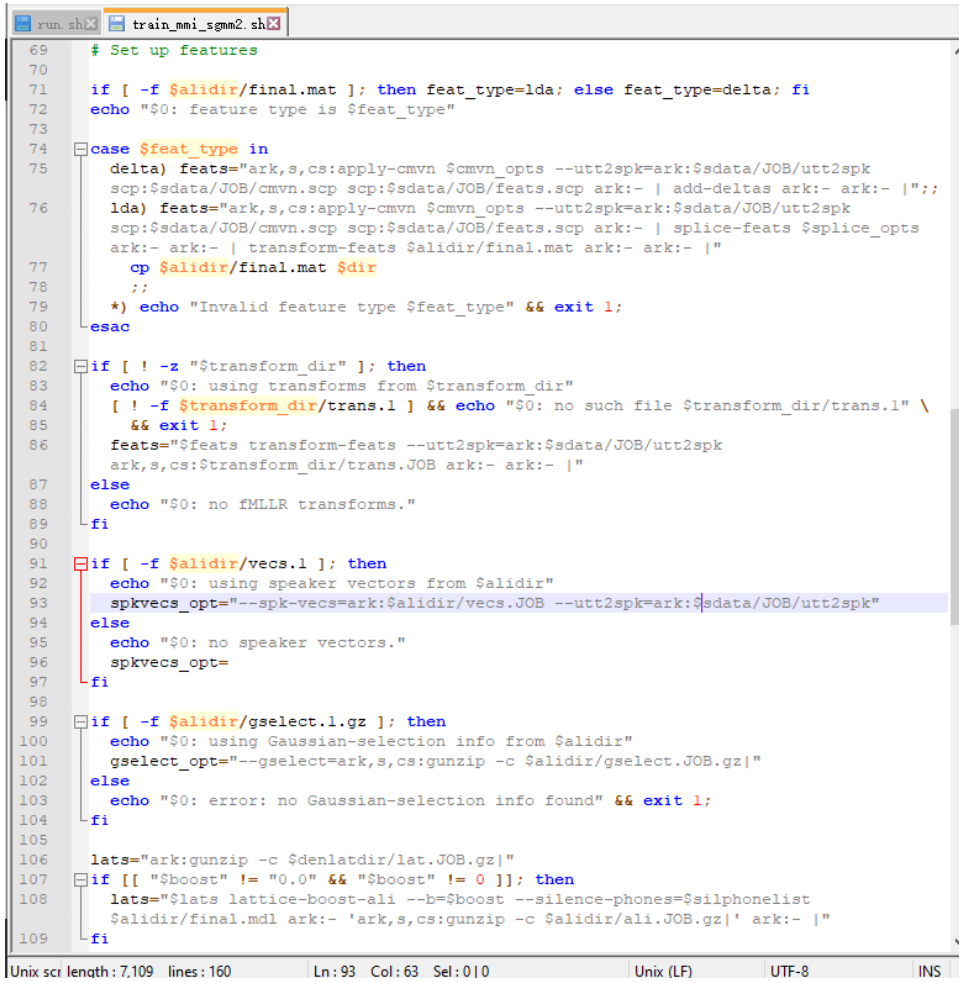
Line 44 of train_mm_sgmm2.sh compared the text of two labeled phones, stop in case of not same. Each phone is related with one identical integer.

Line 47 - 49, get the location of each sentence's features.

alidir/{tree,final.mdl,ali.1.gz} gets the decision tree aligned, model, and alignment information.

denlatdir/lat.1.gz stores word diagram.

Starting from line 69, transforming the features.



final.mat is to transform the features. transform-feats using transform to convert features to ark format for decoding, and then get lattice file.

Line 108 lattice-boost-ali, monitoring the b* (frame phone error) on each arch, increase likelihood of each graph to decrease graph cost, to identify discriminative training, like increasing MMI and changing lattice. The model needs transition to convert pdf-id to phone. With silence-phones parameter, lattice will ignore silence errors, or using max-silence-error. Setting max silence error =1 equivalent to no silence-phones).

11.3 MMI + SGMM2 Output

```
=====================================================
         MMI + SGMM2 Training & Decoding
=====================================================
steps/align_sgmm2.sh --nj 30 --cmd run.pl --max-jobs-
run 10 --transform-dir exp/tri3_ali --use-graphs true
--use-gselect true data/train data/lang exp/sgmm2_4
exp/sgmm2_4_ali
steps/align_sgmm2.sh: feature type is lda
steps/align_sgmm2.sh: using transforms from exp/
tri3_ali
steps/align_sgmm2.sh: aligning data in data/train
using model exp/sgmm2_4/final.alimdl
steps/align_sgmm2.sh: computing speaker vectors (1st
pass)
steps/align_sgmm2.sh: computing speaker vectors (2nd
pass)
steps/align_sgmm2.sh: doing final alignment.
steps/align_sgmm2.sh: done aligning data.
steps/diagnostic/analyze_alignments.sh --cmd run.pl
--max-jobs-run 10 data/lang exp/sgmm2_4_ali
steps/diagnostic/analyze_alignments.sh: see stats in
exp/sgmm2_4_ali/log/analyze_alignments.log
steps/make_denlats_sgmm2.sh --nj 30 --sub-split 30
--acwt 0.2 --lattice-beam 10.0 --beam 18.0 --cmd run.
pl --max-jobs-run 10 --transform-dir exp/tri3_ali
data/train data/lang exp/sgmm2_4_ali exp/sgmm2_4_
denlats
steps/make_denlats_sgmm2.sh: Making unigram grammar
FST in exp/sgmm2_4_denlats/lang
steps/make_denlats_sgmm2.sh: Compiling decoding
graph in exp/sgmm2_4_denlats/dengraph
tree-info exp/sgmm2_4_ali/tree
tree-info exp/sgmm2_4_ali/tree
fsttablecompose exp/sgmm2_4_denlats/lang/L_disambig.
```

```
fst exp/sgmm2_4_denlats/lang/G.fst
fstminimizeencoded
fstdeterminizestar --use-log=true
fstpushspecial
fstisstochastic exp/sgmm2_4_denlats/lang/tmp/LG.fst
1.27271e-05 1.27271e-05
fstcomposecontext --context-size=3 --central-posi-
tion=1 --read-disambig-syms=exp/sgmm2_4_denlats/
lang/phones/disambig.int --write-disambig-syms=exp/
sgmm2_4_denlats/lang/tmp/disambig_ilabels_3_1.int
exp/sgmm2_4_denlats/lang/tmp/ilabels_3_1.7065 exp/
sgmm2_4_denlats/lang/tmp/LG.fst
fstisstochastic exp/sgmm2_4_denlats/lang/tmp/
CLG_3_1.fst
1.27657e-05 0
make-h-transducer --disambig-syms-out=exp/sgmm2_4_
denlats/dengraph/disambig_tid.int --transition-
scale=1.0 exp/sgmm2_4_denlats/lang/tmp/ilabels_3_1
exp/sgmm2_4_ali/tree exp/sgmm2_4_ali/final.mdl
fsttablecompose exp/sgmm2_4_denlats/dengraph/Ha.fst
exp/sgmm2_4_denlats/lang/tmp/CLG_3_1.fst
fstrmepslocal
fstminimizeencoded
fstrmsymbols exp/sgmm2_4_denlats/dengraph/disambig_
tid.int
fstdeterminizestar --use-log=true
fstisstochastic exp/sgmm2_4_denlats/dengraph/HCLGa.
fst
0.000481185 -0.000485819
add-self-loops --self-loop-scale=0.1 --reorder=true
exp/sgmm2_4_ali/final.mdl exp/sgmm2_4_denlats/den-
graph/HCLGa.fst
steps/make_denlats_sgmm2.sh: feature type is lda
steps/make_denlats_sgmm2.sh: using fMLLR transforms
from exp/tri3_ali
```

```
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 1
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 2
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 3
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 4
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 5
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 6
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 7
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 8
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 9
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 10
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 11
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 12
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 13
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 14
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 15
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 16
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 17
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 18
```

```
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 19
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 20
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 21
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 22
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 23
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 24
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 25
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 26
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 27
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 28
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 29
steps/make_denlats_sgmm2.sh: Merging archives for
data subset 30
steps/make_denlats_sgmm2.sh: done generating denomi-
nator lattices with SGMMs.
steps/train_mmi_sgmm2.sh --acwt 0.2 --cmd run.
pl --max-jobs-run 10 --transform-dir exp/tri3_ali
--boost 0.1 --drop-frames true data/train data/lang
exp/sgmm2_4_ali exp/sgmm2_4_denlats exp/sgmm2_4_mmi_
b0.1
steps/train_mmi_sgmm2.sh: feature type is lda
steps/train_mmi_sgmm2.sh: using transforms from exp/
tri3_ali
steps/train_mmi_sgmm2.sh: using speaker vectors from
exp/sgmm2_4_ali
```

```
steps/train_mmi_sgmm2.sh: using Gaussian-selection info from exp/sgmm2_4_ali

Iteration 0 of MMI training

Iteration 0: objf was 0.501016467634615, MMI auxf change was 0.0162648878979182

Iteration 1 of MMI training

Iteration 1: objf was 0.515660655672759, MMI auxf change was 0.00237356455193395

Iteration 2 of MMI training

Iteration 2: objf was 0.51831743276386, MMI auxf change was 0.000636215653485037

Iteration 3 of MMI training

Iteration 3: objf was 0.519198604496081, MMI auxf change was 0.000381818294967297

MMI training finished

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-jobs-run 10 --iter 1 --transform-dir exp/tri3/decode_dev data/lang_test_bg data/dev exp/sgmm2_4/decode_dev exp/sgmm2_4_mmi_b0.1/decode_dev_it1

steps/decode_sgmm2_rescore.sh: using speaker vectors from exp/sgmm2_4/decode_dev

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from exp/tri3/decode_dev

steps/decode_sgmm2_rescore.sh: rescoring lattices with SGMM model in exp/sgmm2_4_mmi_b0.1/1.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-jobs-run 10 --iter 1 --transform-dir exp/tri3/decode_test data/lang_test_bg data/test exp/sgmm2_4/decode_test exp/sgmm2_4_mmi_b0.1/decode_test_it1

steps/decode_sgmm2_rescore.sh: using speaker vectors from exp/sgmm2_4/decode_test

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from exp/tri3/decode_test
```

```
steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/1.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
jobs-run 10 --iter 2 --transform-dir exp/tri3/de-
code_dev data/lang_test_bg data/dev exp/sgmm2_4/de-
code_dev exp/sgmm2_4_mmi_b0.1/decode_dev_it2

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_dev

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_dev

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/2.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
jobs-run 10 --iter 2 --transform-dir exp/tri3/de-
code_test data/lang_test_bg data/test exp/sgmm2_4/
decode_test exp/sgmm2_4_mmi_b0.1/decode_test_it2

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_test

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_test

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/2.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
jobs-run 10 --iter 3 --transform-dir exp/tri3/de-
code_dev data/lang_test_bg data/dev exp/sgmm2_4/de-
code_dev exp/sgmm2_4_mmi_b0.1/decode_dev_it3

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_dev

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_dev

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/3.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
```

```
jobs-run 10 --iter 3 --transform-dir exp/tri3/de-
code_test data/lang_test_bg data/test exp/sgmm2_4/
decode_test exp/sgmm2_4_mmi_b0.1/decode_test_it3

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_test

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_test

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/3.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
jobs-run 10 --iter 4 --transform-dir exp/tri3/de-
code_dev data/lang_test_bg data/dev exp/sgmm2_4/de-
code_dev exp/sgmm2_4_mmi_b0.1/decode_dev_it4

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_dev

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_dev

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/4.mdl

steps/decode_sgmm2_rescore.sh --cmd run.pl --max-
jobs-run 10 --iter 4 --transform-dir exp/tri3/de-
code_test data/lang_test_bg data/test exp/sgmm2_4/
decode_test exp/sgmm2_4_mmi_b0.1/decode_test_it4

steps/decode_sgmm2_rescore.sh: using speaker vectors
from exp/sgmm2_4/decode_test

steps/decode_sgmm2_rescore.sh: feature type is lda

steps/decode_sgmm2_rescore.sh: using transforms from
exp/tri3/decode_test

steps/decode_sgmm2_rescore.sh: rescoring lattices
with SGMM model in exp/sgmm2_4_mmi_b0.1/4.mdl
```

Chapter 12. Dan's DNN

DNN Hybrid Training

Deep Neural Networks (DNNs) are the latest hot topic in speech recognition since around 2010. Google, Microsoft, Facebook, and Amazon name a few are the main players in this area. Kaldi also demonstrated the effectiveness of easily incorporate "Deep Neural Network" (DNN) techniques to improve the recognition performance in almost all recognition tasks.

