# Step by Step to Build a Speech Recognizer

Author: Jiatong Shi & Shuai Guo

**Task Description:**

- Explore Kaldi and its ASR (Automatic Speech Recognition) training process

**Task Aims:**

- Be familiar with the Kaldi training method
- Learn the ability to use Kaldi to train the ASR system with a customized set-up
- Paper Reading and Source Code reading

**Task Preparation:**

- Prepare an Ubuntu 16 / 18 Environment with at least 8G RAM (other Linux platforms are not tested yet)
- Follow the instruction on http://kaldi-asr.org/doc/ to install Kaldi
- TIMIT dataset (please request a copy from your teacher)
- CALL_2K dataset (please request a copy from your teacher)
- Librispeech dev-clean and test-clean dataset (they can be downloaded from http://www.openslr.org/12)

## TASKS:

## TASK 1: (About 3 hours)

From an overall view, the traditional[1] speech recognition system aims to convert an acoustic signal captured by a microphone or telephone to a sequence of words as shown in **Figure 1**.

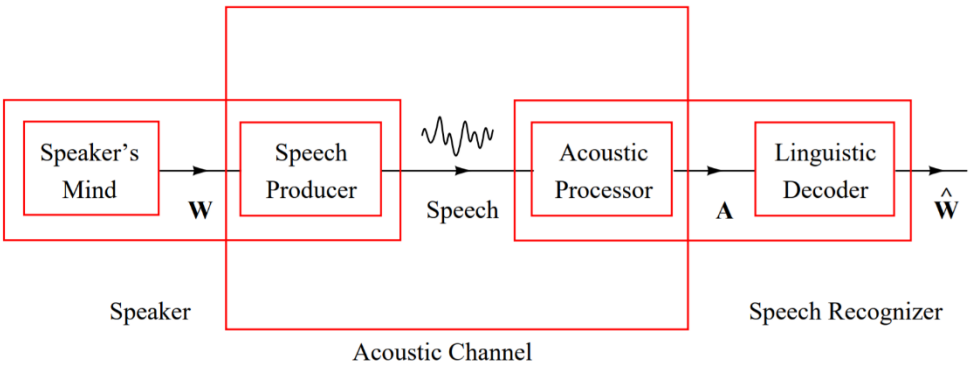


**Figure 1 A Framework of Speech Recognition[2]**

[1] Yes, there are speech recognition models other the traditional ones. Recently, several experiments have shown that end2end ASR system with neural networks can perform better than traditional ones on a large dataset.

[2] The figure and the following formula are from an open talk on "The Basic Mathematics of Automatic Speech Recognition" by Sanjeev Khudanpur, Center for Language and Speech Processing, Johns Hopkins University

Through the figure, we can get that a speech recognition process is like:

$$\widehat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\operatorname{argmax}} P(\boldsymbol{W}|\boldsymbol{A}) = \underset{\boldsymbol{W}}{\operatorname{argmax}} \frac{P(\boldsymbol{A}|\boldsymbol{W})P(\boldsymbol{W})}{P(\boldsymbol{A})} \quad (1)$$

Since $P(\boldsymbol{A})$ is constant for a given speech, we only need to maximize the numerator. Therefore, we can rewrite the above formula into:

$$\widehat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\operatorname{argmax}} P(\boldsymbol{W}|\boldsymbol{A}) = \underbrace{\underset{\boldsymbol{W}}{\operatorname{argmax}}}_{3} \underbrace{P(\boldsymbol{A}|\boldsymbol{W})}_{2} \underbrace{P(\boldsymbol{W})}_{1} \quad (2)$$

Empirically, we can divide the whole process into three sub-processes.

- Language Model: analyze the probability for each word from the language.
- Acoustic Model: analyze the probability for acoustic realizations of a sequence of words.
- Decoding Model: search the word sequence given a language model and an acoustic model.

After decades of development, each sub-process has gotten a classical solution. For language model, N-gram methods is proposed with Markov Assumptions. Hidden Markov Model becomes a benchmark for acoustic model. Finite State Automata theory is adopted to maintain a fast decoding model. Comparing to the other two, HMM-based acoustic model contributes the most to current great progress on ASR and it is generally acknowledged as the most difficult part to understand. It is also the focus of our homework😀

All the homework is planned on Kaldi. It is the most prevail open-source speech-related toolkits for either the academics or the industries[3].

The first task aims to help you learn the basics of Kaldi and tries to teach you how to train a simple ASR system with the TIMIT. Kaldi has left us a runnable code with the TIMIT. In order to use it, we first need to create a personal environment. A runnable workplace should contain 8 elements, including a configuration directory (*conf*), a data directory (*data*), an experimental directory (*exp*), a local directory for some specific scripts of customized dataset (*local*), a feature directory (literally *mfcc*), a path declaration (*path.sh*), and two general libraries from kaldi egs (*steps* & *utils*).

