Combining Speech and Speaker Recognition - A Joint Modeling Approach

by

Hang Su

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Nelson Morgan, Chair
Doctor Steven Wegmann
Professor Keith Johnson
Professor Jerome Feldman

Summer 2018

Combining Speech and Speaker Recognition - A Joint Modeling Approach

Abstract

Combining Speech and Speaker Recognition - A Joint Modeling Approach

by

Hang Su
Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley
Professor Nelson Morgan, Chair

Automatic speech recognition (ASR) and speaker recognition (SRE) are two important fields of research in speech technology. Over the years, many efforts have been made on improving recognition accuracies on both tasks, and many different technologies have been developed. Given the close relationship between these two tasks, researchers have proposed different ways to introduce techniques developed for these tasks to each other.

In the first half of this thesis, I explore ways to improve speaker recognition performance using state-of-the-art speech recognition acoustic models, and then research alternative ways to perform speaker adaptation of deep learning models for ASR using speaker identity vector (i-vector). Experiments from this work shows that ASR and SRE are beneficial to each other and can be used to improve their performance.

In the second part of the thesis, I aim to build joint model for speech and speaker recognition. To implement this idea, I first build an open-source experimental framework, TIK, that connects well-known deep learning toolkit Tensorflow and speech recognition toolkit Kaldi. After reproducing state-of-the-art speech and speaker recognition performance using TIK, I then developed a unified model, JointDNN, that is trained jointly for speech and speaker recognition. Experimental results show that the joint model can effectively perform ASR and SRE tasks. In particular, experiments show that the JointDNN model is more effective in speaker recognition than x-vector system, given a limited amount of training data.

To my parents and my girlfriend.

Contents

List of Figures

List of Tables

Acknowledgments

This section of my thesis probably took me the longest to finish writing. When I look back on my PhD studies, I feel so blessed to have a great many people helping me along the way.

I am honored to have two advisors in ICSI during my PhD studies. My advisor, Nelson Morgan, is a gentleman with all virtues I would expect from a PhD mentor. He is very nice and open, and allows me to explore my research interests the way I like. My second advisor, Steven Wegmann, is also very nice and he shares a great many merits with Morgan. His background in mathematics and engineering makes him both theoretical and practical, and has the ability to grisp the in-depth nature of problems. Morgan and Steven are both great mentors and have been fun to work with.

Apart from my official advisors, I also received a great deal of help from other mentors. Here, I would like to thank Haihua Xu, who guided me greatly during my first year of PhD. His extensive knowledge and experience in developing speech recognition systems let me realize the importance of mastering the tools and infrastructure for system building. I feel very lucky to collaborate with him on a variety of ASR projects. Also, I would like to thank Jim Hieronymus, who guided me through the first paper in my PhD studies and instructed me on linguistic knowledges. Also I would like to thank Frank Seide, who enrolled me twice as a research intern and introduced the whole field of speech recognition to me.

Thanks to Arlo, Shuoyiin, Suman, and TJ Tsai, for setting role models for me to learn from. Every time when I felt frustrated or anxious, I would look up their works and learn from them, and gradually gain peace of mind. Arlo and Shuoyiin are extremely helpful when I consult them regarding PhD studies. Adam and Chuck, together with Arlo, also helped me greatly during my PhD, and they gave me many useful suggestions on the dissertation talk.

Thanks to Misa Zhu and Frank Rao, who introduced me to Rokid Alab / Rlab. The easy access to massive parallel computing resources allowed me to conduct extensive experiments, without which I could never finish what I have planned to do. During my time there, Aiden Fang and Lichun Fan also helped me a great deal setting up the infrastructure. Discussions with Chenghui and Yuanhang on speaker recognition related topics are both enlightening and beneficial. Thanks to all members in Rokid Alab and Rlab - the time I spent there are memorable and priceless.

Lastly, and certainly not least, I would like to thank my parents, Fengquan Su and Xiangge Liu, and my girlfriend, Qing Zhao, for their unconditional support and deep love. I could not imagine how I would accomplish this without them standing by my side and keep me motivated. I will always cherish the whole process.

Chapter 1

Introduction

Automatic Speech Recognition (ASR) and Speaker Recognition (SRE) are two basic tasks in the speech community. Both of the tasks are of great interest in the speech community, and scholars have conducted extensive research on them. The focus of this thesis is to explore connections between these two tasks and to seek ways to improve one's performance with the help of the other. Generally speaking, ASR and SRE share many similarities in that they are dealing with same kind of data and they use similar statistical / deep learning models. Therefore, much research has been done on introducing techniques developed for them to each other. These include speaker adaptation, speaker adaptive training and universal background model for SRE.

Most previous work on this topic focuses on improving evaluation performance on one of the two tasks. Speaker adaptation and speaker adaptive training are techniques for developing speaker-dependent acoustic models so that better ASR performance could be achieved. Some other research exploits ways to improve SRE performance using acoustic models trained for ASR tasks.

The work presented in this thesis consists of two parts. First, I show that ASR and SRE are beneficial to each other. This part extends existing research on introducing ASR models to SRE tasks and speaker adaptation using i-vectors. I will first show that ASR models trained with better speech features can be used to improve baseline SRE models. Then I show that speaker-specific information is useful for acoustic modeling in ASR with a proper regularization method.

The second part of the work is on building a joint model for speech and speaker recognition. To achieve this, I first build an experimental framework on top of Tensorflow and show that state-of-the-art ASR and SRE performance can be reproduced. Then I develop a neural network joint model that is trained in a multi-task way, and show that this model can perform both ASR and SRE tasks with reasonable accuracies. I further explore ways to fine-tune the model so that better performances could be achieved. These two parts together form the main contribution of the thesis, and the descriptions of each chapter are as below:

Chapter 2 provides introductions to automatic speech recognition and speaker recognition. Both traditional and state-of-the-art deep learning approaches are covered, together with common data sets and evaluation metrics.

Chapter 3 first introduces the theory of factor analysis for speaker recognition and then describes research on combining neural-network and i-vector framework for speaker recognition using ASR.

Chapter 4 proposes to use i-vector for neural network adaptation, and introduces a new way of regularization to mitigate the overtraining problem. Experiments show that speaker vectors can be helpful for improving ASR performance.

Chapter 5 describes an experimental framework, TIK, that is designed for speech and speaker recognition. This framework connects the existing deep learning toolkit Tensorflow and the ASR toolkit Kaldi, and supports flexible deep learning models for ASR and speaker recognition. State-of-the-art experimental results are presented and multi-GPU training is covered.

Chapter 6 then proposes a joint-model for speech and speaker, JointDNN, that is trained using multi-task learning. Experiments on speech and speaker recognition are presented and the results are compared with baseline systems. It is shown that this JointDNN model is effective in utilizing limited amount of training data for ASR and speaker recognition.

Chapter 7 concludes the thesis and provide some ideas on possible future work.

Chapter 2

Background

Automatic speech recognition (ASR) and speaker recognition are two major fields of research in speech technology. Over the years, much effort has been made to improve recognition accuracies on both tasks, and many different technologies have been developed. In this chapter, I will give some requisite details of both tasks, including basic modeling approaches, state-of-the-art research directions, typical data sets, and evaluation metrics.

2.1 Automatic Speech Recognition

Automatic speech recognition is an artificial intelligence task that requires translation of spoken language into text automatically. The first ASR system probably dates back to the 1950s when Bell Labs researchers built a system for single-speaker digit recognition. Since then, several waves of innovation have taken place in this area, which led to great progress in both research and industry. After all these years, the focus of research has gone from single speaker to speaker independent systems, and also from isolated word system to large vocabulary continuous speech recognition. With the progress made in this area, people are able to build systems that produce reasonable recognition results for clean conversational speech, with word error rate as low as 5.8%-11% [125], achieving human parity. Though these performance results are encouraging, they do not mean that the problem has been solved: the best-performing system may not be applicable for real life application because of computation or real-time requirement. Also, standard datasets on which state-of-the-art performances are achieved often do not represent speech collected in real life scenarios, such as common acoustic environments in restaurants or cars. Both of these indicates that more efforts are needed to bring current technologies to a higher level.

In this work, I will focus on continuous-time speaker-independent speech recognition, especially for a large vocabulary task. Also I will restrict my experiments to use standard conversational speech datasets.

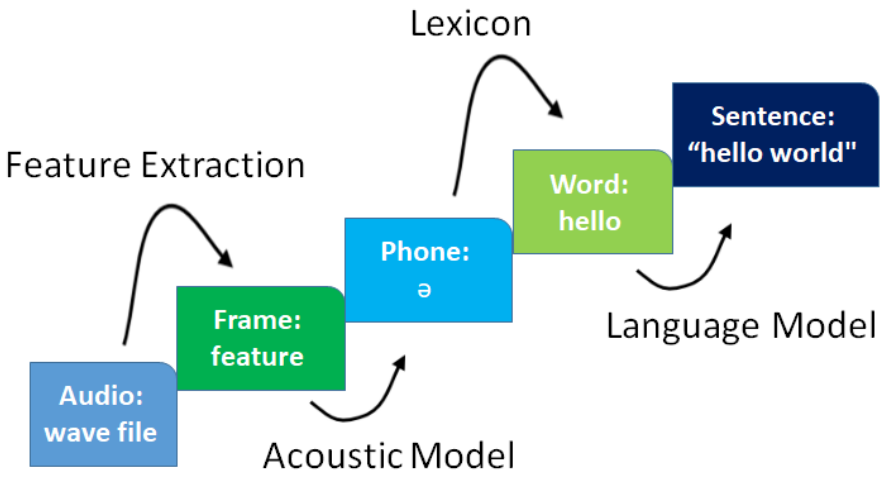

Figure 2.1. Decoding pipeline for ASR

2.1.1 Introduction

Just as the way humans learn to understand speech, building a speech recognizer require extensive amount of speech data called training data. By exploring patterns and characteristics of these training data, we build models that can learn from training data to perform transcriptions of unseen data (called test data). The former part of the process is called *training*, and the latter part is called *testing* (or *decoding*).

In a machine learning approach for speech recognition, models are essentially parameters to be estimated from the training data. Estimation of these parameters could be done by setting an objective function over the training data and optimizing the target. For testing, we perform recognition of test data using the trained model, and compare the outputs with ground-truth transcriptions to get an evaluation metric.

In general, a typical speech recognition system consists of feature extraction, acoustic model, lexicon and language model. Each of these modules could be built separately and then integrated together to perform speech recognition (decoding). Figure 2.1 shows the decoding pipeline for such a system.

From a statistical point of view, ASR is the task of finding the most probable sequence of words W that matches our observation O. Following formula using Bayes rule, we have

$$P(W|O) = \frac{p(O|W)P(W)}{p(O)} \tag{2.1}$$

which breaks our model into two parts: acoustic model $p(O|W)$ and language model $P(W)$. Here $P(W)$ is the probability of the word sequence W, and $p(O|W)$ is the value of the posterior probability density function (p.d.f). Thus, the task of ASR

becomes

$$\arg\max_W \ P(W|O) = \arg\max_W p(O|W)P(W) \tag{2.2}$$

assuming $p(O)$ being constant. We could further decompose the first part by introducing a sequnce of phones $\mathcal{Q}$ as hidden variables:

$$\arg\max_W \ P(W|O) = \arg\max_W \ \sum_Q p(O|\mathcal{Q})P(\mathcal{Q}|W)P(W) \tag{2.3}$$

and these three components correspond to acoustic model, lexicon and language modeling respectively.

Though the focus of this work is on acoustic models $p(O|\mathcal{Q})$, we will first cover the other three components before introducing acoustic modeling techniques.

2.1.2 Feature Extraction

Feature extraction itself is a broad research area in the speech community. Basic approaches in this area target the extraction of frequency information from raw wave files so that the speech recognizer will be less influenced by the presence of noise or reverberaation. Some approaches also consider extracting temporal information from speech. For better performance of ASR systems, several kinds of features have been proposed, including filterbank features, mel-scaled cepstral coefficients (MFCC) [23], perceptual linear prediction (PLP) [46] and Power-normalized cepstral coefficients (PNCC) [58]. Of all these proposed features, MFCC is probably the most widely-used feature in real applications.

A standard feature extractor generally processes segments of speech every 10 ms using a sliding window of 25 ms. These segments (called frames) are then converted into feature vectors using signal processing techniques. The feature vectors X are concatenated and used to represent our observation O, so that the acoustic model becomes $p(X|\mathcal{Q})$.

In this dissertation, MFCCs will be used as default features for all experiments.

2.1.3 Lexicon

A lexicon, also called a dictionary, is a map from written words to their pronunciations in phones. This component helps to bridge the gap between words in written form and their acoustics. The use of a lexicon makes it possible to recognize words that do not appear in the training set. Even for words that do not appear in the lexicon, we can use another technique called "grapheme to phoneme" (G2P) to predict pronunciation for them using their written form.

A phones is a speech segment that possesses distinct physical or perceptual properties [122] and serves as the basic unit of phonetic speech analysis. For a typical spoken language, number of phones may range from 40 to a few hundreds. Phones

are generally either vowels or consonants, and they may last 5 to 15 frames (50 to 150ms).

During testing, the lexicon is used to restrict the search path so that phone strings are valid, and to convert phone strings to words.

2.1.4 Language Modeling

A language model (LM) is a probability distribution over sequences of words, $P(W)$, which may be decomposed as

$$P(W) = P(w_1, w_2, ..., w_n) = P(w_1)P(w_2|w_1) \cdots P(w_n|w_1, \ldots, w_{n-1}) \tag{2.4}$$

And each of these probabilities could be estimated using maximum likelihood methods by checking counts of word sequences $c(w_1, \ldots, w_n)$ in the training corpus.

$$P(w_n|w_1, \ldots, w_{n-1}) = \frac{c(w_1, w_2, \ldots, w_n)}{c(w_1, w_2, ..., w_{n-1})} \tag{2.5}$$

Due to sparsity of training data, typical language modles for ASR make a k-order Markov assumption, i.e.

$$P(w_n|w_1, \ldots, w_{n-1}) = P(w_n|w_{n-k}, \ldots, w_{n-1}) \tag{2.6}$$

In real ASR applications, $k = 2$ and $k = 3$ are the most common settings, and corresponding LMs are called "bigram model" and "trigram model" respectively.

To make sure the recognition outputs may contain n-grams unseen in the training data, many smoothing techniques are developed [17, 59]. Of all these methods, Kneser-Ney (KN) smoothing usually works particularly well on ASR task. In this dissertation, I will also use KN smoothing as my default setup.

2.1.5 Acoustic Modeling

Acoustic modeling is a critical componentfor speech recognition systems. Simply put, acoustic modeling is the task of building statistical models or deep learning models to estimate output likelihoods $p(X|Q)$. Over the years, roughly three generations of acoustic modeling methods have been proposed for continuous speech recognition. In a generative approach, the joint distribution of X and Q, $p(X, Q)$ is modeled. In a discriminative approach, posterior distribution $p(Q|X)$ is directly modeled. In a hybrid approach, both a generative model and a discriminative model are used. We will introduce them in turn in the following subsections.

2.1.5.1 GMM-HMM - A Generative approach

The Hidden Markov Model for continuous time speech recognition dates back to mid-1970s [8, 51]. Later in 1985, Gaussian Mixture Models were introduced to generate HMM output distributions [86]. In this approach, HMMs are used for transition modeling and GMMs are used for frame classification. The acoustic model is thus broken into two parts under the Markov assumption:

$$\begin{aligned} p(X|\mathcal{Q}) &= \sum_{S} p(X|S)P(S|\mathcal{Q}) \\ &= \sum_{S}\prod_{t} p(x_t|s_t)P(s_t|s_{t-1},\mathcal{Q}) \end{aligned} \tag{2.7}$$

where S is a sequence of sub-phone units, x_t and s_t are frame feature and sub-phone unit at frame t.

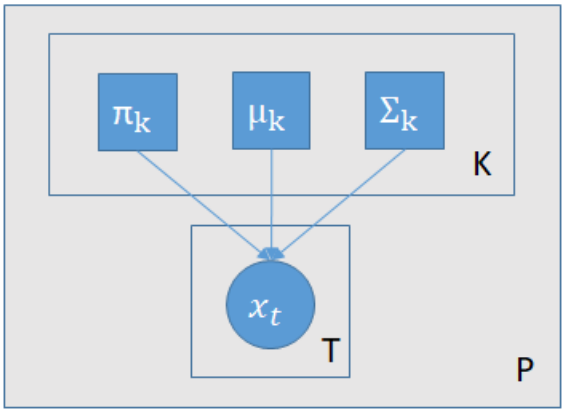

Figure 2.2. Graphical Model for GMM

The first part, $p(x_t|s_t)$, is modeled by Gaussian Mixture Models (GMM):

$$x_t \sim \sum_{k} \pi_k \, \mathcal{N}(\mu_k, \Sigma_k) \tag{2.8}$$

where x_t denotes p-dim speech feature for frame t, π_k is weight for mixture k, μ_k and Σ_k are mean and variance for multivariate Gaussian distribution. Here it is required that $0 \leq \pi_k \leq 1$, and $\sum_k \pi_k = 1$.

Parameters of GMMs are estimated using maximum likelihood estimation (MLE). For GMM, the log-likelihood function could be written as:

$$\log p(X|\theta) = \log \prod_{t} p(x_t|\theta) = \sum_{t} \log \Big(\sum_{k} \pi_k p(x_t|\mu_k, \Sigma_k) \Big) \tag{2.9}$$

where $\theta = \{\pi, \mu, \Sigma\}$ denotes model parameters. This function is difficult to optimize because it contains the log of a summation term. This leads to the use of expectation maximization (EM) algorithm where we introduce a hidden variable c_t that indicates which mixture "generated" each data sample x_i. Figure 2.3 shows the updated GMM graphical model.

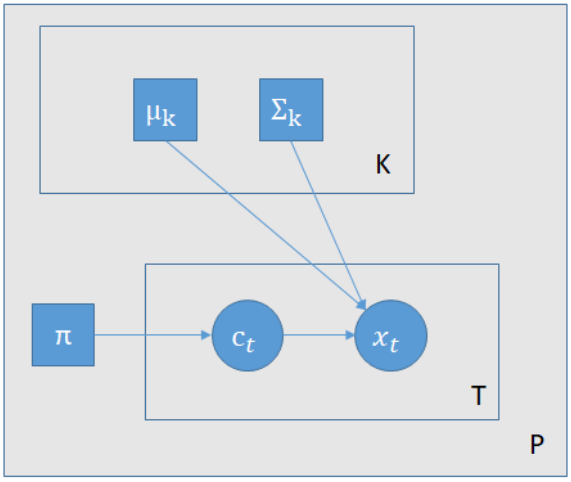

Figure 2.3. Updated Graphical Model for GMM

By introducing the hidden variable c_t, the log-likelihood function is simplified and used to derive the formula for model update:

$$\begin{aligned} \pi_k &= \frac{1}{T}\sum_t P(c_t = k|x_t, \theta) \\ \mu_k &= \frac{\sum_t x_t P(c_t = k|x_t, \theta)}{\sum_t P(c_t = k|x_t, \theta)} \\ \Sigma_k &= \frac{\sum_t P(c_t = k|x_t, \theta)(x_t - \mu_k)(x_t - \mu_k)^\top}{\sum_t P(c_t = k|x_t, \theta)} \end{aligned} \tag{2.10}$$

Here T is the total number of frames for this GMM.

For the second part in Equation 2.7, $P(s_t|s_{t-1}, \mathcal{Q})$, it is modeled by Hidden Markov Models (HMM). The standard GMM-HMM approach models each context dependent tri-phone [60] with an HMM model with 3 to 5 hidden states. Figure 2.4 shows graphical model representation of such an HMM with 3 hidden states. Here, α denotes the transition probability distribution, $s_{p,1}$, $s_{p,2}$ and $s_{p,3}$ denote 3 hidden states of the HMM, and $c_{1,t}$, $c_{2,t}$ and $c_{3,t}$ corresponds to the mixture hidden variables in Figure 2.3. The GMM is then cascaded to the HMM to model emission probabilities.

For HMM, EM is also used for MLE. Here, hidden variables $s_{p,n}$ are estimated by aligning speech features to transcriptions. Details of the derivation and update formula can be found in [51, 30].