12.1 General

Kaldi currently contains two parallel implementations for DNN training.

The first implementation supports Restricted Boltzmann Machines (RBM) pre-training, stochastic gradient descent training using NVidia Graphics Processing Units (GPUs), and discriminative training such as boosted MMI and state-level minimum Bayes risk (sMBR).

The second implementation of DNNs in Kaldi was originally written to support parallel training on multiple CPUs, although it has now been extended to support parallel GPU-based training and it does not support discriminative training.

One is located in code sub-directories nnet/2 and nnetbin/, and is primarily maintained by Karel Vesely. The other is located

in code subdirectories nnet2/ and nnet2bin/, and is primarily maintained by Daniel Povey (this code was originally based on an earlier version of Karel's code, but it has been extensively rewritten). Neither codebase is more "official" than the other. Both are still being developed in parallel.

In this chapter, we will show Dan's DNN mainly.

```
189  echo ============================================================================
190  echo "                    DNN Hybrid Training & Decoding                        "
191  echo ============================================================================
192
193  # DNN hybrid system training parameters
194  dnn_mem_reqs="--mem 1G"
195  dnn_extra_opts="--num_epochs 20 --num-epochs-extra 10 --add-layers-period 1 --shrink-interval 3"
196
197  steps/nnet2/train_tanh.sh --mix-up 5000 --initial-learning-rate 0.015 \
198    --final-learning-rate 0.002 --num-hidden-layers 2  \
199    --num-jobs-nnet "$train_nj" --cmd "$train_cmd" "${dnn_train_extra_opts[@]}" \
200    data/train data/lang exp/tri3_ali exp/tri4_nnet
201
202  [ ! -d exp/tri4_nnet/decode_dev ] && mkdir -p exp/tri4_nnet/decode_dev
203  decode_extra_opts=(--num-threads 6)
204  steps/nnet2/decode.sh --cmd "$decode_cmd" --nj "$decode_nj" "${decode_extra_opts[@]}" \
205    --transform-dir exp/tri3/decode_dev exp/tri3/graph data/dev \
206    exp/tri4_nnet/decode_dev | tee exp/tri4_nnet/decode_dev/decode.log
207
208  [ ! -d exp/tri4_nnet/decode_test ] && mkdir -p exp/tri4_nnet/decode_test
209  steps/nnet2/decode.sh --cmd "$decode_cmd" --nj "$decode_nj" "${decode_extra_opts[@]}" \
210    --transform-dir exp/tri3/decode_test exp/tri3/graph data/test \
211    exp/tri4_nnet/decode_test | tee exp/tri4_nnet/decode_test/decode.log
212
213  echo ============================================================================
214  echo "                    System Combination (DNN+SGMM)                         "
215  echo ============================================================================
216
217  for iter in 1 2 3 4; do
218    local/score_combine.sh --cmd "$decode_cmd" \
219     data/dev data/lang_test_bg exp/tri4_nnet/decode_dev \
220     exp/sgmm2_4_mmi_b0.1/decode_dev_it$iter exp/combine_2/decode_dev_it$iter
221
222    local/score_combine.sh --cmd "$decode_cmd" \
223     data/test data/lang_test_bg exp/tri4_nnet/decode_test \
224     exp/sgmm2_4_mmi_b0.1/decode_test_it$iter exp/combine_2/decode_test_it$iter
225  done
226
```

Dan's DNN adopts a classic Hybrid Training and Decoding framework using a simple deep network with tanh nonlinearities. Moreover, also system combination using minimum Bayes risk decoding is used, and in this case a lattice combination is used to create a union of lattices normalized by removing the total forward cost from them and using the resulting lattice as input the last decoding step.

Line 196 `steps/nnet2/traiin_tanh.sh` trains a fairly vanilla network with tanh nonlinearities.

12.2 Output

```
=====================================================
            DNN Hybrid Training & Decoding
=====================================================
steps/nnet2/train_tanh.sh --mix-up 5000 --initial-
learning-rate 0.015 --final-learning-rate 0.002
--num-hidden-layers 2 --num-jobs-nnet 30 --cmd run.
pl --max-jobs-run 10 data/train data/lang exp/tri3_
ali exp/tri4_nnet
steps/nnet2/train_tanh.sh: calling get_lda.sh
steps/nnet2/get_lda.sh --transform-dir exp/tri3_ali
--splice-width 4 --cmd run.pl --max-jobs-run 10 data/
train data/lang exp/tri3_ali exp/tri4_nnet
steps/nnet2/get_lda.sh: feature type is lda
steps/nnet2/get_lda.sh: using transforms from exp/
tri3_ali
feat-to-dim 'ark,s,cs:utils/subset_scp.pl --qui-
et 333 data/train/split30/1/feats.scp | apply-cmvn
--utt2spk=ark:data/train/split30/1/utt2spk scp:data/
train/split30/1/cmvn.scp scp:- ark:- | splice-feats
--left-context=3 --right-context=3 ark:- ark:- |
transform-feats exp/tri4_nnet/final.mat ark:- ark:- |
transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:- |' -
splice-feats --left-context=3 --right-context=3
ark:- ark:-
transform-feats exp/tri4_nnet/final.mat ark:- ark:-
transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:-
apply-cmvn --utt2spk=ark:data/train/split30/1/utt-
2spk scp:data/train/split30/1/cmvn.scp scp:- ark:-
WARNING (feat-to-dim[5.5.164~1-9698]:Close():Kaldi-
io.cc:515) Pipe utils/subset_scp.pl --quiet 333
data/train/split30/1/feats.scp | apply-cmvn
--utt2spk=ark:data/train/split30/1/utt2spk scp:data/
train/split30/1/cmvn.scp scp:- ark:- | splice-feats
```

```
--left-context=3 --right-context=3 ark:- ark:- |
transform-feats exp/tri4_nnet/final.mat ark:- ark:- |
transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:- | had
nonzero return status 36096

feat-to-dim 'ark,s,cs:utils/subset_scp.pl --qui-
et 333 data/train/split30/1/feats.scp | apply-cmvn
--utt2spk=ark:data/train/split30/1/utt2spk scp:data/
train/split30/1/cmvn.scp scp:- ark:- | splice-feats
--left-context=3 --right-context=3 ark:- ark:- |
transform-feats exp/tri4_nnet/final.mat ark:- ark:- |
transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:- | splice-
feats --left-context=4 --right-context=4 ark:- ark:-
|' -

transform-feats exp/tri4_nnet/final.mat ark:- ark:-

splice-feats --left-context=3 --right-context=3
ark:- ark:-

transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:-

apply-cmvn --utt2spk=ark:data/train/split30/1/utt-
2spk scp:data/train/split30/1/cmvn.scp scp:- ark:-

splice-feats --left-context=4 --right-context=4
ark:- ark:-

WARNING (feat-to-dim[5.5.164~1-9698]:Close():Kaldi-
io.cc:515) Pipe utils/subset_scp.pl --quiet 333
data/train/split30/1/feats.scp | apply-cmvn
--utt2spk=ark:data/train/split30/1/utt2spk scp:data/
train/split30/1/cmvn.scp scp:- ark:- | splice-feats
--left-context=3 --right-context=3 ark:- ark:- |
transform-feats exp/tri4_nnet/final.mat ark:- ark:- |
transform-feats --utt2spk=ark:data/train/split30/1/
utt2spk ark:exp/tri3_ali/trans.1 ark:- ark:- | splice-
feats --left-context=4 --right-context=4 ark:- ark:-
| had nonzero return status 36096

steps/nnet2/get_lda.sh: Accumulating LDA statistics.

steps/nnet2/get_lda.sh: Finished estimating LDA

steps/nnet2/train_tanh.sh: calling get_egs.sh
```

```
steps/nnet2/get_egs.sh --transform-dir exp/tri3_ali
--splice-width 4 --samples-per-iter 200000 --num-
jobs-nnet 30 --stage 0 --cmd run.pl --max-jobs-run 10
--io-opts --max-jobs-run 5 data/train data/lang exp/
tri3_ali exp/tri4_nnet

steps/nnet2/get_egs.sh: feature type is lda

steps/nnet2/get_egs.sh: using transforms from exp/
tri3_ali

steps/nnet2/get_egs.sh: working out number of frames
of training data

utils/data/get_utt2dur.sh: segments file does not
exist so getting durations from wave files

utils/data/get_utt2dur.sh: successfully obtained ut-
terance lengths from sphere-file headers

utils/data/get_utt2dur.sh: computed data/train/utt-
2dur

feat-to-len 'scp:head -n 10 data/train/feats.scp|'
ark,t:-

steps/nnet2/get_egs.sh: Every epoch, splitting the
data up into 1 iterations,

steps/nnet2/get_egs.sh: giving samples-per-iteration
of 37740 (you requested 200000).

Getting validation and training subset examples.

steps/nnet2/get_egs.sh: extracting validation and
training-subset alignments.

copy-int-vector ark:- ark,t:-

LOG    (copy-int-vector[5.5.164~1-9698]:main():copy-
int-vector.cc:83) Copied 3696 vectors of int32.

Getting subsets of validation examples for diagnos-
tics and combination.

Creating training examples

Generating training examples on disk

steps/nnet2/get_egs.sh: rearranging examples into
parts for different parallel jobs

steps/nnet2/get_egs.sh: Since iters-per-epoch == 1,
```

```
just concatenating the data.
Shuffling the order of training examples
(in order to avoid stressing the disk, these won't
all run at once).
steps/nnet2/get_egs.sh: Finished preparing training
examples
steps/nnet2/train_tanh.sh: initializing neural net
Training transition probabilities and setting priors
steps/nnet2/train_tanh.sh: Will train for 15 + 5 ep-
ochs, equalling
steps/nnet2/train_tanh.sh: 15 + 5 = 20 iterations,
steps/nnet2/train_tanh.sh: (while reducing learning
rate) + (with constant learning rate).
Training neural net (pass 0)
Training neural net (pass 1)
Training neural net (pass 2)
......
Training neural net (pass 7)
Training neural net (pass 8)
Training neural net (pass 9)
Training neural net (pass 10)
Training neural net (pass 11)
Training neural net (pass 12)
Mixing up from 1904 to 5000 components
Training neural net (pass 13)
Training neural net (pass 14)
Training neural net (pass 15)
Training neural net (pass 16)
Training neural net (pass 17)
Training neural net (pass 18)
Training neural net (pass 19)
Setting num_iters_final=5
```

```
Getting average posterior for purposes of adjusting the priors.
Re-adjusting priors based on computed posteriors
Done
Cleaning up data
steps/nnet2/remove_egs.sh: Finished deleting examples in exp/tri4_nnet/egs
Removing most of the models
steps/nnet2/decode.sh --cmd run.pl --max-jobs-run 10 --nj 5 --num-threads 6 --transform-dir exp/tri3/decode_dev exp/tri3/graph data/dev exp/tri4_nnet/decode_dev
steps/nnet2/decode.sh: feature type is lda
steps/nnet2/decode.sh: using transforms from exp/tri3/decode_dev
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-jobs-run 10 --iter final exp/tri3/graph exp/tri4_nnet/decode_dev
steps/diagnostic/analyze_lats.sh: see stats in exp/tri4_nnet/decode_dev/log/analyze_alignments.log
Overall, lattice depth (10,50,90-percentile)=(7,32,159) and mean=71.1
steps/diagnostic/analyze_lats.sh: see stats in exp/tri4_nnet/decode_dev/log/analyze_lattice_depth_stats.log
score best paths
score confidence and timing with sclite
Decoding done.
steps/nnet2/decode.sh --cmd run.pl --max-jobs-run 10 --nj 5 --num-threads 6 --transform-dir exp/tri3/decode_test exp/tri3/graph data/test exp/tri4_nnet/decode_test
steps/nnet2/decode.sh: feature type is lda
steps/nnet2/decode.sh: using transforms from exp/tri3/decode_test
```

```
steps/diagnostic/analyze_lats.sh --cmd run.pl --max-
jobs-run 10 --iter final exp/tri3/graph exp/tri4_
nnet/decode_test

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri4_nnet/decode_test/log/analyze_alignments.log

Overall, lattice depth (10,50,90-percen-
tile)=(7,35,183) and mean=84.0

steps/diagnostic/analyze_lats.sh: see stats in exp/
tri4_nnet/decode_test/log/analyze_lattice_depth_
stats.log

score best paths

score confidence and timing with sclite

Decoding done.
```