Some of them can be directly copied from Installed Kaldi (the related directory is *your_installation_place/kaldi/egs/timit/s5*), such as *conf*, *local*, *path.sh*, *steps* and *utils*. But for other files, you will have to create on your own.

A few remarks for the files are as follows

1. change the “path.sh” based on your workspace
2. we recommend you use soft links for “steps” and “utils” as Kaldi does (learn about “ln -s” to form soft links)
3. After setting up with your workspace, you can move your TIMIT data into the data directory *data*.（refer to local/timit_data_prep.sh）

To start training process, you need to export Kaldi path to your path. Since the path is somewhat temporary (especially when you are working on multi-tasks), a better way is to export the path is to use “source path.sh” and “. path.sh” rather than add the kaldi into your PATH.

[3] As far as I know, most the company working on speech-related stuff will take insights from Kaldi or directly use it as the core of their speech products (e.g. Google, Baidu, Tencent, Mitsubishi… more than you can imagine). A great knowledge on it will definitely allows you to gain competitiveness for further work or research.

To help you understand the whole process, please do not directly use the “timit/run.sh” directly. A line by line mode is preferred for further tasks. (You can also create a new .sh file for your own code. But remember change the file permission with chmod is necessary

## **STEP1:** Prepare

Go to your workspace (we use “/home/user/workspace” to stand for it in the follows) and prepare for the task

**Code for STEP 1:**

```
cd /home/usr/workspace
source path.sh
. path.sh
```

## **STEP2:** prepare for your speech data & lexicon data & language data

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. (you can train a language model on your own with irislm in Kaldi as well)

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, welcome to read the TIMIT description (some docs in the dataset. The data collection process is frustrating, and all the datasets are built with great efforts. Therefore, I greatly recommend you read the docs with respect).

**Code for STEP 2:**

```
timit = your_timit_place
local/timit_data_prep.sh $timit
local/timit_prepare_dict.sh
utils/prepare_lang.sh --position-dependent-phones false --num-sil-states 3 \
data/local/dict "sil" data/local/lang_tmp data/lang
local/timit_format_data.sh
```

## **STEP3:** Feature Extraction

We use default MFCC feature as our feature (you are welcome to try other features such as PLP http://kaldi-asr.org/doc/feat.html )
(you can employ more concurrency jobs if your computer can afford them. Here is a reference for

you to learn how to check the CPU number in your computer, https://www.cyberciti.biz/faq/check-how-many-cpus-are-there-in-linux-system)

**Code for STEP 3:**

```
for x in train dev test; do
  steps/make_mfcc.sh --nj 4 data/$x exp/make_mfcc/$x mfcc
  steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x mfcc
done
```

**Question 1:** what the processes for MFCC in Kaldi are? Read the kaldi source code to find out (http://kaldi-asr.org/doc or https://github.com/kaldi-asr/kaldi). Please answer with **detailed functions in the source C++ code**.

**Extra Credits**: what is the difference between Kaldi and the knowledge you learned from your teacher?

## **STEP4:** Mono-Phone Training & Decoding

This step trains a mono-phone model. Firstly, train the mono acoustic model. Next, we make a decoding graph with current configuration for test. Then, we decode our development and test dataset with the graph. A more detailed explanation is as follows.

An ASR system contains two parts: acoustic modeling and decoding. Acoustic model converts speech information to phonetic feature (sometimes accompanied with prosodic features). Its target is to accurately recognize phonemes and outputs a posteriorgram that represents posterior probabilities for phonemes at each frame. The acoustic model should be dependent on speakers, so speaker adaptation methods are often applied at this stage. The decoding model recognizes features (i.e. posteriorgram) into words, which employ language models. For most cases in ASR, the two processes are not strictly separated, and both are combined into the HCLG where "H" is for the Hidden Markov Model, "C" is for the Context-Dependent phonemes, "L" is for the Lexicon Dictionary, and "G" is for Grammar/Language Model. The HCLG is a Finite State Transducer (you may have heard of it from courses on compliers).[4]

The HMM model derives from the Markov Chain model. The Markov Chain model (for this research, the discrete Markov Chain is applied) is a series of random variables. All the finite random processes can be defined as follows:
Let $\mathbf{X} = X_1, X_2, X_3, \dots, X_n$ as a sequence of n random variables chosen from a finite discrete set $O = \{o_1, o_2, o_3, \dots, o_m\}$. According to the Bayes rule, we have

$$P(X_1, X_2, X_3, \dots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i \mid X_1, X_2, X_3, \dots, X_{i-1}) \quad (3)$$