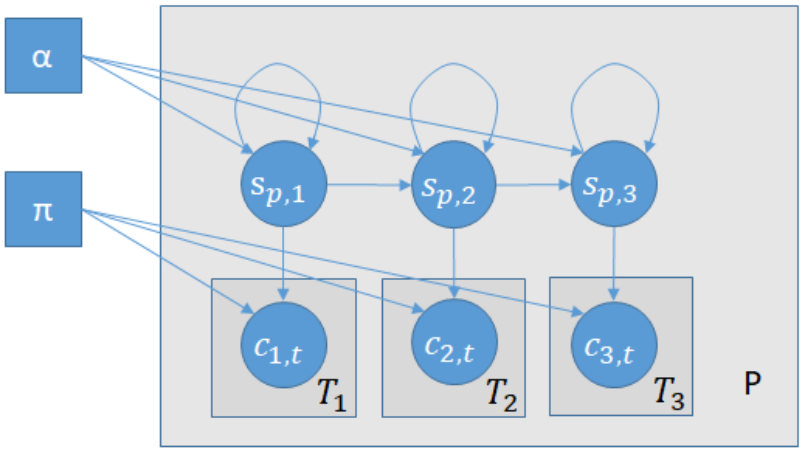

Figure 2.4. Graphical Model for HMM

2.1.5.2 Hybrid Approach

An early hybrid Neural network approach for speech recognition was explored by H. A. Bourlard and N. Morgan[12] in 1990s. In this approach, a neural network is used to predict frame posterior $P(s_t|x_t)$ at each time step, and these posteriors are converted to likelihoods $P(x_t|s_t)$ before sending to the decoder using

$$P(x_t|s_t) \sim \frac{P(s_t|x_t)}{P(s_t)} \tag{2.11}$$

In this approach, an HMM is still used to model transition probability. Following the same hybrid philosophy, recurrent neural networks also showed great potential in the 1990s [94, 95]. In 2010s, a new way of initializing deep neural networks was proposed[47] and then applied to continuous time speech recognition [21, 99]. Since then, researchers have been exploiting massive amount of data and computation to train deep neural networks models for ASR.

2.1.5.3 End-to-end Approaches

Third generation of ASR models were proposed in 2006 when Connectionist Temporal Classification (CTC) was introduced by A. Graves[37]. This method was later explored and improved in other research projects [43, 67, 15, 38].

In this approach, a neural network is used to perform end-to-end speech recognition, i.e. it no longer relies on an HMM model or lexicon, and just leaves transition modeling to a Recurrent Neural Network (RNN). The model is trained with segments of speech as input and CTC loss as its training target. Most of these systems do not even use language model to perform decoding.

In this work, such end-to-end approaches are not covered. I will primarily use the hybrid HMM/neural network approach for my ASR systems.

2.1.6 Speech Corpora

To evaluate different technologies developed for speech recognition and facilitate exchange of research ideas, many speech corpora have been prepared and made public by various parties, with a focus on different scenarios. These includes digit sequences (TIDIGITS), broadcast news, spontaneous telephone speech (Switchboard / CallHome), meetings (ICSI meetings) etc. In this section, three standard speech corpora for training / test are introduced and statistics on these datasets are reported.

2.1.6.1 Switchboard

Switchboard dataset[35] is a collection of about 2,400 two-sided English telephone conversations among 543 speakers. It consists of approximately 300 hours of clean telephone speech. This is dataset is one of the most widely used speech dataset for large vocabulary continuous speech recognition (LVCSR) task.

2.1.6.2 Fisher

Fisher English dataset[20] is another telephone conversation dataset for LVCSR. It contains time-aligned transcript data for 5,850 complete conversations (around 2000 hours).

2.1.6.3 Eval2000 dataset

2000 HUB5 English Evaluation Speech [29] is a collection of conversational telephone speech collected by LDC for evaluation of ASR systems. It consists of 20 unreleased telephone conversations from the Switchboard studies and 20 telephone conversations from CALLHOME American English Speech.

2.1.7 Metric for Performance Measurement

By comparing automatic recognition outputs and human transcription, we can evaluate and compare different ASR systems. Word error rate (WER) is a metric introduced to evaluate recognition accuracy [121]. WER measures the rate of word errors in ASR outputs, which is computed as:

$$WER = \frac{S + D + I}{N} \tag{2.12}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and N is the number of words in the reference. Computation of WER requires aligning each hypothesized word sequence with the reference using dynamic programming.

Although WER might not be a good metric for tasks like speech to speech translation or spoken language understanding[120], in this work, we will stick to this metric, as it is the most widely accepted one, and poor performance on WER often has a major effect on downstream measures.

2.2 Speaker Recognition

Speaker recognition, also called speaker identification (SID), is the task of identifying a person from characteristics of voices. As a sub-task of speaker recognition, speaker verification is about verifying if a speech segment comes from a specific speaker. Since speaker recognition and speaker verification share the same evaluation process under the metric EER (introduce later in 2.2.8), we will just use - "speaker recognition" (or SRE) for simplicity. Also, the terms are sometimes used interchangeably in referenced papers.

Depending on contents underlying speech segments, speaker recognition systems can be classified into two categories: text-dependent and text-independent. In fact, methods developed for a text-independent task can be applied to text-dependent tasks without much change. In this work, I will primarily focus on text-independent speaker recognition.

The core task of all speaker recognition systems is to extract vectors that represents speaker voice characteristics. This idea dates back to the 1960s when the first generation of speaker recognition systems were developed. In this approach, several statistics are computed over the set of training reference utterances provided by each speaker to form a speaker vector for decision making[44, 22]. In the 1970-1980s, pattern matching approaches became more popular as Dynamic Time Warping (DTW) and HMM were introduced to this field [74]. These approaches shared many similarities with those applied to ASR. In late 1990s, Gaussian mixture models (GMMs) have became the dominant approach for text-independent speaker recognition [91] and speaker recognition began sharing methodologies with research in speaker adaptation.

Around 2007, the use of joint factor analysis for speaker modeling showed great recognition performance in NIST speaker recognition evaluations [55, 27]. Later, in 2011, a new model with a similar philosophy was proposed where factor analysis is used to define a low-dimensional space that models both speaker and channel variabilities. The new speaker vector extracted from this model, called "i-vector", has shown better performance compared to old methods[26]. Since then, the i-vector approach became the state-of-the-art technique for speaker recognition.

In this work, factor analysis serves as the baseline system for speaker recognition. The following section will give an introduction for this approach.

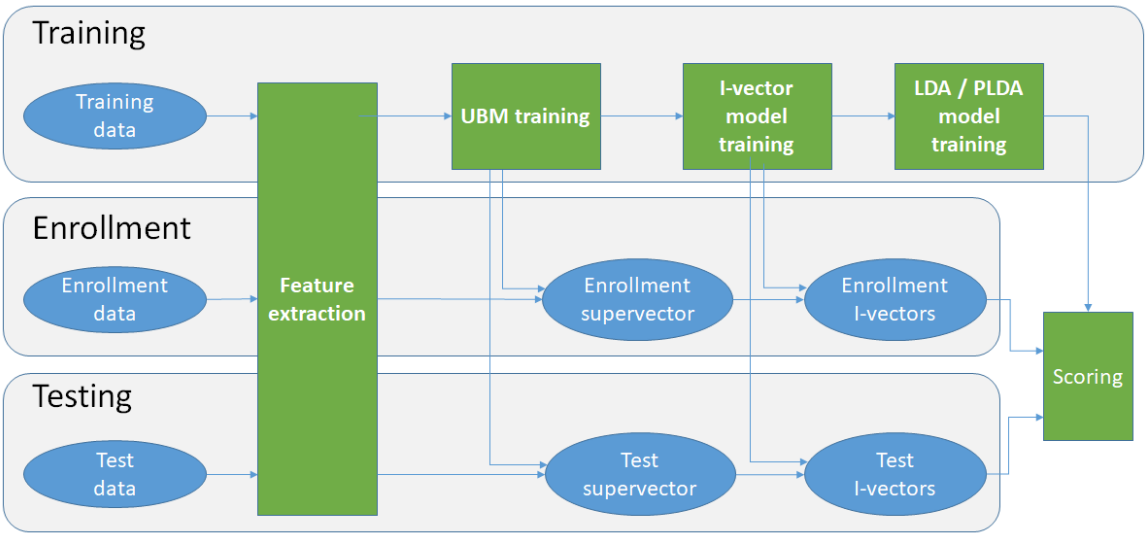

Figure 2.5. Pipeline for speaker recognition systems

2.2.1 Introduction

Similar to the way that researchers develop ASR systems, building speaker recognition systems requires the use of recorded speech data called training data. Depending on methods used for building the system, these training data are not necessarily transcribed into words. In some approaches, even speaker labels are not required, i.e. the system is trained in an unsupervised way. However, supervised models generally improve system performance so they may be cascaded to unsupervised models. During the evaluation phase, two more sets of speech data with speaker information are required: enrollment data and test data. Here enrollment data are used to register specific speakers and test data are used to test the system and compare prediction outputs with ground-truth labels.

Modern factor analysis approach for speaker recognition usually has 4 modules: feature extraction, acoustic modeling, speaker modeling and scoring. The acoustic model, usually a Gaussian mixture model, is built to model speech features. Once this model is trained, speaker models could be estimated using speaker-dependent data and the acoustic model. Scoring is done afterwards when speaker models are used to extract speaker dependent vectors, i.e. speaker embeddings. Figure 2.5 shows the full pipeline of developing such a system. In this approach, acoustic modeling

2.2.2 Feature Extraction

Similar to the feature extraction module in ASR, speaker recognition systems process speech frame by frame (10 ms) using a sliding window of 25 ms, and generate speech features using standard signal processing techniques. Two popular acoustic

features are MFCC and PLP, which were initially developed for ASR and then applied to speaker recognition [89]. Even though the settings of parameters might be different from those used for ASR (typically higher order models are used, to capture more acoustic characteristics of each speaker's voice), the general framework for feature extraction is about the same.

In this dissertation, MFCC will be used as default feature for i-vector systems.

2.2.3 Acoustic modeling

In the standard factor analysis approach, Gaussian mixture models (GMM) are used for acoustic modeling. Differing from a GMM-HMM system for ASR, here a single, speaker-independent background model (called a universal background model, or UBM) is used to represent the speech feature distribution. The reason for this setup is two-fold: 1. training data for speaker recognition does not necessarily come with transcriptions, so HMM alignments will not be available; 2. GMM-HMM system have separate models for phone or sub-phone units so as to discriminate them, while here we focus more on discriminating different speakers. A GMM model with hidden variable is repeated in Equation 2.13.

$$\begin{aligned} x_t|c_t &\sim \mathcal{N}(\mu_{c_t}, \Sigma_{c_t}) \\ P(c_t) = \pi_{c_t}, \quad & \sum_{k=1}^{K} \pi_k = 1 \end{aligned} \tag{2.13}$$

Here x_t denotes p-dim speech feature for frame t, c_t is a hidden variable that indicates the mixture that generates x_t, μ and Σ are mean and variance for Gaussian. K is the total number of mixtures. Historically, UBM with 512-2048 mixture components have been used for modeling speech feature vector distributions [91]. Model parameters are estimated using the expectation maximization (EM) algorithm. Once this model is in place, it could later be used for speaker modeling.

2.2.4 Speaker modeling

As is mentioned earlier, the core part of speaker recognition systems is to extract vectors that represents speakers. In a statistical view, this procedure can be seen as speaker modeling.

A straight-forward way to get speaker-dependent models is to align data to the UBM, and collect first and second order statistics. This is equivalent to fitting a GMM using speaker data with a UBM as seed model [56]. Another way is to perform maximum a posteriori (MAP) adaptation of a UBM [90]. Both methods will create speaker-dependent GMM models. By concatenating the means of these Gaussian mixtures, GMM supervectors could be generated to represent different speakers.

Once GMM supervectors are gathered, factor analysis could be used to learn a speaker subspace from the supervector space. In the classic joint factor analysis (JFA) approach [56], a supervector for a speaker is decomposed into speaker independent, speaker dependent, channel dependent, and residual components. While in the i-vector approach [26], a conversation side supervector is decomposed into side independent and side-dependent component [1]. Both approaches enable us to extract a low-dimensional vector to represent each speech segment (in the latter approach, the low-dimensional vector is called i-vector). Once this vector is extracted, it could be used to compare against enrolled speaker vectors to make a recognition decision.

2.2.5 Scoring methods

The process of comparing test vectors against enrollment speaker vectors is called scoring. A basic scoring method for speaker vectors is cosine scoring. Given v_1 and v_2 as the speaker vectors for enrollment and test speech segments, respectively, the score for the trial is computed as:

$$score(v_1, v_2) = \frac{\langle v_1, v_2 \rangle}{\|v_1\| \|v_2\|} \tag{2.14}$$

This method is easy to use and deploy, and it is shown that consine scoring yields reasonable recognition performance [25].

Since the i-vector model is an unsupervised model, it is natural to apply discriminative methods as a post-processing step so that speaker vectors can better distinguish different speakers. A basic discriminative model for this task is linear discriminant analysis (LDA). This model assumes vectors from each speaker follows a multivariate Gaussian distribution, and these Gaussians share the same covariance matrix. Figure 2.6 shows the graphical model for LDA.

A Bayesian version of LDA, called probabilistic linear discriminant analysis (PLDA), is also widely used. This model was initially proposed for face recognition [50]. It was then introduced to the field of speaker recognition in 2011 [33]. In addition to original LDA assumptions, this model assumes a Gaussian prior for each speaker's mean vector. Figure 2.7 shows the graphical model for PLDA.

Both models are trained using maximum likelihood estimation. For PLDA, the EM algorithm is necessary as it introduces hidden variables [2]. Experiments have shown that both models improve speaker recognition performance effectively [33, 109].

2.2.6 Deep learning approaches

In recent years, using deep neural network to capture speaker characteristics has become more and more popular. A common method of applying neural networks

[1] Details of i-vector approach will be introduced in Chapter 3

[2] Details of the derivation will be covered in Chapter 3

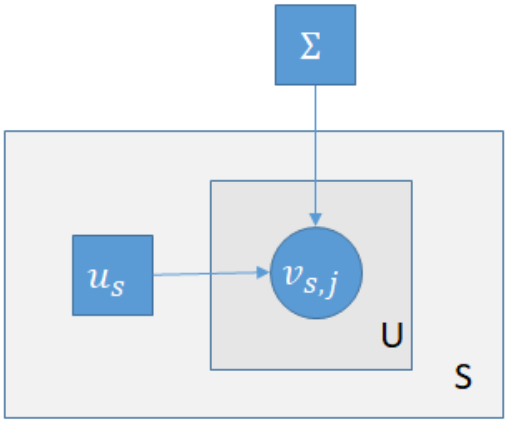

Figure 2.6. Graphical model for LDA

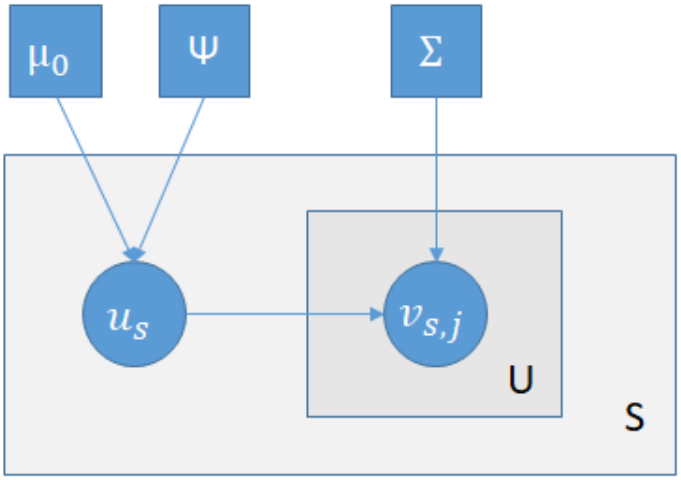

Figure 2.7. Graphical model for PLDA

to speaker recognition is to import frame posteriors from a DNN to the i-vector framework [62]. This method makes use of as ASR acoustic model to gather speaker statistics for i-vector model training. It has been shown that this improvement leads to a 30% relative reduction in equal error rate.

The d-vector is proposed in [114] to tackle text-dependent speaker recognition using neural network. In this approach, a DNN is trained to classify speakers at the frame-level. During enrollment and testing, the trained DNN is used to extract speaker specific features from the last hidden layer. "d-vectors" are then computed by averaging these features and used as speaker models for scoring. This method shows 14% and 25% relative improvement over an i-vector system for clean and noisy conditions respectively.

In [106], a time-delay neural network is trained to extract segment level "x-vectors" for text-independent speech recognition. This network takes in features of speech segments and passes them through a few non-linear layers followed by a pooling layer. Activations from this pooling layer are then sent to some non-linear layers to classify speakers at segment-level. X-vectors extracted by this model later serve as speaker models for enrollment and testing. It is shown that an x-vector DNN can achieve better speaker recognition performance compared to traditional i-vector approach, with the help of data augmentation.

End-to-end approaches based on neural network are explored in [45, 132, 130, 118]. In these papers, a neural network usually takes in a tuple or a triplet of speech segments, and train against a specially designed loss function or binary labels (match / mismatch). These methods all exploit large amounts of training samples for better performance, and are shown to be effective for speaker recognition, especially on short utterances.

2.2.7 Speech Corpora

2.2.7.1 SRE datasets

The NIST Speaker Recognition Evaluation (SRE) is a series of evaluations designed for text independent speaker recognition research [27]. This series dates back to 1997 and has performed 14 evaluations over the years. Data we used in this work includes SRE 2004, SRE 2005, SRE 2006 and SRE 2008 [3].

2.2.7.2 SRE 2010 test set

2010 NIST Speaker Recognition Evaluation Test Set [2] was collected as part of the NIST Speaker Recognition Evaluation [2]. It contains 2,255 hours of American English telephone speech and speech recorded over a microphone channel. The telephone

[3]LDC ids for these data sets are LDC2006S44, LDC2011S01, LDC2011S04, LDC2011S09, LDC2011S10, LDC2012S01, LDC2011S05 and LDC2011S08

speech segments are approximately 10 seconds and 5 minutes, and the microphone excerpts are 3-15 minutes in duration. In this work, speaker recognition experiments use SRE 2010 test set for performance evaluation. Both the "core condition" and one "optional condition" are used. For the core condition, both enrollment data and test data are two-channel telephone conversational excerpts of approximately five minutes total duration. For the optional condition we pick, both enrollment data and test data are two-channel excerpts from telephone conversations estimated to contain approximately 10 seconds of speech.

2.2.8 Metric for Performance Measurement

Each test utterance and enrollment speaker comparison is referred to as a trial. For a binary classification task like speaker verification, true positive rate and false positive rate can be used to evaluate performance of classifiers. A receiver operating characteristic (ROC) curve can then be created by plotting the true positive rate and false positive rate at various thresholds. Alternatively, a detection error tradeoff (DET) curve could be created by plotting the false negative rate versus false positive rate. For DET curve, the x- and y-axes are scaled by logarithmic transformation so that comparison between different DET curves become clearer. Figure 2.8 shows a sample DET curve plot (blue line).

Sometimes, comparing two curves can be complicated. This is especially true when two systems perform well in different regions (e.g. one is good at picking correct hits, while the other is better at rejecting false candidates). To avoid this problem, another metric called equal error rate (EER) is introduced. EER corresponds to the point on DET curve where false negative rate and false positive rate equal. The red point shown in Figure 2.8 is such an example. This meric has the advantage of condensing an entire curve to a single, well understood number, while it has the disadvantage of corresponding to an equal weighting of the 2 types of errors, which rarely corresponds to the concerns of any particular application.

2.3 Conclusion

In this chapter, the fundamentals for both ASR and speaker recognition are covered, which form the basis for the research described in this thesis. The neural network, being the building block for most state-of-the-art systems, is also introduced. The following chapters will then present my thesis research on ASR, speaker recognition and joint modeling.

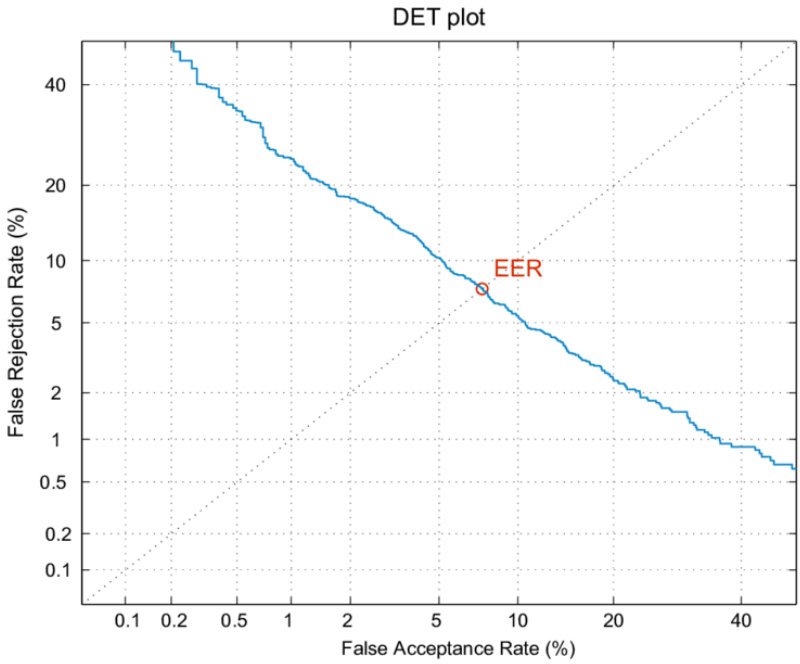

Figure 2.8. DET curve and EER

Chapter 3

Speaker Recognition Using ASR

3.1 Introduction

Factor analysis has become a dominant methodology for speaker verification in the last few years. This model is trained to learn a low-dimensional subspace from high-dimensional Gaussian Mixture Model (GMM) supervector space. The projected low-dimensional vector is used to represent different identities, thus denoted as i-vector (identity vector). I-vectors are usually transformed using a probabilistic linear discriminant analysis (PLDA) model to produce verification scores, which could be seen as a score normalization step. It has been shown that this could improve speaker verification performance significantly.