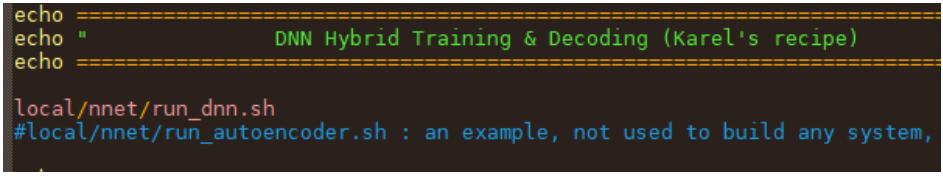
Chapter 13. Karel's DNN

Multiple Stages

Kaldi currently contains two parallel implementations for DNN training. Neither codebase is more "official" than the other. Both are still being developed in parallel. Dan's DNN is described in last Chapter. Here we deliberate Karel's recipe.

Karel's DNN uses layer-wise pre-training based on RBMs (Restricted Boltzmann Machines), per-frame cross-entropy training, and sequence-discriminative training, using lattice framework, optimizing sMBR criterion (State Minimum Bayes Risk). The systems are built on top of LDA-MLLT-fMLLR features obtained from auxiliary GMM (Gaussian Mixture Model) models. Whole DNN training is running in a single GPU using CUDA (Compute Unified Device Architecture, the parallel computing architecture created by Nvidia).

Karel's DNN is located in code sub-directories `nnet/2` and `nnetbin/`, and is primarily maintained by Karel Vesely. The script is pretty simple, just one line:



Karel's DNN is split into several stages.

13.0 Stage 0: Store Features

Here is the run_dnn.sh.

Line 31 - 50, the 40-dimensional features (MFCC, LDA, MLLT, fMLLR with CMN) are stored to simplify the training.

run.sh | score_combine.sh | run_dnn.sh

```
1  #!/usr/bin/env bash
2
3  # Copyright 2012-2014  Brno University of Technology (Author: Karel Vesely)
4  # Apache 2.0
5
6  # This example script trains a DNN on top of fMLLR features.
7  # The training is done in 3 stages,
8  #
9  # 1) RBM pre-training:
10 #    in this unsupervised stage we train stack of RBMs,
11 #    a good starting point for frame cross-entropy trainig.
12 # 2) frame cross-entropy training:
13 #    the objective is to classify frames to correct pdfs.
14 # 3) sequence-training optimizing sMBR:
15 #    the objective is to emphasize state-sequences with better
16 #    frame accuracy w.r.t. reference alignment.
17
18 . ./cmd.sh ## You'll want to change cmd.sh to something that will work on your system.
19           ## This relates to the queue.
20
21 . ./path.sh ## Source the tools/utils (import the queue.pl)
22
23 # Config:
24 gmmdir=exp/tri3
25 data_fmllr=data-fmllr-tri3
26 stage=0 # resume training with --stage=N
27 # End of config.
28 . utils/parse_options.sh || exit 1;
29 #
30
31 if [ $stage -le 0 ]; then
32   # Store fMLLR features, so we can train on them easily,
33   # test
34   dir=$data_fmllr/test
35   steps/nnet/make_fmllr_feats.sh --nj 10 --cmd "$train_cmd" \
36     --transform-dir $gmmdir/decode_test \
37     $dir data/test $gmmdir $dir/log $dir/data || exit 1
38   # dev
39   dir=$data_fmllr/dev
40   steps/nnet/make_fmllr_feats.sh --nj 10 --cmd "$train_cmd" \
41     --transform-dir $gmmdir/decode_dev \
42     $dir data/dev $gmmdir $dir/log $dir/data || exit 1
43   # train
44   dir=$data_fmllr/train
45   steps/nnet/make_fmllr_feats.sh --nj 10 --cmd "$train_cmd" \
46     --transform-dir ${gmmdir}_ali \
47     $dir data/train $gmmdir $dir/log $dir/data || exit 1
48   # split the data : 90% train 10% cross-validation (held-out)
49   utils/subset_data_dir_tr_cv.sh $dir ${dir}_tr90 ${dir}_cv10 || exit 1
50 fi
```

13.1 Stage 1: Pre-training

The implementation of layer-wise RBM (Restricted Boltzmann Machine) pre-training algorithm is Contrastive Divergence with 1-step of Markov Chain Monte Carlo sampling (CD-1). The hyper-parameters of the recipe were tuned on the 100 hours Switchboard subset. If smaller databases are used, mainly the number of epochs N needs to be set to 100 hours/set_size. The training is unsupervised, so it is sufficient to provide single data-directory with input features.

```
51
52  if [ $stage -le 1 ]; then
53    # Pre-train DBN, i.e. a stack of RBMs (small database,
      smaller DNN)
54    dir=exp/dnn4_pretrain-dbn
55    (tail --pid=$$ -F $dir/log/pretrain_dbn.log 2>/dev/null)&
      # forward log
56    $cuda_cmd $dir/log/pretrain_dbn.log \
57      steps/nnet/pretrain_dbn.sh --hid-dim 1024 --rbm-iter 20
        $data_fmllr/train $dir || exit 1;
58  fi
```

When training the RBM with Gaussian-Bernoulli units, there is a high risk of weight-explosion, especially with larger learning rates and thousands of hidden neurons. To avoid weight-explosion a mechanism, which compares the variance of training data with the variance of the reconstruction data in a minibatch has been implemented. If the variance of reconstruction is >2x larger, the weights are shrank and the learning rate is temporarily reduced.

13.2 Stage 2: Frame-level Cross-entropy

In this phase a DNN which classifies frames into triphone-states is trained. This is done by mini-batch Stochastic Gradient Descent. The default is to use Sigmoid hidden units, Softmax output units and fully connected layers AffineTransform. The learning rate by default is 0.008, size of mini-batch 256; no momentum or regularization is used. The optimal learning-rate differs with type of hidden units, the value for sigmoid is 0.008, for tanh 0.00001.

```
60  if [ $stage -le 2 ]; then
61    # Train the DNN optimizing per-frame cross-entropy.
62    dir=exp/dnn4_pretrain-dbn_dnn
63    ali=${gmmdir}_ali
64    feature_transform=exp/dnn4_pretrain-dbn/final.feature_transform
65    dbn=exp/dnn4_pretrain-dbn/6.dbn
66    (tail --pid=$$ -F $dir/log/train_nnet.log 2>/dev/null)& #
      forward log
67    # Train
68    $cuda_cmd $dir/log/train_nnet.log \
69      steps/nnet/train.sh --feature-transform $feature_transform --
      dbn $dbn --hid-layers 0 --learn-rate 0.008 \
70      $data_fmllr/train_tr90 $data_fmllr/train_cv10 data/lang $ali
        $ali $dir || exit 1;
71    # Decode (reuse HCLG graph)
72    steps/nnet/decode.sh --nj 20 --cmd "$decode_cmd" --acwt 0.2 \
73      $gmmdir/graph $data_fmllr/test $dir/decode_test || exit 1;
74    steps/nnet/decode.sh --nj 20 --cmd "$decode_cmd" --acwt 0.2 \
75      $gmmdir/graph $data_fmllr/dev $dir/decode_dev || exit 1;
76  fi
77
78
79  # Sequence training using sMBR criterion, we do Stochastic-GD
80  # with per-utterance updates. We use usually good acwt 0.1
81  dir=exp/dnn4_pretrain-dbn_dnn_smbr
82  srcdir=exp/dnn4_pretrain-dbn_dnn
83  acwt=0.2
```

The input_transform and the pre-trained DBN (i.e. Deep Belief Network, stack of RBMs) is passed into to the script using the options '-input-transform' and '-dbn', only the output layer is initialized randomly. An early stopping criterium is used to prevent over-fitting, for this the objective function on the cross-

validation set (i.e. held-out set) is measured, therefore two pairs of feature-alignment dirs are needed to perform the supervised training.

13.3 Stage 3: Sequence-discriminative Training

In this phase, the neural networks is trained to classify correctly the whole sentences, which is closer to the general ASR objective than frame-level training. The objective of sequence-discriminative training is to maximize the expected accuracy of state labels derived from reference transcriptions, and lattice framework to represent competing hypothesis is used. The training is done by Stochastic Gradient Descent (SGD) with per-utterance updates, low learning rate 1e-5 which is kept constant is used and 3-5 epochs are done. Faster convergence when re-generating lattices after 1st epoch are observed.

```
78
79  # Sequence training using sMBR criterion, we do Stochastic-GD
80  # with per-utterance updates. We use usually good acwt 0.1
81  dir=exp/dnn4_pretrain-dbn_dnn_smbr
82  srcdir=exp/dnn4_pretrain-dbn_dnn
83  acwt=0.2
84
85  if [ $stage -le 3 ]; then
86    # First we generate lattices and alignments:
87    steps/nnet/align.sh --nj 20 --cmd "$train_cmd" \
88      $data_fmllr/train data/lang $srcdir ${srcdir}_ali || exit 1;
89    steps/nnet/make_denlats.sh --nj 20 --cmd "$decode_cmd" --acwt
      $acwt \
90      --lattice-beam 10.0 --beam 18.0 \
91      $data_fmllr/train data/lang $srcdir ${srcdir}_denlats || exit
        1;
92  fi
93
```

MMI, BMMI, MPE and sMBR4 training are all supported. In sMBR optimization, silence frames are excluded from accumulating approximate accuracies.