[4] If you want to learn more about the information about FSTs reads https://cs.nyu.edu/~mohri/pub/hbka.pdf
For a detail discussion, please visit
http://vpanayotov.blogspot.com/2012/06/kaldi-decoding-graph-construction.html

The Markov Chain model is in first-order with the Markov assumption that

$$P(X_i \mid X_1, X_2, X_3, \dots, X_{i-1}) = P(X_i \mid X_{i-1}) \tag{4}$$

Therefore, for the Markov Chain, **Formula (3)** can be rewrite as

$$P(X_1, X_2, X_3, \dots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i \mid X_{i-1}) \tag{5}$$

As the Markov Chain is associated with time-invariant events, the random variable $X_i$ can be represented by finite state $s_i$. Therefore, for a Markov chain with n states, the parameters of it can be summarized as follows:

$$a_{ij} = P(s_i = j \mid s_{i-1} = i) \quad 1 \le i, j \le n \tag{6}$$

$$\pi_i = P(s_1 = i) \quad 1 \le i \le n \tag{7}$$

where $a_{ij}$ is the transition probability from state $i$ to state $j$. And $\pi_i$ is the initial probability for the start of the Markov chain. The sum of $a_{ij}$ and the sum of $\pi_i$ are both 1.

The Markov chain is powerful for building an observable sequence with limited memory cost, but the states in the Markov chain only correspond to deterministically observable output. Therefore, it cannot infer observable symbols from relevant features. To extend the modeling capacity, a non-deterministic process for each state is proposed, which also known as the Hidden Markov Model (HMM). Because of the extension, an HMM has more parameter sets as follows.

$$a_{ij} = P(s_i = j \mid s_{i-1} = i) \quad 1 \le i, j \le n \tag{8}$$

$$\pi_i = P(s_1 = i) \quad 1 \le i \le n \tag{9}$$

$$b_i(k) = P(X_i = o_k \mid s_t = i) \tag{10}$$

Where $b_i(k)$ is an output function that stands for the probability of emitting $o_k$ as in state *i*. The sum of $b_i(k)$ is 1 as well. The set of $a_{ij}$ and $b_i(k)$ can be annotated as **A** and **B**. The model can be sum up to $\Phi(\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ with parameter sets of **A, B**, and $\boldsymbol{\pi}$. A traditional method for the **B** matrix's modeling is to apply Gaussian Mixture Model (GMM) trained with Expectation Maximization (EM) algorithm. Assume the Gaussian Mixture has *M* components. Then the $b_i(k)$ is given by

$$b_i(k) = \sum_{k=1}^{M} w_{ik} b_{ik}(o_t) \tag{11}$$

And for each mixture component, the probability can be given by

$$b_{ik}(o_t) = \frac{1}{2\pi^{\frac{n}{2}} |C_{ik}|^{\frac{1}{2}}} e^{-\frac{1}{2}(o_t - \mu_{ik})^T C_{ik}^{-1} (o_t - \mu_{ik})} \tag{12}$$

where $\mu_{ik}$ denotes the mean of the mixture (n is the size of the output symbol set). $w_{ik}$ is the weight for each mixture. $C_{ik}$ is a covariance matrix and it is set to be diagonal assuming the elements of feature elements are independent[5].

Given an HMM, the probabilities of an output string in $\boldsymbol{O}$ within T speech frames following the state sequence $\theta = < \theta_1, \theta_2, \dots, \theta_T >$ is

$$P(\boldsymbol{O}, \theta) = \pi_{\theta_1} \cdot b_{\theta_1}(o_1) \cdot \prod_{t=2}^{T} a_{\theta_{t-1}\theta_t} \cdot b_{\theta_t}(o_t) \tag{13}$$

The Viterbi Algorithm was always employed to decode the states. It applies dynamic

[5] The main reason for the it is to reduce massive computation cost.

programming when scanning the HMM graph. For each timestamp, the Viterbi algorithm computes probabilities by choosing the optimum previous path. The probability of the Viterbi algorithm at time *t* is

$$V_t(i) = P(X_1^t, S_1^{t-1}, s_t = i \mid \Phi) \tag{14}$$

where $X_1^t$ is the observation till time *t*, the $S_1^{t-1}$ is the previous state sequence. For recursion part, the choosing criteria is

$$V_t(j) = \underset{1 \le i \le N}{\text{Max}}[V_{t-1}(i) \cdot a_{ij}] b_j(X_t) \tag{15}$$

$$B_t(j) = \underset{1 \le i \le N}{Argmax}[V_{t-1}(i) \cdot a_{ij}] \tag{16}$$

Baum-Welch Algorithm is for HMM parameters estimation based on EM. The parameters are iteratively re-estimated, and the process is repeated until the change is accepted by a pre-defined threshold. Given a parameter set $\lambda$ of $\{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$, and $\phi$ as the likelihood function, the target function for Baum-Welch Algorithm can be written as

$$Q(\lambda, \widehat{\lambda}) = \sum_{\theta} \phi(\theta \mid \boldsymbol{O}, \boldsymbol{A}, \boldsymbol{B}) \log(\phi(\theta, \boldsymbol{O} \mid \widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}})) \tag{17}$$