While deep learning has been successfully used for acoustic modeling in speech recognition, it is harder to apply it to speaker verification. The reason for this is two-fold: 1. speaker verification is not a standard classification task where targets are defined during training – unknown speakers may show up during the enrollment phase or the testing phase; 2. training data for speaker models are limited, e.g., each recording may only be used to extract one i-vector for the speaker. Despite all these difficulties, a novel scheme is proposed in [62] where a DNN is introduced to perform frame alignment in GMM supervector generation. This method opens up a new way of using ASR models for speaker recognition, which serves as the basis for the work presented here.

In this chapter, I first introduce the theory of applying factor analysis to speaker recognition, with a focus on comparing two different modeling ideas. In this part, I will give a derivation of factor analysis for speaker verification using a variational Bayesian framework, with a bias term included in hidden variables as is done in Kaldi. Implementation details of factor analysis and PLDA in the Kaldi toolkit are also discussed.

I then further investigate the effectiveness of incorporating ASR acoustic model into factor analysis. Following the scheme in [62], we collect posterior statistics from Deep Neural Networks (DNN) trained with raw MFCC and MFCC with different

feature transformations, including Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transformation (MLLT) and feature-space Maximum Likelihood Linear Regression (fMLLR). We also perform decoding for speech utterances and try to use decoded lattice posteriors for speaker verification. All these methods have shown improvement over a naive DNN trained with MFCC features. This also opens up a basic question for factor analysis based speaker verification: what is the best way to generate posteriors for i-vector extraction?

3.2 Related Work

In Section 2.2.1 it is mentioned that GMM supervector generation serves as the first step for building a traditional i-vector system. To integrate deep learning into the i-vector framework, Lei et al. proposed to use a DNN trained for ASR to collect alignment statistics for GMM supervector estimation [62]. This approach is shown to be effective for speaker verification, where a 30% relative reduction on equal error rate (EER) was achieved. The authors reasoned that this approach allows the system to make use of phonetic content information for speaker recognition. The same method was further investigated in [57] and similar performance is achieved.

Another way to incorporate deep learning into the i-vector framework is by using bottle-neck features. This feature is first proposed in [39] to perform ASR tasks. It is then used as inputs for a GMM-UBM system for speaker recognition in [127]. Later in 2015, bottleneck features were combined with the i-vector framework to perform speaker and language recognitions [92, 93]. The authors show that this method gives better performance than the DNN + i-vector approach proposed in [62].

My work in this chapter mainly focuses on extending the DNN + i-vector approach by using "better" acoustic models trained for ASR. This includes using better features for model training and sequence-discriminative training. On the theory side, I present comparison between two different i-vector modeling ideas, and discuss the implementation details of the i-vector system in Kaldi.

3.3 Factor Analysis applied to Speech

3.3.1 GMM supervector approach

As is mentioned in Section 2.2.1, factor analysis is an unsupervised learning method that is usually used for dimension reduction.

$$x_i = \mu + Az_i + \epsilon_i, \quad z_i \sim \mathcal{N}(0, I), \quad \epsilon_i \sim \mathcal{N}(0, \Psi) \tag{3.1}$$

where x_i is data samples (GMM supervectors) i to be analyzed, μ is global mean of the data, A is an p by q projection matrix where $p > q$, z_i is a q-dimensional latent

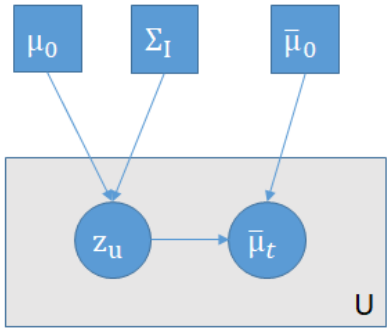

Figure 3.1. Graphical Model for Factor Analysis Applied to GMM Supervector

factor (i.e. i-vector) for sample i, and ϵ_i is Gaussian noise. Ψ is usually assumed to be diagonal, assuming independence between different dimensions of the error. Factor analysis could thus be seen as an extension to principal component analysis (PCA) where ϵ_i are assumed to have equal variance[113].

In the context of speaker recognition, factor analysis could be used to model GMM supervectors for each speaker or conversation [54]. Here we write it in terms of GMM mean vectors rather than a single supervector so that it is better formulated.

$$\mu_{i,c} = \mu_c + A_c z_i + \epsilon_{i,c}, \quad z_i \sim \mathcal{N}(0, I) \tag{3.2}$$

$$\epsilon_{i,c} \sim \mathcal{N}(0, \Psi_{i,c}) \quad \Psi_{i,c} = n_{i,c}^{-1} \Psi_c \tag{3.3}$$

$\mu_{i,c}$ is mean of Gaussian mixture c, and $n_{i,c} = \sum_t p(c|x_{i,t})$ is total number of "counts" for that mixture from speaker i. These are accumulated by aligning each frame t from speaker i to UBM components c using acoustic model. Figure 3.1 shows the graphical model representation for this approach.

Model parameters are estimated using Expectation Maximization (EM), and the auxiliary function is

$$Q(\theta|\theta^{(t)}) = \sum_i \mathbb{E}_{z_i|\mu_i,\theta^{(t)}} \log p(\mu_i, z_i|\theta) \tag{3.4}$$

$$p(\mu_i, z_i|\theta) = p(\mu_i|z_i)p(z_i) \tag{3.5}$$

3.3.2 Gaussian Mixture Factor Analysis Model

The previous GMM supervector approach is straight-forward to implement. However, it assumes an equal prior for each speaker or speech segment depending on the way we feed in our data, which may not make sense if speech data for each speaker / segment is not equally distributed. Also, this model treats GMM means as observed data and use GMM and factor analysis separately, which is not a staight-forward statistical model. To build a Gaussian Mixture Factor Analysis model, we

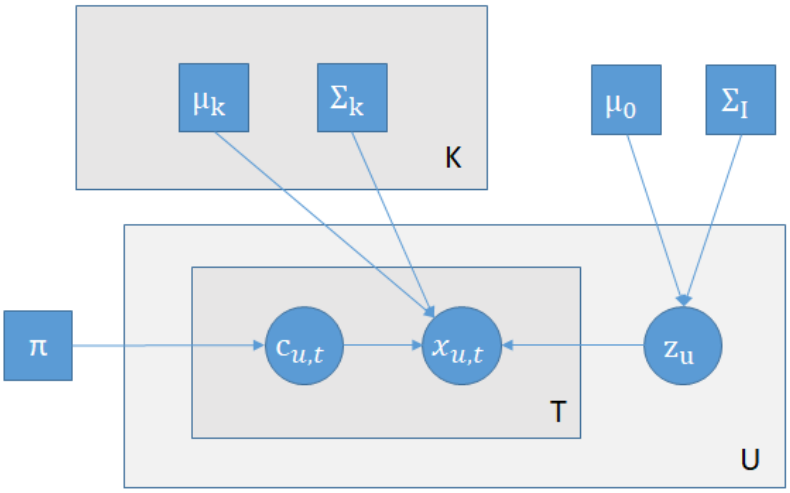

Figure 3.2. Graphical Model for Gaussian Mixture Factor Analysis Model

may want to take speech frames into consideration and incorporate factor analysis into the GMM. Here, I propose to preserve the GMM structure and mixture priors of the model, and derive training formulas using variational Bayes inference. This is different from the model used in [54] where fixed frame alignments are used for model formulation. Our approach is in line with what was mentioned in [62] when alignments are replaced by priors. This derivation makes it clear how we perform EM for mixture factor analysis.

In this model, speech features are modeled by as

$$\begin{aligned} x_{i,t}|c_{i,t}, z_i &\sim \mathcal{N}(A_{c_{i,t}} z_i, \Psi_{c_{i,t}}) \\ z_i &\sim \mathcal{N}(\nu, I), \quad c_{i,t} \sim p_c(k) \end{aligned} \tag{3.6}$$

where $x_{i,t}$ is p-dimensional feature vector for frame t of conversation i. $c_{i,t}$ indicates the mixture that generates $x_{i,t}$. z_i is a q-dimensional latent factor (i.e. i-vector) for this conversation. $A_{c_{i,t}}$ is a p by q projection matrix for mixture $c_{i,t}$ that projects i-vector to feature space, and $\Psi_{c_{i,t}}$ is covariance matrix for mixture $c_{i,t}$. $p_c(k)$ is prior distribution of Gaussian mixtures, with $\sum_k p_c(k) = 1$. The graphical model for this approach is shown in Figure 3.2, and model parameters $\theta = \{A_c, \Psi_c | \forall c\}$.

3.3.3 Model Estimation and Other Details

To perform maximum likelihood estimation (MLE), we use likelihood function as our objective

$$p(x|\theta) = \prod_i p(z_i) \prod_t \sum_c p(x_{i,t}|c_{i,t}, z_i, \theta) p_c(c_{i,t}) \tag{3.7}$$

and it is maximized using EM algorithm with auxiliary function

$$Q(\theta|\theta^t) = \mathbb{E}_{c,z|x,\theta^t} \log p(x, c, z|\theta) \tag{3.8}$$

$$= \mathbb{E}_{z|x,\theta^t} \left[\mathbb{E}_{c|z,x,\theta^t} \log p(x, c, z|\theta) \right] \tag{3.9}$$

$$\log p(x, c, z|\theta) \propto \log p(x, c|z, \theta) + \sum_i \log(p(z_i)) \tag{3.10}$$

$$\log p(x, c|z, \theta) = \sum_i \sum_t \log(p(x_{i,t}|c_{i,t}, z_i) p(c_{i,t}|z_i)) \tag{3.11}$$

where $x = \{x_{i,t}|\forall i, \forall t\}, c = \{c_{i,t}|\forall i, \forall t\}, z = \{z_i|\forall i\}$. Here, both z and c are considered as latent variables in EM framework.

Since z and c are conditionally dependent given x because of the "Explaining Away" effect, there is no closed form solution to update them in a joint fashion. However, we could approximate an auxiliary function and a posterior distribution assuming conditional independence between z and c

$$Q(\theta|\theta^t) \approx \mathbb{E}_{z|x,\theta^t} \left[\mathbb{E}_{c|x,\theta^t} \log p(x, c, z|\theta) \right] \tag{3.12}$$

Following the derivation of EM for GMM in [11], the auxiliary function could be simplified as

$$Q(\theta|\theta^t) \propto \mathbb{E}_{z|x,\theta^t} \sum_i \left[\log p(z_i) + \sum_t \sum_k \log p(x_{i,t}|k, z_i) \gamma_{i,t}^k \right] \tag{3.13}$$

where $\gamma_{i,t}^k$ denotes the posterior distribution of $c_{i,t}$ given $x_{i,t}$, i.e. $p_{c|x}(k|x_{i,t})$.

From here we need the posterior distribution of z given x to proceed.

$$\begin{aligned} p(z_i|x_i) &\propto p(z_i) p(x_i|z_i) \\ &\propto p(z_i) \prod_t \sum_c p(x_{i,t}|c_{i,t}, z_i) p(c_{i,t}) \end{aligned} \tag{3.14}$$

Here $x_i = \{x_{i,t}|\forall t\}$, and similarly $c_i = \{c_{i,t}|\forall t\}$.

The analytical solution for this is also intractable. However, we could use a variational Bayes method[6] to approximate it by

$$\begin{aligned} p(z_i|x_i) &\approx p(z_i) e^{\mathbb{E}_{c_i|x_i} \log p(x_i, c_i|z_i, \theta)} \\ &\approx p(z_i) \prod_t \prod_k p(x_{i,t}|k, z_i)^{\gamma_{i,t}^k} \end{aligned} \tag{3.15}$$

So the E-step gives

$$\begin{aligned} \mathbb{E}_{z_i|x_i} &= \mathrm{Var}_{z_i|x_i} \cdot \left(\sum_k A_k^\top \Psi_k^{-1} \sum_t \gamma_{i,t}^k x_{i,t} + \nu \right) \\ \mathrm{Var}_{z_i|x_i} &= \left(\sum_k \left(A_k^\top \sum_t \gamma_{i,t}^k \Psi_k^{-1} A_k \right) + I \right)^{-1} \end{aligned} \tag{3.16}$$

and maximizing the auxiliary function w.r.t. z_i gives

$$\begin{aligned} A_k &= \Big[\sum_i \sum_t \gamma_{i,t}^k x_{i,t} \mathbb{E}_{z_i|x_i} z_i\Big] \Big[\sum_i \sum_t \gamma_{i,t}^k \mathbb{E}_{z_i|x_i} z_i z_i^\top\Big]^{-1} \\ \Psi_k &= \frac{1}{\sum_i \sum_t \gamma_{i,t}^k} \mathbb{E}_{z|x} \sum_i \sum_t \gamma_{i,t}^k (x_{i,t} - A_k z_i)(x_{i,t} - A_k z_i)^\top \end{aligned} \tag{3.17}$$

These formulas are consistent with those derived in [54] using posteriors in the model formulation.

After updating the projection matrix and the covariance matrix, a Minimum-Divergence (MD) [13] step could be used to speed up model learning, which includes an extra step to update the prior ν. An extra transformation (Householder transformation) [1] is used to complete the update of the priors. These steps are already included in the open-source toolkit Kaldi.

3.3.4 PLDA model for scoring

Several formulations of PLDA have been proposed by researchers, and they can be unified as the same one[102]. Kaldi's PLDA follows the formulation proposed in [50].

$$x_{i,j} = \mu_g + A u_{i,j}, \quad u_{i,j} \sim \mathcal{N}(v_i, I), \quad v_i \sim \mathcal{N}(0, \Psi) \tag{3.18}$$

where $x_{i,j}$ is sample j from speaker i (in this case, they are i-vectors extracted from previous step), μ_g is global mean of the data sample. $u_{i,j}$ is sample-specific latent vector in transformed space, and A is the transformation. v_i is speaker specific latent vector for speaker i, and its variance Ψ is a diagonal matrix. This model assumes equal variance for different identities, which could be seen as score normalization model.

Though this model could be trained by EM directly, the training process becomes easier if we convert it to two-covariance form [14]

$$\begin{aligned} y_i &\sim \mathcal{N}(0, \Sigma_B) \\ x_{i,j}|y_i &\sim \mathcal{N}(\mu_g + y_i, \Sigma_W) \end{aligned} \tag{3.19}$$

Here, y_i is the latent vector for speaker i. Σ_B is between-class variance and Σ_W is within-class variance. The graphical model for this simplified model (shown in Figure 3.3) is only slightly different from what we introduced in Section 2.2.5.

The conversion is done by setting

$$y_i = A v_i, \quad \Sigma_B = A^\top \Psi A, \quad \Sigma_W = A^\top A \tag{3.20}$$

μ_g is estimated as global mean of training data and is fixed during model training.

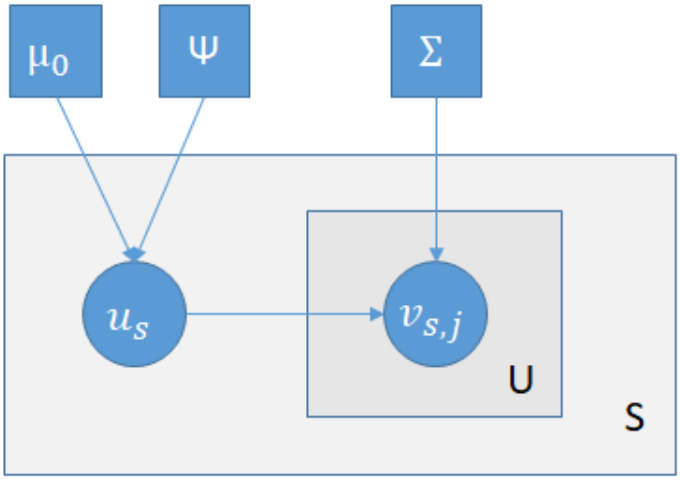

Figure 3.3. Graphical Model for simplified PLDA

Kaldi uses an EM algorithm that is slight different from what was described in [102]. Model learning is speeded up by introducing $m_i = \frac{1}{n_i}\sum_j x_{i,j}$, so the model becomes

$$\begin{aligned} y_i &\sim \mathcal{N}(0, \Sigma_B) \\ m_i|y_i &\sim \mathcal{N}(\mu_g + y_i, n_i^{-1}\Sigma_W) \end{aligned} \tag{3.21}$$

and auxiliary function is

$$\begin{aligned} Q(\theta|\theta^t) &= \sum_i \mathbb{E}_{y_i|m_i,\theta^t} \log p(m_i, y_i|\theta) \\ p(m_i, y_i|\theta) &= p(m_i|y_i)p(y_i) \end{aligned} \tag{3.22}$$

In E-step, conditional expectation are derived using conjugate prior

$$\begin{aligned} \mathbb{E}_{y_i|m_i} &= (n_i\Sigma_W^{-1} + \Sigma_B^{-1})^{-1} n_i \Sigma_W^{-1}(m_i - \mu_g) \\ \mathrm{Var}_{y_i|m_i} &= (n_i\Sigma_W^{-1} + \Sigma_B^{-1})^{-1} \end{aligned} \tag{3.23}$$

and the M-step model update formulae is

$$\begin{aligned} \Sigma_W &= \frac{1}{N}\sum_i \mathbb{E}_{y_i|m_i} n_i (m_i - \mu_g - y_i)(m_i - \mu_g - y_i)^\top \\ \Sigma_B &= \frac{1}{N}\sum_i \mathbb{E}_{y_i|m_i} y_i y_i^\top \end{aligned} \tag{3.24}$$

The model is then converted back to the form shown in Equ. (3.18) by performing Cholesky decomposition of Σ_W and eigenvalue decomposition of transformed Σ_B.

Once a model is trained, transformed vectors $u_{i,j}$ could be extracted from i-vector $x_{i,j}$, and then used for inference against enrollment data. This part is covered in Section 3.1 in [50].

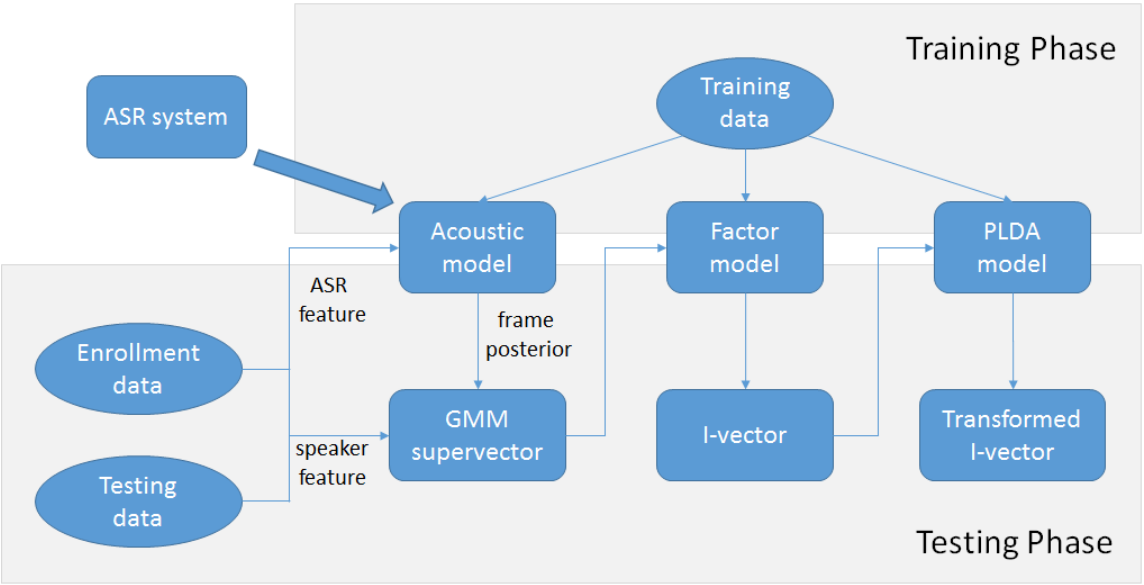

Figure 3.4. Framework for Speaker verification using Neural Network

3.4 Neural Networks for Speaker Recognition

3.4.1 General Framework

A general speaker verification framework using a neural network (shown in Figure 3.4) was proposed in [62]. In this approach, a DNN trained for ASR is used to produce frame alignments. These alignments are used as $\gamma_{i,t}^{k}$ in equation 3.16 in our formulation. It was stated that this pipeline integrates information from speech content directly into statistics. To better understand how neural networks could be used to improve speaker recognition performance, we investigate the effectiveness of better senone posteriors.