13.4 Stage 4: Iteration of sMBR

In this phase, we iterated the sMBRs to get better result.

```
if [ $stage -le 4 ]; then
  # Re-train the DNN by 6 iterations of sMBR
  steps/nnet/train_mpe.sh --cmd "$cuda_cmd" --num-
iters 6 --acwt $acwt \
    --do-smbr true \
    $data_fmllr/train data/lang $srcdir ${srcdir}_
ali ${srcdir}_denlats $dir || exit 1
  # Decode
  for ITER in 1 6; do
    steps/nnet/decode.sh --nj 20 --cmd "$decode_
cmd" \
      --nnet $dir/${ITER}.nnet --acwt $acwt \
      $gmmdir/graph $data_fmllr/test $dir/decode_
test_it${ITER} || exit 1
    steps/nnet/decode.sh --nj 20 --cmd "$decode_
cmd" \
      --nnet $dir/${ITER}.nnet --acwt $acwt \
      $gmmdir/graph $data_fmllr/dev $dir/decode_
dev_it${ITER} || exit 1
  done
fi

echo Success
exit 0
```

13.5 Final Output

```
=====================================================
 DNN Hybrid Training & Decoding (Karel's recipe)
=====================================================
steps/nnet/make_fmllr_feats.sh --nj 10 --cmd run.pl
--max-jobs-run 10 --transform-dir exp/tri3/decode_
test data-fmllr-tri3/test data/test exp/tri3 data-
fmllr-tri3/test/log data-fmllr-tri3/test/data
steps/nnet/make_fmllr_feats.sh: feature type is lda_
fmllr
utils/copy_data_dir.sh: copied data from data/test
to data-fmllr-tri3/test
utils/validate_data_dir.sh: Successfully validated
data-directory data-fmllr-tri3/test
steps/nnet/make_fmllr_feats.sh: Done!, type lda_
fmllr, data/test --> data-fmllr-tri3/test, using :
raw-trans None, gmm exp/tri3, trans exp/tri3/decode_
test
steps/nnet/make_fmllr_feats.sh --nj 10 --cmd run.
pl --max-jobs-run 10 --transform-dir exp/tri3/de-
code_dev data-fmllr-tri3/dev data/dev exp/tri3 data-
fmllr-tri3/dev/log data-fmllr-tri3/dev/data
steps/nnet/make_fmllr_feats.sh: feature type is lda_
fmllr
utils/copy_data_dir.sh: copied data from data/dev to
data-fmllr-tri3/dev
utils/validate_data_dir.sh: Successfully validated
data-directory data-fmllr-tri3/dev
steps/nnet/make_fmllr_feats.sh: Done!, type lda_
fmllr, data/dev --> data-fmllr-tri3/dev, using : raw-
trans None, gmm exp/tri3, trans exp/tri3/decode_dev
steps/nnet/make_fmllr_feats.sh --nj 10 --cmd run.pl
--max-jobs-run 10 --transform-dir exp/tri3_ali da-
ta-fmllr-tri3/train data/train exp/tri3 data-fmllr-
tri3/train/log data-fmllr-tri3/train/data
steps/nnet/make_fmllr_feats.sh: feature type is lda_
```

```
fmllr
utils/copy_data_dir.sh: copied data from data/train
to data-fmllr-tri3/train
utils/validate_data_dir.sh: Successfully validated
data-directory data-fmllr-tri3/train
steps/nnet/make_fmllr_feats.sh: Done!, type lda_
fmllr, data/train --> data-fmllr-tri3/train, using :
raw-trans None, gmm exp/tri3, trans exp/tri3_ali
Speakers, src=462, trn=416, cv=46 /tmp/sv_WRwQQ/
speakers_cv
utils/data/subset_data_dir.sh: reducing #utt from
3696 to 3328
utils/data/subset_data_dir.sh: reducing #utt from
3696 to 368
# steps/nnet/pretrain_dbn.sh --hid-dim 1024 --rbm-
iter 20 data-fmllr-tri3/train exp/dnn4_pretrain-dbn
# Started at Fri Jan  4 13:23:49 CST 2019
#
steps/nnet/pretrain_dbn.sh --hid-dim 1024 --rbm-iter
20 data-fmllr-tri3/train exp/dnn4_pretrain-dbn
# INFO
steps/nnet/pretrain_dbn.sh : Pre-training Deep Be-
lief Network as a stack of RBMs
      dir       : exp/dnn4_pretrain-dbn
      Train-set : data-fmllr-tri3/train '3696'


LOG     ([5.5.164~1-9698]:main():cuda-gpu-available.
cc:49)


### IS CUDA GPU AVAILABLE? 'HP' ###
WARNING   ([5.5.164~1-9698]:SelectGpuId():cu-device.
cc:203) Not in compute-exclusive mode.  Suggestion:
use 'nvidia-smi -c 3' to set compute exclusive mode
LOG   ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.
cc:323) Selecting from 1 GPUs
```

```
LOG ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSetDevice(0): GeForce GTX 1070 free:7750M, used:365M, total:8116M, free/total:0.954946

LOG ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to select device: 0 (automatically), mem_ratio: 0.954946

LOG ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:404) Success selecting device 0 free mem ratio: 0.954946

LOG ([5.5.164~1-9698]:FinalizeActiveGpu():cu-device.cc:258) The active GPU is [0]: GeForce GTX 1070 free:7684M, used:431M, total:8116M, free/total:0.946814 version 6.1

### HURRAY, WE GOT A CUDA GPU FOR COMPUTATION!!! ##


### Testing CUDA setup with a small computation (setup = cuda-toolkit + gpu-driver + Kaldi):

### Test OK!


# PREPARING FEATURES

copy-feats --compress=true scp:data-fmllr-tri3/train/feats.scp ark,scp:/tmp/Kaldi.1A01/train.ark,exp/dnn4_pretrain-dbn/train_sorted.scp

LOG (copy-feats[5.5.164~1-9698]:main():copy-feats.cc:143) Copied 3696 feature matrices.

# 'apply-cmvn' not used,

feat-to-dim 'ark:copy-feats scp:exp/dnn4_pretrain-dbn/train.scp ark:- |' -

copy-feats scp:exp/dnn4_pretrain-dbn/train.scp ark:-

WARNING (feat-to-dim[5.5.164~1-9698]:Close():Kaldi-io.cc:515) Pipe copy-feats scp:exp/dnn4_pretrain-dbn/train.scp ark:- | had nonzero return status 36096

# feature dim : 40 (input of 'feature_transform')

+ default 'feature_transform_proto' with splice +/-5 frames
```

```
nnet-initialize --binary=false exp/dnn4_pretrain-dbn/splice5.proto exp/dnn4_pretrain-dbn/tr_splice5.nnet

VLOG[1] (nnet-initialize[5.5.164~1-9698]:Init():nnet-nnet.cc:314) <Splice> <InputDim> 40 <OutputDim> 440 <BuildVector> -5:5 </BuildVector>

LOG (nnet-initialize[5.5.164~1-9698]:main():nnet-initialize.cc:63) Written initialized model to exp/dnn4_pretrain-dbn/tr_splice5.nnet

# compute normalization stats from 10k sentences

compute-cmvn-stats ark:- exp/dnn4_pretrain-dbn/cmvn-g.stats

nnet-forward --print-args=true --use-gpu=yes exp/dnn4_pretrain-dbn/tr_splice5.nnet 'ark:copy-feats scp:exp/dnn4_pretrain-dbn/train.scp.10k ark:- |' ark:-

WARNING (nnet-forward[5.5.164~1-9698]:SelectGpuId():cu-device.cc:203) Not in compute-exclusive mode. Suggestion: use 'nvidia-smi -c 3' to set compute exclusive mode

LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting from 1 GPUs

LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSetDevice(0): GeForce GTX 1070 free:7750M, used:365M, total:8116M, free/total:0.954946

LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to select device: 0 (automatically), mem_ratio: 0.954946

LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:404) Success selecting device 0 free mem ratio: 0.954946

LOG (nnet-forward[5.5.164~1-9698]:FinalizeActiveGpu():cu-device.cc:258) The active GPU is [0]: GeForce GTX 1070 free:7684M, used:431M, total:8116M, free/total:0.946814 version 6.1
```

```
copy-feats   scp:exp/dnn4_pretrain-dbn/train.scp.10k
ark:-
LOG   (copy-feats[5.5.164~1-9698]:main():copy-feats.
cc:143) Copied 3696 feature matrices.
LOG   (nnet-forward[5.5.164~1-9698]:main():nnet-for-
ward.cc:192) Done 3696 files in 0.0376807min, (fps
497522)
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:168) Wrote global
CMVN stats to exp/dnn4_pretrain-dbn/cmvn-g.stats
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:171) Done accumu-
lating CMVN stats for 3696 utterances; 0 had errors.
# + normalization of NN-input at 'exp/dnn4_pretrain-
dbn/tr_splice5_cmvn-g.nnet'
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/tr_splice5.nnet
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating cmvn-to-nnet exp/dnn4_pretrain-
dbn/cmvn-g.stats -|
cmvn-to-nnet exp/dnn4_pretrain-dbn/cmvn-g.stats -
LOG    (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-
nnet.cc:114) Written cmvn in 'nnet1' model to: -
LOG    (nnet-concat[5.5.164~1-9698]:main():nnet-con-
cat.cc:82) Written model to exp/dnn4_pretrain-dbn/
tr_splice5_cmvn-g.nnet


### Showing the final 'feature_transform':
nnet-info   exp/dnn4_pretrain-dbn/tr_splice5_cmvn-g.
nnet
LOG     (nnet-info[5.5.164~1-9698]:main():nnet-info.
cc:57) Printed info about exp/dnn4_pretrain-dbn/tr_
splice5_cmvn-g.nnet
num-components 3
input-dim 40
output-dim 440
```

```
number-of-parameters 0.00088 millions
component 1 : <Splice>, input-dim 40, output-dim 440,
  frame_offsets [ -5 -4 -3 -2 -1 0 1 2 3 4 5 ]
component 2 : <AddShift>, input-dim 440, output-dim
440,
   shift_data ( min -0.168308, max 0.0655773, mean
-0.00402776, stddev 0.0406097, skewness -2.33079,
kurtosis 6.28039 ) , lr-coef 0
component 3 : <Rescale>, input-dim 440, output-dim
440,
   scale_data ( min 0.320696, max 0.967125, mean
0.762129, stddev 0.15756, skewness -0.640707, kurto-
sis -0.34954 ) , lr-coef 0
###


# PRE-TRAINING RBM LAYER 1
# initializing 'exp/dnn4_pretrain-dbn/1.rbm.init'
# pretraining 'exp/dnn4_pretrain-dbn/1.rbm' (input
gauss, lrate 0.01, iters 40)
# converting RBM to exp/dnn4_pretrain-dbn/1.dbn
rbm-convert-to-nnet exp/dnn4_pretrain-dbn/1.rbm exp/
dnn4_pretrain-dbn/1.dbn
LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-
convert-to-nnet.cc:69) Written model to exp/dnn4_
pretrain-dbn/1.dbn


# PRE-TRAINING RBM LAYER 2
# computing cmvn stats 'exp/dnn4_pretrain-dbn/2.
cmvn' for RBM initialization
WARNING                    (nnet-forward[5.5.164~1-
9698]:SelectGpuId():cu-device.cc:203) Not in com-
pute-exclusive mode. Suggestion: use 'nvidia-smi -c
3' to set compute exclusive mode
LOG                        (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting
```

```
from 1 GPUs
LOG                          (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:338)   cudaSet-
Device(0): GeForce GTX 1070 free:7750M,    used:366M,
total:8116M, free/total:0.954884
LOG                          (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to
select device: 0 (automatically), mem_ratio: 0.954884
LOG                          (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:404)    Success
selecting device 0 free mem ratio: 0.954884
LOG                          (nnet-forward[5.5.164~1-
9698]:FinalizeActiveGpu():cu-device.cc:258) The ac-