Since the output sequences are modeled with GMM, **Formula (15)** can be written as

$$Q(\lambda, \widehat{\lambda}) = c - \frac{1}{2} \sum_{t=1}^{T} \sum_{m=1}^{M} \gamma_m(t) \left( c_m + \log\left|\widehat{\Sigma_m}\right| + (o_t - \widehat{\mu_m})^T \widehat{\Sigma_m^{-1}} (o_t - \widehat{\mu_m}) \right) \tag{18}$$

where *M* is the number of Gaussian Mixture components. $c_m$ and $c$ are constants in respect to $\lambda$. The $\gamma_m(t)$ denotes the probability of the state in m[th] mixture component at time *t*.

For acoustic modeling, the basic unit is a phoneme. Instead of words that are various, phonemes are accurate, trainable and generalizable for acoustic modeling [1][6]. In a mono-phone system, a phoneme is divided into three states (beginning, processing, and ending). Each state corresponds to an HMM cell. The decoding process is rather complex. If you want to dig more, welcome to learn more in http://kaldi-asr.org/doc/hmm.html .

**Code for STEP4 4:**

```
steps/train_mono.sh --nj 4 data/train data/lang exp/mono
utils/mkgraph.sh data/lang_test_bg exp/mono exp/mono/graph
steps/decode.sh --nj 4 exp/mono/graph data/dev exp/mono/decode_dev
steps/decode.sh --nj 4 exp/mono/graph data/test exp/mono/decode_test
```

## **STEP5:** Tri-phone- Delta + Delta-Delta Training

A mono-phone model (consider each phoneme an individual unit) cannot model the context dependency problem given the Markov assumption. Consequently, the context dependency model was proposed with the triphone [2]. A triphone model considers neighboring phones as a unit other than a single phone. In addition, stress also affects the phonetic feature. Researches have a consensus that stressed phonemes tend to have higher pitches, longer duration and more energy

[6] There are only less than 50 phonemes for English under different criteria, while the words are countless.

comparing to unstressed ones [3]. Therefore, vowels are divided into stressed, unstressed, secondary stressed in the system as well.

Though triphone has a significant effect on the acoustic model, the computation and memory costs are at exponential expenses. Another point is that triphone assumes that the triphone's context is different from each other. However, there are similarities between the effect of neighboring phonemes. Therefore, Huang proposed clustering methods based on linguistic questions [4]. Through asking questions on linguistic features (such as nasal, sonorant or voiced), triphones are compressed into clusters with decision tree. The cluster from tri-phonetic events names senone [5]. A simple structure of HMM with senone is defined in **Figure 2**. There are two sub-HMM models. After clustering, the first two phonemes of both HMM are clustered to the same senone while the last two are different because of changes in context. By adjusting the decision tree for clustering, the number of senones can be modified to a specific range (around thousands) where allow us to balance the efficiency and accuracy.

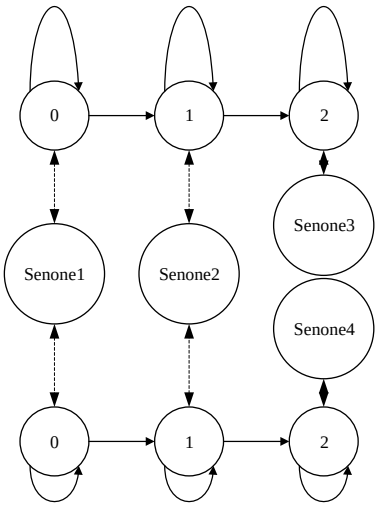


**Figure 2 An HMM Structure with Senones**

**Code for STEP 5:**

```
steps/align_si.sh --boost-silence 1.25 --nj 4 \
data/train data/lang exp/mono exp/mono_ali
# Train tri1, which is deltas + delta-deltas, on train data.
steps/train_deltas.sh \
 2500 15000 data/train data/lang exp/mono_ali exp/tri1
utils/mkgraph.sh data/lang_test_bg exp/tri1 exp/tri1/graph
steps/decode.sh --nj 4 exp/tri1/graph data/dev exp/tri1/decode_dev
steps/decode.sh --nj 4 exp/tri1/graph data/test exp/tri1/decode_test
```

**Question 2: A**s the tri-phone model was proved to be more powerful than mono-phone model, why don't we directly run a tri-phone model? What about delta + delta-delta features?