Many techniques that improve ASR performance are based on transformation of feature / model, and another family of methods called sequence-discriminative training[80] analyzes conditional dependence between frames and optimizes objectives defined with regard to whole utterances.

3.4.1.1 Linear Discriminant Analysis for speech recognition

LDA is a well-known technique for speech recognition[41, 123]. In general, we seek to obtain a transformation so that it maximizes the separability of transformed data. This is usually done by solving a generalized eigen-value decomposition problem. In Kaldi, LDA transformation matrix is computed to project MFCC features (with delta and acceleration) into a 40-dim subspace with triphone senones as class labels.

3.4.1.2 Maximum Likelihood Linear Transformation

MLLT (also known as Global Semi-tied Covariance) is another important technique for speech recognition[31, 32]. It is a global transformation matrix used to maximize frame log-likelihood with respect to some constraint. This is usually done using Expectation Maximization. In Kaldi, MLLT is performed on top of LDA features and is performed in feature space.

3.4.1.3 fMLLR transforms

fMLLR (also known as CMLLR) is a useful technique for speaker-adaptive training (SAT) of speech recognition[61]. It is a speaker-specific feature-space affine transformation that maximize frame log-likelihood, estimated using EM. Kaldi performs SAT on top of LDA and MLLT.

3.4.1.4 Sequence discriminative training

Sequence discriminative training was developed to address the sequential nature of speech. In brief, it tries to optimize objectives that are closely related to sequence classification accuracy[80]. Popular objectives include Maximum Mutual Information (MMI) [7], boosted MMI [82], Minimum Phone Error [83] and state-level Minimum Bayes Risk [36].

3.5 Experiments

3.5.1 Data

In this chapter, we use the 300-hour Switchboard-I Training set for ASR model training.

The UBM and i-vector model training data consists of SWB and NIST SREs. The SWB data contains 21,254 utterances from 6,820 speakers of SWB 2 Phases I, II and III. The SRE dataset consists 18,715 utterances / channels from 3,009 speakers of SREs from 2004 to 2006. PLDA model is trained using NIST SREs from 2004 to 2008, which consists of 28,579 utterances from 5,321 speakers.

We evaluate our systems on the condition 5 extended task of SRE10[2]. The evaluation consists of conversational telephone speech in both enrollment and test utterances. There are 387,112 trials, over 98% of which are non-target comparisons.

3.5.2 Setup

The Kaldi toolkit[81] is used for both speech and speaker recognition. For the speech recognition system, the standard 13-dim MFCC feature is extracted and used

for maximum likelihood GMM model training. Features are then transformed using LDA+MLLT before SAT training. After GMM training is done, three tanh-neuron DNN-HMM hybrid systems are trained using different kinds of features: 1. MFCC; 2. LDA + MLLT transformed MFCC; 3. LDA + MLLT + fMLLR transformed MFCC. Details of DNN training follows Section 2.2 in [116].

For the speaker verification system, we follow the setup in [103]. The front-end consists of 20 MFCCs with a 25ms frame-length. The features are mean-normalized over a 3 second window. Delta and acceleration are appended to create 60 dimensional frame-level feature vectors. I-vector dimension is set to 600.

To get fMLLR transformations, we need to perform ASR for all speaker verification data and also a pre-ASR Voice Activity Detection (VAD). VAD is done by performing phone decoding with a limited search beam, and speaker independent decoding and fMLLR decoding are done in an iterative fashion. These steps are time-consuming in practice, making it impractical for real-time scenarios at this time.

3.5.3 Results and Analysis

Table 3.1 shows EERs of factor analysis systems trained with different posteriors [1], and Figure 3.5 plots the corresponding DET curve for these systems. All the experiments in this table use standard speaker ID MFCC features [2]. As is shown, significant improvements are achieved when we use posteriors from a DNN trained with transformations [3]. We could also see that improvements on EER aligns with speech recognition performance of ASR systems, and the best performance is from sequence discriminative training with LDA, MLLT and fMLLR transformation.

	eval2000 WER	EER		
		male	female	all
UBM (4096)	–	5.92	6.80	6.36
UBM (8192)	–	5.83	6.80	6.31
DNN-MFCC (8824)	19.4	5.63	7.05	6.39
+ LDA + MLLT	16.3	4.07	5.43	4.84
+ SAT (fMLLR)*	14.9	3.98	5.02	**4.55**
+ MPE*	13.5	3.58	4.75	**4.38**
GMM-fMLLR-latpost*	21.8	4.50	5.99	5.45
DNN-fMLLR-latpost*	15.0	4.16	5.00	4.66

Table 3.1. EER of speaker recognition systems trained with different posteriors

[1] EERs in these experiments are worse than those reported in [103] because we use less data for UBM, FA and PLDA model training. Specifically, we left out Switchboard Cellular, SRE 2005 test set, SRE 2006 test set and SRE 2008 due to computation issue.

[2] where a higher model order is used compared with ASR

[3] Asterisk (*) indicates experiments require decoding of speech

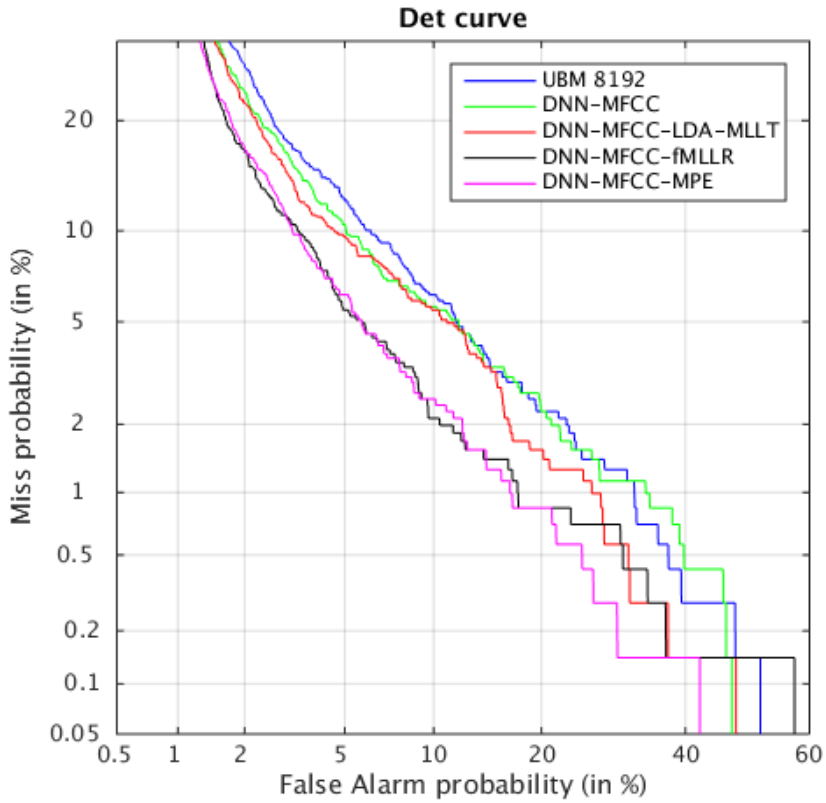

Figure 3.5. DET curve for speaker ID performance

We also try to incorporate phonetic content using posteriors from decode lattices. We could see from Table 3.1 that these posteriors give comparable results as those come right out of acoustic models. However, they do require more computation, so in general these are not good alternatives for this task.

3.5.4 Using ASR features for speaker verification

We learn from Section 3.5.3 that posteriors generated from fMLLR-DNN benefit speaker verification a lot. This is somewhat surprising because fMLLR transformation is believed to remove speaker specific information. However, it gives better posterior estimates, and thus help speaker verification.

In this section, we would like to use transformed features for speaker verification directly. Table 3.2 compares different features front-end for factor analysis, where they all share the same posteriors from fMLLR based DNN. "Default" denotes the standard MFCC feature used in previous experiments, "ASR LDA+MLLT" denotes the MFCC feature transformed by LDA and MLLT, and "ASR fMLLR" denotes the

MFCC feature transformed by LDA, MLLT and fMLLR. We could see from the table that both LDA+MLLT and fMLLR features degrade system performances, which is consistent with our knowledge. Meanwhile, it is interesting to note that these features, though transformed to remove speaker specific characteristics, still contain speaker information and can be used for speaker ID. This might raise an issue when one wants to protect speaker information by applying an fMLLR transform on speech features and transmit them over the Internet.

Speaker recog feats	EER		
	male	female	all
Default	3.98	5.02	**4.55**
ASR LDA+MLLT	5.43	7.24	6.35
ASR fMLLR	7.85	9.42	8.84

Table 3.2. fMLLR posteriors with different feature front-end

3.6 Conclusion

In this chapter, I study the effectiveness of state-of-the-art ASR techniques for speaker verification. It was found that speaker verification performance aligns with speech recognition performance when posteriors are imported from acoustic models trained for ASR. Out of all the systems, the one using a DNN trained with fMLLR features and MPE objective produces posteriors that benefit factor analysis most. I also presented a derivation of factor analysis in the framework of GMM with mixture priors, using variational Bayes inference, and explain some of the implementation details for the Kaldi toolkit.

Chapter 4

Speaker Adaptation Using I-vectors

4.1 Introduction

Speaker adaptation refers to methods whereby a speech recognition system is adapted to the acoustics of specific speakers. The need of such techniques arises since differences between training and testing speakers usually introduce recognition errors. Even for human, listening to strangers speaking with unknown accent or dialect may take much efforts. To mitigate this problem, several approaches may be taken:

1. Use large amounts of training data that covers different kinds of accents / dialects.
2. Build separate speaker-dependent ASR systems.
3. Adapt speaker-independent system to different speakers.
4. Normalize speech feature before sending them into ASR systems.

Adding training data that covers various speakers make sense in that it forces models to generalize to different speaker characteristics. However, this does require more efforts spent on collecting data, and it usually requires an increase in model complexity. Moreover, out-of-sample test speakers will always introduce some inconsistency, so this method may not cure the whole problem. Building speaker dependent ASR systems would certainly improve recognition accuracy given the same amount of data, but it requires labeled speech from target speakers, which may not be possible for out-of-sample speakers.

Methods that fall into the 3rd and 4th approaches generally do not require large amounts of speaker data, nor do they increase model complexity too much. These methods are considered to be speaker adaptation techniques. Depending on whether

transcriptions of speaker data are given, these methods could be either supervised or unsupervised. Some unsupervised normalization techniques even do not require training on speaker data, e.g., cepstral mean variance normalization (CMVN) [42] and vocal tract length normalization (VTLN) [28]. Methods that require the use of speaker data can be divided into the maximum a posteriori (MAP) adaptation family and transformation-based adaptation family [124]. Of these methods, constrained maximum likelihood linear regression (CMLLR), also know as feature-space MLLR (fMLLR) is one of the most widely used method [31].

Speaker adaptation is also closely related to the concept of speaker adaptive training [4, 30]. During the model training phase, speaker independent systems necessarily processes data from a large number of speakers. As was mentioned earlier, this may make the acoustic model waste a large number of parameters encoding differences between speakers rather than differences between words / phones. Thus, we may also normalize speech before sending it into the system during training time, and this is referred to as speaker adaptive training (SAT) [5]. Oftentimes people use these two words interchangeably in research papers.

The use of neural networks for ASR opens up a new area for speaker adaptation. Traditional feature-space speaker adaptation methods developed for GMM-HMM systems could be applied to neural network ASR systems naturally. Apart from these, adaptation technologies within the framework of neural network are also developed, including using transformations [128], adjusting hidden activation states [111] or retrain part of the model with some regularization on a loss function [64, 129]. These methods result in a new model, or part-model, for each speaker which adds significant complexity and storage for cloud-based applications [101].

Recent research on using the i-vector for speaker recognition points to a new direction of speaker adaptation. Within the framework of deep learning, one may easily incorporate speaker information by introducing i-vectors to acoustic modeling. Various ways of incorporating i-vectors have been studied in different research papers [96, 101, 40, 71], and they all show improvements over baseline speaker-independent systems.

My work in this chapter adds to the research on using i-vectors for speaker adaptation of DNN acoustic models. By studying the characteristics underlying this approach, I proposed a new regularization idea that helps solve the data scarcity issue for i-vectors. This method is shown to be effective in improving speaker adaptation performances. Also, I compare the proposed idea with the traditional feature-space maximum likelihood linear regression (fMLLR) method, and find that these two methods show a similar effect in ASR performance.

In this chapter, I will first introduce speaker adaptation using i-vectors in general, and then go on to describe my proposal of using channel i-vectors with regularization for speaker adaptation. Experimental setup and results will be presented in Section 4.4 and Section 4.5 respectively. And then I will conclude this chapter in Section 4.6.

4.2 Related work

Using i-vectors for speaker adaptation of DNN based ASR systems serves as a convenient way to avoid system design complexity [96, 101, 40]. In this approach, i-vectors are fed into neural networks during training as as additional source of information, and this makes the neural network use these speaker-specific characteristics to perform phone state classification. Another similar approach uses a sub-network for i-vector inputs and adds hidden activations of this sub-network to the original speech features in hope they help normalize them [70, 71]. Both of these approaches show promising results on improving ASR performances.

The first approach mentioned above appears to be much easier to implement and straight-forward to understand, thus attracting great attention. This work also follows this idea where i-vectors are appended to the original speech features before being used for neural network training. An inherent problem with using i-vectors this way is that i-vectors are usually of a higher dimension compared with speech features. Even if one performs context expansion of the speech feature, i-vector dimension usually appears similar to that of expanded features. However, the number of data samples for an i-vector is significantly lower than for speech features. For example, on the Switchboard dataset [35], there are around 520 speakers in 4800+ recording sides, totaling 100+ million frames of data. A neural network structure, which exploits huge amount of data to learn the non-linear relationship between feature and targets, may not have enough i-vector samples for training compared with their high dimension. This is called the curse of dimensionality in machine learning research [49]. This problem may cause a neural network to be over-trained on training samples, and perform badly on test data.

This problem is closely studied in [101]. It is shown that adding 300-dimention utterance i-vectors directly as input features does not improve ASR performance. Also, statistics collected along the training progress shows the neural network is over-fitting to the i-vector and is unable to use this information during decoding. The author tries reducing the dimension of the i-vectors, and a slight improvement over baseline system can be got when the dimension is set to 20. Though dimension reduction can help avoid curse of dimensionality, it also reduces the i-vectors' capability for speaker modeling [1]. Another method proposed in this paper is to perform regularization. It begins with a network trained without any i-vector information, and the input layer of the network is augmented with weights for i-vectors. Then the network is trained with i-vector inputs, but with L2 regularization back to the original weights. This methods shows similar performance improvement using a tuned regularization parameter.

A similar approach for using i-vectors is explored in [96]. Different from the one just covered, this research uses speaker based i-vectors for adaptation. The au-

[1] Standard i-vector speaker recognition systems uses dimension 400-600

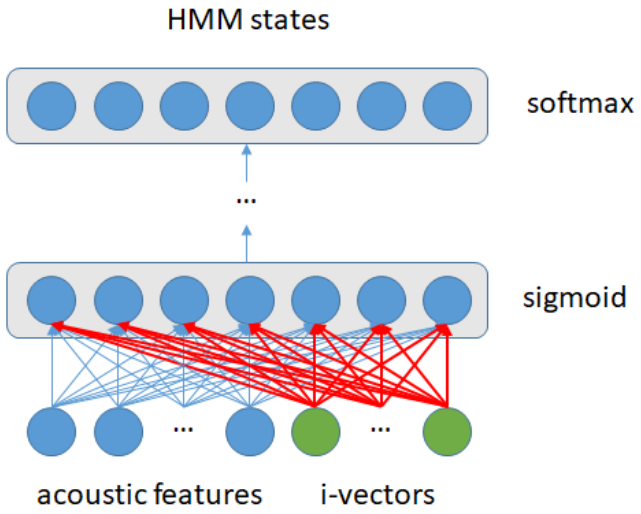

Figure 4.1. Adaptation of DNN using i-vectors

thors show better performance with higher dimensional speaker i-vectors (100-200). They also demonstrated that i-vectors combined well with a conventional CMLLR adaptation techniques.

The Kaldi toolkit [81] also has an implementation that utilizes i-vectors for ASR. It differs from the previous two approaches in that it extract i-vectors in an online fashion. To be specific, the i-vector is estimated in a left-to-right way, meaning that at a certain time t, it sees input from time 0 to t. This recipe is implemented to address the online decoding task, where traditional transformation based speaker adaptation methods cannot be applied effectively.

4.3 Speaker adaptation using i-vectors

My work in this section shares the same model structure as the previous ones, but uses channel-based ivectors. The reason for this is two-fold: 1. channel i-vectors are more stable than utterance i-vectors / online i-vectors (when they are available). This is because longer segments tend to cover different ranges of phones that can help i-vector estimation. 2. it does not require speaker information being recorded beforehand, which makes it more applicable to different scenarios. On the other hand, I introduce a new method for regularization to mitigate the curse of dimensionality issue. This method avoids reducing the i-vector dimension so as to preserve speaker information, and also it does not require another model to regularize back to.

Figure 4.1 shows the model structure. The inputs to the neural network are split into two parts: acoustic features and i-vectors. For the acoustic feature, I use MFCC transformed by linear discriminative analysis (LDA) and maximum likelihood linear

transformation (MLLT). These transformations are learned from a baseline GMM-HMM system. For i-vectors, I extract them using a standard factor analysis model trained on MFCC features. Input features are passed through several layers of affine transformation followed by sigmoid nonlinear activations. The final layer is a softmax layer that predicts HMM states. We denote i-vector inputs and weights associated to them as "i-vector sub-network".

During training, we first exclude the "i-vector sub-network" and initialize the model as a deep belief network. Evidence has shown that a deep belief network pre-training algorithm will help speed up convergence of deep learning models [47]. This model is pre-trained layer-wise, using gradient-based Contrastive Divergence algorithm for Restricted Boltzmann Machines (RBM) [47, 72]. After the pre-training, speech features and i-vectors are combined and sent into the network to perform the backpropagation (BP) algorithm. Model is updated using stochastic gradient descent (SGD). I use the "newbob" learning rate schedule, which is to start with a fixed learning rate for the first few iterations, and began halving the learning rate when relative improvement of loss on cross-validation set is less than 1%. The training goes on until relative improvement of loss becomes less than 0.01%.

During decoding, channel i-vectors are extracted for each side of the recordings. Together with MFCC features, these inputs are passed into the neural network for frame posterior prediction. These posteriors are then converted to likelihoods before being sent into a WFST decoder.

4.3.1 Regularization for i-vector sub-network

Regularization for this model is done by adding a L2 regularization term of the i-vector sub-network. Suppose the original cross-entropy loss function for neural network training is

$$L_{ce} = -\sum_{t}\sum_{c} y_{c,t} \log P(c|x_t) \tag{4.1}$$

where x_t is feature vector for frame t, $y_{c,t}$ is a binary indicator (0 or 1) that equals 1 if class c is the correct state for frame t. $P(c|x_t)$ is the predicted probability that frame t is of state c. Then the new loss function with regularization term is

$$L_{re} = L_{ce} + \beta \|w_{ivec}\|^2 \tag{4.2}$$

where $\|w_{ivec}\|^2$ denotes L2 regularization of weights directly associated with i-vectors (these weights are shown in red in Figure 4.1). β is a hyper-parameter to be tuned for appropriate value.

4.4 Experimental Setup

The Kaldi toolkit[81] is used for baseline GMM-HMM and DNN-HMM system training. Standard 13-dim MFCC feature are extracted, and together with its deltas and accelerations, used for maximum likelihood GMM model training. Features are then transformed using LDA+MLLT before SAT training. After GMM training is done, alignments of training data are prepared for DNN model training. Note that SAT training is used here only to get better alignments. The transformations learned in this process are not necessary for DNN model training.

The 300-hour Switchboard data set [35] is used in this work for model training. Statistics of the data set are provided in Table 4.1. To facilitate system development and parameter tuning, a smaller dataset is constructed by randomly picking recordings and utterances from the full data set. To make sure the utterances are evenly distributed, we restrict equal number of utterances per channel. Statistics of this smaller data set is also shown in Table 4.1. The test data for this work is the NIST 2000 Hub5 evaluation set (Eval2000) [29]. Recognition results of both Switchboard portion (denoted as "swbd") and Call Home portion (denoted as "callhm") are reported.

		channels	utterances	hours
Switchboard	10-hour set	438	7,825	9.83
	full set	4809	192,390	284.76
Eval2000	swbd	40	1,831	2.09
	callhm	40	2,635	1.61

Table 4.1. Statistics for training and test set

4.5 Results and Discussion

4.5.1 Effects of regularization

Table 4.2 shows experimental results of models trained on the 10-hour subset of Switchboard data. The last line of the table shows results from the baseline acoustic feature only system. Here LDA transformed feature is used as the default acoustic feature. The top line of the table shows results from i-vector adaptation model. We can see that i-vector adaptation model performs better than the baseline model in both validation set frame accuracy and evaluation set decoding WER.