tive GPU is [0]: GeForce GTX 1070     free:7684M,
used:432M, total:8116M, free/total:0.946752 version
6.1
nnet-concat     exp/dnn4_pretrain-dbn/final.feature_
transform exp/dnn4_pretrain-dbn/1.dbn -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/final.feature_
transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating exp/dnn4_pretrain-dbn/1.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to -
copy-feats   scp:exp/dnn4_pretrain-dbn/train.scp.10k
ark:-
LOG   (copy-feats[5.5.164~1-9698]:main():copy-feats.
cc:143) Copied 3696 feature matrices.
LOG   (nnet-forward[5.5.164~1-9698]:main():nnet-for-
ward.cc:192) Done 3696 files in 0.0902158min, (fps
207802)
LOG                    (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:168) Wrote global
CMVN stats to standard output
LOG                    (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:171) Done accumu-
```

```
lating CMVN stats for 3696 utterances; 0 had errors.
LOG (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-nnet.cc:114) Written cmvn in 'nnet1' model to: exp/dnn4_pretrain-dbn/2.cmvn
initializing 'exp/dnn4_pretrain-dbn/2.rbm.init'
pretraining 'exp/dnn4_pretrain-dbn/2.rbm' (lrate 0.4, iters 20)
# appending RBM to exp/dnn4_pretrain-dbn/2.dbn
nnet-concat exp/dnn4_pretrain-dbn/1.dbn 'rbm-convert-to-nnet exp/dnn4_pretrain-dbn/2.rbm - |' exp/dnn4_pretrain-dbn/2.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:53) Reading exp/dnn4_pretrain-dbn/1.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:65) Concatenating rbm-convert-to-nnet exp/dnn4_pretrain-dbn/2.rbm - |
rbm-convert-to-nnet exp/dnn4_pretrain-dbn/2.rbm -
LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-convert-to-nnet.cc:69) Written model to -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:82) Written model to exp/dnn4_pretrain-dbn/2.dbn


# PRE-TRAINING RBM LAYER 3
# computing cmvn stats 'exp/dnn4_pretrain-dbn/3.cmvn' for RBM initialization
WARNING (nnet-forward[5.5.164~1-9698]:SelectGpuId():cu-device.cc:203) Not in compute-exclusive mode. Suggestion: use 'nvidia-smi -c 3' to set compute exclusive mode
LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting from 1 GPUs
LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSetDevice(0): GeForce GTX 1070 free:7751M, used:365M, total:8116M, free/total:0.954984
```

```
LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to select device: 0 (automatically), mem_ratio: 0.954984
LOG (nnet-forward[5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:404) Success selecting device 0 free mem ratio: 0.954984
LOG (nnet-forward[5.5.164~1-9698]:FinalizeActiveGpu():cu-device.cc:258) The active GPU is [0]: GeForce GTX 1070 free:7685M, used:431M, total:8116M, free/total:0.946853 version 6.1
nnet-concat exp/dnn4_pretrain-dbn/final.feature_transform exp/dnn4_pretrain-dbn/2.dbn -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:53) Reading exp/dnn4_pretrain-dbn/final.feature_transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:65) Concatenating exp/dnn4_pretrain-dbn/2.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:82) Written model to -
copy-feats scp:exp/dnn4_pretrain-dbn/train.scp.10k ark:-
LOG (copy-feats[5.5.164~1-9698]:main():copy-feats.cc:143) Copied 3696 feature matrices.
LOG (nnet-forward[5.5.164~1-9698]:main():nnet-forward.cc:192) Done 3696 files in 0.1077min, (fps 174068)
LOG (compute-cmvn-stats[5.5.164~1-9698]:main():compute-cmvn-stats.cc:168) Wrote global CMVN stats to standard output
LOG (compute-cmvn-stats[5.5.164~1-9698]:main():compute-cmvn-stats.cc:171) Done accumulating CMVN stats for 3696 utterances; 0 had errors.
LOG (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-nnet.cc:114) Written cmvn in 'nnet1' model to: exp/dnn4_pretrain-dbn/3.cmvn
initializing 'exp/dnn4_pretrain-dbn/3.rbm.init'
```

```
pretraining 'exp/dnn4_pretrain-dbn/3.rbm' (lrate
0.4, iters 20)

# appending RBM to exp/dnn4_pretrain-dbn/3.dbn

nnet-concat exp/dnn4_pretrain-dbn/2.dbn 'rbm-con-
vert-to-nnet exp/dnn4_pretrain-dbn/3.rbm - |' exp/
dnn4_pretrain-dbn/3.dbn

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/2.dbn

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating rbm-convert-to-nnet exp/dnn4_
pretrain-dbn/3.rbm - |

rbm-convert-to-nnet exp/dnn4_pretrain-dbn/3.rbm -

LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-
convert-to-nnet.cc:69) Written model to -

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to exp/dnn4_pretrain-dbn/3.dbn


# PRE-TRAINING RBM LAYER 4

# computing cmvn stats 'exp/dnn4_pretrain-dbn/4.
cmvn' for RBM initialization

WARNING (nnet-forward[5.5.164~1-
9698]:SelectGpuId():cu-device.cc:203) Not in com-
pute-exclusive mode. Suggestion: use 'nvidia-smi -c
3' to set compute exclusive mode

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting
from 1 GPUs

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSet-
Device(0): GeForce GTX 1070 free:7750M, used:365M,
total:8116M, free/total:0.954946

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to
select device: 0 (automatically), mem_ratio: 0.954946

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:404) Success
```

```
selecting device 0 free mem ratio: 0.954946
LOG                          (nnet-forward[5.5.164~1-
9698]:FinalizeActiveGpu():cu-device.cc:258) The ac-
tive GPU is [0]: GeForce GTX 1070     free:7684M,
used:431M, total:8116M, free/total:0.946814 version
6.1
nnet-concat     exp/dnn4_pretrain-dbn/final.feature_
transform exp/dnn4_pretrain-dbn/3.dbn -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/final.feature_
transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating exp/dnn4_pretrain-dbn/3.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to -
copy-feats   scp:exp/dnn4_pretrain-dbn/train.scp.10k
ark:-
LOG   (copy-feats[5.5.164~1-9698]:main():copy-feats.
cc:143) Copied 3696 feature matrices.
LOG   (nnet-forward[5.5.164~1-9698]:main():nnet-for-
ward.cc:192) Done 3696 files in 0.11935min, (fps
157076)
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:168) Wrote global
CMVN stats to standard output
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:171) Done accumu-
lating CMVN stats for 3696 utterances; 0 had errors.
LOG    (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-
nnet.cc:114) Written cmvn in 'nnet1' model to: exp/
dnn4_pretrain-dbn/4.cmvn
initializing 'exp/dnn4_pretrain-dbn/4.rbm.init'
pretraining   'exp/dnn4_pretrain-dbn/4.rbm'   (lrate
0.4, iters 20)
# appending RBM to exp/dnn4_pretrain-dbn/4.dbn
nnet-concat   exp/dnn4_pretrain-dbn/3.dbn   'rbm-con-
```

```
vert-to-nnet exp/dnn4_pretrain-dbn/4.rbm - |' exp/
dnn4_pretrain-dbn/4.dbn

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/3.dbn

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating rbm-convert-to-nnet exp/dnn4_
pretrain-dbn/4.rbm - |

rbm-convert-to-nnet exp/dnn4_pretrain-dbn/4.rbm -

LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-
convert-to-nnet.cc:69) Written model to -

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to exp/dnn4_pretrain-dbn/4.dbn


# PRE-TRAINING RBM LAYER 5

# computing cmvn stats 'exp/dnn4_pretrain-dbn/5.
cmvn' for RBM initialization

WARNING (nnet-forward[5.5.164~1-
9698]:SelectGpuId():cu-device.cc:203) Not in com-
pute-exclusive mode. Suggestion: use 'nvidia-smi -c
3' to set compute exclusive mode

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting
from 1 GPUs

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSet-
Device(0): GeForce GTX 1070 free:7750M, used:366M,
total:8116M, free/total:0.954907

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to
select device: 0 (automatically), mem_ratio: 0.954907

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:404) Success
selecting device 0 free mem ratio: 0.954907

LOG (nnet-forward[5.5.164~1-
9698]:FinalizeActiveGpu():cu-device.cc:258) The ac-
tive GPU is [0]: GeForce GTX 1070 free:7684M,
used:432M, total:8116M, free/total:0.946775 version
```

```
6.1
nnet-concat     exp/dnn4_pretrain-dbn/final.feature_
transform exp/dnn4_pretrain-dbn/4.dbn -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/final.feature_
transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating exp/dnn4_pretrain-dbn/4.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to -
copy-feats   scp:exp/dnn4_pretrain-dbn/train.scp.10k
ark:-
LOG   (copy-feats[5.5.164~1-9698]:main():copy-feats.
cc:143) Copied 3696 feature matrices.
LOG   (nnet-forward[5.5.164~1-9698]:main():nnet-for-
ward.cc:192) Done 3696 files in 0.13468min, (fps
139197)
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:168) Wrote global
CMVN stats to standard output
LOG                   (compute-cmvn-stats[5.5.164~1-
9698]:main():compute-cmvn-stats.cc:171) Done accumu-
lating CMVN stats for 3696 utterances; 0 had errors.
LOG    (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-
nnet.cc:114) Written cmvn in 'nnet1' model to: exp/
dnn4_pretrain-dbn/5.cmvn
initializing 'exp/dnn4_pretrain-dbn/5.rbm.init'
pretraining   'exp/dnn4_pretrain-dbn/5.rbm'   (lrate
0.4, iters 20)
# appending RBM to exp/dnn4_pretrain-dbn/5.dbn
nnet-concat  exp/dnn4_pretrain-dbn/4.dbn  'rbm-con-
vert-to-nnet exp/dnn4_pretrain-dbn/5.rbm - |' exp/
dnn4_pretrain-dbn/5.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/4.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
```

```
cc:65) Concatenating rbm-convert-to-nnet exp/dnn4_
pretrain-dbn/5.rbm - |

rbm-convert-to-nnet exp/dnn4_pretrain-dbn/5.rbm -

LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-
convert-to-nnet.cc:69) Written model to -

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to exp/dnn4_pretrain-dbn/5.dbn


# PRE-TRAINING RBM LAYER 6

# computing cmvn stats 'exp/dnn4_pretrain-dbn/6.
cmvn' for RBM initialization