**Question 3:** How is the WER (word error rate) of your model up-to-now? You can find related code for showing your model performance from "/kaldi/egs/timit/s5/RESULTS". (Warning: do not directly paste the results in the file to here)

**References:**

[1] Huang, X., Acero, A., Hon, H. W., & Reddy, R. (2001). Spoken language processing: A guide to theory, algorithm, and system development (Vol. 1). Upper Saddle River: Prentice hall PTR.

[2] Deng, L., Lennig, M., Seitz, F., & Mermelstein, P. (1990). Large vocabulary word recognition using context-dependent allophonic hidden Markov models. Computer Speech & Language, 4(4), 345-357.

[3] Tamburini, F. (2003). Automatic prosodic prominence detection in speech using acoustic features: an unsupervised system. In Eighth European Conference on Speech Communication and Technology.

[4] Huang, X., Acero, A., Alleva, F., Hwang, M., Jiang, L., & Mahajan, M. (1996). From Sphinx-II to whisper—making speech recognition usable. In Automatic Speech and Speaker Recognition (pp. 481-508). Springer, Boston, MA.

[5] Hon, H. W., & Lee, K. F. (1991, April). CMU robust vocabulary-independent speech recognition system. In [Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing (pp. 889-892). IEEE.

# TASK 2: (About 1 hours)

## STEP1: MLLR

In this step, we will dig into some basic speaker adaptation techniques. MLLR is a baseline adaptation model that introduced in [1] and developed in [2]. It re-estimates GMM's parameters with linear transformation given a speaker independent acoustic model. There are two forms of MLLR, known as unconstrained MLLR and constrained MLLR (cMLLR, also known as fMLLR). And both of their estimation processes apply the EM algorithm. Experiments had shown that there is not a superior method comparing MLLR and fMLLR [3]. However, for large speech corpora, fMLLR simplifies the training process and thus has a better runtime performance.

When taking the speaker effect for speakers $\boldsymbol{R} = < s_1, \dots, s_R >$ into the HMM training process, the $\widehat{\mu_m}$ and $\widehat{\Sigma_m}$, are

$$\widehat{\mu_m} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_{rm}(t)o_{rt}}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_{rm}(t)} \tag{19}$$

$$\widehat{\Sigma_m} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_{rm}(t)(o_{rt} - \widehat{\mu_m})(o_{rt} - \widehat{\mu_m})^T}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_{rm}(t)} \tag{20}$$

where $\gamma_{rm}(t)$ denotes the speaker $r$'s probability of the state at time $t$ for GMM mixture $m$.

For MLLR, $\widehat{\mu_m}$ and $\widehat{\Sigma_m}$ are adapted as **Formula (21)** and **Formula (22)**:

$$\hat{\mu} = \boldsymbol{W}\mu + \boldsymbol{b} \tag{21}$$

$$\hat{\Sigma} = \boldsymbol{B}\Sigma\boldsymbol{B}^T \tag{22}$$

For fMLLR, $\widehat{\mu_m}$ and $\widehat{\Sigma_m}$ are changed as **Formula (23)** and **Formula (24)** where the transformation matrix is constrained as the same $\boldsymbol{A}$:

$$\hat{\mu} = \boldsymbol{A}\mu + \boldsymbol{b} \tag{23}$$

$$\hat{\Sigma} = \boldsymbol{A}\Sigma\boldsymbol{A}^T \tag{24}$$

Speaker Adaptive Training (SAT) can adapt to speaker variations based on the fMLLR. It focuses on speaker-dependent transforms. Another advantage of fMLLR is its efficiency in Speaker Adaptation Training (SAT). Comparing to MLLR, fMLLR can be fitted into SAT procedures with minimum changes [2].

The fMLLR is effective for the GMM models, but it is not useable to the DNN structure. In the GMM, means and variances have statistical meanings and can be transformed together within the model. Unlike the GMM's parameters, the weight factors in DNN have no well-formed structure for the linear transformation. Therefore, when discussing the DNN methods, traditional fMLLR cannot work. There are researches that studied similar strategies as fMLLR in a DNN structure [4], but it is trained under Cross Entropy criterion other than Maximum Likelihood. Another substitute method is to apply fMLLR that estimates with GMM-HMMs to get speaker adapted features. However, it has to be admitted that fMLLR are trained assuming with the GMM-HMMs other than DNN-HMM. The adapted fMLLR features are not sure to be suitable for the DNN. To put forward a "DNN-like" speaker adaptation method, Saon et al. proposed an I-vector method that extracts speaker information through EM processes [5]. I-vector method is a popular technique for studies in speaker recognition or verification, which aims to find a linear dependence from

Universal Background Model (UBM, The UBM is a GMM trained with speaker independent audio wave. Therefore, it can be considered as speaker independent and a useful verification tool to verify whether the feature is speaker dependent) to speaker dependent distribution. The estimated I-vectors are cascaded after basic MFCC features in the DNN input feature.