The three lines in the middle of Table 4.2 shows effects of regularization. In fact, the top line (where no regularization is used) and bottom line (where no i-vectors are used) corresponds to regularization weight β set to 0 and $+\infty$ respectively. Figure 4.2 plots the WER of Eval2000 swbd portion and frame accuracy on cv set in one plot.

	Validation set		Eval 2000 WER	
	XENT	frame acc	swbd	callhm
With i-vector ($\beta = 0$)	2.875	40.21	24.9	39.7
$\beta = 8e-6$	2.857	40.24	24.6	39.3
$\beta = 8e-5$	2.860	40.60	24.5	38.5
$\beta = 8e-4$	2.892	39.98	24.9	39.2
No i-vector ($\beta = +\infty$)	2.952	38.35	26.5	41.5

Table 4.2. WER of systems trained on 10-hour training set

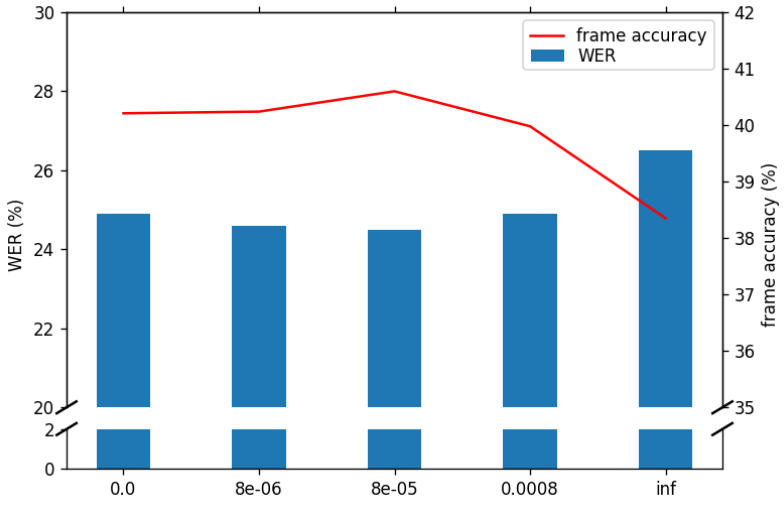

Figure 4.2. Comparing systems with different regularization weights

We can see that models with moderate regularization do achieve better system performances, and the frame accuracy on development set is a good indicator of ASR recognition performance.

4.5.2 Comparing with feature space adaptation method

The left half of Table 4.3 shows recognition performance of different systems trained on the full Switchboard dataset. We can see from the numbers that i-vector adaptation and regularization works well on the full data set. On the other hand, we also want to compare i-vector adaptation with traditional fMLLR based adaptation. The right half of the table shows systems trained on fMLLR adapted acoustic features. Compared with the baseline system, we can see the fMLLR system achieves a WER reduction from 16.0 to 14.9, which is about the same as that achieved by i-vector adapted system with regularization. Also, when these two are combined, a slight better WER can be achieved (14.3). From these results we see that effects from

i-vector and fMLLR adaptation are similar.

feature	ac feature		+fMLLR	
data	swbd	callhm	swbd	callhm
acoustic feature	16.0	28.5	14.9	25.6
+ i-vector	15.2	27.1	14.4	25.7
+ regularization	**14.6**	**26.3**	**14.3**	**24.9**

Table 4.3. WER of systems trained on full training set

4.6 Conclusion

In this chapter, I proposed to use channel i-vectors for speaker adaptation of DNN hybrid system, and used L2 regularization of "i-vector sub-network" to mitigate the curse of dimensionality. Experimental results show that regularization can effective improve adaptation performance, and the final result matches the approach that uses the transformation based fMLLR adaptation method. In terms of real application, the i-vector adaptation method is handier than the fMLLR method as its pipeline is much shorter; i.e., it does not require performing ASR using a speaker independent model and estimating speaker specific transformations.

Future work may include:

1. Perform speaker adaptation on recurrent neural networks.

2. Substitute i-vector with speaker embeddings generated by neural network.

Chapter 5

TIK: An Open-source Toolkit Connecting Tensorflow and Kaldi

5.1 Introduction

In recent years, the resurgence of neural network research has generated a need for flexible and efficient deep learning frameworks. As a result, a bunch of open-source implementations have been actively developed. Among them are Tensorflow[3], PyTorch[77], Caffe[52], MXNet[18] and CNTK[98]. All of these toolkits support flexible model development at large scale and in heterogeneous environments, bringing a positive impact to deep learning research and production.

Tensorflow, being one of the most popular deep learning toolkits, has been widely used in many speech research areas. These include automatic speech recognition [134, 19], speaker recognition [118, 45], speaker diarization [119], etc. The support for programmable network structure greatly facilitates development and testing of a research idea, adding to the booming of deep learning research.

Meanwhile, another open source toolkit, Kaldi, has been very popular in the field of speech research [81]. As we have introduced in earlier chapters, Kaldi has a set of useful tools written in C++ that helps to build efficient ASR systems conveniently. It also supports speaker recognition and language recognition tasks. One particular advantage of Kaldi over other speech recognition toolkits is that many state-of-the-art recipes for various datasets are publicly available, which greatly reduces the time needed to start a project. In terms of neural network research, Kaldi has three different implementations for neural network model building and training. Out of these three, the "nnet3" is intended to support programmable network structures, i.e., model building that does not require any actual coding.

Compared with Kaldi's nnet3, the Tensorflow toolkit is more actively developed and maintained. By introducing Tensorflow Lite, it becomes possible to deploy deep learning algorithms on mobile and embedded devices. Also the Tensor Processing

Unit (TPU) that has been developed by Google makes Tensorflow a promising tool for deep learning applications (for those who have access to a TPU).

My work here aims to bridge the gap between Tensorflow and Kaldi. By using Tensorflow for acoustic modeling and Kaldi for all other speech related tasks, one may exploit the convenience brought by both toolkits and conduct rapid deep learning research. TIK, which stands for Tensorflow integration with Kaldi, is developed to address this need. This open source tool uses the Tensorflow toolkit for deep learning based acoustic modeling, and connects to the Kaldi toolkit for alignments, ASR decoding, and scoring, etc. In terms of applications, it supports both speech and speaker recognition. It also has recipes for switchboard and SRE 2010 data sets. With this framework in place, researchers and developers using Kaldi will be able to use TensorFlow to explore deep learning models in Kaldi speech recognition pipelines.

I will briefly cover related work in Section 5.2. Then, I will introduce my design of the experimental framework in 5.3, which supports both speech and speaker recognition. This framework is built on top of the Tensorflow toolkit for neural network modeling, and it connects to the Kaldi toolkit for alignments, ASR decoding, and scoring, etc. In Section 5.4, I will show state-of-the-art experimental results in speech and speaker recognition using my implementation. Finally, I will describe the multi-GPU training implementation in Section 5.5 and show its effectiveness on acceleration of model training.

5.2 Related Work

In 2016, an open source tool called "tfkaldi" was developed by V. Renkens [87]. This tool contains a set of Python scripts that support training and decoding of an end-to-end ASR model. It uses Tensorflow for acoustic model building and training. During decoding, frame-likelihoods are dumped into files and passed into a Kaldi decoder for transcriptions. Later, this project evolved into another tool called "Nabu" that does end-to-end ASR with neural networks [88]. Nabu is an experimental framework built on top of Tensorflow that focusses on adaptability, and it no longer connects to the Kaldi toolkit, nor does it require Kaldi scripts / binaries. Some other ASR projects developed using Tensorflow also follow the end-to-end approach [75], and they only work on ASR tasks.

In another open-source project, "tfdnn-kaldi", the author implemented a simple DNN for acoustic modelling for ASR [107]. This project also uses Kaldi's decoder by dumping frame-likelihoods into files. It does not support any network structure other than a feed-forward DNN.

In 2017, a RNNLM extension to Kaldi was developed by Y. Carmiel and H. Xu [126]. This extension uses Tensorflow for neural network language modeling, and supports rescoring of decoded word lattices generated by Kaldi. Though this extension is reported to have "Tensorflow integration", it only supports language

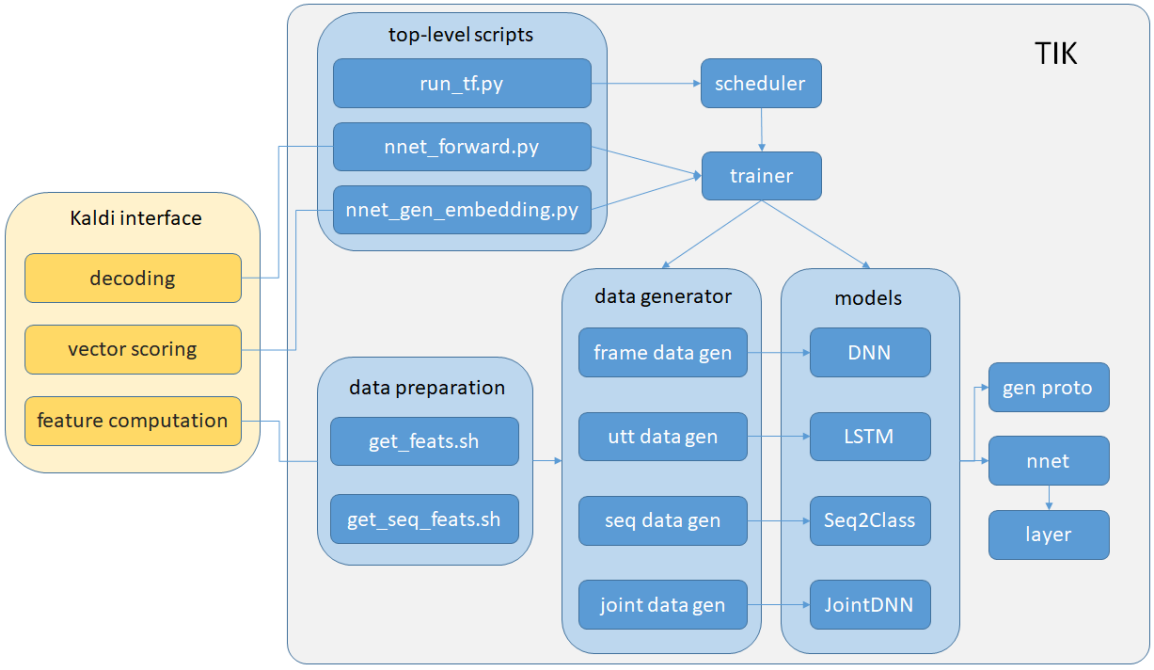

Figure 5.1. System Design of TIK

modeling and lattice rescoring, which limits the capability of Tensorflow.

Another speech recognition toolkit EESEN [68] was built out of the Kaldi toolkit. This tool develops end-to-end acoustic model in C++ on top of Kaldi's matrix library, and uses Kaldi's WFST decoder for decoding. As it does not connect to any flexible deep learning toolkit, it does not support programmable network structure.

My work in this chapter differentiates itself from existing tools in 3 aspects:

- It supports acoustic modeling using Tensorflow
- It integrates with Kaldi decoder through a pipe
- It covers both speech and speaker recognition tasks

5.3 A Tensorflow framework

The TIK implementation is separated into six components: top-level scripts, data preparation scripts, data generators, network models, network trainer and other helper modules. The overall system design is shown in Figure 5.1.

5.3.1 Network trainer

The network trainer is designed as the core of the framework, and all modules are implemented to support the trainer. This abstract class is responsible for the following tasks:

- Initialize / write / read neural network models
- Train model through one iteration of data
- Perform forward pass on given input data

This class is instantiated in all top-level scripts so operations on the network model could be done.

5.3.2 Data Preparation

The amount of training data for speech applications are often quite significant, ranging from 100 million to a few billions samples, which makes it impossible to load them all into the memory during model training. To speed up the entire training process, these data are scheduled to be loaded into memory in blocks. Data generators are designed to perform these tasks. During each iteration, data generators are responsible for loading blocks of training data from a file into the memory one at a time, pack them into numpy `ndarray` format, and provide them to the trainer when it requires them. These data generators keep processing until they run out of data, and then instruct the trainer to end this iteration.

Training data randomization is another important point to note as it has been proven to be critical for training of neural networks [12, 9, 99]. This is because for mini-batch stochastic gradient descent method, it is important that each minibatch approximately represent data distribution of the full data set. However, random access to data samples during training is usually expensive, so it is necessary to perform the randomization before the training starts. Randomization is done in two levels: file randomization and memory randomization. `get_feats.sh` and `get_seq_feats.sh` are coded to perform utterance level randomization, and save randomized data to file before any training scripts are called. During training, we also perform frame / utterance level randomization in memory.

5.3.3 Model and proto file

In TensorFlow, a neural network model is represented as a dataflow graph that can perform computation in terms of the dependencies between individual operations. Nodes in this graph could either be tensors that store model parameters or operators

that perform computations. A sample Tensorflow graph for an LSTM model is shown in Figure 5.2 [1].

In my implementation, 4 types of models are supported: DNN, LSTM, SEQ2CLASS and JOINTDNN. DNN and LSTM are designed for ASR acoustic modeling, and SEQ2CLASS model is designed for x-vector speaker recognition, which will be introduced in Section 5.4.3. JOINTDNN model is a joint model for ASR and speaker recognition, and this model will be covered in Chapter 6.

Neural network proto files are required to initialize these models. The proto file specifies the network structure and model-related hyper-parameters, including number of hidden layers, hidden notes, or number of cells. The use of proto file facilitates experiments with different model hyper-parameters. A sample proto file for bi-directional LSTM is shown below:

```
<NnetProto>
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<BLSTM> <NumCells> 1024 <UsePeepHoles> True
<AffineTransform> <InputDim> 1024 <OutputDim> 8815 <BiasMean> 0.000000 \
<BiasRange> 0.000000 <ParamStddev> 0.049901
</NnetProto>
```

Once a proto file is given, it is passed to "nnet" and "layer" module to construct the model graph. During training, gradient-related operators are added to the graph and used to update the model.

5.3.4 Training and scheduler

Training a neural network using stochastic gradient descent (SGD) usually requires working through data samples for several passes. To achieve better convergence, change of learning rates during training are necessary. Schedulers are designed to support various learning rate schedules. Currently, two schedulers are implemented: `exponential_scheduler` and `newbob_scheduler`. For the first one, learning rate for model updates decay exponentially from the initial value to the final value settled beforehand, as is shown below.

$$\mathtt{lr}_i = \mathtt{lr}_{init} \cdot (\mathtt{lr}_{final}/\mathtt{lr}_{init})^{\frac{i}{iters}} \tag{5.1}$$

Here $\mathtt{lr}_i$ is the learning rate for iteration i, $\mathtt{lr}_{init}$ and $\mathtt{lr}_{final}$ are the initial and final learning rates, and $iters$ are total number of iterations for training the network.

[1] This graph is generated by the Tensorboard

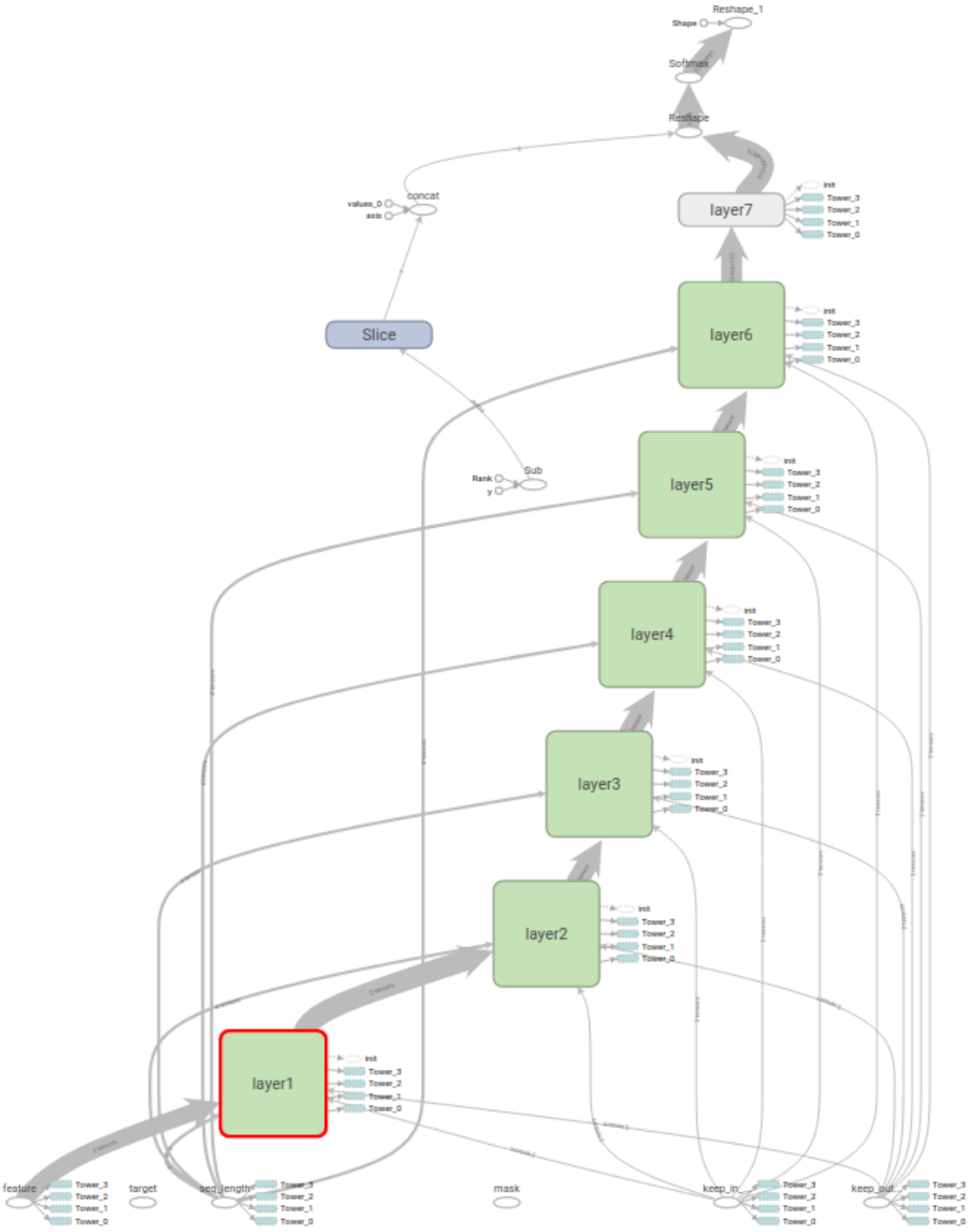

Figure 5.2. Sample Tensorflow graph for an LSTM model

The newbob scheduler works in the way introduced in Section 4.3. In short, it starts with a fixed initial learning rate for the first few iterations, and then begin halving by half when relative improvement of loss on cross-validation set is less than a pre-set value. The training process goes on until relative improvement of the loss on cv set becomes less than another value.

5.3.5 Connection to Kaldi

TIK connects to Kaldi via pipes. These include the following scenarios:

- Load data from disk into memory for model training.
- Predict frame log-likelihood for ASR decoding.
- Generate speaker embeddings for speaker recognition.

By using pipes for integration, fewer intermediate files are necessary.

5.4 Experiments and Results

With the design of TIK in place, one can easily construct deep learning models for ASR and speaker recognition. In this section, I will present experiments using TIK.

5.4.1 DNN and LSTM for ASR

As is mentioned earlier in Section 2.1, the GMM-HMM and hybrid approach are two basic approaches for ASR. For the hybrid approach, DNN is the most basic setup. Apart from DNN, other neural network structures also show great potential for acoustic modeling, including the time delay neural network (TDNN) [117, 78], maxout neural network [133] and long-short term memory neural network (LSTM) [73]. In this section, I will present experiments using feed-forward DNN models and bi-directional LSTM models.

A traditional tri-phone GMM-HMM system serves as our baseline system, and its training framework follows the standard pipeline in the Kaldi toolkit [81]. The standard 13-dim MFCC feature and its delta and acceleration are extracted and used for maximum likelihood GMM model training. Features are then transformed using Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transformation (MLLT) and fMLLR.

With the baseline GMM-HMM system developed, I then use frame alignments generated from the system to train a DNN-HMM hybrid system [116]. 40-dimension fMLLR features are used as inputs to the DNN, and HMM state alignments are used as targets. Kaldi's DNN-HMM system was trained using RBM-based pretraining

and then stochastic gradient descent, while TIK's DNN system does not require any pretraining. The DNN used in this experiment has 6 hidden layers with 2048 hidden nodes in each layer. Sigmoid functions are used as nonlinear activations.