WARNING (nnet-forward[5.5.164~1-
9698]:SelectGpuId():cu-device.cc:203) Not in com-
pute-exclusive mode. Suggestion: use 'nvidia-smi -c
3' to set compute exclusive mode

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:323) Selecting
from 1 GPUs

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:338) cudaSet-
Device(0): GeForce GTX 1070 free:7751M, used:365M,
total:8116M, free/total:0.954999

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:385) Trying to
select device: 0 (automatically), mem_ratio: 0.954999

LOG (nnet-forward[5.5.164~1-
9698]:SelectGpuIdAuto():cu-device.cc:404) Success
selecting device 0 free mem ratio: 0.954999

LOG (nnet-forward[5.5.164~1-
9698]:FinalizeActiveGpu():cu-device.cc:258) The ac-
tive GPU is [0]: GeForce GTX 1070 free:7685M,
used:431M, total:8116M, free/total:0.946868 version
6.1

nnet-concat exp/dnn4_pretrain-dbn/final.feature_
transform exp/dnn4_pretrain-dbn/5.dbn -

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/final.feature_
```

```
transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:65) Concatenating exp/dnn4_pretrain-dbn/5.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:82) Written model to -
copy-feats scp:exp/dnn4_pretrain-dbn/train.scp.10k ark:-
LOG (copy-feats[5.5.164~1-9698]:main():copy-feats.cc:143) Copied 3696 feature matrices.
LOG (nnet-forward[5.5.164~1-9698]:main():nnet-forward.cc:192) Done 3696 files in 0.148966min, (fps 125848)
LOG (compute-cmvn-stats[5.5.164~1-9698]:main():compute-cmvn-stats.cc:168) Wrote global CMVN stats to standard output
LOG (compute-cmvn-stats[5.5.164~1-9698]:main():compute-cmvn-stats.cc:171) Done accumulating CMVN stats for 3696 utterances; 0 had errors.
LOG (cmvn-to-nnet[5.5.164~1-9698]:main():cmvn-to-nnet.cc:114) Written cmvn in 'nnet1' model to: exp/dnn4_pretrain-dbn/6.cmvn
initializing 'exp/dnn4_pretrain-dbn/6.rbm.init'
pretraining 'exp/dnn4_pretrain-dbn/6.rbm' (lrate 0.4, iters 20)
# appending RBM to exp/dnn4_pretrain-dbn/6.dbn
nnet-concat exp/dnn4_pretrain-dbn/5.dbn 'rbm-convert-to-nnet exp/dnn4_pretrain-dbn/6.rbm - |' exp/dnn4_pretrain-dbn/6.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:53) Reading exp/dnn4_pretrain-dbn/5.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.cc:65) Concatenating rbm-convert-to-nnet exp/dnn4_pretrain-dbn/6.rbm - |
rbm-convert-to-nnet exp/dnn4_pretrain-dbn/6.rbm -
LOG (rbm-convert-to-nnet[5.5.164~1-9698]:main():rbm-convert-to-nnet.cc:69) Written model to -
```

```
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to exp/dnn4_pretrain-dbn/6.dbn


# REPORT

# RBM pre-training progress (line per-layer)

exp/dnn4_pretrain-dbn/log/rbm.1.log:progress:
[59.4867  53.2198  52.0588  51.4113  51.068  50.8504
50.702 50.5875 50.5049 50.4403 50.3839 50.3431 50.3531
50.3546  50.3716  50.3526  50.3494  50.3537  50.3309
50.3397 50.3387 50.3246 50.325 50.3117 ]

exp/dnn4_pretrain-dbn/log/rbm.2.log:progress:
[6.72542  5.54473  5.45044  5.38909  5.33717  5.28904
5.23951 5.19439 5.14991 5.10887 5.06825 5.04026 ]

exp/dnn4_pretrain-dbn/log/rbm.3.log:progress:
[5.87008  4.54053  4.45563  4.41395  4.37866  4.34733
4.31496 4.28637 4.25737 4.23046 4.20576 4.18703 ]

exp/dnn4_pretrain-dbn/log/rbm.4.log:progress:
[4.41877  3.54689  3.48168  3.45219  3.42853  3.40722
3.38683 3.36887 3.35155 3.3345 3.31848 3.30785 ]

exp/dnn4_pretrain-dbn/log/rbm.5.log:progress:
[4.11132  3.10282  3.0477  3.02518  3.00828  2.99136
2.97569 2.96107 2.94746 2.93379 2.92229 2.91543 ]

exp/dnn4_pretrain-dbn/log/rbm.6.log:progress:
[3.17977  2.53633  2.4901  2.47436  2.46173  2.44852
2.43675 2.42552 2.41665 2.40667 2.39865 2.39314 ]


Pre-training finished.

# Removing features tmpdir /tmp/Kaldi.1A01 @ HP

train.ark

# Accounting: time=1934 threads=1

# Ended (code 0) at Fri Jan  4 13:56:03 CST 2019,
elapsed time 1934 seconds

# steps/nnet/train.sh --feature-transform exp/dnn4_
pretrain-dbn/final.feature_transform --dbn exp/dnn4_
pretrain-dbn/6.dbn --hid-layers 0 --learn-rate 0.008
data-fmllr-tri3/train_tr90    data-fmllr-tri3/train_
```

```
cv10 data/lang exp/tri3_ali exp/tri3_ali exp/dnn4_
pretrain-dbn_dnn
# Started at Fri Jan  4 13:56:03 CST 2019
#
steps/nnet/train.sh  --feature-transform  exp/dnn4_
pretrain-dbn/final.feature_transform --dbn exp/dnn4_
pretrain-dbn/6.dbn --hid-layers 0 --learn-rate 0.008
data-fmllr-tri3/train_tr90   data-fmllr-tri3/train_
cv10 data/lang exp/tri3_ali exp/tri3_ali exp/dnn4_
pretrain-dbn_dnn


# INFO
steps/nnet/train.sh : Training Neural Network
      dir       : exp/dnn4_pretrain-dbn_dnn
      Train-set : data-fmllr-tri3/train_tr90 3328,
exp/tri3_ali
      CV-set      : data-fmllr-tri3/train_cv10 368
exp/tri3_ali


LOG     ([5.5.164~1-9698]:main():cuda-gpu-available.
cc:49)


### IS CUDA GPU AVAILABLE? 'HP' ###
WARNING   ([5.5.164~1-9698]:SelectGpuId():cu-device.
cc:203) Not in compute-exclusive mode.  Suggestion:
use 'nvidia-smi -c 3' to set compute exclusive mode
LOG   ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.
cc:323) Selecting from 1 GPUs
LOG       ([5.5.164~1-9698]:SelectGpuIdAuto():cu-de-
vice.cc:338)    cudaSetDevice(0):    GeForce    GTX
1070 free:7750M, used:365M, total:8116M, free/to-
tal:0.954953
LOG   ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.
cc:385) Trying to select device: 0 (automatically),
mem_ratio: 0.954953
```

```
LOG ([5.5.164~1-9698]:SelectGpuIdAuto():cu-device.cc:404) Success selecting device 0 free mem ratio: 0.954953
LOG ([5.5.164~1-9698]:FinalizeActiveGpu():cu-device.cc:258) The active GPU is [0]: GeForce GTX 1070 free:7684M, used:431M, total:8116M, free/total:0.946822 version 6.1
### HURRAY, WE GOT A CUDA GPU FOR COMPUTATION!!! ##


### Testing CUDA setup with a small computation (setup = cuda-toolkit + gpu-driver + Kaldi):
### Test OK!


# PREPARING ALIGNMENTS
Using PDF targets from dirs 'exp/tri3_ali' 'exp/tri3_ali'
hmm-info exp/tri3_ali/final.mdl
copy-transition-model --binary=false exp/tri3_ali/final.mdl exp/dnn4_pretrain-dbn_dnn/final.mdl
LOG (copy-transition-model[5.5.164~1-9698]:main():copy-transition-model.cc:62) Copied transition model.


# PREPARING FEATURES
# re-saving features to local disk,
copy-feats --compress=true scp:data-fmllr-tri3/train_tr90/feats.scp ark,scp:/tmp/Kaldi.pqzk/train.ark,exp/dnn4_pretrain-dbn_dnn/train_sorted.scp
LOG (copy-feats[5.5.164~1-9698]:main():copy-feats.cc:143) Copied 3328 feature matrices.
copy-feats --compress=true scp:data-fmllr-tri3/train_cv10/feats.scp ark,scp:/tmp/Kaldi.pqzk/cv.ark,exp/dnn4_pretrain-dbn_dnn/cv.scp
LOG (copy-feats[5.5.164~1-9698]:main():copy-feats.cc:143) Copied 368 feature matrices.
```

```
# importing feature settings from dir 'exp/dnn4_pre-
train-dbn'
# cmvn_opts='' delta_opts='' ivector_dim=''
# 'apply-cmvn' is not used,
feat-to-dim 'ark:copy-feats scp:exp/dnn4_pretrain-
dbn_dnn/train.scp.10k ark:- |' -
copy-feats scp:exp/dnn4_pretrain-dbn_dnn/train.
scp.10k ark:-
WARNING (feat-to-dim[5.5.164~1-9698]:Close():Kaldi-
io.cc:515) Pipe copy-feats scp:exp/dnn4_pretrain-
dbn_dnn/train.scp.10k ark:- | had nonzero return sta-
tus 36096
# feature dim : 40 (input of 'feature_transform')
# importing 'feature_transform' from 'exp/dnn4_pre-
train-dbn/final.feature_transform'


### Showing the final 'feature_transform':
nnet-info exp/dnn4_pretrain-dbn_dnn/imported_final.
feature_transform
LOG (nnet-info[5.5.164~1-9698]:main():nnet-info.
cc:57) Printed info about exp/dnn4_pretrain-dbn_dnn/
imported_final.feature_transform
num-components 3
input-dim 40
output-dim 440
number-of-parameters 0.00088 millions
component 1 : <Splice>, input-dim 40, output-dim 440,
  frame_offsets [ -5 -4 -3 -2 -1 0 1 2 3 4 5 ]
component 2 : <AddShift>, input-dim 440, output-dim
440,
   shift_data ( min -0.168308, max 0.0655773, mean
-0.00402776, stddev 0.0406097, skewness -2.33079,
kurtosis 6.28039 ) , lr-coef 0
component 3 : <Rescale>, input-dim 440, output-dim
440,
```

```
   scale_data ( min 0.320696, max 0.967125, mean
0.762129, stddev 0.15756, skewness -0.640707, kurto-
sis -0.34954 ) , lr-coef 0
###


# NN-INITIALIZATION
# getting input/output dims :
feat-to-dim 'ark:copy-feats scp:exp/dnn4_pretrain-
dbn_dnn/train.scp.10k ark:- | nnet-forward "nnet-
concat exp/dnn4_pretrain-dbn_dnn/final.feature_
transform '\''exp/dnn4_pretrain-dbn/6.dbn'\'' -|"
ark:- ark:- |' -
copy-feats scp:exp/dnn4_pretrain-dbn_dnn/train.
scp.10k ark:-
nnet-forward "nnet-concat exp/dnn4_pretrain-dbn_dnn/
final.feature_transform 'exp/dnn4_pretrain-dbn/6.
dbn' -|" ark:- ark:-
LOG (nnet-forward[5.5.164~1-9698]:SelectGpuId():cu-
device.cc:128) Manually selected to compute on CPU.
nnet-concat exp/dnn4_pretrain-dbn_dnn/final.feature_
transform exp/dnn4_pretrain-dbn/6.dbn -
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn_dnn/final.fea-
ture_transform
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating exp/dnn4_pretrain-dbn/6.dbn
LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to -
WARNING (feat-to-dim[5.5.164~1-9698]:Close():Kaldi-
io.cc:515) Pipe copy-feats scp:exp/dnn4_pretrain-dbn_
dnn/train.scp.10k ark:- | nnet-forward "nnet-concat
exp/dnn4_pretrain-dbn_dnn/final.feature_transform
'exp/dnn4_pretrain-dbn/6.dbn' -|" ark:- ark:- | had
nonzero return status 36096
# genrating network prototype exp/dnn4_pretrain-dbn_
dnn/nnet.proto
```

```
# initializing the NN 'exp/dnn4_pretrain-dbn_dnn/
nnet.proto' -> 'exp/dnn4_pretrain-dbn_dnn/nnet.init'

nnet-initialize --seed=777 exp/dnn4_pretrain-dbn_
dnn/nnet.proto exp/dnn4_pretrain-dbn_dnn/nnet.init

VLOG[1] (nnet-initialize[5.5.164~1-
9698]:Init():nnet-nnet.cc:314) <AffineTransform>
<InputDim> 1024 <OutputDim> 1904 <BiasMean> 0.000000
<BiasRange> 0.000000 <ParamStddev> 0.091474

VLOG[1] (nnet-initialize[5.5.164~1-
9698]:Init():nnet-nnet.cc:314) <Softmax> <InputDim>