In Kaldi, the MLLR is implemented with a simplified version of MLLT (Maximum Likelihood Linear Transform). Explanation of it can be found in http://kaldi-asr.org/doc/transform.html

**Question 1**: Please prove the fact that fMLLR only conducts feature space transform and leaves the model parameters still.

**Extra Credits**: The primary use of I-vector is for speaker identification. Read the paper [6], think about why speaker identification tasks only use mean-only adaptation other than consider variance and GMM weights?

**Code for STEP1 1:**

```
steps/align_si.sh --nj 4 data/train data/lang exp/tri1 exp/tri1_ali
steps/train_lda_mllt.sh --splice-opts "--left-context=3 --right-context=3" \
 2500 15000 data/train data/lang exp/tri1_ali exp/tri2
utils/mkgraph.sh data/lang_test_bg exp/tri2 exp/tri2/graph
steps/decode.sh --nj 4 exp/tri2/graph data/dev exp/tri2/decode_dev
steps/decode.sh --nj 4 exp/tri2/graph data/test exp/tri2/decode_test
# Align tri2 system with train data.
steps/align_si.sh  --nj  4  --use-graphs  true  data/train  data/lang  exp/tri2
exp/tri2_ali
# From tri2 system, train tri3 which is LDA + MLLT + SAT.
steps/train_sat.sh 2500 15000 data/train data/lang exp/tri2_ali exp/tri3
utils/mkgraph.sh data/lang_test_bg exp/tri3 exp/tri3/graph

steps/decode_fmllr.sh --nj 4 exp/tri3/graph data/dev exp/tri3/decode_dev
steps/decode_fmllr.sh --nj 4 exp/tri3/graph data/test exp/tri3/decode_test
```

## **STEP2:** Extension-Reading Subspace Gaussian Mixture Model (SGMM)

The Subspace Gaussian Mixture Model (SGMM) is also a GMM method that asks all HMM states share the same GMM structure with the same number of Gaussians in each state. A more detailed version can be found in a paper from Daniel Povey, the main developer of the Kaldi (https://www.danielpovey.com/files/csl10_sgmm_preprint.pdf )

**Code for STEP 2:**

```
steps/align_fmllr.sh --nj 4 data/train data/lang exp/tri3 exp/tri3_ali
steps/train_ubm.sh 400 data/train data/lang exp/tri3_ali exp/ubm4
steps/train_sgmm2.sh 7000 9000 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 4 --transform-dir exp/tri3/decode_dev \
 exp/sgmm2_4/graph data/dev exp/sgmm2_4/decode_dev
steps/decode_sgmm2.sh --nj 4 --transform-dir exp/tri3/decode_test \
 exp/sgmm2_4/graph data/test exp/sgmm2_4/decode_test
```

**Question 2:** How is the WER (word error rate) of your model up-to-now?

**References:**


[1] Leggetter, C. J., & Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer speech & language*, *9*(2), 171-185.

[2] Gales, M. J. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, *12*(2), 75-98.

[3] Ganitkevitch, J. (2005, August). Speaker adaptation using maximum likelihood linear regression. In *Rheinish-Westflesche Technische Hochschule Aachen, the course of Automatic Speech Recognition, www-i6. informatik. rwthaachen. de/web/Teaching/Seminars/SS05/ASR/Juri Ganitkevitch Ausarbeitung. pdf*.

[4] Seide, F., Li, G., Chen, X., & Yu, D. (2011, December). Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding* (pp. 24-29). IEEE.

[5] Saon, G., Soltau, H., Nahamoo, D., & Picheny, M. (2013, December). Speaker adaptation of neural network acoustic models using i-vectors. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding* (pp. 55-59). IEEE.

[6] Campbell, W. M., Sturim, D. E., Reynolds, D. A., & Solomonoff, A. (2006, May). SVM based speaker verification using a GMM supervector kernel and NAP variability compensation. In 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings (Vol. 1, pp. I-I). IEEE.

## TASK 3: Deep Neural Network (About 6 hours (training takes up most of the time))

Deep learning has been very popular in recent years. Therefore, we assume that you have learn some of it. A basic framework of DNN-HMM for acoustic modeling is shown in **Figure 3**. Based on the senones and the deep structure, it can substitute the GMM part in the traditional architecture which brings much improvement with a minimum-modified decoding process [1].