Following the recipe proposed in [73], I develop a deep bi-directional LSTM (BLSTM) RNN model. In this implementation, BLSTM is used to model short segments of speech (40 frames), and predict frame-level HMM states. The model has 6 hidden layers and 512 LSTM cells for each direction. A standard dropout mechanism, peephole connection and gradient clipping are adopted to stabilize model training. Like in DNN-HMM training, fMLLR feature and HMM state alignments are used as inputs and targets respectively.

	development		Eval2000		
	XENT	acc	swbd	callhome	all
Kaldi GMM	–	–	21.4	34.8	28.2
Kaldi DNN	1.800	54.78	14.9	25.6	20.3
TIK DNN	1.875	53.97	14.5	25.5	20.0
TIK BLSTM	1.067	72.50	13.6	24.3	19.0

Table 5.1. Speech Recognition Performance

Table 5.1 shows that TIK DNN shows comparable ASR accuracy as Kaldi DNN, with a WER of 14.5 on Switchboard portion of Eval2000. The BLSTM model further reduces this WER to 13.6. On Callhome portion of Eval2000, similar patterns are observed.

5.4.2 DNN + i-vector for Speaker Recognition

For speaker recognition, a DNN+i-vector approach and x-vector approach are tested.

Importing DNN posteriors to an i-vector framework is shown to be effective on improving speaker recognition performance, which has been described in Chapter 3. Here, I reproduce state-of-the-art speaker recognition performance using a DNN trained with the TIK toolkit, and compare the results with Kaldi's sre10 recipe. DNNs trained on Switchboard and Fisher English data sets are compared. This comparison helps to understand how speaker recognition benefit from DNNs trained with more speech data.

Table 5.2 shows the comparison between Kaldi and TIK systems. Here the UBM system is the baseline i-vector system combined with UBM. SUP-GMM stands for supervised-GMM method, which is introduced in [104]. This approach creates a GMM based on DNN posteriors and speaker recognition features, so as to model phonetic content in a lightweight sense. It requires less computation during enrollment and testing phase compared to SUP-DNN approach.

	Cosine	LDA	PLDA
Kaldi UBM	6.91	3.36	2.51
Kaldi SUP-GMM-swbd	6.07	2.75	1.94
Kaldi SUP-DNN-swbd	4.00	1.83	1.27
Kaldi SUP-GMM-fisher	6.11	2.65	2.02
Kaldi SUP-DNN-fisher	3.54	1.60	1.21
TIK SUP-GMM-swbd	6.19	2.75	1.97
TIK SUP-DNN-swbd	4.53	2.00	1.27
TIK SUP-GMM-fisher	6.19	2.73	1.95
TIK SUP-DNN-fisher	3.93	1.74	1.19

Table 5.2. Speaker Recognition Performance using Kaldi and TIK

SUP-DNN is used to denote our DNN + i-vector approach. As we can see from the table, SUP-DNN systems generally perform better than SUP-GMM systems. Also, by comparing DNNs trained with Switchboard and Fisher data set, we can observe similar performance. This indicates that the DNN + i-vector approach may not benefit from a huge data set for DNN training.

5.4.3 X-vector for Speaker Recognition

As is mentioned in 2.2.6, using the x-vector for speaker recognition is proposed in [106] to extract speaker embeddings from segments of speech. The network takes in features of speech segments, passes them though several hidden layers and trains against speaker labels at the segment-level. Even though this approach does not perform better than the now traditional i-vector approach given same amount of training data, its capability to effectively exploit data augmentation helps it surpasses i-vector systems. A typical x-vector model is shown in Figure 5.3. Here Relu stands for rectified linear unit [66], which is an activation function that performs

$$f(x) = max(0, x) \tag{5.2}$$

and Batchnorm stands for batch normalization [117], which is proposed to accelerate the training process.

In this section, I implemented a network structure that is similar to Kaldi's setup, and achieved reasonable speaker recognition performance. Some differences between them are detailed below.

Firstly, due to restrictions on graph compilation in Tensorflow, the dimension of the inputs for the graph must be determined beforehand. To handle speech segments of variable length, the so-called bucket-training was implemented to address this issue. To be specific, speech segments for training are placed into different buckets according to their lengths. During graph compilation, input placeholders of different bucket sizes are defined and connected to the model. In the training phase, speech

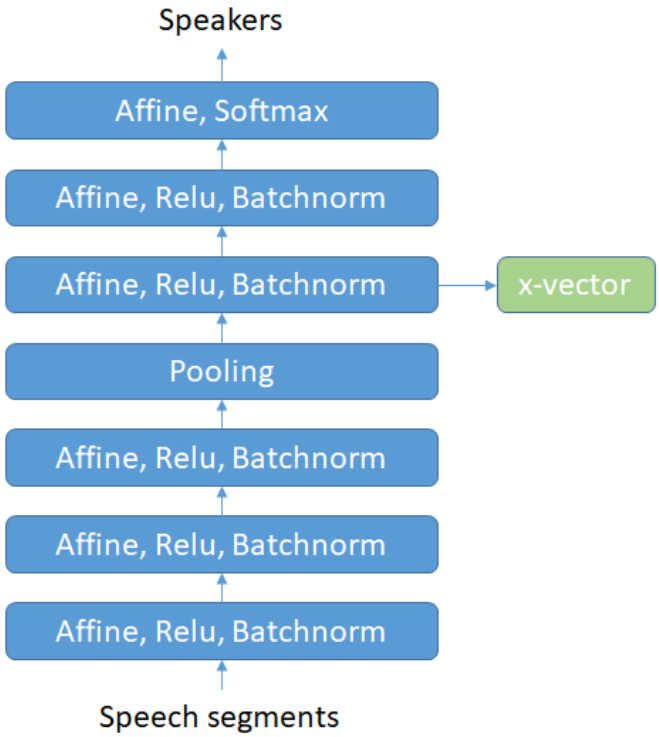

Figure 5.3. Structure of x-vector model in Kaldi

segments in the same bucket are passed into the network trainer in batches to perform SGD based model update. This speeds up training and helps the model converge. Buckets for testing are also necessary, as test speech segments are of variable lengths. For SRE 2010 data set, some speech segments may last 30k frames, which makes graph compilation a big problem. In TIK, this problem is solved by cutting speech segments into shorter ones. X-vectors are computed over each segments, and then averaged to get the final embedding. Maximum speech segments supported by the graph is set to 10k frames.

Another difference exists in the model structure. In my setup, the relu nonlinearity is found to be unstable for x-vector training. So I chose to use the sigmoid as the activation function for hidden layers. Batch normalization is used for the second part of the model as we observe better convergence in this setup. Model structure for my x-vector implementation is shown in Figure 5.4.

Table 5.3 shows the EERs of x-vector experiments on the standard SRE10 test set. As is shown in the table, both x-vector systems perform a bit worse than the i-vector system. The TIK system gives less accurate recognition performance compared to Kaldi's implementation, mostly due to differences mentioned earlier. Figure 5.5 shows the DET curve for these three systems.

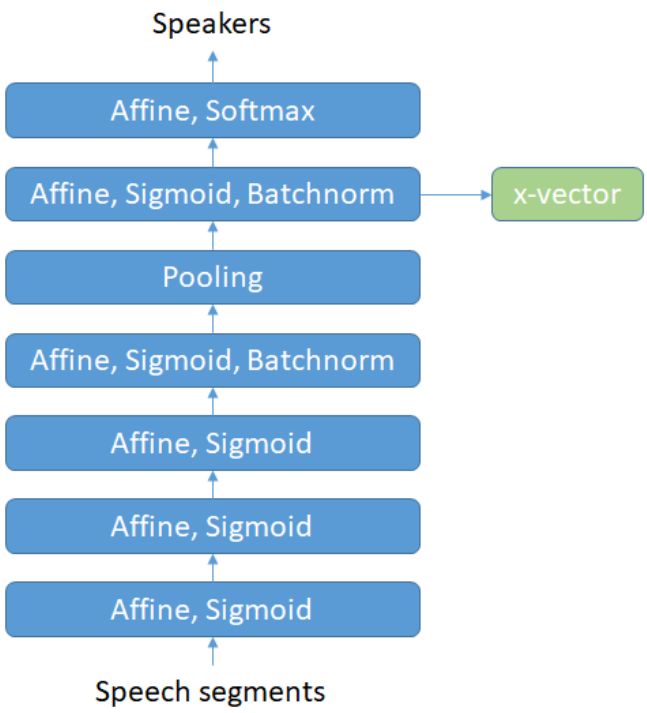

Figure 5.4. Structure of x-vector model in TIK

	Cosine	LDA	PLDA	LDA-PLDA
Kaldi UBM i-vector	6.91	3.36	2.51	2.86
Kaldi x-vector	45.36	8.05	3.52	3.08
TIK x-vector	41.23	7.99	4.85	4.53

Table 5.3. Comparing TIK and Kaldi's x-vector EER

5.5 Multi-GPU training

5.5.1 Introduction

Since the resurgence of deep neural networks, large / complex models and massive training data have been the top two driving forces for great recognition performance. However, these two factors also slow down the training procedure.

Parallelization of DNN training has been a popular topic since the revival of neural networks. Several different strategies have been proposed to tackle this problem. Multiple thread CPU parallelization and single GPU implementation are compared in [97, 115], and it is shown that a single GPU version could beat multi-threaded CPU implementation by a factor of 2. DistBelief proposed in [24] reports that 8 CPU machines train 2.2 times faster than a single GPU machine on a moderately sized

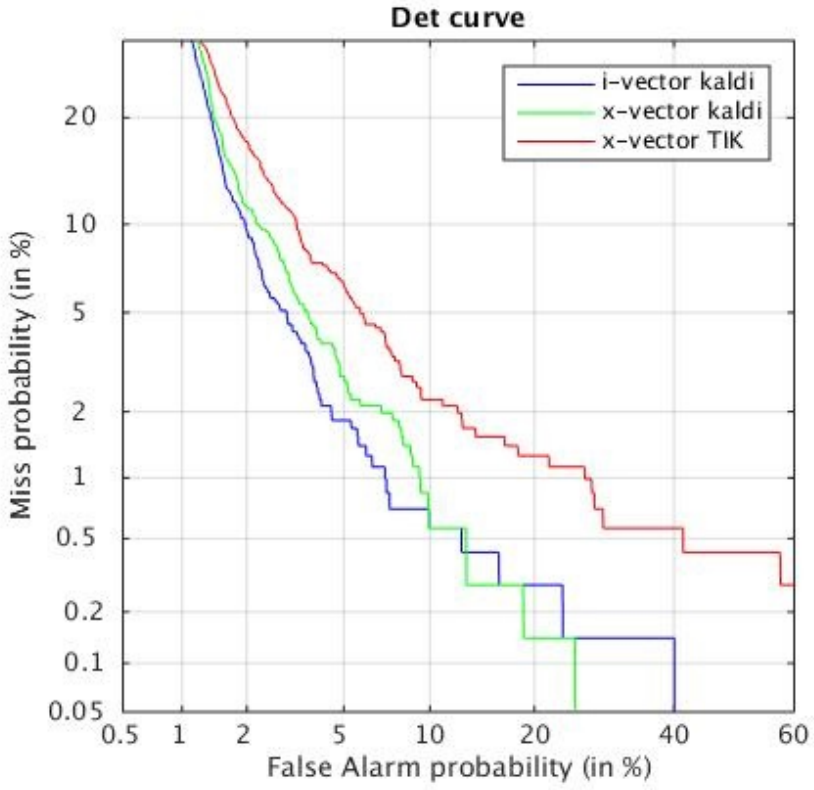

Figure 5.5. DET curve for i-vector, Kaldi's x-vector and TIK's x-vector systems

speech model. Asynchronous SGD using multiple GPUs achieved a 3.2x speed-up on 4 GPUs [131].

Distributed model averaging is used in [133, 69], and a further improvement is done using NG-SGD [84]. In this approach, separate models are trained on multiple nodes using different partitions of data, and model parameters are averaged after each epoch. It is shown that NG-SGD can effectively improve convergence and ensure a better model trained using the model averaging framework.

My work in [108] utilizes multiple GPUs in neural networks training via MPI, which allows the training process to perform model averaging more frequently and efficiently. I also compared the "all-reduce" strategy with "butterfly", which reduces the bandwidth requirement. On the 300h Switchboard dataset, 9.3x and 17x speedups could be achieved using 16 and 32 GPUs respectively.

In the next section, I will briefly introduce my implementation of multi-GPU training in TIK, and then present experimental results comparing models trained in parallel and using single GPU.

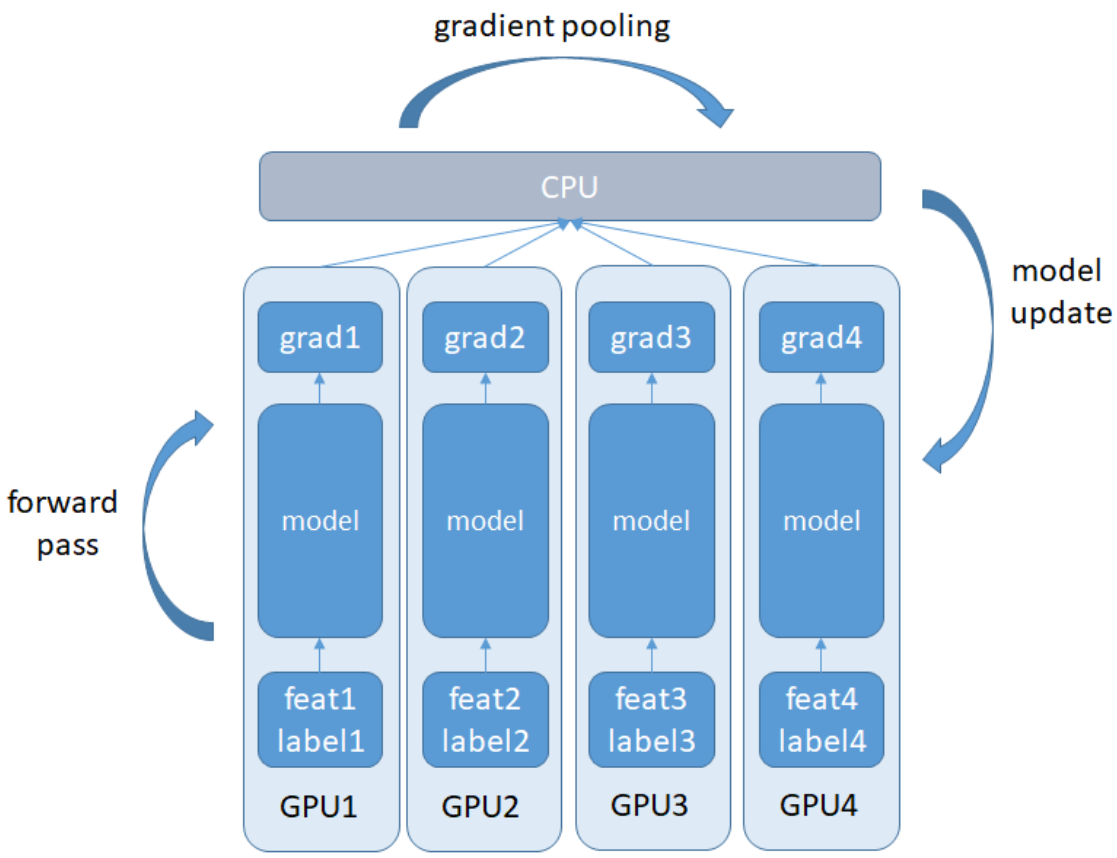

Figure 5.6. Multi-GPU training of deep learning models

5.5.2 Multi-GPU training in Tensorflow

Figure 5.6 shows the pipeline of my implementation of multi-GPU training. The data are first prepared in CPU memory using a data generator. Then these data are split into batches and sent to different GPUs. Each GPU is responsible to perform a forward and backward pass using its portion of data. Gradients computed by each GPU are copied to CPU for pooling. The pooled gradients are then sent back to each GPU for model update. Losses and other training statistics on these batches are also gathered and pooled in the CPU.

5.5.3 Experiments and Analysis

Table 5.4 shows effectiveness of multi-GPU training. As is shown in the table, the final ASR performances using different numbers of GPUs are about the same. On the other hand, by exploiting more GPUs, the time for model training can be greatly reduced.

GPUs		1	2	4	8
training	batch size	128	128*2	128*4	128*8
speedup/iter	training	1	1.98	3.50	5.21
	validation	1	1.62	2.97	4.89
development	XENT	1.067	1.090	1.086	1.096
	Acc (%)	72.45	41.92	72.10	71.20
evaluation	swbd	13.6	14.0	13.8	14.0
	callhm	24.2	24.8	24.6	24.7

Table 5.4. Multi-GPU training speed up and ASR performance

One thing to note here is that using more GPUs naturally requires a bigger minibatch for model update, and this usually causes slower convergence. One could use a smaller minibatch for each GPU during training, while this approach may reduce the speedup factor. Alternatively, a slight bigger learning rate can compensate the change in convergence speed.

For multi-GPU to be effective for training speedup, one may need to make sure the time spent on GPU is more than that spent on CPU gradient pooling.

5.6 Conclusion

In this work, I design an open-source experimental framework, TIK, that connects Tensorflow and Kaldi. This framework utilizes both toolkits to facilitate deep learning research in speech and speaker recognition. Details of the design are covered, and state-of-the-art experimental results are presented. Effectiveness of multi-GPU training are also explored. It is shown that using 8 GPUs at the same time may be able to speed up the training process by a factor of 5.

Some directions for future developments include:

- Add support for bottleneck neural network and end-to-end CTC model for ASR
- Tuning x-vector setup and add end-to-end approaches for speaker ID
- Expand to other speech research areas like language recognition and speech diarization.

Chapter 6

Joint Modeling of Speaker and Speech

6.1 Introduction

In the last few sections, I have been discussing speech and speaker recognition separately. I showed that acoustic models trained for ASR can effectively be used to improve speaker recognition performance, and speaker i-vectors based speaker adaptation are effective in improving ASR accuracy. In this chapter, I seek to combine ASR and speaker recognition, and find a solution that tackles both tasks at the same time.

The connections between speech and speaker recognition has long been recognized by researchers. However, compared with the efforts spent on speaker adaptation and speaker recognition, fewer attempts have been made on joint modeling of speech and speaker. There are many reasons leading to this:

1. Most research projects are set up to study one particular problem rather than tackling two or more of them all together.

2. Focusing on ASR or speaker recognition individually usually yields better results compared to multi-tasking.

3. Speech data sets are mostly designed to address either ASR or speaker recognition, which makes it hard to conduct research on joint modeling.

Despite of all these reasons, joint-modeling itself is an important topic to study. Firstly, the human brain is able to perform two or more task simultaneously using unconscious mind. When picking up phone calls, or listening to TV shows / radios, people tend to recognize the speaker and contents at the same time. Secondly, as we have shown, ASR and speaker recognition can be beneficial to each other, so it is natural to think that a joint model could benefit from learning to perform both

tasks. Last but not least, this topic itself is worthy of exploration. In a recent paper published by Google Research, the authors propose to use one deep learning model to perform 8 tasks at the same time [53]. This motivates us to seek better joint-modeling methods for speech.

In this chapter, I propose a JointDNN model to perform ASR and speaker recognition together. This model is built using the TIK toolkit developed in Chapter 5, and is tested on popular data sets for both ASR and speaker recognition. Related work is introduced in Section 6.2, followed by the design and implementation of a JointDNN in Section 6.3. Experimental setup and results for the joint-modeling are presented in Section 6.4 and Section 6.5 respectively. Section 6.6 concludes this chapter.

6.2 Related Work

Multi-task learning (MTL) is probably the most straight-forward approach for joint modeling. In this approach, a network is trained to perform both the primary classification task and one or more secondary tasks using some shared representations. The idea behind this approach is that secondary tasks are beneficial for neural networks to gain useful information from the features, yielding a better representation of the data.

The topic of MTL has been studied in much of the deep learning literature. For example, in [100], the authors propose to use phone or state context prediction as a secondary task in additional to the main task of predicting context-independent states. In [76] and [65] where MTL is used for an isolated digits recognition, the network was trained to predict both digit labels and clean speech feature vectors given noisy feature vectors as inputs. Other multi-task learning research was conducted on noise robustness [85], semi-supervised training [110] and multi-language transfer learning [48], etc.

For joint speech and speaker recognition, a Twin-Output Multi-Layer Perceptron (TO-MLP) was proposed in [34] that can be used for both ASR and SRE. In this approach, a TO-MLP is trained for each speaker by adapting a speaker-independent MLP. These speaker-dependent TO-MLPs can then be used for speaker dependent ASR and for SRE. This approach requires training of separate neural network models for each enrolled speaker, which adds much model complexity.