1904 <OutputDim> 1904

VLOG[1] (nnet-initialize[5.5.164~1-
9698]:Init():nnet-nnet.cc:314) </NnetProto>

LOG (nnet-initialize[5.5.164~1-9698]:main():nnet-
initialize.cc:63) Written initialized model to exp/
dnn4_pretrain-dbn_dnn/nnet.init

nnet-concat exp/dnn4_pretrain-dbn/6.dbn exp/dnn4_
pretrain-dbn_dnn/nnet.init exp/dnn4_pretrain-dbn_
dnn/nnet_dbn_dnn.init

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:53) Reading exp/dnn4_pretrain-dbn/6.dbn

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:65) Concatenating exp/dnn4_pretrain-dbn_dnn/nnet.
init

LOG (nnet-concat[5.5.164~1-9698]:main():nnet-concat.
cc:82) Written model to exp/dnn4_pretrain-dbn_dnn/
nnet_dbn_dnn.init


# RUNNING THE NN-TRAINING SCHEDULER

steps/nnet/train_scheduler.sh --feature-transform
exp/dnn4_pretrain-dbn_dnn/final.feature_transform
--learn-rate 0.008 exp/dnn4_pretrain-dbn_dnn/nnet_
dbn_dnn.init ark:copy-feats scp:exp/dnn4_pretrain-
dbn_dnn/train.scp ark:- | ark:copy-feats scp:exp/
dnn4_pretrain-dbn_dnn/cv.scp ark:- | ark:ali-to-pdf
exp/tri3_ali/final.mdl "ark:gunzip -c exp/tri3_ali/
ali.*.gz |" ark:- | ali-to-post ark:- ark:- | ark:ali-
```

```
to-pdf exp/tri3_ali/final.mdl "ark:gunzip -c exp/
tri3_ali/ali.*.gz |" ark:- | ali-to-post ark:- ark:-
| exp/dnn4_pretrain-dbn_dnn

CROSSVAL PRERUN AVG.LOSS 7.7600 (Xent),

ITERATION 01: TRAIN AVG.LOSS 2.0913, (lrate0.008),
CROSSVAL AVG.LOSS 1.9001, nnet accepted (nnet_dbn_
dnn_iter01_learnrate0.008_tr2.0913_cv1.9001)

ITERATION 02: TRAIN AVG.LOSS 1.3923, (lrate0.008),
CROSSVAL AVG.LOSS 1.7871, nnet accepted (nnet_dbn_
dnn_iter02_learnrate0.008_tr1.3923_cv1.7871)

ITERATION 03: TRAIN AVG.LOSS 1.1886, (lrate0.008),
CROSSVAL AVG.LOSS 1.7467, nnet accepted (nnet_dbn_
dnn_iter03_learnrate0.008_tr1.1886_cv1.7467)

ITERATION 04: TRAIN AVG.LOSS 1.0431, (lrate0.008),
CROSSVAL AVG.LOSS 1.7458, nnet accepted (nnet_dbn_
dnn_iter04_learnrate0.008_tr1.0431_cv1.7458)

ITERATION 05: TRAIN AVG.LOSS 0.8836, (lrate0.004),
CROSSVAL AVG.LOSS 1.6402, nnet accepted (nnet_dbn_
dnn_iter05_learnrate0.004_tr0.8836_cv1.6402)

ITERATION 06: TRAIN AVG.LOSS 0.8084, (lrate0.002),
CROSSVAL AVG.LOSS 1.5782, nnet accepted (nnet_dbn_
dnn_iter06_learnrate0.002_tr0.8084_cv1.5782)

ITERATION 07: TRAIN AVG.LOSS 0.7764, (lrate0.001),
CROSSVAL AVG.LOSS 1.5362, nnet accepted (nnet_dbn_
dnn_iter07_learnrate0.001_tr0.7764_cv1.5362)

ITERATION 08: TRAIN AVG.LOSS 0.7625, (lrate0.0005),
CROSSVAL AVG.LOSS 1.5088, nnet accepted (nnet_dbn_
dnn_iter08_learnrate0.0005_tr0.7625_cv1.5088)

ITERATION 09: TRAIN AVG.LOSS 0.7554, (lrate0.00025),
CROSSVAL AVG.LOSS 1.4934, nnet accepted (nnet_dbn_
dnn_iter09_learnrate0.00025_tr0.7554_cv1.4934)

ITERATION 10: TRAIN AVG.LOSS 0.7511, (lrate0.000125),
CROSSVAL AVG.LOSS 1.4856, nnet accepted (nnet_dbn_
dnn_iter10_learnrate0.000125_tr0.7511_cv1.4856)

ITERATION 11: TRAIN AVG.LOSS 0.7481, (lrate6.25e-05),
CROSSVAL AVG.LOSS 1.4820, nnet accepted (nnet_dbn_
dnn_iter11_learnrate6.25e-05_tr0.7481_cv1.4820)
```

```
ITERATION 12: TRAIN AVG.LOSS 0.7461, (lrate3.125e-05),
CROSSVAL AVG.LOSS 1.4804, nnet accepted (nnet_dbn_
dnn_iter12_learnrate3.125e-05_tr0.7461_cv1.4804)

ITERATION 13: TRAIN AVG.LOSS 0.7448, (lrate1.5625e-05),
CROSSVAL AVG.LOSS 1.4797, nnet accepted (nnet_dbn_
dnn_iter13_learnrate1.5625e-05_tr0.7448_cv1.4797)

finished, too small rel. improvement 0.000472845

steps/nnet/train_scheduler.sh: Succeeded training
the Neural Network : 'exp/dnn4_pretrain-dbn_dnn/fi-
nal.nnet'

steps/nnet/train.sh: Successfuly finished. 'exp/
dnn4_pretrain-dbn_dnn'

steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-
run 10 --acwt 0.2 exp/tri3/graph data-fmllr-tri3/
test exp/dnn4_pretrain-dbn_dnn/decode_test

# Removing features tmpdir /tmp/Kaldi.pqzk @ HP

cv.ark

train.ark

# Accounting: time=333 threads=1

# Ended (code 0) at Fri Jan  4 14:01:36 CST 2019,
elapsed time 333 seconds

steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-
run 10 --acwt 0.2 exp/tri3/graph data-fmllr-tri3/dev
exp/dnn4_pretrain-dbn_dnn/decode_dev

steps/nnet/align.sh --nj 20 --cmd run.pl --max-jobs-
run 10 data-fmllr-tri3/train data/lang exp/dnn4_pre-
train-dbn_dnn exp/dnn4_pretrain-dbn_dnn_ali

steps/nnet/align.sh: aligning data 'data-fmllr-tri3/
train' using nnet/model 'exp/dnn4_pretrain-dbn_dnn',
putting alignments in 'exp/dnn4_pretrain-dbn_dnn_
ali'

steps/nnet/align.sh: done aligning data.

steps/nnet/make_denlats.sh --nj 20 --cmd run.pl --max-
jobs-run 10 --acwt 0.2 --lattice-beam 10.0 --beam
18.0 data-fmllr-tri3/train data/lang exp/dnn4_pre-
train-dbn_dnn exp/dnn4_pretrain-dbn_dnn_denlats
```

```
Making unigram grammar FST in exp/dnn4_pretrain-dbn_
dnn_denlats/lang

Compiling decoding graph in exp/dnn4_pretrain-dbn_
dnn_denlats/dengraph

tree-info exp/dnn4_pretrain-dbn_dnn/tree

tree-info exp/dnn4_pretrain-dbn_dnn/tree

fsttablecompose   exp/dnn4_pretrain-dbn_dnn_denlats/
lang/L_disambig.fst   exp/dnn4_pretrain-dbn_dnn_den-
lats/lang/G.fst

fstminimizeencoded

fstdeterminizestar --use-log=true

fstpushspecial

fstisstochastic   exp/dnn4_pretrain-dbn_dnn_denlats/
lang/tmp/LG.fst

1.27271e-05 1.27271e-05

fstcomposecontext  --context-size=3  --central-posi-
tion=1   --read-disambig-syms=exp/dnn4_pretrain-dbn_
dnn_denlats/lang/phones/disambig.int    --write-dis-
ambig-syms=exp/dnn4_pretrain-dbn_dnn_denlats/lang/
tmp/disambig_ilabels_3_1.int  exp/dnn4_pretrain-dbn_
dnn_denlats/lang/tmp/ilabels_3_1.17584 exp/dnn4_pre-
train-dbn_dnn_denlats/lang/tmp/LG.fst

fstisstochastic   exp/dnn4_pretrain-dbn_dnn_denlats/
lang/tmp/CLG_3_1.fst

1.27657e-05 0

make-h-transducer  --disambig-syms-out=exp/dnn4_pre-
train-dbn_dnn_denlats/dengraph/disambig_tid.int
--transition-scale=1.0    exp/dnn4_pretrain-dbn_dnn_
denlats/lang/tmp/ilabels_3_1  exp/dnn4_pretrain-dbn_
dnn/tree exp/dnn4_pretrain-dbn_dnn/final.mdl

fstrmepslocal

fsttablecompose   exp/dnn4_pretrain-dbn_dnn_denlats/
dengraph/Ha.fst   exp/dnn4_pretrain-dbn_dnn_denlats/
lang/tmp/CLG_3_1.fst

fstrmsymbols  exp/dnn4_pretrain-dbn_dnn_denlats/den-
graph/disambig_tid.int
```

```
fstminimizeencoded
fstdeterminizestar --use-log=true
fstisstochastic  exp/dnn4_pretrain-dbn_dnn_denlats/dengraph/HCLGa.fst
0.000473932 -0.000485819
add-self-loops --self-loop-scale=0.1 --reorder=true exp/dnn4_pretrain-dbn_dnn/final.mdl  exp/dnn4_pretrain-dbn_dnn_denlats/dengraph/HCLGa.fst
steps/nnet/make_denlats.sh: generating denlats from data 'data-fmllr-tri3/train', putting lattices in 'exp/dnn4_pretrain-dbn_dnn_denlats'
steps/nnet/make_denlats.sh: done generating denominator lattices.
steps/nnet/train_mpe.sh --cmd run.pl --gpu 1 --numiters 6 --acwt 0.2 --do-smbr true data-fmllr-tri3/train data/lang exp/dnn4_pretrain-dbn_dnn exp/dnn4_pretrain-dbn_dnn_ali  exp/dnn4_pretrain-dbn_dnn_denlats exp/dnn4_pretrain-dbn_dnn_smbr
Pass 1 (learnrate 0.00001)
 TRAINING FINISHED; Time taken = 1.02159 min; processed 18350.9 frames per second.
 Done 3696 files, 0 with no reference alignments, 0 with no lattices, 0 with other errors.
 Overall average frame-accuracy is 0.871751 over 1124823 frames.
Pass 2 (learnrate 1e-05)
 TRAINING FINISHED; Time taken = 1.02279 min; processed 18329.4 frames per second.
 Done 3696 files, 0 with no reference alignments, 0 with no lattices, 0 with other errors.
 Overall average frame-accuracy is 0.878631 over 1124823 frames.
Pass 3 (learnrate 1e-05)
 TRAINING FINISHED; Time taken = 1.01812 min; processed 18413.4 frames per second.
```

```
 Done 3696 files, 0 with no reference alignments, 0
with no lattices, 0 with other errors.

 Overall average frame-accuracy is 0.882521 over
1124823 frames.

Pass 4 (learnrate 1e-05)

 TRAINING FINISHED; Time taken = 1.02529 min; pro-
cessed 18284.6 frames per second.

 Done 3696 files, 0 with no reference alignments, 0
with no lattices, 0 with other errors.

 Overall average frame-accuracy is 0.885317 over
1124823 frames.

Pass 5 (learnrate 1e-05)

 TRAINING FINISHED; Time taken = 1.03147 min; pro-
cessed 18175.1 frames per second.

 Done 3696 files, 0 with no reference alignments, 0
with no lattices, 0 with other errors.

 Overall average frame-accuracy is 0.887541 over
1124823 frames.

Pass 6 (learnrate 1e-05)

 TRAINING FINISHED; Time taken = 1.03413 min; pro-
cessed 18128.3 frames per second.

 Done 3696 files, 0 with no reference alignments, 0
with no lattices, 0 with other errors.

 Overall average frame-accuracy is 0.889426 over
1124823 frames.

MPE/sMBR training finished

Re-estimating priors by forwarding 10k utterances
from training set.

steps/nnet/make_priors.sh --cmd run.pl --max-jobs-
run 10 --nj 20 data-fmllr-tri3/train exp/dnn4_pre-
train-dbn_dnn_smbr

Accumulating prior stats by forwarding 'data-fmllr-
tri3/train' with 'exp/dnn4_pretrain-dbn_dnn_smbr'

Succeeded creating prior counts 'exp/dnn4_pretrain-
dbn_dnn_smbr/prior_counts'  from  'data-fmllr-tri3/
```

```
train'
steps/nnet/train_mpe.sh: Done. 'exp/dnn4_pretrain-dbn_dnn_smbr'
steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-run 10 --nnet exp/dnn4_pretrain-dbn_dnn_smbr/1.nnet --acwt 0.2 exp/tri3/graph data-fmllr-tri3/test exp/dnn4_pretrain-dbn_dnn_smbr/decode_test_it1
steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-run 10 --nnet exp/dnn4_pretrain-dbn_dnn_smbr/1.nnet --acwt 0.2 exp/tri3/graph data-fmllr-tri3/dev exp/dnn4_pretrain-dbn_dnn_smbr/decode_dev_it1
steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-run 10 --nnet exp/dnn4_pretrain-dbn_dnn_smbr/6.nnet --acwt 0.2 exp/tri3/graph data-fmllr-tri3/test exp/dnn4_pretrain-dbn_dnn_smbr/decode_test_it6
steps/nnet/decode.sh --nj 20 --cmd run.pl --max-jobs-run 10 --nnet exp/dnn4_pretrain-dbn_dnn_smbr/6.nnet --acwt 0.2 exp/tri3/graph data-fmllr-tri3/dev exp/dnn4_pretrain-dbn_dnn_smbr/decode_dev_it6
Success
```