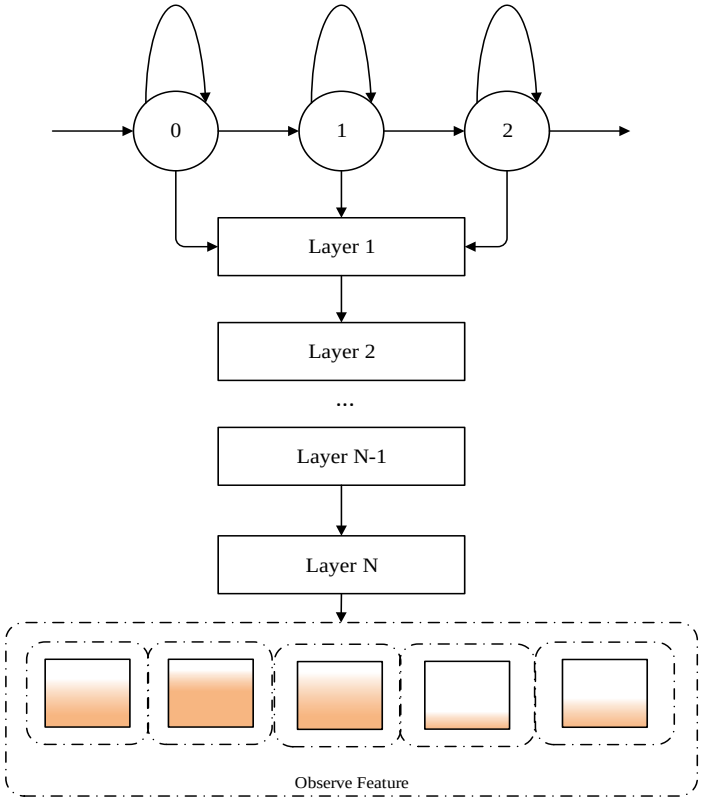


**Figure 3 A DNN-HMM Structure**

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), Minimum Phone Error (MPE), boosted MMI (BMMI) and Minimum Bayesian Risk (MBR). According to the report from [2], the introduction of the criteria can offer an improvement of 1.5% to 2% comparing to DNN with CE loss. Traditional discriminative training processes require lattices generated from a preliminary model such as GMM-HMMs and DNN-HMMs with CE loss [3]. The lattices are used to provide a simple approximation for possible phoneme sequences (or word sequences) which can limit the computation cost to a controllable range. A typical MMI loss can be computed as follows.

The $\boldsymbol{o}^m = < \boldsymbol{o}_1^m, \boldsymbol{o}_2^m, \dots, \boldsymbol{o}_{T_m}^m >$ is defined as the observed sequence of the m$^{th}$ speech where $T_m$

is the frame number, and the $\boldsymbol{w}^m = < \boldsymbol{w}_1^m, \boldsymbol{w}_2^m, \ldots, \boldsymbol{w}_{N_m}^m >$ is defined as words' caption of the m[th] speech where $N_m$ is the number of words. For the whole training set that has $M$ speech samples (denoted as $\mathbf{S}$), the MMI is

$$\mathcal{L}_{MMI}(\theta; \boldsymbol{S}) = \sum_{m=1}^{M} \mathcal{L}_{MMI}(\theta; \boldsymbol{o}^m, \boldsymbol{w}^m)$$

$$= \sum_{m=1}^{M} logP(\boldsymbol{w}^m \mid \theta; \boldsymbol{o}^m)$$

$$= \sum_{m=1}^{M} log\left(\frac{p(\boldsymbol{o}^m|\boldsymbol{s}^m; \theta)^{\kappa} P(\boldsymbol{w}^m)}{\sum_{\boldsymbol{w}} p(\boldsymbol{o}^m|\boldsymbol{s}^m; \theta)^{\kappa} P(\boldsymbol{w})}\right) \quad (25)$$

where $\theta$ is the parameter set for the model (i.e. DNN) and $\boldsymbol{s}^m$ is for states in the HMM. $\kappa$ is a hyperparameter as the acoustic scaling factor.

According to chain rules, the derivative of $\mathcal{L}_{MMI}(\theta; \boldsymbol{S})$ is as **Formula (26)**:

$$\nabla_{\theta} \mathcal{L}_{MMI}(\theta; \boldsymbol{S}) = \sum_{m=1}^{M} \sum_{t=1}^{T} \nabla_{z_{mt}} \mathcal{L}_{MMI}(\theta; \boldsymbol{S}) \cdot \frac{\partial z_{mt}}{\partial \theta}$$