Then, in 2015, a joint modeling idea was proposed in to tackle text-dependent speaker verification[16]. This work stems from the "d-vector" approach where speech features are trained against speakers [114] in a frame-by-frame fashion. The proposed j-vector model is shown in Figure 6.1. In this approach, speech feature vectors are passed into the network in batches, and the network is trained to classify speakers and phrases at the same time. During speaker enrollment and testing, activations from the last hidden layer are pooled to form j-vectors, which will then be used for scoring. It is shown that the j-vector approach can get a large improvement over other

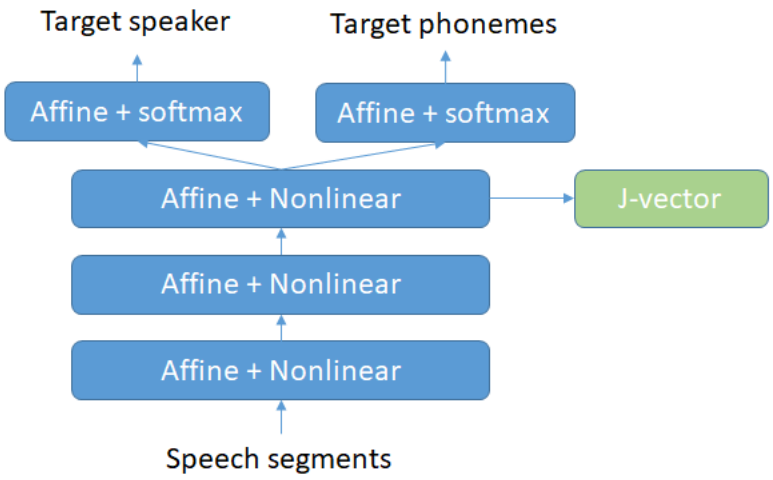

Figure 6.1. Structure of j-vector model

deep learning approaches, like d-vectors or tandem deep features, in the context of text-dependent speaker verification.

A similar network structure was utilized in [79] to perform ASR. In this paper, a RNN-LSTM is used for acoustic modeling. The neural network is trained to classify phoneme-states and speakers at the same time. An additional "SIL" class is introduced and used as the target speaker if the input is in silence at that moment. Experimental results on the TIMIT phone recognition task are reported, and it is shown that these results are comparable to traditional single task learning models.

While all the MTL research mentioned above emphasize performance on the primary task, some efforts have been made on conducting both tasks together.

In 2003, a combined system for text-dependent speaker recognition and for speech recognition was studied [10]. In this work, an artificial neural network (ANN) model and a Gaussian mixture model are combined and trained jointly. The ANN is used to recognize words and the GMM is for MAP based speaker recognition. Recognition of both the word and the speaker identity is done on the following criterion:

$$(\hat{W}, \hat{S}) = \arg\max_{\{W,S\}} \left[\log P(W|\theta_s, X) + \log P(X|\lambda_s)\right] \tag{6.1}$$

Here X denotes speech features, W denotes possible words, S denotes possible speakers, and θ_s and λ_s denote parameters for the ANN and the GMM model respectively.

In 2016, a multi-task recurrent model for speech and speaker recognition was proposed [112]. The authors of this work advocates using the output of one task as part of the input for the other. While this would not be feasible because of deadlock issue, the author chose to use output of the tasks at a previous time step instead, leading to a recurrent architecture. The structure of this implementation is

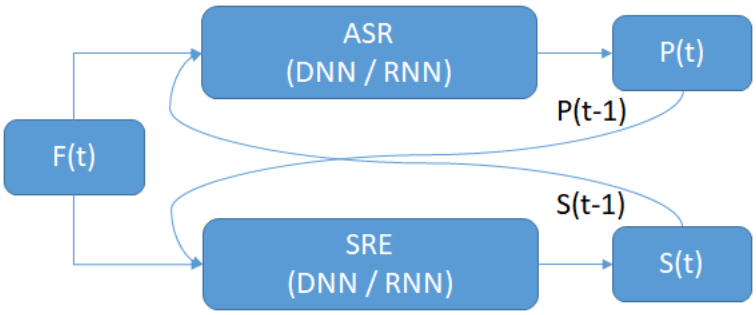

Figure 6.2. Structure of multi-task recurrent model

shown in Figure 6.2. In this plot, $F(t)$ denotes the speech feature for frame t, $P(t)$ denotes phone identities and $S(t)$ denotes speaker identities. The author of the paper conducted experiments on the WSJ data set, and showed that the joint model can achieve comparable performance as baseline LSTM and i-vector systems. Later on, this similar structure was further applied to joint language and speaker recognition [63].

My work on JointDNN model differs from the research mentioned above in a number of aspects:

1. It focuses on conversational ASR and text-independent speaker recognition.
2. It is a joint neural network model rather than a combination of different models.
3. It is evaluated on standard test sets for both ASR and speaker recognition.

6.3 Joint Modeling of Speaker and Speech

6.3.1 General Design of JointDNN

I designed the JointDNN model for ASR and speaker recognition using multi-task learning. The model takes segments of speech features as inputs, passes them through a number of shared hidden layers, and then separates out into sub-networks that predict HMM states and speaker identity respectively. The structure of the proposed model is shown in Figure 6.3. A pooling layer is placed inside the SRE sub-nnet to average out sequence of activations. These pooled activations are then passed on to predict speaker identity. During testing, the ASR sub-nnet can be used to generate frame log-likelihoods for WFST decoder, and the SRE sub-nnet can be used to generate speaker embeddings for speaker recognition, referred to as jd-vector.

Since the pooling layer reduces the dimension of activations, there will be a dimension mismatch between layers before and after pooling. To update the network in a minibatch fashion, special cares must be taken while preparing the data. To

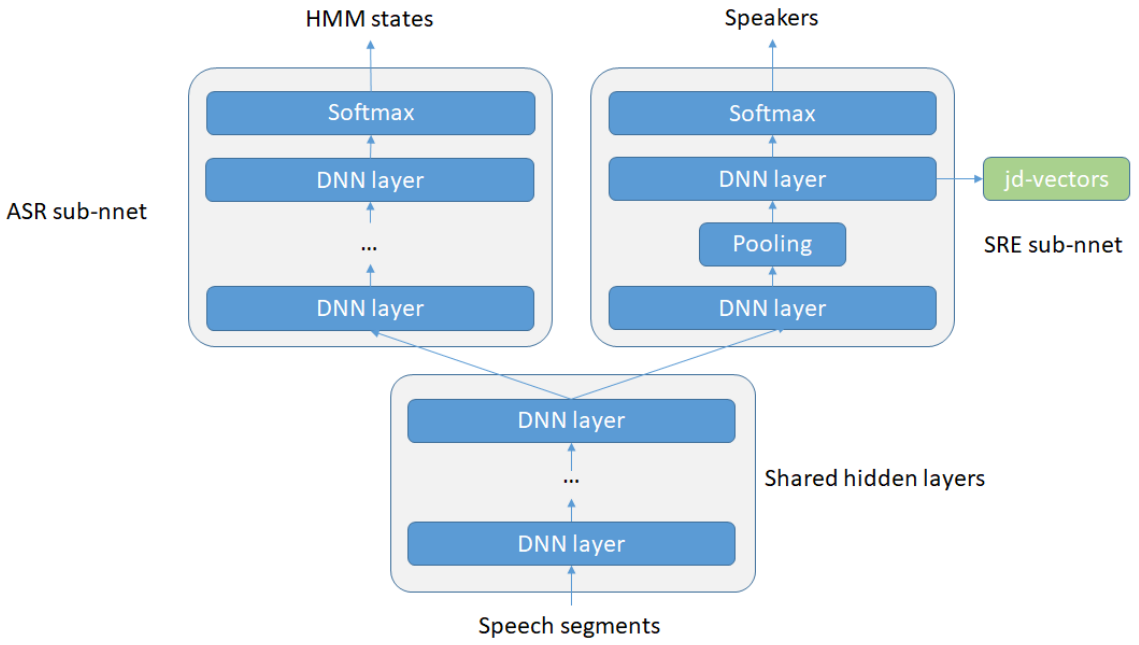

Figure 6.3. Structure of JointDNN model

be specific, input data for the network are packed into a 3-dimensional matrix of size [num_batches, segment_length, feature_dim], and activations of all DNN layers before the pooling layer [1] are of size [num_batches, segment_length, hidden_units]. The pooling layer take averages of hidden activations over speech segments, so the output of the pooling layer becomes [num_batches, hidden_units]. The labels for ASR sub-nnet and SRE sub-nnet are of sizes [num_batches, segment_length, num_states] and [num_batches, num_speakers] respectively.

6.3.2 Data preparation

Since joint modeling requires data labeled with both text and speaker information, only limited choices are available for model training. Here in this project, the Switchboard data set is chosen as the main corpus, which contains 192k utterances from 520 speakers. These utterances are randomized and saved sequentially on disk before training starts. Filter-bank features are used as input features in this project as they are low level representation of speech signals compared to other well-developed ASR or speaker recognition features.

During training, blocks of utterances are loaded into memory one at a time. The data generator packs utterances into batches before sending them over for model backpropagation. For utterances longer than a pre-defined segment length L, they

[1] including those in ASR sub-nnet

are broken into pieces using a sliding window. For those that are shorter than L, zeros are padded to the end of those utterances. The data generator also needs to prepare ASR labels and speaker labels along the way.

Special care must be taken regarding silence inside speech utterances. Since silence frames are not useful for speaker recognition, they must be excluded during pooling. To achieve this operation, a mask is prepared for each segment using pre-computed voice activity detection information.

6.3.3 Loss function

The loss function for model training is defined by:

$$L(\theta) = -\sum_{s=1}^{S}\sum_{t=1}^{s_T} h_{s,t} \log P(h_{s,t}|o_{s,t}) - \beta \sum_{s=1}^{S} x_s \log P(x_s|o_s) \tag{6.2}$$

which is an interpolation of cross-entropy losses for ASR and speaker recognition. Here $h_{s,t}$ denotes the HMM state for frame t of speech segment s, and $o_{s,t}$ is the observed feature vector that corresponds to $h_{s,t}$, x_s is the correct speaker for segment s and o_s is speech features for segment s. β is the interpolation weight.

6.3.4 Making predictions

To evaluate this model, ASR and speaker recognition are tested on standard data set. For ASR decoding, the ASR branch of the network are used to generate frame log-likelihoods. These log-likelihoods are passed into Kaldi's WFST decoder via a pipe to generate decoding outputs. For speaker recognition, activations after the pooling layer are collected as speaker embeddings, just as the way x-vector is generated. These speaker embeddings, referred to as jd-vector, is used for scoring methods, like cosine scoring or PLDA scoring.

6.3.5 Buckets for training and testing

To ensure SRE sub-nnet generalizes well to speech segments of different lengths, bucket training is implemented in a similar way as is done in Section 5.4.3 for x-vector. During data preparation, speech segments for training are fed into buckets of different sizes. Then, in training phase, speech segments in the same bucket are passed into the model trainer in batches to perform an SGD based model update. During model evaluation, buckets are also used to generate jd-vectors for speech segments of different sizes.

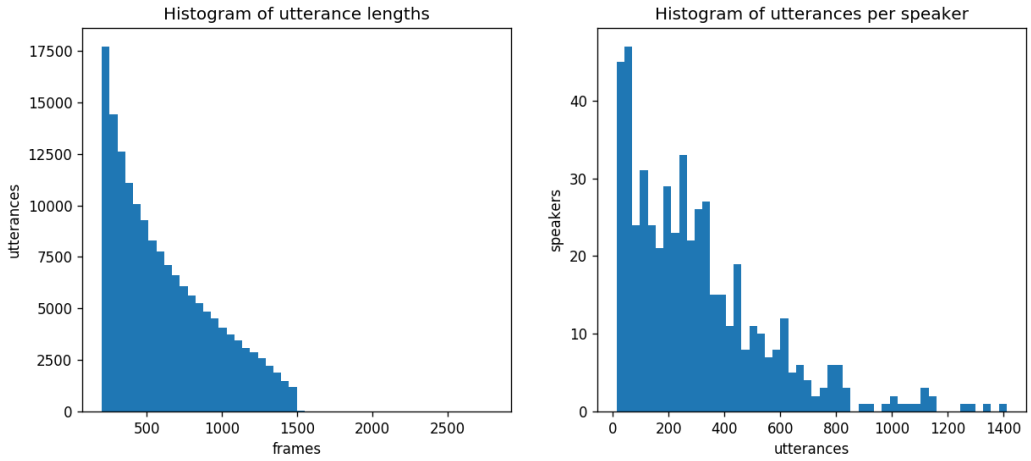

Figure 6.4. Histograms of utterance lengths and utterances per speaker

6.4 Experimental Setup

The JointDNN model used in this section has 3 shared hidden layers with 2048 hidden nodes per layer. For the ASR sub-nnet, 3 more hidden layers with 2048 nodes per layer are used, before a final softmax layer of 5238. For the SRE sub-nnet, the number of hidden nodes are projected down to 1500 before pooling. The layer after the pooling layer further reduces number of hidden nodes to 512, before sending activations to a final softmax layer. Jd-vectors are extracted after the pooling layer, which has 512 hidden nodes.

The Switchboard data set [35] is used for model training. To make sure the pooling operation in SRE sub-nnet is stable during training, I pre-select utterances that are 2 seconds or longer, which leaves 158k utterances from 520 speakers, totaling 270 hours of speech. Histograms of utterance lengths and utterances per speaker are shown in Figure 6.4.

Standard filter-bank features (24-dimension) are extracted, and then expanded to 264 dimensions to include context information by concatenating features of neighboring frames. Sliding window mean normalization is performed on the features before a global mean and variance normalization. Alignments for ASR sub-nnet is generated using a baseline GMM-HMM system. This baseline system is built following the standard Kaldi recipe for Switchboard data set, where LDA, MLLT and SAT techniques are used during model training. Newbob method is used for learning rate scheduling, and standard SGD is used for model update.

For model evalutation, the NIST 2000 Hub5 evaluation set (Eval2000) [29] is used for ASR experiments, and SRE 2010 data set [2] is used for speaker recognition experiments.

6.5 Results and analysis

6.5.1 SRE performance

Testing of speaker recognition is evaluated on the SRE 2010 data set. Table 6.1 compares recognition performances of different systems on the test set. As is shown in the table, the baseline i-vector system performs best in terms of EER. This is because the training data is very limited in this experiment - only around 300 hours in total. The i-vector model, as a generative one, usually performs better than discriminative models when limited data are available. This observation is also compatible with that published in [105]. The Kaldi x-vector system and the TIK x-vector system give comparable results regarding EER. The JointDNN implementation, however, outperformed both x-vector systems and achieves a EER of 5.30 under LDA+PLDA scoring. This result shows that phonetic content could help the JointDNN to model speaker embeddings.

	cosine	LDA	PLDA	LDA+PLDA
Baseline i-vector	28.12	6.263	4.85	4.95
Kaldi x-vector	39.60	13.78	8.94	8.89
TIK x-vector	38.61	12.54	8.16	8.80
TIK jd-vector (beta0.01)	37.45	8.20	4.75	4.85

Table 6.1. EER of JointDNN model for speaker recognition

Figure 6.5 shows the DET curve of all four systems on SRE 2010 test set. As we can see from the figure, performances of i-vector and jd-vector systems are quite close to each other. However, in the low miss-rate region, the i-vector system becomes better. In general, jd-vector performs better than both x-vector systems.

6.5.2 ASR performance

Testing of speech recognition is evaluated on an Eval2000 dataset [29]. Figure 6.2 shows the word error rate (WER) for JointDNN model. Currently, we do not observe recognition improvement over the baseline DNN model. Adding additional targets of speaker labels introduces slight negative influences on the speech recognition part [2].

6.5.3 Adjusting Beta

As is covered in the last section, the loss function for training JointDNN model is a weighted sum of ASR loss and SRE loss. Here β is used to adjust the weight

[2]Note that the WER here are worse than state-of-the-art DNN systems because we are not using any feature transformations like LDA, MLLT and SAT

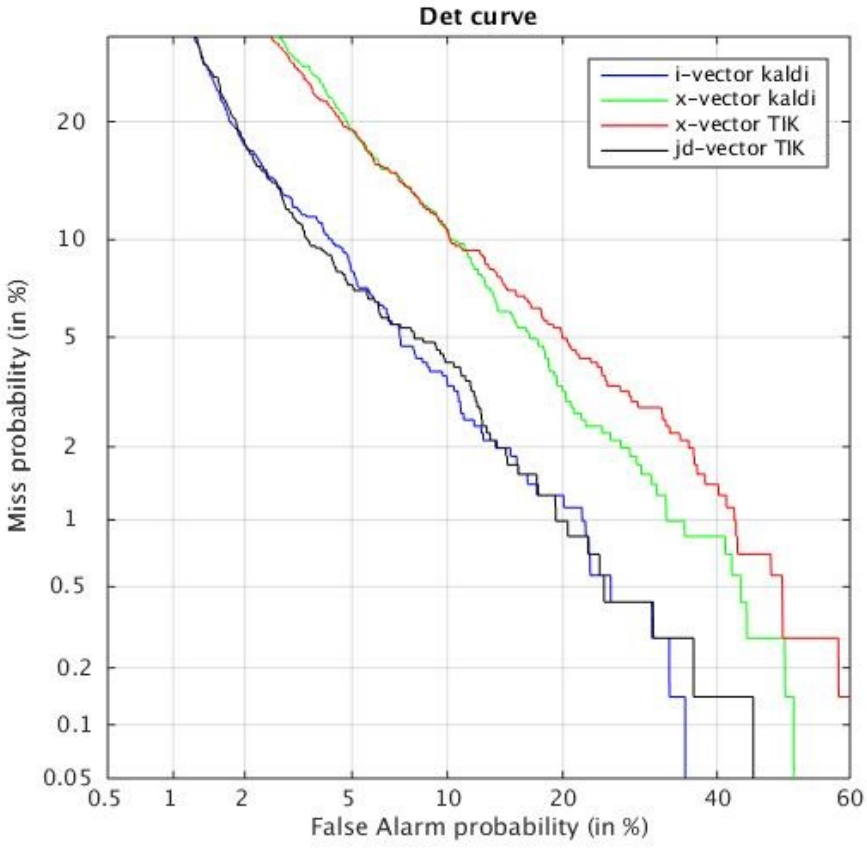

Figure 6.5. DET curve for speaker recognition performance

	swbd	callhome	all
Baseline DNN	16.1	28.4	22.3
JointDNN (beta 0.01)	16.8	29.0	23.9

Table 6.2. WER of JointDNN model for speech recognition

placed on speaker recognition. Table 6.3 shows ASR and SRE performances of models trained with different βs.

	Validation		Evaluation	
beta	frame acc (%)	speaker acc (%)	EER (%)	swbd WER (%)
0.1	39.07	97.22	5.10	16.7
0.01	39.20	94.10	4.75	16.8
0.001	38.60	85.36	9.19	17.2
0.0001	38.59	41.95	13.25	17.0

Table 6.3. EER of JointDNN model with different interpolation weight

Validation columns show frame accuracy and speaker accuracy on cross-validation set when the last iteration of model training is finished. Evaluation columns show EER on SRE 2010 test set and WER on Hub5 Eval 2000 Switchboard portion.

As is shown in the table, the SRE portion general performs better as β becomes bigger. The best SRE performance is achieved when β is set to 0.1. However, as β gets bigger, the ASR performance begins heading worse. The trade-off between ASR and SRE performances can be balanced by adjusting β.

6.6 Conclusion

I conclude the final technical chapter of this thesis by introducing a joint model for ASR and speaker recognition. This model, referred to as JointDNN, is built using the TIK tool developed in Chapter 5. Multi-task learning is used for model training. After training, the joint model can be used for both ASR and SRE. For speaker recognition, speaker embeddings, referred to as jd-vectors, are extracted for enrollment and testing. Experiments show that JointDNN model is effective in using a limited amount of training data for a neural network based speaker recognition model. On ASR, the JointDNN performs comparably to baseline DNN systems without feature transformations.

This thesis only presents some preliminary results on joint modeling of speaker and speech, yet there is much more to explore in this field. The recurrent approach proposed in [112] is novel in utilizing higher-level representations of the data, but it is slow to train because of the recurrent structure. The JointDNN model, being a feed

forward network, is much faster during training, but a bit weaker in utilizing high-level representations. The trade-off between model complexity and training speed is worth more investigation. Other possible research directions for joint modeling includes:

1. Using powerful network layers that could capture temporal information, like LSTM or TDNN;
2. Using a large amount of training data or using perturbation of training data;
3. Exploring different structures for joint modeling;
4. Using an end-to-end approach for joint modeling of speaker and speech;
5. Fine-tuning of model parameters after joint training of the model.