Chapter 14. Final Results

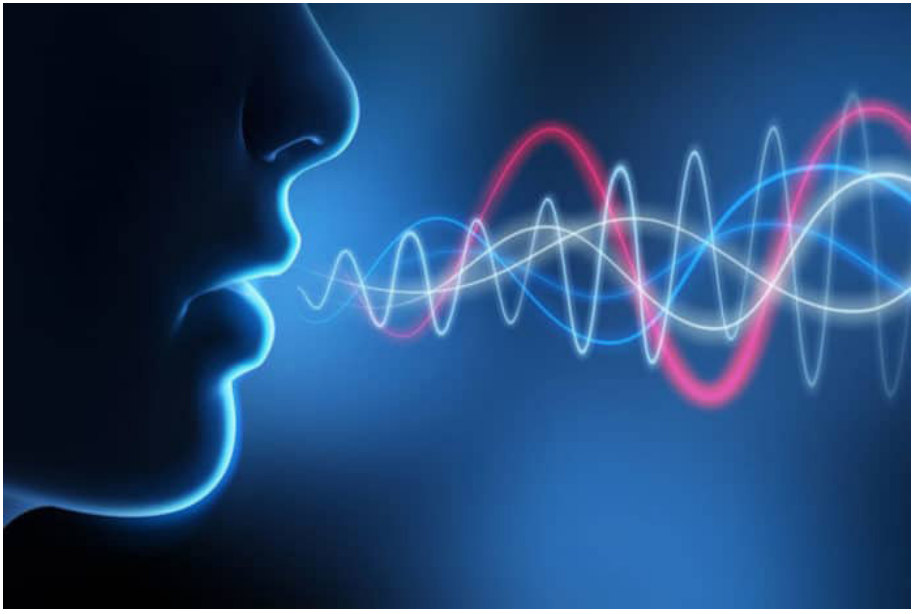
from MonoPhone to DNN

```
=====================================================================
        Getting Results [see RESULTS file]
=====================================================
%WER 31.8 | 400 15057 | 71.6 19.6 8.8 3.4 31.8 100.0 |
-0.506 | exp/mono/decode_dev/score_5/ctm_39phn.filt.
sys
%WER 24.8 | 400 15057 | 79.2 15.8 5.0 4.0 24.8 99.8
|  -0.147  |  exp/tri1/decode_dev/score_10/ctm_39phn.
filt.sys
%WER 22.6 | 400 15057 | 81.0 14.2 4.7 3.6 22.6 99.5
|  -0.217  |  exp/tri2/decode_dev/score_10/ctm_39phn.
filt.sys
%WER 20.5 | 400 15057 | 82.6 12.8 4.7 3.1 20.5 99.8
|  -0.588  |  exp/tri3/decode_dev/score_10/ctm_39phn.
filt.sys
%WER 23.2 | 400 15057 | 80.0 14.8 5.2 3.3 23.2 99.5 |
-0.188  |  exp/tri3/decode_dev.si/score_10/ctm_39phn.
filt.sys
%WER 21.0 | 400 15057 | 81.9 12.5 5.6 2.9 21.0 99.8 |
-0.525 | exp/tri4_nnet/decode_dev/score_5/ctm_39phn.
filt.sys
%WER 18.1 | 400 15057 | 85.2 11.0 3.8 3.3 18.1 99.5
| -0.662 | exp/sgmm2_4/decode_dev/score_6/ctm_39phn.
filt.sys
%WER 18.3 | 400 15057 | 84.8 11.3 3.9 3.2 18.3
99.5 | -0.279 | exp/sgmm2_4_mmi_b0.1/decode_dev_it1/
score_8/ctm_39phn.filt.sys
```

```
%WER 18.5 | 400 15057 | 84.2 11.4 4.4 2.7 18.5
99.5 | -0.138 | exp/sgmm2_4_mmi_b0.1/decode_dev_it2/
score_10/ctm_39phn.filt.sys

%WER 18.5 | 400 15057 | 84.3 11.4 4.4 2.8 18.5
99.5 | -0.148 | exp/sgmm2_4_mmi_b0.1/decode_dev_it3/
score_10/ctm_39phn.filt.sys

%WER 18.5 | 400 15057 | 84.3 11.4 4.3 2.8 18.5
99.5 | -0.155 | exp/sgmm2_4_mmi_b0.1/decode_dev_it4/
score_10/ctm_39phn.filt.sys

%WER 17.4 | 400 15057 | 84.8 10.4 4.7 2.3 17.4 99.3
| -0.616 | exp/dnn4_pretrain-dbn_dnn/decode_dev/
score_6/ctm_39phn.filt.sys

%WER 17.3 | 400 15057 | 85.1 10.4 4.5 2.4 17.3 99.3 |
-0.581 | exp/dnn4_pretrain-dbn_dnn_smbr/decode_dev_
it1/score_6/ctm_39phn.filt.sys

%WER 17.0 | 400 15057 | 85.9 10.2 3.9 2.9 17.0 99.5 |
-0.610 | exp/dnn4_pretrain-dbn_dnn_smbr/decode_dev_
it6/score_6/ctm_39phn.filt.sys

%WER 16.9 | 400 15057 | 85.9 11.0 3.2 2.8 16.9 99.5
| -0.126 | exp/combine_2/decode_dev_it1/score_6/
ctm_39phn.filt.sys

%WER 16.9 | 400 15057 | 86.0 10.9 3.1 2.8 16.9 99.3
| -0.126 | exp/combine_2/decode_dev_it2/score_6/
ctm_39phn.filt.sys

%WER 16.8 | 400 15057 | 85.8 10.8 3.4 2.6 16.8 99.3
| -0.036 | exp/combine_2/decode_dev_it3/score_7/
ctm_39phn.filt.sys

%WER 16.9 | 400 15057 | 86.3 10.9 2.8 3.2 16.9 99.0
| -0.260 | exp/combine_2/decode_dev_it4/score_5/
ctm_39phn.filt.sys

%WER 32.1 | 192 7215 | 71.1 19.4 9.5 3.3 32.1 100.0
| -0.468 | exp/mono/decode_test/score_5/ctm_39phn.
filt.sys

%WER 26.1 | 192 7215 | 77.6 16.9 5.5 3.6 26.1 100.0
| -0.136 | exp/tri1/decode_test/score_10/ctm_39phn.
filt.sys

%WER 24.0 | 192 7215 | 79.7 15.0 5.4 3.7 24.0 99.5
```

```
| -0.260 | exp/tri2/decode_test/score_10/ctm_39phn.
filt.sys

%WER 21.2 | 192 7215 | 82.1 13.2 4.7 3.3 21.2 99.5
| -0.695 | exp/tri3/decode_test/score_9/ctm_39phn.
filt.sys

%WER 24.0 | 192 7215 | 79.3 15.4 5.2 3.4 24.0 99.5 |
-0.281 | exp/tri3/decode_test.si/score_9/ctm_39phn.
filt.sys

%WER 22.8 | 192 7215 | 80.3 13.7 6.0 3.1 22.8 100.0
| -0.423 | exp/tri4_nnet/decode_test/score_5/
ctm_39phn.filt.sys

%WER 19.5 | 192 7215 | 82.8 12.1 5.1 2.3 19.5 100.0 |
-0.120 | exp/sgmm2_4/decode_test/score_10/ctm_39phn.
filt.sys

%WER 19.7 | 192 7215 | 84.3 11.9 3.8 3.9 19.7 100.0
| -0.673 | exp/sgmm2_4_mmi_b0.1/decode_test_it1/
score_6/ctm_39phn.filt.sys

%WER 20.0 | 192 7215 | 83.1 12.3 4.6 3.1 20.0 100.0
| -0.179 | exp/sgmm2_4_mmi_b0.1/decode_test_it2/
score_9/ctm_39phn.filt.sys

%WER 20.0 | 192 7215 | 83.1 12.4 4.5 3.2 20.0 100.0
| -0.195 | exp/sgmm2_4_mmi_b0.1/decode_test_it3/
score_9/ctm_39phn.filt.sys

%WER 19.9 | 192 7215 | 83.8 12.1 4.1 3.7 19.9 99.5
| -0.521 | exp/sgmm2_4_mmi_b0.1/decode_test_it4/
score_7/ctm_39phn.filt.sys

%WER 18.3 | 192 7215 | 84.4 11.2 4.4 2.7 18.3 100.0
| -1.177 | exp/dnn4_pretrain-dbn_dnn/decode_test/
score_4/ctm_39phn.filt.sys

%WER 18.4 | 192 7215 | 84.7 11.2 4.2 3.1 18.4 100.0 |
-1.168 | exp/dnn4_pretrain-dbn_dnn_smbr/decode_test_
it1/score_4/ctm_39phn.filt.sys

%WER 18.5 | 192 7215 | 84.9 11.4 3.7 3.4 18.5 100.0 |
-0.779 | exp/dnn4_pretrain-dbn_dnn_smbr/decode_test_
it6/score_5/ctm_39phn.filt.sys

%WER 17.8 | 192 7215 | 84.9 11.6 3.5 2.7 17.8 99.5
| -0.143 | exp/combine_2/decode_test_it1/score_6/
```

```
ctm_39phn.filt.sys
%WER 18.2 | 192 7215 | 85.1 11.6 3.3 3.2 18.2 99.5
|  -0.236  |  exp/combine_2/decode_test_it2/score_5/
ctm_39phn.filt.sys
%WER 18.1 | 192 7215 | 84.8 11.7 3.5 2.9 18.1 99.5
|  -0.124  |  exp/combine_2/decode_test_it3/score_6/
ctm_39phn.filt.sys
%WER 18.0 | 192 7215 | 84.7 11.8 3.5 2.8 18.0 99.5
|  -0.130  |  exp/combine_2/decode_test_it4/score_6/
ctm_39phn.filt.sys
=====================================================

Finished successfully on Fri Jan 4 14:16:46 CST 2019
=====================================================
```



Next Step

When voice assistants began to emerge in 2011 with the introduction of Siri, no one could have predicted that this novelty would become a driver for tech innovation. Now nearly 10 years later, it's estimated that every one in six Americans own a smart speaker (Google Home, Amazon Echo) and eMarketer forecasts that nearly 100 million smartphone users will be using voice assistants in 2020.

This is only the beginning of voice technology as we will see major advancements in the user interface in the years to come. Voice is the future of brand interaction and customer experience. There is a lot of opportunity for much deeper and much more conversational experiences with customers.

The question is, are you willing to jump on this opportunity?

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z