$$= \sum_{m=1}^{M} \sum_{t=1}^{T} \left( \kappa \left( \frac{\alpha_t^{num}(\boldsymbol{r}) \beta_t^{num}(\boldsymbol{r})}{\sum_r^N \alpha_t^{num}(\boldsymbol{r})} - \frac{\alpha_t^{de}(\boldsymbol{r}) \beta_t^{de}(\boldsymbol{r})}{\sum_r^N \alpha_t^{de}(\boldsymbol{r})} \right) \right) \cdot \frac{\partial z_{mt}}{\partial \theta} \quad (26)$$

where $z_{mt}$ is the input of the final Softmax layer. The computation of $\frac{\partial z_{mt}}{\partial \theta}$ is the same as CE. $\boldsymbol{r}$ represents the state sequence. $\frac{\alpha_t^{num}(\boldsymbol{r}) \beta_t^{num}(\boldsymbol{r})}{\sum_r^N \alpha_t^{num}(\boldsymbol{r})}$ denotes the posterior probability vector for $\boldsymbol{r}$[7]. It is computed via a forward-backward algorithm (like the Baum-Welch algorithm discussed above) on the numerator lattice graph and the denominator lattice graph. The traditional MMI discriminate training must apply the lattice, otherwise, the computation cost is huge. However, the lattice also introduces some losses in accuracy, and it is still time-consuming. The framework is shown in **Figure 4**.

[7] The Formula 26 is simplified for interpretation. A full version can be found in [3].

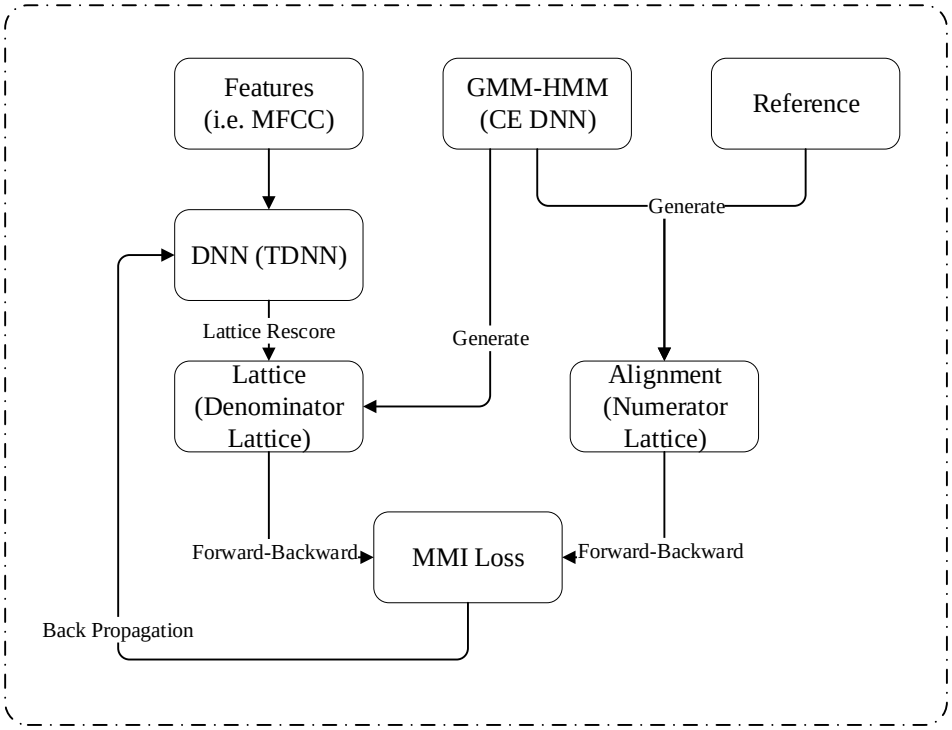


**Figure 4 The MMI Training Process**

**Question 1:** Another popular loss is CTC (Connectionist Temporal Classification). What are the weaknesses of CTC compared to discriminative training process?

**Question 2:** What are the differences between maximum likelihood and discriminative training?

**COMPETITION:** there are several neural network recipes in Kaldi. Tune hyperparameters for different networks (time delay neural networks, deep neural networks, recurrent neural networks and chain model) and achieve a best result on TIMIT~

Reference:

[1] Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, *20*(1), 30-42.

[2] Veselý, K., Ghoshal, A., Burget, L., & Povey, D. (2013, August). Sequence-discriminative training of deep neural networks. In *Interspeech* (Vol. 2013, pp. 2345-2349).

[3] Povey, D. (2005). *Discriminative training for large vocabulary speech recognition* (Doctoral dissertation, University of Cambridge).

## Task 4: Train a new dataset on your own (About 4 hours)

After working so much, let's train an entirely new dataset on your own.

1) Create a new workspace
2) train a tri-phone model with a dataset that combining CALL_2K dataset and Librispeech dev-clean dataset (add some self-recorded data is also welcome).
3) Decode the model with Librispeech dev-test dataset.

Report the WER to your teacher~