Chapter 7

Conclusions

7.1 Contributions

The contribution of this thesis has two components: first, I show that ASR and SRE are beneficial to each other; secondly, I build an opensource tool, "TIK", on top of which I design a joint model for ASR and speaker recognition.

In Chapter 3, I explore ways to improve speaker recognition performance using acoustic models trained for ASR. It is shown that speaker verification performance aligns well with ASR performance when posteriors are imported from acoustic models trained for ASR. This demonstrate that ASR can be used to aid speaker recognition systems.

In Chapter 4, I conduct research on speaker adaptation of DNN hybrid systems using i-vectors. A novel regularization method is introduced that helps to solve the curse of dimensionality. Experimental results show that speaker adaptation using i-vectors can effectively improve ASR performance of DNN systems, especially when regularization is placed on a "i-vector sub-network".

Chapter 5 describes the open-source tool, "TIK", that connects Tensorflow and Kaldi for deep learning research in ASR and SRE. This tool makes it easier to conduct deep learning research using flexible network structures. Design of the framework is detailed, and state-of-the-art ASR and speaker recognition results are presented.

In Chapter 6 I describe the building of a JointDNN model using the TIK tool I developed earlier. This model is trained using multi-task learning and is able to perform both ASR and SRE tasks. Experiments show that the JointDNN model is more effective in speaker recognition than x-vector systems, given a limited amount of training data.

7.2 Future Work and Beyond

Since the first generation of speech applications, spoken language technology has evolved over 60 years. Recently, the rapid popularization of smart home devices like Amazon Echo and Google Home has made it possible for humans to connect to machines through voice. Other mobile devices like cell-phones or smart watches also support more and more speech applications. Because of the huge number of devices and requests, it is usually necessary to design embedded algorithms for these devices rather than implementing all of the speech application on the server side. This will require that models take up limited space, and that algorithms do not occupy huge amounts of memory and use excessive amounts of computation. In this regard, a joint model for ASR and SRE might then become a reasonable choice. Further research on compressing network weights might contribute to the advantage of joint modeling.

The end-to-end approach also comes into play as an efficient solution. Since the resurgence of neural network research, the trend of using end-to-end neural networks for AI tasks has been kept moving forward. Even though some problems with this approach remain unsolved [1], it is worthwhile to build end-to-end joint models for speech and speaker recognition.

Joint modeling of speaker and speech only serves as a first step towards an all-around AI agent. Ideally, all speech-related AI tasks shall share some lower-level representations. Further research on joint training might consider coverage of language recognition, gender recognition, emotion analysis, etc. Training data might become a problem when more tasks are taken into consideration, so research on utilizing multiple data sets for joint model is definitely worth exploration.

[1] E.g. a solution for out-of-vocabulary words or integration of a domain-specific language model.

Bibliography

[1] Householder transformation. `http://en.wikipedia.org/wiki/Householder_transformation`. Accessed: 2016-03-23.

[2] The nist year 2010 speaker recognition evaluation plan. `http://www.nist.gov/itl/iad/mig/upload/NIST_SRE10_evalplan-r6.pdf`. Accessed: 2016-03-22.

[3] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`. Software available from tensorflow.org.

[4] Anastasakos, T., McDonough, J., and Makhoul, J. (1997). Speaker adaptive training: A maximum likelihood approach to speaker normalization. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 2, pages 1043–1046. IEEE.

[5] Anastasakos, T., McDonough, J., Schwartz, R., and Makhoul, J. (1996). A compact model for speaker-adaptive training. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 2, pages 1137–1140. IEEE.

[6] Attias, H. (2000). A variational bayesian framework for graphical models. *Advances in neural information processing systems*, 12(1-2):209–215.

[7] Bahl, L., Brown, P. F., De Souza, P. V., and Mercer, R. L. (1986). Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *proc. icassp*, volume 86, pages 49–52.

[8] Baker, J. (1975). The dragon system–an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29.

[9] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

[10] BenZeghiba, M. F. and Bourlard, H. (2003). On the combination of speech and speaker recognition. In *Eighth European Conference on Speech Communication and Technology*.

[11] Bilmes, J. A. (1998). A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126.

[12] Bourlard, H. A. and Morgan, N. (1993). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA.

[13] Brümmer, N. (2009). The em algorithm and minimum divergence. *Agnitio Labs Technical Report*.

[14] Brümmer, N. and De Villiers, E. (2010). The speaker partitioning problem. In *Odyssey*.

[15] Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE.

[16] Chen, N., Qian, Y., and Yu, K. (2015a). Multi-task learning for text-dependent speaker verification. In *Sixteenth annual conference of the international speech communication association*.

[17] Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

[18] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015b). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274.

[19] Chiu, C.-C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J., and Bacchiani, M. (2017). State-of-the-art speech recognition with sequence-to-sequence models. *arXiv preprint arXiv:1712.01769*.

[20] Cieri, C., Miller, D., and Walker, K. (2004). Fisher english training speech parts 1 and 2. *Philadelphia: Linguistic Data Consortium*.

[21] Dahl, G. E., Yu, D., Deng, L., and 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.

[22] Das, S. and Mohn, W. (1971). A scheme for speech processing in automatic speaker verification. *IEEE transactions on Audio and electroacoustics*, 19(1):32–43.

[23] Davis, S. B. and Mermelstein, P. (1990). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In *Readings in speech recognition*, pages 65–74. Elsevier.

[24] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M., Ranzato, M. A., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231.

[25] Dehak, N., Kenny, P., Dehak, R., Glembek, O., Dumouchel, P., Burget, L., Hubeika, V., and Castaldo, F. (2009). Support vector machines and joint factor analysis for speaker verification. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4237–4240. IEEE.

[26] Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798.

[27] Doddington, G. R., Przybocki, M. A., Martin, A. F., and Reynolds, D. A. (2000). The nist speaker recognition evaluation–overview, methodology, systems, results, perspective. *Speech Communication*, 31(2):225–254.

[28] Eide, E. and Gish, H. (1996). A parametric approach to vocal tract length normalization. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 346–348. IEEE.

[29] Fiscus, J., Fisher, W. M., Martin, A. F., Przybocki, M. A., and Pallett, D. S. (2000). 2000 nist evaluation of conversational speech recognition over the telephone: English and mandarin performance results. In *Proc. Speech Transcription Workshop*. Citeseer.

[30] Gales, M. and Young, S. (2008). The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304.

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

[32] Gales, M. J. (1999). Semi-tied covariance matrices for hidden markov models. *Speech and Audio Processing, IEEE Transactions on*, 7(3):272–281.

[33] Garcia-Romero, D. and Espy-Wilson, C. Y. (2011). Analysis of i-vector length normalization in speaker recognition systems. In *Twelfth Annual Conference of the International Speech Communication Association*.

[34] Genoud, D., Ellis, D., and Morgan, N. (1999). Combined speech and speaker recognition with speaker-adapted connectionist models. In *Proc. ASRU*.

[35] Godfrey, J. J. and Holliman, E. (1997). Switchboard-1 release 2. *Linguistic Data Consortium, Philadelphia*, 926:927.

[36] Goel, V. and Byrne, W. J. (2000). Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135.

[37] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

[38] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.

[39] Grézl, F., Karafiát, M., Kontár, S., and Cernocky, J. (2007). Probabilistic and bottle-neck features for lvcsr of meetings. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–757. IEEE.

[40] Gupta, V., Kenny, P., Ouellet, P., and Stafylakis, T. (2014). I-vector-based speaker adaptation of deep neural networks for french broadcast audio transcription. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6334–6338. IEEE.

[41] Haeb-Umbach, R. and Ney, H. (1992). Linear discriminant analysis for improved large vocabulary continuous speech recognition. In *ICASSP*. IEEE.

[42] Hain, T., Woodland, P. C., Niesler, T. R., and Whittaker, E. W. D. (1999). The 1998 htk system for transcription of conversational telephone speech. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 57–60 vol.1.

[43] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

[44] Hargreaves, W. A. and Starkweather, J. A. (1963). Recognition of speaker identity. *Language and Speech*, 6(2):63–67.

[45] Heigold, G., Moreno, I., Bengio, S., and Shazeer, N. (2016). End-to-end text-dependent speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5115–5119. IEEE.

[46] Hermansky, H. (1990). Perceptual linear predictive (plp) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752.

[47] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[48] Huang, J.-T., Li, J., Yu, D., Deng, L., and Gong, Y. (2013). Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7304–7308. IEEE.

[49] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM.

[50] Ioffe, S. (2006). Probabilistic linear discriminant analysis. In *European Conference on Computer Vision*, pages 531–542. Springer.

[51] Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556.

[52] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM.

[53] Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One model to learn them all. *arXiv preprint arXiv:1706.05137*.

[54] Kenny, P., Boulianne, G., and Dumouchel, P. (2005). Eigenvoice modeling with sparse training data. *Speech and Audio Processing, IEEE Transactions on*, 13(3):345–354.

[55] Kenny, P., Boulianne, G., Ouellet, P., and Dumouchel, P. (2007). Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1435–1447.

[56] Kenny, P. and Dumouchel, P. (2004). Experiments in speaker verification using factor analysis likelihood ratios. In *ODYSSEY04-The Speaker and Language Recognition Workshop*.

[57] Kenny, P., Gupta, V., Stafylakis, T., Ouellet, P., and Alam, J. (2014). Deep neural networks for extracting baum-welch statistics for speaker recognition. In *Proc. Odyssey*, pages 293–298.

[58] Kim, C. and Stern, R. M. (2012). Power-normalized cepstral coefficients (pncc) for robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4101–4104. IEEE.

[59] Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

[60] Lee, K.-F. (1990). Context-dependent phonetic hidden markov models for speaker-independent continuous speech recognition. In *Readings in speech recognition*, pages 347–365. Elsevier.

[61] Leggetter, C. J. and Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171–185.

[62] Lei, Y., Nicolas, S., Ferrer, L., and McLaren, M. (2014). A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *ICASSP*. IEEE.

[63] Li, L., Tang, Z., Wang, D., Abel, A., Feng, Y., and Zhang, S. (2017). Collaborative learning for language and speaker recognition. In *National Conference on Man-Machine Speech Communication*, pages 58–69. Springer.

[64] Liao, H. (2013). Speaker adaptation of context dependent deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7947–7951. IEEE.

[65] Lu, Y., Lu, F., Sehgal, S., Gupta, S., Du, J., Tham, C. H., Green, P., and Wan, V. (2004). Multitask learning in connectionist speech recognition. In *Proceedings of the Australian International Conference on Speech Science and Technology*.

[66] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3.

[67] Miao, Y., Gowayyed, M., and Metze, F. (2015a). Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *Automatic Speech*

Recognition and Understanding (ASRU), 2015 IEEE Workshop on, pages 167–174. IEEE.

[68] Miao, Y., Gowayyed, M., and Metze, F. (2015b). EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding. *CoRR*, abs/1507.08240.

[69] Miao, Y., Zhang, H., and Metze, F. (2014a). Distributed learning of multilingual dnn feature extractors using gpus.

[70] Miao, Y., Zhang, H., and Metze, F. (2014b). Towards speaker adaptive training of deep neural network acoustic models. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[71] Miao, Y., Zhang, H., and Metze, F. (2015c). Speaker adaptive training of deep neural network acoustic models using i-vectors. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(11):1938–1949.

[72] Mohamed, A.-r., Sainath, T. N., Dahl, G., Ramabhadran, B., Hinton, G. E., and Picheny, M. A. (2011). Deep belief networks using discriminative features for phone recognition. In *ICASSP*. IEEE.

[73] Mohamed, A.-r., Seide, F., Yu, D., Droppo, J., Stoicke, A., Zweig, G., and Penn, G. (2015). Deep bi-directional recurrent networks over spectral windows. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 78–83. IEEE.

[74] Naik, J. M. (1990). Speaker verification: A tutorial. *IEEE Communications Magazine*, 28(1):42–48.

[75] pannous. tfasr. `https://github.com/pannous/tensorflow-speech-recognition`.

[76] Parveen, S. and Green, P. (2003). Multitask learning in connectionist robust asr using recurrent neural networks. In *Eighth European Conference on Speech Communication and Technology*.

[77] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[78] Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[79] Pironkov, G., Dupont, S., and Dutoit, T. (2016). Speaker-aware long short-term memory multi-task learning for speech recognition. In *Signal Processing Conference (EUSIPCO), 2016 24th European*, pages 1911–1915. IEEE.

[80] Povey, D. (2005). *Discriminative training for large vocabulary speech recognition*. PhD thesis, University of Cambridge.

[81] Povey, D., Ghoshal, A., G.Boulianne, Burget, L., O.Glembek, Goel, N., Hannermann, M., Motlíček, P., Qian, Y., Schwartz, P., Silovský, J., Stemmer, G., and Veselý, K. (2011). The kaldi speech recognition toolkit. In *ASRU*. IEEE.

[82] Povey, D., Kanevsky, D., Kingsbury, B., Ramabhadran, B., Saon, G., and Visweswariah, K. (2008). Boosted mmi for model and feature-space discriminative training. In *ICASSP*. IEEE.

[83] Povey, D. and Woodland, P. C. (2002). Minimum phone error and i-smoothing for improved discriminative training. In *ICASSP*. IEEE.

[84] Povey, D., Zhang, X., and Khudanpur, S. (2014). Parallel training of deep neural networks with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*.

[85] Qian, Y., Yin, M., You, Y., and Yu, K. (2015). Multi-task joint-learning of deep neural networks for robust speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 310–316. IEEE.

[86] Rabiner, L., Juang, B.-H., Levinson, S., and Sondhi, M. (1985). Recognition of isolated digits using hidden markov models with continuous mixture densities. *Bell Labs Technical Journal*, 64(6):1211–1234.

[87] Renkens, V. (2016). Kaldi with tensorflow neural net. `https://github.com/vrenkens/tfkaldi`.

[88] Renkens, V. (2017). Code for end-to-end asr with neural networks. `https://github.com/vrenkens/nabu`.

[89] Reynolds, D. A. (1994). Experimental evaluation of features for robust speaker identification. *IEEE Transactions on Speech and Audio Processing*, 2(4):639–643.

[90] Reynolds, D. A. (1997). Comparison of background normalization methods for text-independent speaker verification. In *Fifth European Conference on Speech Communication and Technology*.

[91] Reynolds, D. A., Quatieri, T. F., and Dunn, R. B. (2000). Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1-3):19–41.

[92] Richardson, F., Reynolds, D., and Dehak, N. (2015a). Deep neural network approaches to speaker and language recognition. *Signal Processing Letters, IEEE*, 22(10):1671–1675.

[93] Richardson, F., Reynolds, D., and Dehak, N. (2015b). A unified deep neural network for speaker and language recognition. *arXiv preprint arXiv:1504.00923*.

[94] Robinson, T. and Fallside, F. (1991). A recurrent error propagation network speech recognition system. *Computer Speech and Language*, 5(3).

[95] Robinson, T., Hochberg, M., and Renals, S. (1996). The use of recurrent neural networks in continuous speech recognition. In *Automatic speech and speaker recognition*, pages 233–258. Springer.

[96] Saon, G., Soltau, H., Nahamoo, D., and Picheny, M. (2013). Speaker adaptation of neural network acoustic models using i-vectors. In *ASRU*, pages 55–59.

[97] Scanzio, S., Cumani, S., Gemello, R., Mana, F., and Laface, P. (2010). Parallel implementation of artificial neural network training. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 4902–4905. IEEE.

[98] Seide, F. and Agarwal, A. (2016). Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM.

[99] Seide, F., Li, G., and Yu, D. (2011). Conversational speech transcription using context-dependent deep neural networks. In *Twelfth Annual Conference of the International Speech Communication Association*.

[100] Seltzer, M. L. and Droppo, J. (2013). Multi-task learning in deep neural networks for improved phoneme recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6965–6969. IEEE.

[101] Senior, A. and Lopez-Moreno, I. (2014). Improving dnn speaker independence with i-vector inputs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 225–229. IEEE.

[102] Sizov, A., Lee, K. A., and Kinnunen, T. (2014). Unifying probabilistic linear discriminant analysis variants in biometric authentication. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 464–475. Springer.

[103] Snyder, D., Garcia-Romero, D., and Povey, D. (2015a). Time delay deep neural network-based universal background models for speaker recognition. In *ASRU*. IEEE.

[104] Snyder, D., Garcia-Romero, D., and Povey, D. (2015b). Time delay deep neural network-based universal background models for speaker recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 92–97. IEEE.

[105] Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. In *Proc. Interspeech*, pages 999–1003.

[106] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust dnn embeddings for speaker recognition. *Submitted to ICASSP*.

[107] srinivr (2016). Tensorflow dnn with kaldi. `https://github.com/srinivr/tfdnn-kaldi`.

[108] Su, H. and Chen, H. (2015). Experiments on parallel training of deep neural network using model averaging. *arXiv preprint arXiv:1507.01239*.

[109] Su, H. and Wegmann, S. (2016). Factor analysis based speaker verification using asr. In *Interspeech*, pages 2223–2227.

[110] Su, H. and Xu, H. (2015). Multi-softmax deep neural network for semi-supervised training. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[111] Swietojanski, P. and Renals, S. (2014). Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 171–176. IEEE.

[112] Tang, Z., Li, L., and Wang, D. (2016). Multi-task recurrent model for speech and speaker recognition. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2016 Asia-Pacific*, pages 1–4. IEEE.

[113] Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

[114] Variani, E., Lei, X., McDermott, E., Moreno, I. L., and Gonzalez-Dominguez, J. (2014). Deep neural networks for small footprint text-dependent speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4052–4056. IEEE.

[115] Veselỳ, K., Burget, L., and Grézl, F. (2010). Parallel training of neural networks for speech recognition. In *Text, Speech and Dialogue*, pages 439–446. Springer.

[116] Veselý, K., Hannemann, M., and Burget, L. (2013). Semi-supervised training of deep neural networks. In *ASRU*. IEEE.

[117] Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition. *Neural computation*, 1(1):39–46.

[118] Wan, L., Wang, Q., Papir, A., and Moreno, I. L. (2017). Generalized end-to-end loss for speaker verification. *arXiv preprint arXiv:1710.10467*.

[119] Wang, Q., Downey, C., Wan, L., Mansfield, P. A., and Moreno, I. L. (2017). Speaker diarization with lstm. *arXiv preprint arXiv:1710.10468*.

[120] Wang, Y.-Y., Acero, A., and Chelba, C. (2003). Is word error rate a good indicator for spoken language understanding accuracy. In *Automatic Speech Recognition and Understanding, 2003. ASRU'03. 2003 IEEE Workshop on*, pages 577–582. IEEE.

[121] Wikipedia (2018). Word error rate — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Word%20error%20rate&oldid=839599774`. [Online; accessed 07-May-2018].

[122] Wikipedia contributors (2018). Phone (phonetics) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Phone_(phonetics)&oldid=845974307`. [Online; accessed 16-June-2018].

[123] Wood, L. C., Pearce, D. J., and Novello, F. (1991). Improved vocabulary-independent sub-word hmm modelling. In *ICASSP*. IEEE.

[124] Woodland, P. C. (2001). Speaker adaptation for continuous density hmms: A review. In *ISCA Tutorial and Research Workshop (ITRW) on Adaptation Methods for Speech Recognition*.

[125] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*.

[126] Xu, H., Chen, T., Gao, D., Wang, Y., Li, K., Goel, N., Carmiel, Y., Povey, D., and Khudanpur, S. (2018). A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition.

[127] Yaman, S., Pelecanos, J., and Sarikaya, R. (2012). Bottleneck features for speaker recognition. In *Odyssey 2012-The Speaker and Language Recognition Workshop*.

[128] Yao, K., Yu, D., Seide, F., Su, H., Deng, L., and Gong, Y. (2012). Adaptation of context-dependent deep neural networks for automatic speech recognition. In *Spoken Language Technology Workshop (SLT), 2012 IEEE*, pages 366–369. IEEE.

[129] Yu, D., Yao, K., Su, H., Li, G., and Seide, F. (2013). Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7893–7897. IEEE.

[130] Zhang, C. and Koishida, K. (2017). End-to-end text-independent speaker verification with triplet loss on short utterances. In *Proc. of Interspeech*.

[131] Zhang, S., Zhang, C., You, Z., Zheng, R., and Xu, B. (2013). Asynchronous stochastic gradient descent for dnn training. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6660–6663. IEEE.

[132] Zhang, S.-X., Chen, Z., Zhao, Y., Li, J., and Gong, Y. (2016). End-to-end attention based text-dependent speaker verification. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*, pages 171–178. IEEE.

[133] Zhang, X., Trmal, J., Povey, D., and Khudanpur, S. (2014). Improving deep neural network acoustic models using generalized maxout networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 215–219. IEEE.

[134] Zhang, Y., Chan, W., and Jaitly, N. (2017). Very deep convolutional networks for end-to-end speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4845–4849. IEEE.